מכון ויצמן למדע

WEIZMANN INSTITUTE OF SCIENCE

Thesis for the degree
Doctor of Philosophy

חיבור לשם קבלת התואר
דוקטור לפילוסופיה

By
Tal Moran

מאת
טל מורן

קריפטוגרפיה בידיי העם, למען העם

Cryptography by the People, for the People

Published papers format

Advisor
Prof. Moni Naor

מנחה
פרופ' מוני נאור

July 2008

תמוז התשס"ח

Submitted to the Scientific Council of the
Weizmann Institute of Science
Rehovot, Israel

מוגש למועצה המדעית של
מכון ויצמן למדע
רחובות, ישראל

Abstract

In this thesis we investigate methods for designing and analyzing cryptographic schemes which take into account the differences between humans and computers. On the one hand, humans have trouble performing complex algorithms and mathematical computations. On the other, humans live in a physical world, and can make use of "real" objects in ways that purely electronic algorithms cannot.

Tamper-Evident Seals. In the first part of the thesis (Chapters 2 and 3), we construct cryptographic protocols that rely on the properties of everyday objects (such as locks, envelopes or scratch-off cards). In Chapter 2, we formally define the properties of tamper-evident seals. We consider several variants of the "intuitive" definition, and study their relation to basic cryptographic primitives (such as fair coin-flipping, bit-commitment and oblivious transfer), giving both positive and negative results. In Chapter 3, we use scratch-off cards and envelopes to construct protocols for securely conducting polls of sensitive questions: the responders can safely answer truthfully while maintaining *plausible deniability* (the pollster will never be certain of their actual answer), but at the same time cannot bias the results of the poll. The protocols are simple enough to be used in practice.

Voting Protocols. The distinction between the capabilities of humans and computers is especially evident in the context of voting, where voters need to be certain that their ballots were taken into account, even if they mistrust the computers running the election. The second part of the thesis (Chapters 4 and 5) deals specifically with protocols for secure elections. We present two protocols for universally-verifiable voting; both allow voters to verify that their ballots were cast correctly and allow anyone to verify that the published final tally is correct — even if the computers running the election behave maliciously. Both protocols also have the property of *everlasting privacy*: the data published during the election contains no information at all about voters' choices (beyond the final tally). This ensures that even if the cryptographic assumptions used by these protocols are subsequently broken, voter privacy will not be violated.

Formal Analysis. One of our major aims is to provide rigorous proofs of security for cryptographic protocols involving humans. We give formal proofs of security for our protocols in the Universal Composability model, which gives very powerful guarantees about the security of the protocols when run concurrently with other protocols and when used as building-blocks in larger protocols.

The following published papers are included in this thesis:

Chapter 2: Tal Moran and Moni Naor. Basing cryptographic protocols on tamper-evident seals. In *ICALP 2005*, volume 3580 of *LNCS*, pages 285–297, 2005.

Chapter 3: Tal Moran and Moni Naor. Polling with physical envelopes: A rigorous analysis of a human-centric protocol. In *EUROCRYPT 2006*, pages 88–108, 2006.

Chapter 4: Tal Moran and Moni Naor. Receipt-free universally-verifiable voting with everlasting privacy. In *CRYPTO 2006*, volume 4117 of *LNCS*, pages 373–392, 2006.

Chapter 5: Tal Moran and Moni Naor. Split-ballot voting: Everlasting privacy with distributed trust. In *CCS 2007*, pages 246–255, 2007.

Acknowledgments

Like learning to juggle, starting out in research is a lot easier if your mentor can catch everything you throw — and pass it back in a perfect parabola. I feel privileged to have had such a mentor in Moni Naor, my thesis advisor. Most of the balls I'm holding aloft myself can be traced to the exhilarating passing sessions that conversations with Moni usually resemble. My deepest thanks to Moni for the many balls he threw my way, for his guidance and for being willing to talk even at odd hours.

I would like to express my gratitude to Josh Benaloh, my mentor during a delightful summer at Microsoft, who took me under his wing and to the other side of the world.

Many thanks to Adi Shamir for his comments, support and for appearing in the audience, asking insightful questions, no matter where in the world I gave a talk.

I would like to thank Ronen Shaltiel for his support and encouragement. After talking to Ronen I always find myself looking at things from a new perspective.

I also thank the faculty and students at the Weizmann Institute, in whose company even stale cookies can be a source of inspiration. Thanks to my office-mates, past and present: Yuval Emek, Ariel Gabizon, Ronen Gradwohl, Iftach Haitner, Danny Harnik, Erez Kantor, Dana Moshkovitz, Gil Segev and Amir Yehudayoff. Special thanks are due to Gil, a coauthor and academic sibling, with whom collaborating has been both fruitful and fun. Heartfelt thanks to Dana, with whom I've shared a cubicle, an academic path and many deep discussions about life, the universe and everything.

Thanks to my family, friends and Simba the cat, for their unwavering support and for his help in keeping my keyboard warm during cold nights. Finally, deepest thanks to my Maya, for her love and understanding.

Contents

Chapter 1

Introduction

The 2000 U.S. elections were plagued by highly publicized failures in several voting systems. In response, election boards across the U.S. replaced mechanical machines and hand-counted ballots with new, electronic voting machines. However, the result was not necessarily an improvement: in 2007, a panel of cryptography and security experts was commissioned by California's Secretary of State to conduct a detailed study of the voting machines from the three major manufacturers. Their findings led to the decertification of all three models due to critical security vulnerabilities.

This example highlights three points that motivate my research in general, and underpin the results that constitute my thesis:

1. Electronic systems are becoming steadily more pervasive elements of our society's infrastructure, and thus we are increasingly reliant on their security. While their capabilities and functionality are growing rapidly, their security is not keeping pace. Moreover, in the "analog" physical world, small errors usually have minor consequences (e.g., stuffing one ballot box is unlikely to change the outcome of an election). In the digital realm, on the other hand, a tiny change can have a huge effect (one bad line of code can change the outcome completely and undetectably).

2. Our intuitions of security for physical systems often fail to hold for digital ones. The electronic voting machines were originally certified after "rigorous" testing. Their testing included resistance to vibration and temperature extremes — but missed multiple glaring security holes in the software [74, 12].

3. Modern cryptography gives us tools we can use to deal with the first two points. Secure protocols have been developed for many tasks, while rigorous definitions and formal proofs of security let us gain better intuitions for security in the digital world, and relieve us from having to rely entirely on them.

In this thesis, I approach these problems from two related directions. The first is putting everyday objects in a formal framework, allowing us to construct and rigorously analyze the security of protocols that utilize them. The second direction, which was also a motivation for the first, is a focus on cryptographic protocols that explicitly involve humans.

Humans, Computers and Everyday Objects

Until recently, cryptographic protocols were designed with the assumption that the participants in the protocol can run arbitrary algorithms (as long as they are feasible for a computer). Even if the participants are actually humans, the ubiquity of cheap computing devices made such an assumption appear reasonable.

However, there is an important distinction between operations a human performs herself and those performed on her behalf by a computer: a human knows what actions she performed as part of the protocol (barring inadvertent errors). A computer, on the other hand, is opaque to most users – even for a programmer it can be difficult to verify that the computer is doing what it is supposed to.

The problem is exemplified by electronic voting machines; voters have no way to check that the machine actually recorded a vote for the candidate selected by the voter. Indeed, the experience of the U.S. elections

shows that even when machines were later proven to have recorded votes incorrectly (as determined by inconsistencies in the final tallies), this was not spotted by voters.

Thus, there is a strong motivation to find cryptographic protocols that are more transparent, and thus more trustworthy, to their human users. The issue of trust in complex protocols is not a new one, and exists on two levels. The first is that the protocol itself may be hard to understand, and its security may not be evident to a layman (even though it may be formally proved). The second is that the computers and operating system actually implementing the protocol may not be trusted (even though the protocol itself is).

Relying on the properties of everyday objects (such as locks, envelopes or scratch-off cards), is one method to make protocols intuitive to the "average" user. This is the approach we take in the first part of the thesis (Chapters 2 and 3). A summary of our results in this area appears in Section 1.1.

For tasks with more complex requirements, such as secure voting, we do not know how to construct completely transparent protocols that are also (provably) secure. In that case, we would still like to allow "human verification" of the protocol's correct implementation combined with a formal proof of security. Thus, a person can be convinced by the testimony of many independent experts that the protocol is secure, and can verify herself that the implementation is correct. It is important that the security proof take into account the fact that humans are involved in the protocol (as this may have security implications). This is the approach we take in our constructions of verifiable voting protocols, in the second part of the thesis (Chapters 4 and 5). Section 1.2 contains a summary of our work in this area.

1.1 Basing Protocols on Tamper-Evident Seals

A tamper-evident seal is a primitive based on very intuitive physical models: the sealed envelope and the locked box. In the cryptographic and popular literature, these are often used as illustrations for a number of basic cryptographic primitives. For instance, when Alice sends an encrypted message to Bob, she is often depicted as placing the message in a locked box and sending the box to Bob (who needs the key to read the message).

Bit commitment, another well known primitive, is usually illustrated using a sealed envelope. In a bit-commitment protocol one party, Alice, commits to a bit b to Bob in such a way that Bob cannot tell what b is. At a later time Alice can reveal b, and Bob can verify that this is indeed the bit to which she committed. The standard illustration used for a bit-commitment protocol is Alice putting b in a sealed envelope, which she gives to Bob. Bob cannot see through the envelope (so cannot learn b). When Alice reveals her bit, she lets Bob open the envelope so he can verify that she didn't cheat.

The problem with the above illustration is that a physical "sealed envelope", used in the simple manner described, is insufficient for bit-commitment: Bob can always tear open the envelope before Alice officially allows him to do so. Even a locked box is unlikely to suffice; many protocols based on bit-commitment remain secure only if no adversary can *ever* open the box without a key. A more modest security guarantee seems to be more easily obtained: an adversary may be able to tear open the envelope but Alice will be able to recognize this when she sees the envelope again.

"Real" closures with this property are commonly known as "tamper-evident seals". These are used widely, from containers for food and medicines to high-security government applications. Another common application that embodies these properties is the "scratch-off card", often used as a lottery ticket. This is usually a printed cardboard card which has some areas coated by an opaque layer (e.g., the possible prizes to be won are covered). The text under the opaque coating cannot be read without scratching off the coating, but it is immediately evident that this has been done (so the card issuer can verify that only one possible prize has been uncovered).

In Chapter 2, we attempt to clarify what it means to use a sealed envelope or locked box in a cryptographic protocol (a preliminary version of this work appeared in [53]). Our focus is on constructing cryptographic protocols that use physical tamper-evident seals as their basis.

1.1.1 Seals in Different Flavours

The intuitive definition of a tamper-evident seal does not specify its properties precisely. We consider three variants of containers with tamper-evident seals. The differences arise from two properties: whether or not sealed containers can be told apart and whether or not an honest player can break the seal.

Distinguishable vs. Indistinguishable. One possibility is that containers can always be uniquely identified, even when sealed (e.g., the containers have a serial number engraved on the outside). We call this a "distinguishable" model. A second possibility is that containers can be distinguished only when open; all closed containers look alike, no matter who sealed them (this is similar to the paper-envelope voting model, where the sealed envelopes can't be told apart). We call this an "indistinguishable" model.

Weak Lock vs. Envelope. The second property can be likened to the difference between an envelope and a locked box: an envelope is easy to open for anyone. A locked box, on the other hand, may be difficult for an "honest" player to open without a key, although a dishonest player may know how to break the lock. We call the former an "envelope" model and the latter a "weak lock" model. Formal definitions for the different models appear in Section 2.2.

1.1.2 Our Results: Theoretical Foundations

In Chapter 2, we show that tamper-evident seals can be used to implement standard cryptographic protocols. We construct protocols for some of the most basic cryptographic primitives, formally analyze them and show separations between the different models of tamper-evident seals.

Basic Cryptographic Primitives. A good intuition for the "cryptographic power" of a new model can be developed by studying its relation to elemental cryptographic primitives. In our different models for tamper-evident seals, we study the possibility of implementing coin flipping (CF), zero-knowledge protocols, bit-commitment (BC) and oblivious transfer (OT), some of the most fundamental primitives in modern cryptography; OT is sufficient by itself for secure function evaluation without additional complexity assumptions [43, 50]. OT implies bit-commitment, which in turn implies zero-knowledge proofs for any language in NP [42] and weakly-fair coin flipping (in a weakly-fair coin-flipping protocol an honest player may abort when it detects the other party cheating) [10]. None of these primitives are possible in the "bare" model (where adversaries are computationally unbounded and there are no physical assumptions beyond error-free communication channels).

Formal Analysis. An important contribution of our work is the *formal* analysis for the models and protocols we construct. We prove the security of our protocols for CF, BC and OT in Cannetti's *Universal Composability* framework [16]. This allows us to use them securely as "black-boxes" in larger constructions.

On the negative side, we give impossibility results for BC and OT (note that we show the impossibility of *any* type of bit-commitment or oblivious transfer, not just universally composable realizations). The proofs are based on information-theoretic methods: loosely speaking, we show that the sender has too much information about what the receiver knows. When this is the case, BC is impossible because the sender can decide in advance what the receiver will accept (so either the receiver knows the committed bit or it is possible to equivocate), while OT is impossible because the transfer cannot be "oblivious" (the sender knows how much information the receiver has on each of his bits).

Our results show a separation between the different models of tamper-evident seals and the "bare" model, summarized in the following table:

Model	Possible	Impossible
Bare		CF, BC, OT
Dist. Weak Locks	Coin Flip	BC, OT
Dist. Envelopes	Coin Flip, Bit-Commitment, Strongly Fair Coin Flip($1/r$)	OT
Indist. Weak Locks	Coin Flip, Bit-Commitment, Oblivious Transfer	??

Strongly-Fair Coin-Flipping. One significant outcome of our work is a protocol for "strongly-fair" coin-flipping with optimal bias. In a strongly-fair coin-flipping protocol, the result for an honest player must be

either 0 or 1, even if the other player quits before finishing the protocol. In 1986, Cleve showed that in any r-round coin-flipping protocol (under standard cryptographic assumptions), one of the parties can bias the result by $\Omega(\frac{1}{r})$ (i.e., cause the honest side to output a specific bit with probability $\frac{1}{2} + \Omega(\frac{1}{r})$) [24]. A careful reading of [24] shows that this lower bound holds in all three of our tamper-evident seal models as well.

Until recently, however, the best known protocol for coin-flipping allowed one side to bias the results by $\Omega(\frac{1}{\sqrt{r}})$. In [53], we construct an r-round coin-flipping protocol in the distinguishable envelope model with bias bounded by $O(\frac{1}{r})$ (which is optimal in this model). This protocol was a stepping stone on the way to a very recent result showing that strongly fair coin-flipping with bias $O(\frac{1}{r})$ is possible in the standard model [57]; one of the main ideas in the new protocol, that of using a secret "threshold round", originated in our search for a strongly-fair coin-flipping protocol based on tamper-evident seals (see Section 2.4.3). Loosely speaking, the insight was to determine the outcome of the coin-flip at the threshold round, but let the parties learn which round was the threshold only after the fact.

Our coin-flipping protocol in the distinguishable envelope model, together with the impossibility proof for OT, also proves a more general black-box separation between coin-flipping with optimal bias and OT. This is interesting, because the new protocol for coin-flipping in the standard model [57] makes fundamental use of oblivious transfer (it relies on secure function evaluation). On the other hand, for a large class of protocols (including all known coin-flipping protocols that do not require oblivious transfer in the standard model), an unpublished result of Cleve and Impagliazzo [25] shows that any coin-flipping protocol with r rounds has at least $\Omega(\frac{1}{\sqrt{r}})$ bias.

1.1.3 Our Results: Secure Polling Protocols

Polling Schemes. Polling schemes are closely related to voting, but usually have slightly less exacting requirements. In a polling scheme the purpose of the pollster is to get a good statistical profile of the responses, however some degree of error is admissible. Unlike voting, absolute secrecy is generally not a requirement for polling, but some degree of response privacy is often necessary to ensure respondents' cooperation.

The issue of privacy arises because polls often contain questions whose answers may be incriminating or stigmatizing (e.g., questions on immigration status, drug use, religion or political beliefs). Even if promised that the results of the poll will be used anonymously, the accuracy of the poll is strongly linked to the trust responders place in the pollster. A useful rule of thumb for polling sensitive questions is "better privacy implies better data": the more respondents trust that their responses cannot be used against them, the likelier they are to answer truthfully. Using polling techniques that clearly give privacy guarantees can significantly increase the accuracy of a poll.

Randomized Response. One of the most well-known ideas is for providing such guarantees is the Randomized Response Technique (RRT) [75]. A randomized response protocol involves two parties, a pollster and a responder. The responder has a secret input bit b (this is the true response to the poll question). In the ideal case, the pollster learns a bit c, which is equal to b with probability p (p is known to the pollster) and to $1-b$ with probability $1-p$. Since p is known to the pollster, the distribution of responders' secret inputs can be easily estimated from the distribution of the pollster's outputs.

The essential property we require of a Randomized Response protocol is *plausible deniability*: A responder should be able to claim that, with reasonable probability, the bit learned by the pollster is not the secret bit b. This should be the case even if the pollster maliciously deviates from the protocol.

Cryptographic Randomized Response. A *Cryptographic* Randomized Response (CRRT) protocol is a Randomized Response protocol that satisfies an additional requirement, *bounded bias*: The probability that $c = b$ must be at most p, even if the responder maliciously deviates from the protocol. The bounded bias requirement ensures that malicious responders cannot bias the results of the poll (other than by changing their own vote).

In Chapter 3, we propose two very simple protocols for cryptographic randomized response polls, based on tamper-evident seals (a preliminary version of this work appeared in [54]). Our CRRT protocols are meant to be implemented using physical envelopes (or scratch-off cards) rather than computers. Since the properties of physical envelopes are intuitively understood, even by a layman, it is easy to verify that the

implementation is correct. The protocols are also simple enough that the privacy guarantee can be intuitively understood from the description of the protocol.

Unlike previous works concerning human-implementable protocols, we give a formal definition and a rigorous proof of security for our protocols. The security is unconditional: it relies only on the physical tamper-evidence properties of the envelopes, not on any computational assumption. Furthermore, we show that the protocols are "universally composable". Using Canetti's Composition Theorem [16], this implies that the security guarantees hold even under general concurrent composition (this is important in our polling scenario, where multiple pollsters and multiple responders may be operating concurrently).

Our protocols implement a relaxed version version of CRRT (called *weakly secure* in [3]). We also give an inefficient strong CRRT protocol (that requires a large number of rounds), and give impossibility results and lower bounds for strong CRRT protocols with a certain range of parameters. These suggest that constructing a strong CRRT protocol using scratch-off cards may be difficult (or even impossible if we require a constant number of rounds).

1.2 Human Aware Voting Protocols

A tenet of democracy is that all citizens have an equal voice in their government. The embodiment of this voice is the election. Thus, it is vitally important that elections be unbiased and fair.

A "perfect" voting protocol must satisfy a long list of requirements. Among the most important are:

Accuracy The final tally must reflect the voters' wishes.

Privacy A voter's vote must not be revealed to other parties.

Receipt-Freeness A voter should not be able to prove how she voted (this is important in order to prevent vote-buying and coercion).

Universal Verifiability Voters should be able to verify that their votes were counted correctly (this helps increase voters' trust in the system).

One of the main problems with traditional systems is that the accuracy of the election is entirely dependent on the people who count the votes. In modern systems, this usually consists of fairly small committees: if an entire committee colludes, they can manufacture their own results. Even worse, depending on the exact setup, it may be feasible to stuff ballot boxes, destroy votes or perform other manipulations.

The problems with assuring election integrity were a large factor in the introduction of mechanical voting machines and, more recently, optical scan and "Direct Recording Electronic" (DRE) machines. These perform a function identical to a ballot box and paper ballots, using a different medium: the basic protocol remains the same. While alleviating some of the problems (such as ballot stuffing), in some cases they actually aggravate the main one: instead of relying on a large number of election committees (each of which has a limited potential for harm), their security relies on a much smaller number of programmers (who may be able to undetectably change the results of the entire election).

Almost as important as the actual integrity of elections is the citizens' trust that the outcome accurately reflects the voters' intent. This requirement for transparency to humans makes voting protocols a good proving ground for human-aware cryptography.

1.2.1 Receipt-Free Human Verifiable Voting with Everlasting Privacy

In Chapter 4, we construct a voting scheme that is both receipt-free and universally verifiable even in the presence of untrusted computers, while providing voters with information-theoretic privacy (a preliminary version of this work appeared in [55]).

Everlasting Privacy. A voting protocol is said to provide *information-theoretic privacy* if a computationally unbounded adversary does not gain any information about individual votes (apart from the final tally). If the privacy of the votes depends on computational assumptions, we say the protocol provides *computational privacy*. Computational assumptions that are true today, however, may not remain so in the near future

(e.g., Adi Shamir estimated that existing public-key systems, with key-lengths in use today, will remain secure for less than thirty years). Thus, protocols that provide computational privacy may not be proof against coercion: the voter may fear her vote becoming public some time in the future.

While integrity that depends on computational assumptions only requires the assumptions to hold during the election, privacy that depends on computational assumptions requires them to hold forever. To borrow a term from Aumann et al. [5], we can say that information-theoretic privacy is *everlasting* privacy.

To achieve everlasting privacy, our voting protocol is based on statistically-hiding commitment rather than encryption (this technique is also what gives the protocol of Cramer et al. [27] the same property). We present a general scheme that can be based on any statistically-hiding string-commitment. In addition, we give a more efficient scheme based on the hardness of discrete log (that uses Pedersen commitments).

Protocol Construction. Our voting protocol follows the same general outline as Neff's voting scheme [62]: it is meant to be implemented in a traditional election setting, where voters cast their ballots in a voting booth by interacting with a DRE. Whereas in Neff's scheme the DRE publishes an *encryption* of the ballot, in our protocols the DRE publishes a statistically-hiding commitment to the contents of the ballot. As in Neff's scheme, the DRE then proves in zero knowledge to the voter that the commitment is to the voter's actual choice, while providing fake proofs for all the other candidates. After all votes have been cast, the DRE announces the final tally and proves in zero-knowledge that the published commitments correspond to the tally.

As in the case of tamper-evident seals (see Section 1.1), we provide a proof of security in the Universal Composability (UC) framework.

Formally Defining Receipt-Freeness. One of the significant contributions of this paper is a formal definition of receipt-freeness in the general multi-party computation setting (we also prove that our protocol satisfies this definition). Our definition is a generalization of Canetti and Gennaro's definition for an incoercible computation [17]. To the best of our knowledge, this is the first definition to capture receipt-freeness in the general case (most previous papers that deal with receipt-freeness do not provide a formal definition at all).

1.2.2 Split-Ballot Voting: Everlasting Privacy With Distributed Trust

One disadvantage of the voting scheme in Chapter 4 is that the privacy of the election has a single point of failure: the DRE knows all the votes, and if it is corrupt we can no longer ensure secrecy. The common way to solve this is by using a small number of trustees instead of a single authority. In most of the electronic voting schemes, it is enough that one of trustees remain uncorrupted. Because we use commitment instead of encryption, the standard mix techniques for sharing the trust between multiple authorities do not work.

In Chapter 5, we propose a new voting protocol that overcomes this problem (a preliminary version of this work appears in [56]). Our new protocol, which we call the "split-ballot" voting protocol, is also universally-verifiable, receipt-free and provides everlasting privacy for individual votes. In contrast to our previous protocol, in the split-ballot protocol the information about the votes is split between two separate voting authorities.

The Challenge. What makes this difficult is that the voting authorities *must* receive information about the votes in order to perform the tally (since we do not require voters to actively participate in tallying). Thus, everlasting privacy seems impossible to achieve in this setting. We sidestep the problem by relaxing the requirements slightly: we require the *public* information (the information used to verify the tally) to be information-theoretically hiding, but allow the data seen by the voting authorities to be only computationally hiding.

The security of the scheme degrades gracefully when authorities are corrupted: accuracy is guaranteed even if both authorities are malicious and colluding, computational privacy is guaranteed as long as one of the authorities is honest, and both everlasting privacy and receipt-freeness are guaranteed if both authorities are honest.

Splitting the Ballot. In order to prevent either voting authority from learning the voter's choice, the voter must perform what amounts to computing shares of her choice using a secret-sharing scheme (each voting authority receives a share). Although trivial for a computer, we require this computation to be feasible for

the "average voter". Hence, the user interface is extremely important.

The user interface for the "split-ballot" voting protocol was inspired by the Punchscan voting scheme [22]. Like Punchscan, the split-ballot voting protocol uses pre-printed paper ballots and does not require a computer for ballot casting. Ballots are composed of three separate pages: a standard (bottom) page that is marked by the voter and later used as the ballot, and two (top) pages containing secret information (one from each voting authority). When the pages are stacked, the information on the top pages tells the voter where to mark the bottom page in order to vote for her candidate. The voter then destroys the secret information in the top pages (e.g., by shredding them), and exits the polling booth with the bottom page.

By combining the mark on the bottom page with the secret information on the top page it sent, each authority can compute its secret share of the voter's choice. Note that the secret sharing operation is implicit in the ballot construction — the voter is never required to explicitly perform "a computation". Without the secret information, the marks on the bottom page reveal nothing about the voter's choice; the fact that the voter can lie about the secret information on the pages she destroyed gives us receipt-freeness.

We formally prove the security of the protocol in the Universal Composability framework, based on number-theoretic assumptions. Interestingly, we prove that the protocol is secure in the UC framework even though the underlying commitment and encryption schemes are not universally composable. We also show our protocol is receipt-free (under the simulation-based definition from Chapter 4).

An additional result in this paper (that affirms the importance of rigorous definitions for receipt-freeness) is an explicit vote-buying attack against Punchscan [22] (one of the "competing" voting schemes). This attack is possible even though the scheme is incoercible by Canetti and Gennaro's definition.

Chapter 2

Basing Cryptographic Protocols on Tamper-Evident Seals

2.1 Introduction

In this paper we consider the use of "tamper-evident seals" in cryptographic protocols. A tamper-evident seal is a primitive based on very intuitive physical models: the sealed envelope and the locked box. In the cryptographic and popular literature, these are often used as illustrations for a number of basic cryptographic primitives. For instance, when Alice sends an encrypted message to Bob, she is often depicted as placing the message in a locked box and sending the box to Bob (who needs the key to read the message).

Bit commitment, another well known primitive, is usually illustrated using a sealed envelope. In a bit-commitment protocol one party, Alice, commits to a bit b to Bob in such a way that Bob cannot tell what b is. At a later time Alice can reveal b, and Bob can verify that this is indeed the bit to which she committed. The standard illustration used for a bit-commitment protocol is Alice putting b in a sealed envelope, which she gives to Bob. Bob cannot see through the envelope (so cannot learn b). When Alice reveals her bit, she lets Bob open the envelope so he can verify that she didn't cheat.

The problem with the above illustration is that a physical "sealed envelope", used in the simple manner described, is insufficient for bit-commitment: Bob can always tear open the envelope before Alice officially allows him to do so. Even a locked box is unlikely to suffice; many protocols based on bit-commitment remain secure only if no adversary can *ever* open the box without a key. A more modest security guarantee seems to be more easily obtained: an adversary may be able to tear open the envelope but Alice will be able to recognize this when she sees the envelope again.

"Real" closures with this property are commonly known as "tamper evident seals". These are used widely, from containers for food and medicines to high-security government applications. Another common application that embodies these properties is the "scratch-off card", often used as a lottery ticket. This is usually a printed cardboard card which has some areas coated by an opaque layer (e.g., the possible prizes to be won are covered). The text under the opaque coating cannot be read without scratching off the coating, but it is immediately evident that this has been done (so the card issuer can verify that only one possible prize has been uncovered).

In this paper we attempt to clarify what it means to use a sealed envelope or locked box in a cryptographic protocol. Our focus is on constructing cryptographic protocols that use physical tamper-evident seals as their basis. In particular, we study their applicability to coin flipping (CF), zero-knowledge protocols, bit commitment (BC) and oblivious transfer (OT), some of the most fundamental primitives in modern cryptography; Oblivious transfer is sufficient by itself for secure function evaluation [43, 50] without additional complexity assumptions. Oblivious transfer implies bit-commitment, which in turn implies zero-knowledge proofs for any language in NP [42] and (weakly-fair) coin flipping [10].

Note that encryption is very simple to implement using tamper-evident containers (given authenticated channels), which is why we do not discuss in depth in this paper. For example, Alice and Bob can agree

on a secret key by sending random bits in sealed containers. A bit in a container that arrives unopened is guaranteed (by the tamper-evidence property) to be completely unknown to the adversary. The case where only the creator of a container can tell whether it has been opened requires only slightly more complex protocols.

2.1.1 Seals in Different Flavours

The intuitive definition of a tamper-evident seal does not specify its properties precisely. In this paper, we consider three variants of containers with tamper-evident seals. The differences arise from two properties: whether or not sealed containers can be told apart and whether or not an honest player can break the seal.

Distinguishable vs. Indistinguishable One possibility is that containers can always be uniquely identified, even when sealed (e.g., the containers have a serial number engraved on the outside). We call this a "distinguishable" model. A second possibility is that containers can be distinguished only when open; all closed containers look alike, no matter who sealed them (this is similar to the paper-envelope voting model, where the sealed envelopes can't be told apart). We call this an "indistinguishable" model.

Weak Lock vs. Envelope The second property can be likened to the difference between an envelope and a locked box: an envelope is easy to open for anyone. A locked box, on the other hand, may be difficult for an "honest" player to open without a key, although a dishonest player may know how to break the lock. We call the former an "envelope" model and the latter a "weak lock" model. In Section 2.2 we give formal definitions for the different models.

Any envelope model is clearly stronger than the corresponding weak-lock model (since in the envelope model the honest player is more powerful, while the adversary remains the same). We show that there are protocols that can be implemented in the indistinguishable models that cannot be realized in any of the distinguishable models. It is not clear however, that any indistinguishable model is strictly stronger than any distinguishable model. Although all four combinations are possible, the indistinguishable envelope model does not appear significantly stronger than the indistinguishable weak lock model, and in this paper we discuss only the latter. Note that in the standard model of cryptography, where the parties exchange messages and there is no access to outside physical resources, we do not know how to implement any of these closures.

Additional Variants The definitions of tamper-evident seals we consider in this paper are by no means the only possible ones. They do, however, represent a fairly weak set of requirements for a physical implementation. In particular, we don't require the containers to be unforgeable by their creator (this relaxation is captured by allowing the creator of the container to change its contents and reseal it).

2.1.2 Our Results

In this paper we show that tamper-evident seals can be used to implement standard cryptographic protocols. We construct protocols for "weakly-fair" coin flipping (in which the result is 0, 1 or *invalid*), bit-commitment and oblivious transfer using tamper-evident seals as primitives.

A possibly practical application of our model is the "cryptographic randomized response technique" (CRRT), defined by Ambainis et al. [3]. "Randomized response" is a polling technique used when some of the answers to the poll may be stigmatizing (e.g., "do you use drugs?"). The respondent lies with some known probability, allowing statistical analysis of the results while letting the respondent disavow a stigmatizing response. In a CRRT, there is the additional requirement that a malicious respondent cannot bias the results more than by choosing a different answer. The techniques described by Ambainis et al. achieve this, but require "heavy" cryptographic machinery (such as OT), or quantum cryptography. In a follow-up paper [54], we show a simple protocol for CRRT using scratch-off cards.

One of the most interesting results is a protocol for "strongly-fair" coin flipping (where the result for an honest player must be either 0 or 1 even if the other player quits before finishing the protocol) with bias

bounded by $O(\frac{1}{r})$, where r is the number of rounds. This protocol was a stepping-stone to the subsequent construction of an optimal protocol for strongly-fair coin flipping in the standard model [57].

An important contribution of this paper is the *formal* analysis for the models and protocols we construct. We show that the protocols are *Universally Composable* in the sense of Canetti [16]. This allows us to use them securely as "black-boxes" in larger constructions.

On the negative side, we show that our protocol for strongly-fair CF using sealed envelopes is optimal: it is impossible to do better than $O(\frac{1}{r})$ bias (this follows from a careful reading of the proof in [24]). We also give impossibility results for BC and OT (note that we show the impossibility of *any* type of bit-commitment or oblivious transfer, not just universally composable realizations). The proofs are based on information-theoretic methods: loosely speaking, we show that the sender has too much information about what the receiver knows. When this is the case, BC is impossible because the sender can decide in advance what the reciever will accept (so either the reciever knows the committed bit or it is possible to equivocate), while OT is impossible because the transfer cannot be "oblivious" (the sender knows how much information the receiver has on each of his bits).

Our results show a separation between the different models of tamper-evident seals and the "bare" model, summarized in the following table:

Model	Possible	Impossible
Bare		CF, BC, OT
Dist. Weak Locks	Coin Flip	BC, OT
Dist. Envelopes	Coin Flip, Bit-Commitment, Strongly-Fair Coin Flip($1/r$)	OT
Indist. Weak Locks	Coin Flip, Bit-Commitment, Oblivious Transfer	??

2.1.3 Related Work

To the best of our knowledge, this is the first attempt at using tamper evident seals for cryptographic protocols. Ross Anderson discusses "packaging and seals" in the context of security engineering [4], however the use of tamper-evidence does not extend to more complex protocols. Blaze gives some examples of the reverse side of the problem: cryptanalysis of physical security systems using techniques from computer science [8, 9]. Using scratch-off cards in the lottery setting can be described as a very weak form of CF, however we do not believe this has ever been formally analyzed (or used in more complex protocols).

On the other hand, basing cryptographic protocols on physical models is a common practice. Perhaps the most striking example is the field of quantum cryptography, where the physics of quantum mechanics are used to implement cryptographic operations – some of which are impossible in the "bare" model. One of the inspirations for this work was the idea of "Quantum Bit Escrow" (QBE) [2], a primitive that is very similar to a tamper-evident seal and that can be implemented in a quantum setting. There are, however, significant differences between our definitions of tamper-evident seals and QBE. In particular, in QBE the adversary may "entangle" separate escrowed bits and "partially open" commitments. Thus, while unconditionally secure bit-commitment is impossible in the pure quantum setting [52, 51], it is possible in ours.

Much work has been done on basing BC and OT on the physical properties of communication channels, using the random noise in a communication channel as the basis for security. Both BC and OT were shown to be realizable in the *Binary Symmetric Channel* model [30, 29], in which random noise is added to the channel in both directions with some known, constant, probability. Later work shows that they can also be implemented, under certain conditions, in the weaker (but more convincing) *Unfair Noisy Channel* model [34, 32], where the error probability is not known exactly to the honest parties, and furthermore can be influenced by the adversary. Our construction for 1-2 OT uses some of the techniques and results from [34].

One of the motivations for this work was the attempt to construct cryptographic protocols that are implementable by humans without the aid of computers. This property is useful, for example, in situations where computers cannot be trusted to be running the protocol they claim, or where "transparency" to humans is a requirement (such as in voting protocols). Many other examples exist of using simple physical objects as a basis for cryptographic protocols that can be performed by humans, some are even folklore: Sarah Flannery [39] recounts a childhood riddle that uses a doubly-locked box to transfer a diamond between two

parties, overcoming the corrupt postal system (which opens any unlocked boxes) despite the fact that the two parties have never met (and can only communicate through the mail). Fagin, Naor and Winkler [38] assembled a number of solutions to the problem of comparing secret information without revealing anything but the result of the comparison using a variety of different physical methods. Schneier devised a cipher [70] that can be implemented by a human using a pack of cards. In a lighter vein, Naor, Naor and Reingold [59] give a protocol that provides a "zero knowledge proof of knowledge" of the correct answer to the children's puzzle "Where's Waldo" using only a large newspaper and scissors. A common thread in these works is that they lack a formal specification of the model they use, and a formal proof of security.

2.1.4 Organization of the Paper

In Section 2.2, we give formal definitions for the different models of tamper-evident seals and the functionalities we attempt to realize using them. In Section 2.3 we discuss the capabilities of the Distinguishable Weak Lock model, show that bit-commitment is impossible in this model and give a protocol for weakly-fair coin-flipping. In Section 2.4 we discuss the capabilities of the Distinguishable Envelope model, showing that OT is impossible and giving protocols for BC and strongly-fair CF with bias $1/r$. Section 2.5 contains a discussion of the indistinguishable weak lock model and a protocol for oblivious transfer in this model. The proofs of security for the protocols we describe are given in Sections 2.6,2.8.1,2.7,2.9 and 2.10. The proofs are fairly technical, and can be skipped on first reading. Section 2.11 contains the discussion and some open problems.

2.2 The Model: Ideal Functionalities

2.2.1 Ideal Functionalities and the UC Framework

Many two-party functionalities are easy to implement using a trusted third party that follows pre-agreed rules. In proving that a two-party protocol is secure, we often want to say that it behaves "as if it were performed using the trusted third party". A formalization of this idea is the "Universally Composable" model defined by Canetti [16].

In the UC model, the trusted third party is called the *ideal functionality*. The ideal functionality is described by a program (formally, it is an interactive Turing machine) that can communicate by authenticated, private channels with the participants of the protocol.

The notion of security in the UC model is based on simulation: a protocol securely realizes an ideal functionality in the UC model if any attack on the protocol in the "real" world, where no trusted third party exists, can be performed against the ideal functionality with the same results. Attacks in the ideal world are carried out by an "ideal adversary", that can also communicate privately with the functionality. The ideal adversary can corrupt honest parties by sending a special **Corrupt** command to the functionality, at which point the adversary assumes full control of the corrupted party. This allows the functionality to act differently depending on which of the parties are corrupted. Additional capabilities of the adversary are explicitly defined by the ideal functionality.

Proving protocol security in the UC model provides two main benefits: First, the functionality definition is an intuitive way to describe the desired properties of a protocol. Second (and the original motivation for the definition of the UC model), protocols that are secure in the UC have very strong security properties, such as security under composition and security that is retained when the protocol is used as a sub-protocol to replace an ideal functionality. This security guarantee allows us to simplify many of our proofs, by showing separately the security of their component sub-protocols.

Note that our impossibility results are not specific to the UC model: the impossibility results for bit-commitment (Section 2.3.3), oblivious transfer (Section 2.4.1) and the lower bound for strongly-fair coin flipping (Section 2.4.4) hold even for the weaker "standard" notions of these functionalities.

In this section we formally define the different models for tamper-evident seals in terms of their ideal functionalities. For completeness, we also give the definitions of the primitives we are trying to implement (CF, BC and OT). We restrict ourselves to the two-party case, and to adversaries that decide at the beginning of the protocol whether to corrupt one of the parties or neither.

For readability, we make a few compromises in strict formality when describing the functionalities. First, the description is in natural language rather than pseudocode. Second, we implicitly assume the following for all the descriptions:

- All functionalities (unless explicitly specified) have a **Halt** command that can be given by the adversary at any time. When a functionality receives this command, it outputs $\perp$ to all parties. The functionality then halts (ignoring further commands). In a two party protocol, this is equivalent to a party halting prematurely.
- When a functionality receives an invalid command (one that does not exist or is improperly formatted), it proceeds as if it received the **Halt** command.
- When we say that the functionality "verifies" some condition, we mean that if the condition does not hold, the functionality proceeds as if it received the **Halt** command.

2.2.2 Tamper-Evident Seals

These are the functionalities on which we base the protocols we describe in the paper. In succeeding sections, we assume we are given one of these functionalities and attempt to construct a protocol for a "target" functionality (these are described in Section 2.2.3).

Distinguishable Weak Locks

This functionality models a tamper-evident container that has a "weak lock": an honest party requires a key to open the container, but the adversary can break the lock without help. Functionality $\mathcal{F}^{(DWL)}$ contains an internal table that consists of tuples of the form $(id, value, creator, holder, state)$. The table represents the state and location of the tamper-evident containers. It contains one entry for each existing container, indexed by the container's id and creator. We denote $value_{id}$, $creator_{id}$, $holder_{id}$ and $state_{id}$ the corresponding values in the table in row id (assuming the row exists). The table is initially empty. The functionality is described as follows, running with parties $P_1, \ldots, P_n$ and ideal adversary $\mathcal{I}$:

Seal $(id, value)$ This command creates and seals a container. On receiving this command from party P_i, the functionality verifies that id has the form $(P_i, \{0,1\}^*)$ (this form of id is a technical detail to ensure that ids are local to each party). If this is the first **Seal** message with id id, the functionality stores the tuple $(id, value, P_i, P_i, \textbf{sealed})$ in the table. If this is not the first **Seal** with id id, it verifies that $creator_{id} = holder_{id} = P_i$ and, if so, replaces the entry in the table with $(id, value, P_i, P_i, \textbf{sealed})$.

Send (id, P_j) On receiving this command from party P_i, the functionality verifies that an entry for container id appears in the table and that $holder_{id} = P_i$. If so, it outputs $(\textbf{Receipt}, id, creator_{id}, P_i, P_j)$ to P_j and $\mathcal{I}$ and replaces the entry in the table with $(id, value_{id}, creator_{id}, P_j, state_{id})$.

Open id On receiving this command from P_i, the functionality verifies that an entry for container id appears in the table, that $holder_{id} = P_i$ and that either P_i is corrupted or $state_{id} = \textbf{unlocked}$. It then sends (**Opened**, id, $value_{id}$, $creator_{id}$) to P_i. If $state_{id} \neq \textbf{unlocked}$ it replaces the entry in the table with $(id, value_{id}, creator_{id}, holder_{id}, \textbf{broken})$.

Verify id On receiving this command from P_i, the functionality verifies that an entry for container id appears in the table and that $holder_{id} = P_i$. It then considers $state_{id}$. If $state_{id} = \textbf{broken}$ it sends $(\textbf{Verified}, id, \textbf{broken})$ to P_i. Otherwise, it sends $(\textbf{Verified}, id, \textbf{ok})$ to P_i.

Unlock id On receiving this command from P_i, the functionality verifies that an entry for container id appears in the table, that $creator_{id} = P_i$ and that $state_{id} = \textbf{sealed}$. If so, it replaces the entry in the table with $(id, value_{id}, creator_{id}, holder_{id}, \textbf{unlocked})$ and sends $(\textbf{Unlocked}, id)$ to $holder_{id}$.

Distinguishable Envelopes

Functionality $\mathcal{F}^{(DE)}$ models a tamper-evident "envelope": in this case honest parties can open the envelope without need for a key (although the opening will be evident to the envelope's creator if the envelope is returned). This functionality is almost exactly identical to $\mathcal{F}^{(DWL)}$, except the **Open** command allows anyone holding the container to open it. The functionality description is identical to $\mathcal{F}^{(DWL)}$, except that the new handling of the **Open** command is:

Open *id* On receiving this command from P_i, the functionality verifies that an entry for container *id* appears in the table and that $holder_{id} = P_i$. It sends $(\textbf{Opened}, id, value_{id}, creator_{id})$ to P_i. It also replaces the entry in the table with $(id, value_{id}, creator_{id}, holder_{id}, \textbf{broken})$.

The **Unlock** command is now irrelevant, but still supported to make it clear that this model is strictly stronger than the weak lock model.

Indistinguishable Weak Locks

This functionality models tamper-evident containers with "weak locks" that are indistinguishable from the outside. The indistinguishability is captured by allowing the players to shuffle the containers in their possession using an additional **Exchange** command. To capture the fact that the indistinguishability applies only to *sealed* containers, the internal table contains an addition column: *sid*, the "sealed id". This is a unique id that is shuffled along with the rest of the container contents and is revealed when the container is opened[1]

Functionality $\mathcal{F}^{(IWL)}$ can be described as follows, running with parties $P_1, \ldots, P_n$ and adversary $\mathcal{I}$:

Seal $(id, sid, value)$ This command creates and seals a container. On receiving this command from party P_i, the functionality verifies that *id* and *sid* have the form $(P_i, \{0,1\}^*)$.

- Case 1: *This is the first **Seal** message with id id and sid sid.* In this case the functionality stores the tuple $(id, sid, value, P_i, P_i, \textbf{sealed})$ in the table.
- Case 2: *This is the first **Seal** message with sid sid but id has been used before.* In this case, the functionality verifies that $holder_{id} = P_i$. It then replaces the entry in the table with $(id, sid, value, P_i, P_i, \textbf{sealed})$.
- Case 3: *This is the first **Seal** message with id id but sid has been used before.* In this case the functionality proceeds as if it has received the **Halt** command.

Send (id, P_j) On receiving this command from party P_i, the functionality verifies that an entry for container *id* appears in the table and that $holder_{id} = P_i$. If so, it sends $(\textbf{Receipt}, id, P_i, P_j)$ to P_j and $\mathcal{I}$ and replaces the entry in the table with $(id, sid_{id}, value_{id}, creator_{id}, P_j, state_{id})$.

Open *id* On receiving this command from P_i, the functionality verifies that an entry for container *id* appears in the table, that $holder_{id} = P_i$ and that either P_i is corrupted or $state_{id} = \textbf{unlocked}$. It then sends (**Opened**, *id*, sid_{id}, $value_{id}$, $creator_{id}$) to P_i. If $state_{id} \neq \textbf{unlocked}$ it replaces the entry in the table with (*id*, sid_{id}, $value_{id}$, $creator_{id}$, $owner_{id}$, **broken**).

Verify *id* On receiving this command from P_i, the functionality verifies that an entry for container *id* appears in the table and that $holder_{id} = P_i$. It then considers $state_{id}$. If $state_{id} = \textbf{broken}$ it sends $(\textbf{Verified}, id, \textbf{broken})$ to P_i. Otherwise, it sends $(\textbf{Verified}, id, \textbf{ok})$ to P_i.

Unlock *sid* On receiving this command from P_i, the functionality verifies that an entry exists in the table for which $sid_{id} = sid$, that $creator_{id} = P_i$. If $state_{id} = \textbf{sealed}$, it replaces the entry in the table with (*id*, sid_{id}, $value_{id}$, $creator_{id}$, $holder_{id}$, **unlocked**). Otherwise, it does nothing. Note that this command does not send any messages (so it cannot be used to determine who is holding a container).

Exchange (id_1, id_2) On receiving this command from P_i the functionality verifies that both id_1 and id_2 exist in the table, and that $holder_{id_1} = holder_{id_2} = P_i$. It then exchanges the two table rows; the tuples in the table are replaced with (id_2, sid_{id_1}, $value_{id_1}$, $creator_{id_1}$, P_i, $state_{id_1}$) and (id_1, sid_{id_2}, $value_{id_2}$, $creator_{id_2}$, P_i, $state_{id_2}$).

[1] Technically, the *sid* can be used to encode more than a single bit in a container. We do not make use of this property in any of our protocols, but changing the definition to eliminate it would make it unduly cumbersome.

A Note About Notation In the interests of readability, we will often refer to parties "preparing" a container or envelope instead of specifying that they send a **Seal** message to the appropriate functionality. Likewise we say a party "verifies that a container is sealed" when the party sends a **Verify** message to the functionality, waits for the response and checks that the resulting **Verified** message specifies an **ok** status. We say a party "opens a container" when it sends an **Open** message to the functionality and waits for the **Opened** response. We say the party "shuffles" a set of containers according to some permutation (in the indistinguishable model) when it uses the appropriate **Exchange** messages to apply the permutation to the containers' ids.

2.2.3 Target Functionalities

These are the "standard" functionalities we attempt to implement using the tamper-evident seals.

Weakly-Fair Coin Flipping

This functionality models coin flipping in which the result of the coin flip can be 0, 1 or $\perp$. The result of the flip c should satisfy: $\Pr[c = 0] \leq \frac{1}{2}$ and $\Pr[c = 1] \leq \frac{1}{2}$. This is usually what is meant when talking about "coin flipping" (for instance, in Blum's "Coin Flipping Over the Telephone" protocol [10]). The $\perp$ result corresponds to the case where one of the parties noticeably deviated from (or prematurely aborted) the protocol. Under standard cryptographic assumptions (such as the existence of one-way functions), weakly-fair coin flipping is possible. Conversely, in the standard model the existence of weakly-fair coin flipping implies one-way functions [46].

Functionality $\mathcal{F}^{(WCF)}$ is described as follows, with parties Alice and Bob and adversary $\mathcal{I}$:

Value The sender of this command is Alice (the other party is Bob). When this command is received, the functionality chooses a uniform value $d \in \{0, 1\}$. If one of the parties is corrupted, the functionality outputs $(\textbf{Approve}, d)$ to $\mathcal{I}$ (the adversary). In that case, the functionality ignores all input until it receives either a **Continue** command or a **Halt** command from $\mathcal{I}$. If no party is corrupted, the functionality proceeds as if $\mathcal{I}$ had sent a **Continue** command.

Halt When this command is received from $\mathcal{I}$ (in response to an **Approve** message) the functionality outputs $\perp$ to all parties and halts.

Continue When this command is received from $\mathcal{I}$ (in response to an **Approve** message), the functionality outputs $(\textbf{Coin}, d)$ to all parties and halts.

Note: if only one of the parties can cheat in the coin flip, we say the coin flip has *one-sided error*.

Strongly-Fair Coin Flipping with Bias p.

This functionality (adapted from [16]) models a coin flip between two parties with a bounded bias. If both parties follow the protocol, they output an identical uniformly chosen bit. Even if one party does not follow the protocol, the other party outputs a random bit d that satisfies: $|\Pr[d = 0] - \Pr[d = 1]| \leq 2p$. Note that we explicitly deal with premature halting; the standard **Halt** command is *not* present in this functionality.

Functionality $\mathcal{F}^{(SCF)}$ is described as follows:

Value When this command is received for the first time from any party, $\mathcal{F}^{(SCF)}$ chooses a bit b, such that $b = 1$ with probability p and 0 with probability $1 - p$ (this bit signifies whether it will allow the adversary to set the result). If $b = 1$, the functionality sends the message **ChooseValue** to $\mathcal{I}$. Otherwise, it chooses a random bit $d \in \{0, 1\}$ and outputs $(\textbf{Coin}, d)$ to all parties and to $\mathcal{I}$. If this command is sent more than once, all invocations but the first are ignored.

Bias d When this command is received, the functionality verifies that the sender is corrupt, that the **Value** command was previously sent by one of the parties and that $b = 1$ (if any of these conditions are not met, the command is ignored). The functionality then outputs $(\textbf{Coin}, d)$ to all parties.

Bit Commitment

Functionality $\mathcal{F}^{(BC)}$ (adapted from [16]) can be described as follows:

Commit b The issuer of this command is called the sender, the other party is the receiver. On receiving this command the functionality records b and outputs **Committed** to the receiver. It then ignores any other commands until it receives the **Open** command from the sender.

Open On receiving this command from the sender, the functionality outputs (**Opened**, b) to the receiver.

Oblivious Transfer

Functionality $\mathcal{F}^{(OT)}$ (taken from [32]) is as follows:

Send (b_0, b_1) The issuer of this command is called the sender, the other party is the receiver. On receiving this command the functionality records (b_0, b_1) and outputs **QueryChoice** to the receiver. It then ignores all other commands until it receives a **Choice** command from the receiver

Choice c On receiving this command from the receiver, the functionality verifies that $c \in \{0, 1\}$. It then sends b_c to the receiver.

2.2.4 Intermediate Functionalities

In order to simplify the presentation, in the following sections we will present protocols that realize functionalities that are slightly weaker than the ones we want. We then use standard amplification techniques to construct the "full" functionalities from their weak version. In this section we define these intermediate functionalities and state the amplification lemmas we use to construct the stronger versions of these primitives. These definitions are in the spirit of [34].

p-Weak Bit-Commitment

This functionality models bit-commitment where a corrupt receiver can cheat with probability p. Note that an ϵ-WBC protocol is a regular bit-commitment protocol when ϵ is negligible. Formally, functionality $\mathcal{F}^{(p-WBC)}$ proceeds as follows:

Commit b The issuer of this command is called the sender, the other party is the receiver. On receiving this command the functionality records b and outputs **Committed** to the receiver. It ignores any additional **Commit** commands.

Open b On receiving this command from the sender, the functionality verifies that the sender previously sent a **Commit** b command. If so, the functionality outputs (**Opened**, b) to the receiver.

Break On receiving this command from a corrupt receiver, the functionality verifies that the sender previously sent a **Commit** b command. With probability p it sends (**Broken**, b) to the receiver and with probability $1 - p$ it sends $\perp$ to the receiver. Additional **Break** commands are ignored.

The following theorem allows us to amplify any p-WBC protocol when $p < 1$, meaning that the existence of such a protocol implies the existence of regular bit-commitment.

Theorem 2.1. *For any $p < 1$ and $\epsilon > 0$, there exists a protocol that realizes $\mathcal{F}^{(\epsilon-WBC)}$ using $O\left(\log\left(\frac{1}{\epsilon}\right)\right)$ instances of $\mathcal{F}^{(p-WBC)}$*

The proof for this theorem is given in Section 2.9.3.

p-Remotely Inspectable Seal

This functionality is used in our protocol for strongly-fair CF. It is a strengthened version of a tamper-evident seal. With a tamper-evident seal, only its holder can interact with it. Thus, either the sender can check if it was opened, or the receiver can verify that the sealed contents were not changed, but not both at the same time. A remotely inspectable seal is one that can be tested "remotely" (without returning it to the sender). Unfortunately, we cannot realize this "perfect" version in the DE model, therefore relax it somewhat: we allow a corrupt receiver to learn the committed bit during the verification process and only then decide (assuming he did not previously break the seal) whether or not the verification should succeed. Our definition is actually a further relaxation[2]: the receiver may cheat with some probability: A corrupt receiver who opens the commitment before the verify stage will be caught with probability $1-p$.

Formally, the functionality maintains an internal state variable $v = (v_b, v_s)$ consisting of the committed bit v_b and a "seal" flag v_s. It accepts the commands:

Commit b The issuer of this command is called the sender, the other party is the receiver. b can be either 0, 1. The functionality sets $v \leftarrow (b, \textbf{sealed})$. The functionality outputs **Committed** to the receiver and ignores any subsequent **Commit** commands.

Open This command is sent by the receiver. The functionality outputs $(\textbf{Opened}, v_b)$ to the receiver. If $v_s = \textbf{sealed}$, with probability $1-p$ the functionality sets $v_s \leftarrow \textbf{open}$

Verify If $v_s \neq \textbf{sealed}$, the functionality outputs $(\textbf{Verifying}, \perp)$ to the receiver and $\perp$ to the sender. Otherwise (no opening was detected), the functionality outputs $(\textbf{Verifying}, v_b)$ to the receiver. If the receiver is corrupt, the functionality waits for a response. If the adversary responds with **ok**, the functionality outputs **Sealed** to the sender, otherwise it outputs $\perp$ to the sender. If the receiver is not corrupt, the functionality behaves as if the adversary had responded with **ok**. After receiving this command from the sender (and responding appropriately), the functionality ignores any subsequent **Verify** and **Open** commands.

We call 0-RIS simply "RIS". When $\epsilon$is negligible, ϵ-RIS is statistically indistinguishable from RIS. The following theorem states that a p-RIS functionality can be amplified for any $p < 1$ to get RIS:

Theorem 2.2. *For any $p < 1$ and $\epsilon > 0$, there exists a protocol that realizes $\mathcal{F}^{(RIS)}$ using $O\left(\log\left(\frac{1}{\epsilon}\right)\right)$ instances of $\mathcal{F}^{(p-RIS)}$*

The proof for this theorem appears in Section 2.8.2.

Possibly Cheating Weak Oblivious Transfer

The ideal functionality for WOT is defined in [34]. Loosely, a (p,q)-WOT protocol is a 1-2 OT protocol in which a corrupt sender gains extra information and can learn the receiver's bit with probability at most p, while a corrupt receiver gains information that allows it to learn the sender's bit with probability at most q. Here we define a very similar functionality, (p,q)-*Possibly-Cheating Weak Oblivious Transfer*

This functionality differs from WOT in two ways: First, a corrupt sender or receiver learns whether or not cheating will be successful before committing to their bits. Second, a corrupt sender that cheats successfully is not committed to her bits — the sender can choose which bit the receiver will receive as a function of the receiver's bit.

Formally, functionality $\mathcal{F}^{(p,q-PCWOT)}$ proceeds as follows:

CanCheat When this command is first received the functionality chooses a uniformly random number $x \in [0,1]$ and records this number. x is returned to the issuer of the command and further **CanCheat** commands are ignored. This command can only be sent by a corrupt party.

[2]This second relaxation is only for convenience; we can remove it using amplification as noted in Theorem 2.2

Send (b_0, b_1) The issuer of this command is called the sender, the other party is the receiver. On receiving this command the functionality records (b_0, b_1) and outputs **QueryChoice** to the receiver. If the receiver is corrupt and $x < q$ it also outputs $(\textbf{Broken}, b_0, b_1)$ to the receiver. It then ignores all other commands until it receives a **Choice** command from the receiver

Choice c On receiving this command from the receiver, the functionality verifies that $c \in \{0, 1\}$. If the sender is corrupt and $x < p$, it sends $(\textbf{Broken}, c)$ to the sender and waits for a **Resend** command. Otherwise, it sends b_c to the receiver. Any further **Choice** commands are ignored.

Resend b On receiving this command from a corrupt sender, and if $x < p$, the functionality sends b to the receiver.

In [34], Damgård et al. prove that (p, q)-WOT implies OT iff $p + q < 1$. A careful reading of their proof shows that this is also the case for (p, q)-PCWOT, giving the following result:

Theorem 2.3. *For any $p + q < 1$ and any $\epsilon > 0$, there exists a protocol that realizes $\mathcal{F}^{(\epsilon,\epsilon-PCWOT)}$ using $O\left(\log^2\left(\frac{1}{\epsilon}\right)\right)$ instances of $\mathcal{F}^{(p,q-PCWOT)}$.*

2.2.5 Proofs in the UC Model

Formally, the UC model defines two "worlds", which should be indistinguishable to an outside observer called the "environment machine" (denoted $\mathcal{Z}$).

The "ideal world" contains two "dummy" parties, the "target" ideal functionality, $\mathcal{Z}$ and an "ideal adversary", $\mathcal{I}$. The parties in this world are "dummy" parties because they pass any input they receive directly to the target ideal functionality, and write anything received from the ideal functionality to their local output. $\mathcal{I}$ can communicate with $\mathcal{Z}$ and the ideal functionality, and can corrupt one of the parties. $\mathcal{I}$ sees the input and any communication sent to the corrupted party, and can control the output of that party. The environment machine, $\mathcal{Z}$, can set the inputs to the parties and read their local outputs, but cannot see the communication with the ideal functionality.

The "real world" contains two "real" parties, $\mathcal{Z}$ and a "real adversary", $\mathcal{A}$. In addition it may contain the "service" ideal functionalities (in our case the distinguishable envelope functionality). $\mathcal{A}$ can communicate with $\mathcal{Z}$ and the "service" ideal functionalities, and can corrupt one of the parties. The uncorrupted parties follow the protocol, while corrupted parties are completely controlled by $\mathcal{A}$. As in the ideal world, $\mathcal{Z}$ can set the inputs for the parties and see their outputs, but not internal communication (other than what is known to the adversary).

The protocol securely realizes an ideal functionality in the UC model, if there exists $\mathcal{I}$ such that for any $\mathcal{Z}$ and $\mathcal{A}$, $\mathcal{Z}$ cannot distinguish between the ideal world and the real world. Our proofs of security follow the general outline for a proof typical of the UC model: we describe the ideal adversary, $\mathcal{I}$, that "lives" in the ideal world. Internally, $\mathcal{I}$ simulates the execution of the "real" adversary, $\mathcal{A}$. We can assume w.l.o.g. that $\mathcal{A}$ is simply a proxy for $\mathcal{Z}$, sending any commands received from the environment to the appropriate party and relaying any communication from the parties back to the environment machine. $\mathcal{I}$ simulates the "real world" for $\mathcal{A}$, in such a way that $\mathcal{Z}$ cannot distinguish between the ideal world when it is talking to $\mathcal{I}$ and the real world. In our case we will show that $\mathcal{Z}$'s view of the execution is not only indistinguishable, but actually identical in both cases.

All the ideal adversaries used in our proofs have, roughly, the same idea. They contain a "black-box" simulation of the real adversary, intercepting its communication with the tamper-evident container functionalities and replacing it with a simulated interaction with simulated tamper-evident containers. The main problem in simulating a session that is indistinguishable from the real world is that the ideal adversary does not have access to honest parties' inputs, and so cannot just simulate the honest parties. Instead, the ideal adversary makes use of the fact that in the ideal world the "tamper-evident seals" are simulated, giving it two tools that are not available in the real world:

First, the ideal adversary does not need to commit in advance to the contents of containers (it can decide what the contents are at the time they are opened), since, in the real world, the contents of a container don't affect the view until the moment it is opened.

Second, the ideal adversary knows exactly what the real adversary is doing with the simulated containers *at the time the real adversary performs the action*, since any commands sent by the real adversary to the simulated tamper-evident container functionality are actually received by the ideal adversary. This means the ideal adversary knows when the real adversary is cheating. The target functionalities, when they allow cheating, fail completely if successful cheating gives the corrupt party "illegal" information: in case cheating is successful they give the adversary the entire input of the honest party. Thus, the strategy used by the ideal adversary is to attempt to cheat (by sending a command to the target ideal functionality) when it detects the real adversary cheating. If it succeeds, it can simulate the rest of the protocol identically to a real honest party (since it now has all the information it needs). If it fails to cheat, the ideal adversary uses its "inside" information to cause the real adversary to be "caught" in the simulation.

2.3 Capabilities of the Distinguishable Weak-Lock Model

This is the weakest of the four primitives we consider. We show that unconditionally secure bit commitment and oblivious transfer are impossible in this model. However, this model is still strictly stronger than the bare model, as weak coin flipping is possible in this model.

2.3.1 A Weakly-Fair Coin Flipping Protocol

We give a protocol that securely realizes $\mathcal{F}^{(WCF)}$ using calls to $\mathcal{F}^{(DWL)}$. Here Alice learns the result of the coin flip first. Note that when this protocol is implemented in the Distinguishable Envelope Model, a trivial change allows it to have one-sided error (only Bob can cheat). In this case, Bob learns the result of the coin flip first.

Protocol 2.1 (WCF)**.**

1. Alice prepares and sends to Bob $4n$ containers arranged in quads. Each quad contains two containers with the value 0 and two with the value 1. The order of the 0s and 1s within the quad is random.

2. If Alice halts before completing the previous stage, Bob outputs a random bit and halts. Otherwise, Bob chooses one container from every quad and sends the chosen containers to Alice.

3. Alice verifies that all the containers Bob sent are still sealed (if not, or if Bob halts before sending all the containers, she outputs $\perp$ and halts). She then unlocks all the remaining containers, outputs the xor of the bits in the containers she received from Bob and halts.

4. Bob opens all the containers in his possession. If any triplet of open containers is improper ($(0,0,0)$ or $(1,1,1)$), Bob outputs a random bit and halts. If Alice quits before unlocking the containers, Bob outputs $\perp$ and halts. Otherwise he outputs the xor of the bits in the containers that remain in his possession and halts. In the DE model, Bob can open the containers without help from Alice, so he never outputs $\perp$.

The following theorem (whose proof appears in Section 2.6) states the security properties for the protocol:

Theorem 2.4. *Protocol 2.1 securely realizes $\mathcal{F}^{(WCF)}$ in the UC model.*

2.3.2 Oblivious Transfer is Impossible

Any protocol in the DWL model is also a protocol in the DE model (see Section 2.4). We show in Section 2.4.1 that OT is impossible in the DE model, hence it must also be impossible in the DWL model.

2.3.3 Bit-Commitment is Impossible

To show bit-commitment is impossible in the DWL model, we define a small set of properties that every bit-commitment protocol must satisfy in order to be considered "secure". We then show that no protocol in the DWL model can satisfy these properties simultaneously.

A bit-commitment protocol is a protocol between two players, a sender and a receiver. Formally, we can describe the protocol using four PPTs, corresponding to the commitment stage and the opening stage for each party.

$P^S_{\textbf{Commit}}(b, 1^n)$ receives an input bit and plays the sender's part in the commit stage of the protocol. The PPT can communicate with P^R_{Commit} and with the $\mathcal{F}^{(DWL)}$ functionality. It also has an output tape whose contents are passed to P^S_{Open}

$P^R_{\textbf{Commit}}(1^n)$ plays the receiver's part in the commit stage of the protocol. It can communicate with P^S_{Commit} and with the $\mathcal{F}^{(DWL)}$ functionality. It also has an output tape whose contents are passed to P^R_{Open}

$P^S_{\textbf{Open}}(1^n)$ receives the output tape of P^S_{Commit}, and can communicate with P^R_{Open} and with the $\mathcal{F}^{(DWL)}$ functionality (note that $\mathcal{F}^{(DWL)}$ retains its state between the commit and open stage).

$P^R_{\textbf{Open}}(1^n)$ receives the output tape of P^R_{Commit}, and can communicate with P^R_{Open} and with the $\mathcal{F}^{(DWL)}$ functionality. $P^R_{\text{Open}}(1^n)$ outputs either a bit b' or $\perp$.

A bit-commitment protocol is complete if it satisfies:

Definition 2.5 (Completeness)**.** If b is the input to P^S_{Commit}, and both parties follow the protocol, the probability that the output of $P^R_{\text{Open}}(1^n)$ is not b is a neglible function in n.

We say a bit-commitment protocol is *secure* if it satisfies the following two properties:

Definition 2.6 (Hiding)**.** Let the sender's input b be chosen uniformly at random. Then for any adversary B substituted for P^R_{Commit} in the protocol, the probability that B can guess b is at most $\frac{1}{2} + \epsilon(n)$, where ϵ is a negligible function.

Definition 2.7 (Binding)**.** For any adversary $A = (A_{\text{Commit}}, A_{\text{Open}}(x))$ substituted for P^S in the protocol, if $x \in \{0, 1\}$ is chosen independently and uniformly at random after the end of the commit stage, the probability (over A and P^R's random coins and over x) that P^R_{Open} outputs x is at most $\frac{1}{2} + \epsilon(n)$, where ϵ is a negligible function.

Implementing bit-commitment that is secure against computationally unbounded players using only the $\mathcal{F}^{(DWL)}$ functionality is impossible. We show this is the case not only for universally composable bit-commitment (which is a very strong notion of bit commitment), but even for a fairly weak version: there is no bit commitment protocol that is both unconditionally hiding and unconditionally binding in the DWL model.

Intuitively, the reason that bit-commitment is impossible is that in the DWL model the sender has access to all the information the receiver has about the sender's bit. This information cannot completely specify the bit (since in that case the hiding requirement of the commitment protocol is not satisfied), hence there must be valid decommitments for both 0 and 1 (that the receiver will accept). Since the sender knows what information the receiver has, she can determine which decommitments will be accepted (contradicting the binding requirement).

More formally, the proof proceeds in three stages. First, we show that we can assume w.l.o.g. that a BC protocol in the DWL model ends the commit phase with all containers returned to their creators. Second, we show that if the receiver is honest, the sender can compute everything the receiver knows about her bit and her random string. We then combine these facts to show that either the receiver knows her bit (hence the protocol is not hiding) or the sender can decommit to two different values (hence the protocol is not binding).

Let $P = \left(P^S_{\text{Commit}}, P^S_{\text{Open}}, P^R_{\text{Commit}}, P^R_{\text{Open}}\right)$ be a bit commitment protocol using calls to $\mathcal{F}^{(DWL)}$, where P^S denotes the sender's side of the protocol and P^R the receiver's side. Let Alice be the sender in the

commitment protocol and Bob the receiver. Denote Alice's input bit by b and her random string by r_A. Denote Bob's random string r_B and Bob's view of the protocol at the end of the commit stage V_{Bob} (w.l.o.g, this is assumed to be the output of P^R_{Commit}). We can assume w.l.o.g. that both parties know which is the final message of the commit stage (since both parties must agree at some point that the commit stage is over).

Let P' the protocol in which, at the end of the commit stage, Alice unlocks all the containers she created and Bob opens all the containers in his possession, records their contents and returns them to Alice. Formally, the protocol is defined as follows:

- P'^R_{Commit} runs P^R_{Commit} using the same input and random coins, keeping track of the locations of all containers created by the sender. When P^R_{Commit} terminates, P'^R_{Commit} waits for all containers it holds to be unlocked, then opens all of them, records their contents and returns them to P'^S.
- P'^S_{Commit} runs P^S_{Commit} using the same input and random coins, keeping track of the locations of all containers it creates. When P^S_{Commit} terminates, P'^S_{Commit} unlocks all the containers created by P^S and still held by the receiver, then waits for the containers to be returned.
- P'^S_{Open} runs P^S_{Open}, but when P^S_{Open} sends an **Unlock** command to $\mathcal{F}^{(DWL)}$ for a container that was created by P'^S_{Commit}, P'^S_{Open} instead sends a special "unlock" message to P'^R_{Open}.
- P'^R_{Open} runs P^R_{Open}, converting the special "unlock" messages sent by P'^S_{Open} to simulated **Unlocked** messages from $\mathcal{F}^{(DWL)}$. It also intercepts requests to open containers that were created by P'^S_{Commit} and simulates the responses using the recorded contents. It's output is the output of P^R_{Open}.

Lemma 2.8. *If P is both hiding and binding, so is P'*

Proof. *P' is binding.* If P' is not binding, it means there is some adversary $A' = (A'_{\text{Commit}}, A'_{\text{Open}}(x))$ such that when A' is substituted for P'^S, P'^R will output x with probability at least $\frac{1}{2} + \text{poly}(\frac{1}{n})$.

We can construct an adversary A that will have the same probability of success when substituted for P^S in protocol P: A_{Commit} runs A'_{Commit} until P^R_{Commit} terminates, recording the contents and locations of any containers A' creates. It then continues to run A'_{Commit}, discarding any **Unlock** commands A' sends after this point, and simulating the receipt of all containers created by A' and still held by P^R (if A' asks to verify a container, A simulates an **ok** response from $\mathcal{F}^{(DWL)}$, and if it asks to open a container, A simulates the correct **Opened** response using the recorded contents).

$A_{\text{Open}}(x)$ runs $A'_{\text{Open}}(x)$. When $A'_{\text{Open}}(x)$ sends a special unlock message to P^R, $A_{\text{Open}}(x)$ sends the corresponding real unlock command to $\mathcal{F}^{(DWL)}$. Given the same input and random coins, the simulated version of P^R_{Open} under P' has a view identical to the real P^R_{Open} under P, hence the output must be the same. Therefore the probability that A is successful is identical to the probability that A' is successful. This contradicts the hypothesis that P is binding.

P' is hiding. If P' is not hiding, there is some adversary $B' = B'_{\text{Commit}}$ that, substituted for P'^R_{Commit} in protocol P' can guess b with probability $\frac{1}{2} + \text{poly}(\frac{1}{n})$. We can construct an adversary B for the protocol P as follows: B behaves identically to B' until P^S_{Commit} terminates. It then breaks all the containers that remain in its possession and continues running B', simulating the **Unlock** messages from P^S. Since the simulation of B' under B and the real B' in protocol P' see an identical view (given the same random coins and input), B and B' will have the same output, guessing b successfully with non-negligible advantage. This contradicts the hypothesis that P is hiding. □

Denote P'' the protocol in which, at the end of P_{Commit}, Alice returns all of Bob's containers to him and Bob uses them only in P''_{Open} (or ignores them if they are never used).

Formally, P''^S_{Commit} runs P^S_{Commit} until it terminates, keeping track of the containers created by P''^R. It then returns all of those containers that it still holds to P''^R. P''^R_{Commit} runs P^R_{Commit} until it terminates, and records the ids of the containers received from P''^S_{Commit}.

P''^S_{Open} runs P^S_{Open}, replacing **Send** commands to $\mathcal{F}^{(DWL)}$ for containers sent by P''^S with special "send" messages to P''^R. When P^S_{Open} attempts to open one of the containers sent by P''^S, P''^S sends a special "return" message to P''^R and waits for it to send that container.

P''^R_{Open} runs P^R_{Open}, intercepting the special "send" and "return" messages from P''^S. In response to a "send" message it simulates a **Receipt** message from $\mathcal{F}^{(DWL)}$, and in response to a "return" message it gives the corresponding **Send** command to $\mathcal{F}^{(DWL)}$.

Lemma 2.9. *If P is both hiding and binding then so is P''*

Proof. *P'' is binding.* Suppose P'' is not. Then there exists some adversary $A'' = (A''_{\text{Commit}}, A''_{\text{Open}}(x))$ such that when A'' is substituted for P''^S, P''^R will output x with probability at least $\frac{1}{2} + \text{poly}(\frac{1}{n})$.

We can construct an adversary A that will have the same probability of success when substituted for P^S in protocol P: A_{Commit} runs A''_{Commit} until P^R_{Commit} terminates. It then continues to run A''_{Commit}, intercepting any **Send** commands A'' sends after this point.

$A_{\text{Open}}(x)$ runs $A''_{\text{Open}}(x)$. When $A''_{\text{Open}}(x)$ sends a special "send" message to P''^R, $A_{\text{Open}}(x)$ instead sends the corresponding real container to P^R. When A'' sends a special "return" message to P''^R, A simulates the receipt of the container from P''^R (this is possible because the container was never actually sent).

Given the same input and random coins, the simulated version of P^R_{Open} under P'' has a view identical to the real P^R_{Open} under P, hence the output must be the same. Therefore the probability that A is successful is identical to the probability that A'' is successful. This contradicts the hypothesis that P is binding.

P'' is hiding. Suppose it is not, then there is some adversary $B'' = B''_{\text{Commit}}$ that, substituted for P'^R_{Commit} in protocol P'' can guess b with probability $\frac{1}{2} + \text{poly}(\frac{1}{n})$. We can construct an adversary B for the protocol P as follows: B runs B'' until P^S_{Commit} terminates, recording the contents and locations of containers it creates. B then simulates the receipt of all containers it created that were still held by P^S and continues running B''. If B'' tests whether a container is sealed, B simulates an **ok** response for all containers (note that since P^S is an honest party, it cannot break any lock, so this response is always correct). If B'' opens a container, B simulates the proper response using the last recorded contents for that container (since only the creator of the container can alter the contents, this response is always correct).

Given the same input and random coins, the views of B'' when P'' is running and the simulated B'' when P is running are identical, hence the output must be the same. Therefore B can also guess b with probability $\frac{1}{2} + \text{poly}(\frac{1}{n})$, contradicting the hypothesis that P is hiding. □

Lemma 2.10. *If neither Alice (the sender) nor Bob (the receiver) break open containers (open a container that is not unlocked), Alice can compute $b, r_A \mid V_{Bob}$ (the information Bob has about b and Alice's random string at the end of the commitment phase).*

Proof. Bob's view, V_{Bob}, is composed of some sequence of the following:

1. **Seal** messages for his own containers
2. **Receipt** messages for containers received from Alice.
3. **Send** messages for containers sent to Alice.
4. **Open** messages sent for containers he created and Alice holds (there's no point in Bob opening a container he created and also holds – he already knows what it contains)
5. **Opened** messages generated by Alice opening a container she created and he holds.
6. **Verify** messages he sent
7. **Verified** messages received as a response to his **Verify** messages.
8. **Unlock** messages he sent
9. Plaintext communication

Any information Bob has about b, r_A must derive from his view of the protocol. Any messages sent by Bob do not add information about b or r_A: the contents of the message are determined solely by r_B, which is independent of b and r_A, and by the prefix of the protocol. Therefore, the **Seal**, **Send**, **Open**, **Verify** and **Unlock** messages do not contribute information about b or r_A.

The response to a **Verify** message will always be **ok**, since Alice never breaks open a container. Therefore **Verified** messages do not contain any information about b or r_A.

It follows that all the information Bob has about b, r_A must reside in the **Receipt** and **Opened** messages and plaintext communication. However, this information is also available to Alice: Every **Receipt** message is generated by a **Send** message from Alice (so she knows the contents of all **Receipt** messages received by Bob). On the other hand, since Bob never breaks open a container, every **Open** message he sends must be preceded by an **Unlock** message from Alice. Thus, Alice must know which containers he opened (and since she created them, she knows their contents) And, of course, Alice also knows anything she sent in plaintext to Bob. □

Theorem 2.11. *$\mathcal{F}^{(BC)}$ cannot be securely realized against computationally unbounded adversaries using $\mathcal{F}^{(DWL)}$ as a primitive.*

Proof. From Lemmas 2.8 and 2.9, we can assume w.l.o.g that at the end of the Commit phase, all of Alice's containers are held by Alice and all of Bob's containers are held by Bob

From Lemma 2.10, Alice knows everything Bob knows about b and r_A. Therefore she can compute all the possible pairs b', r'_A which are consistent with Bob's view of the protocol.

Assume, in contradiction, that with non-negligible probability (over b and both parties' random coins), in at least $\text{poly}(\frac{1}{n})$ of the pairs $b' = 0$ and in at least $\text{poly}(\frac{1}{n})$ of the pairs $b' = 1$. Consider the following adversary $A = (A_{\text{Commit}}, A_{\text{Open}})$: A_{Commit} runs P^S_{Commit} with a random input b. $A_{\text{Open}}(x)$ actions depend on b:

Case 1: If $b = x$, it runs P^S_{Open}.

Case 2: if $b = 1-x$, but in at least $\text{poly}(\frac{1}{n})$ of the pairs $b' = x$, it chooses r'_A randomly from this set of pairs and simulates $P^S_{\text{Commit}}(x)$, using r'_A for the random coins, intercepting all commands to $\mathcal{F}^{(DWL)}$ but **Seal** commands and simulating the correct responses using the recorded view (note that the contents and ids of Bob's containers must be identical no matter which r'_A is chosen, because Bob's view is identical for all these pairs). A can send **Seal** commands for the containers because it currently holds all the containers it created. $A_{\text{Open}}(x)$ then runs P^S_{Open} using the output from the simulation of $P^S_{\text{Commit}}(x)$.

Case 3: If $b = 1 - x$, but only a negligible fraction of the pairs $b' = x$, it fails.

By the completeness property, the probability that P^R_{Open} outputs something other than x must be negligible in cases 1 and 2. Case 1 occurs with probability $\frac{1}{2}$ and, by our hypothesis, case 2 occurs with non-negligible probability. This contradicts the binding property of the protocol.

Assume that the probability that both $b' = 0$ and $b' = 1$ in a non-negligible fraction of the pairs is negligible. Consider the following adversary B: B_{Commit} runs P^R_{Commit}. It then outputs the majority value of b' on all the pairs b', r_A consistent with it's view. By our hypothesis, with overwhelming probability $b' = b$, contradicting the hiding property of the protocol. Thus, the protocol is either not binding or not hiding. □

2.4 Capabilities of the Distinguishable Envelope Model

This model is clearly at least as strong as the Distinguishable Weak Lock model (defined in Section 2.2.2), since we only added capabilities to the honest players, while the adversary remains the same. In fact, we show that it is strictly stronger, by giving a protocol for bit-commitment in this model (in Section 2.3.3 we prove that bit-commitment is impossible in the DWL model). We also give a protocol for $\frac{1}{r}$-Strong Coin Flipping in this model and show that Oblivious transfer is impossible.

2.4.1 Oblivious Transfer is Impossible

Let Alice be the sender and Bob the receiver. Consider Alice's bits a_0 and a_1, as well as Bob's input c, to be random variables taken from some arbitrary distribution. Alice's view of a protocol execution can also be considered a random variable $V_A = (a_0, a_1, r_A, N_1, \ldots, N_n)$, consisting of Alice's bits, random coins (r_A) and the sequence of messages that comprise the transcript as seen by Alice. In the same way we denote

Bob's view with $V_B = (c, r_B, M_1, \ldots, M_n)$, consisting of Bob's input and random coins and the sequence of messages seen by Bob.

The essence of oblivious transfer (whether universally composable or not) is that Bob gains information about one of Alice's bits, but Alice does not know which one. We can describe the information Bob has about Alice's bits using Shannon entropy, a basic tool of information theory. The Shannon entropy of a random variable X, denoted $H(X)$ is a measure of the "uncertainty" that resides in X. When X has finite support: $H(X) = -\sum_x \Pr[X = x] \log \Pr[X = x]$.

Suppose Bob's view of a specific protocol transcript is v_B. What Bob learns about a_i ($i \in \{0, 1\}$) can be described by the conditional entropy of a_i given Bob's view of the protocol. We write this $H(a_i \mid V_B = v_B)$. If Bob knows a_i at the end of the protocol then $H(a_i \mid V_B = v_B) = 0$ since there is no uncertainty left about the value of a_i given Bob's view. If Bob has no information at all about a_i then $H(a_i \mid V_B = v_B) = 1$, since there are two equally likely values of a_i given Bob's view.

We show that in any protocol in the DE Model, Alice can calculate the amount of information Bob has about each of her bits:

Theorem 2.12. *For any protocol transcript where $V_A = v_A$ and $V_B = v_B$, both $H(a_0 \mid V_B = v_B)$ and $H(a_1 \mid V_B = v_B)$ are completely determined by v_A*

Proof. We will show how to compute $H(a_0 \mid V_B = v_B)$ using the value of V_A. Computing $H(a_1 \mid V_B = v_B)$ works in the same way, replacing a_0 with a_1.

For any injection f and any random variable, the event $Y = y$ is identical to the event $f(Y) = f(y)$. Therefore, for any two random variables X and Y, it holds that $H(X \mid Y = y) = H(X \mid f(Y) = f(y))$. We will describe an injection from V_B to a variable that Alice can (almost) compute:

1. Denote by C the set of all pairs $(id, value_{id})$ that appear in some **Opened** message from $M_1, \ldots, M_n$ and such that id is one of Alice's envelopes.

2. Denote by O the multiset of all pairs $(id, state)$ that appear in some **Verified** message from $M_1, \ldots, M_n$ This is a multiset because the same envelope may be verified multiple times. We only count the first **Verified** message after a **Receipt** message for the same envelope, however (i.e. if Bob verified the same envelope more than once without sending it to Alice between verifications, we ignore all but the first).

3. Denote M' the subsequence of the messages $M_1, \ldots, M_n$ consisting only of **Receipt** messages from $\mathcal{F}^{(DE)}$ and plaintext messages from Alice. We consider M' to contain the indices of the messages in the original sequence.

Let $f(V_B) = (O, C, c, r_B, M')$. To show that f is one-to-one, we show that given (O, C, c, r_B, M') it is possible to compute V_B by simulating Bob. The simulation proceeds as follows:

1. Run Bob (using c for the input and r_B for the random coins) until Bob either sends a message to $\mathcal{F}^{(DE)}$ or should receive a message from Alice (we an assume w.l.o.g. that Bob always knows when he is supposed to receive a message). If Bob asks to send a message to Alice the simulation pretends to have done so.

2. If Bob sends a message to $\mathcal{F}^{(DE)}$, we simulate a response from $\mathcal{F}^{(DE)}$:

   (a) If Bob sends an **Open** message for one of Alice's envelopes, we can look up the contents in C and respond with a simulated **Opened** message

   (b) If Bob sends an **Verify** message for one of his envelopes, we can look up the result in O and respond with a simulated **Verified** message (if the envelope was verified multiple times, we return the result corresponding to the current request from the multiset, or the previous returned result if Bob did not send the envelope to Alice between verifications).

   (c) If Bob sends an **Seal** message, we store the value (and do nothing, since no response is expected).

(d) If Bob sends an **Open** message for one of his own envelopes, we respond with an **Opened** message using the value stored earlier.

(e) The simulation also keeps track of the locations of simulated envelopes (so that it can respond correctly if Bob tries an illegal operation, such as opening an envelope that is not in his possession).

3. If Bob should receive a message, we simulate either a plaintext message from Alice or a **Receipt** message from $\mathcal{F}^{(DE)}$ by looking it up in M'.

Given r_B, Bob is deterministic, so the simulation transcript must be identical to the original protocol view.

Finally note that the random variables a_0 and (c, r_B) must be independent (otherwise, even before beginning the protocol, Bob has information about Alice's input bits). Hence, for any random variable X: $H(a_0 \mid X, c, r_B) = H(a_0 \mid X)$. In particular, $H(a_0 \mid O, C, c, r_B, M') = H(a_0 \mid O, C, M')$.

However, Alice can compute O, C, M' from V_A: Alice can compute M' since any **Receipt** messages Bob received must have been a response to a **Send** message sent by Alice, and all messages sent by Alice (including plaintext messages) can be computed from her view of the protocol.

We can assume w.l.o.g. that Bob opens all the envelopes that remain in his possession at the end of the protocol (if the protocol is secure, the protocol in which Bob opens the envelopes at the end must be secure as well, since a corrupt Bob can always do so without getting caught). Likewise, we can assume w.l.o.g. that both players verify all of their envelopes as they are returned by the other player (again, this can be done by a corrupt player without leaking any information to the other player, so the protocol that includes this step cannot be less secure than the same protocol without it).

C consists of the contents of all of Alice's envelopes that Bob opened. Obviously, Alice knows the contents of all her envelopes (since she created them). To compute C, she only needs to know which of them were opened by Bob. Each of her envelopes is either in her possession or in Bob's possession at the end of the protocol; Alice can tell which is the case by checking if the envelope was sent more times than it was received. If it's not in her possession, she can assume Bob opened it. If it is in her possession, she verified the seal on the envelope every time it was received from Bob and the results of the verification are in her view of the protocol. If Bob opened it, at least one of the verifications must have failed. Thus, Alice can compute C. Similarly, her view tells her which of Bob's envelopes she opened and how many times each envelope was sent to Bob. Since she can assume Bob verified each envelope every time it was returned to him, she can compute the results of the **Verified** messages Bob received (and so she can compute O).

Thus, Alice can compute $H(a_0 \mid O, C, M') = H(a_0 \mid f(V_B) = f(v_B)) = H(a_0 \mid V_B = v_B)$. □

2.4.2 Bit Commitment

In this section we give a protocol for bit-commitment using distinguishable envelopes. The protocol realizes a weak version of bit commitment (defined in Section 2.2.4). Theorem 2.1 implies that WBC is sufficient to realize "standard" bit-commitment.

Protocol 2.2 ($\frac{3}{4}$-WBC)**.**

To implement **Commit** b:

1. The receiver prepares four sealed envelopes, two containing a 0 and two a 1 in random order. The receiver sends the envelopes to the sender.

2. The sender opens three envelopes (chosen randomly) and verifies that they are not all the same. Let r be the value in the remaining (sealed) envelope. The sender sends $d = b \oplus r$ to the receiver.

To implement **Open**:

1. The sender sends b and the sealed envelope to the receiver.

2. The receiver verifies that the envelope is sealed, then opens it to extract r. He verifies that $d = b \oplus r$.

The proof for the security of this protocol, stated as the following theorem, appears in Section 2.9:

Theorem 2.13. *Protocol 2.2 securely realizes* $\mathcal{F}^{(\frac{3}{4}-WBC)}$ *in the UC model.*

2.4.3 A Strongly-Fair Coin Flipping Protocol with Bias $O(\frac{1}{r})$

The construction uses remotely inspectable seals (defined in Section 2.2.4), which we then show how to implement in the DE model. The idea is similar to the "standard" coin flipping protocol using bit-commitment: Alice commits to a random bit a. Bob sends Alice a random bit b, after which Alice opens her commitment. The result is $a \oplus b$.

The reason that this is not a strongly-fair CF protocol is that Alice learns the result of the toss before Bob and can decide to quit before opening her commitment. Using RIS instead of BC solves this problem, because Bob can open the commitment without Alice's help.

Ideally, we would like to replace BC with RIS (and have Alice verify that Bob didn't break the seal before sending b). This almost works; If Bob quits before verification, or if the verification fails, Alice can use a as her bit, because Bob had to have decided to quit before seeing a. If Bob quits after verification (and the verification passed), Alice can use $a \oplus b$, since Bob sent b before learning a.

The reason this idea fails is that RIS allows Bob to see the committed bit *during* verification. If he doesn't like it, he can cause the verification to fail.

We can overcome the problem with probability $1 - \frac{1}{r}$ by doing the verification in r rounds. The trick is that Alice secretly decides on a "threshold round": after this round a failure in verification won't matter. Bob doesn't know which is the threshold round (he can guess with probability at most $1/r$). If Bob decides to stop before the threshold round, either he did not attempt to illegally open a commitment (in which case his decision to stop cannot depend on the result of the coin flip), or he illegally opened all the remaining commitments (opening less than that gives no information about the result). In this case all subsequent verifications will fail, so he may as well have simply stopped at this round (note that the decision to open is made before knowing the result of the coin flip). Clearly, anything Bob does after the threshold round has no effect on the result. Only if he chooses to illegally open commitments during the threshold round can this have an effect on the outcome (since in this case, whether or not the verification fails determines whether Alice outputs a or $a \oplus b$).

The full protocol follows:

Protocol 2.3 ($\frac{1}{r}$-SCF)**.** The protocol uses r instances of $\mathcal{F}^{(RIS)}$:

1. Alice chooses r random bits $a_1, \ldots, a_r$ and sends **Commit** a_i to $\mathcal{F}_i^{(RIS)}$ (this is done in parallel). Denote $a = a_1 \oplus \cdots \oplus a_r$.

2. Bob chooses a random bit b. If Alice halts before finishing the commit stage, Bob outputs b. Otherwise, he sends b to Alice.

3. If Bob halts before sending b, Alice outputs a. Otherwise, Alice chooses a secret index $j \in \{1, \ldots, r\}$.

4. The protocol now proceeds in r rounds. Round i has the following form:

   (a) Alice verifies that Bob did not open the commitment for a_i.

   (b) Bob opens the commitment for a_i (this actually occurs during the RIS verification step).

5. If the verification for round j and all preceeding rounds was successful, Alice outputs $a \oplus b$. Otherwise, Alice outputs a.

6. Bob always outputs $a \oplus b$ (If Alice halts before completing the verification rounds, Bob opens the commitments himself (instead of waiting for verification).

The proof of the following theorem appears in Section 2.7:

Theorem 2.14. *Protocol 2.3 securely realizes* $\mathcal{F}^{(\frac{1}{r}-SCF)}$ *in the UC model.*

Implementation of Remotely Inspectable Seals

We give protocol that realizes $\frac{1}{2}$-RIS. We can then apply Theorem 2.2 to amplify it to ϵ-RIS for some negligible ϵ. In addition to the $\mathcal{F}^{(DE)}$ functionality, the protocol utilises a weak coin flip functionality with one-sided error (only Bob can cheat). This can be implemented using distinguishable envelopes. The WCF protocol in the DWL model, described in Section 2.3.1, has one-sided error in the DE model (although we don't give a formal proof in this paper). Alternatively, Blum's protocol for coin flipping also has this property, and can be implemented using bit-commitment.

Protocol 2.4 ($\frac{1}{2}$-RIS)**.**
To implement **Commit** b:

1. Alice sends two envelopes, denoted e_0 and e_1 to Bob, both containing the bit b.

To implement **Verify**:

1. Alice initiates a weakly-fair coin flip with Bob (the coin flip has one-sided error, so that Alice is unable to cheat).

2. Denote the result of the coin flip r. Bob opens envelope e_{1-r} and outputs (**Verifying**, b) (where b is the contents of the envelope. Bob returns envelope e_r to Alice.

3. Alice waits for the result of the coin flip and the envelope from Bob. If the result of the coin flip is $\perp$, or if Bob does not return an envelope, Alice outputs $\perp$. Otherwise, Alice verifies that Bob returned the correct envelope and that it is still sealed. If either of these conditions is not satisfied, she outputs $\perp$, otherwise she outputs **Sealed**.

To implement **Open**:

1. Bob randomly chooses one of the envelopes in his possession. He opens the envelope and outputs (**Opened**, b) (where b is the contents of the envelope). Bob opens the other envelope as well.

The proof of the following theorem appears in Section 2.8.1:

Theorem 2.15. *Protocol 2.4 securely realizes* $\mathcal{F}^{(\frac{1}{2}-RIS)}$ *in the UC model.*

2.4.4 Lower Bound for Strongly-Fair Coin Flipping

In [24], Cleve proves that for any coin flipping protocol in the standard model, one of the parties can bias the result by $\Omega(1/r)$ where r is the number of rounds. This is true even if all we allow the adversary to do is to stop early. An inspection of his proof shows that this is also true in the DE model:

Theorem 2.16. *Any r-round strongly-fair coin flipping protocol in the DE model can be biased by* $\Omega(\frac{1}{r})$

The main idea in Cleve's proof is to construct a number of adversaries for each round of the protocol. He then proves that the average bias for all the adversaries is at least $\Omega(\frac{1}{r})$, so there must be an adversary that can bias the result by that amount. Each adversary runs the protocol correctly until it reaches "its" round. It then computes what an honest player would output had the other party stopped immediately after that round. Depending on the result, it either stops in that round or continues for one more round and then stops.

The only difficulty in implementing such an adversary in the DE model is that to compute its result it might need to open envelopes, in which case it may not be able to continue to the next round. The solution is to notice that it *can* safely open any envelopes that would not be sent to the other party at the end of the round (since it will stop in the next round in any case). Also, it must be able to compute the result without the envelopes it's about to send (since if the other party stopped after the round ends he would no longer have access to the envelopes). Therefore Cleve's proof is valid in the DE model as well.

2.5 Capabilities of the Indistinguishable Weak-Lock Model

The addition of indistinguishability makes the tamper-evident seal model startlingly strong. Even in the Weak Lock variant, unconditionally secure oblivious transfer is possible (and therefore so are bit-commitment and coin flipping). In this section we construct a 1-2 OT protocol using the $\mathcal{F}^{(IWL)}$ functionality. We show a $(\frac{1}{2}, \frac{1}{3})$-PCWOT protocol (for a definition of the functionality, see 2.2.4). We can then use Theorem 2.3 to construct a full 1-2 OT protocol.

2.5.1 A $(\frac{1}{2}, \frac{1}{3})$-Possibly Cheating Weak Oblivious Transfer Protocol

The basic idea for the protocol is that the sender can encode information in the order of containers, not just in their contents. When the containers are indistinguishable, the sender can shuffle containers (thus changing the information encoded in their order) without knowing the identities of the containers themselves; this gives us the obliviousness.

In order to get a more intuitive understanding of the protocol it is useful to first consider a protocol that works only against an "honest but curious" adversary:

1. the sender prepares two containers containing the bits $(0, 1)$, and sends them to the receiver.

2. the receiver prepares two containers of his own, also containing $(0, 1)$. If his bit is 0, he returns both pairs to the sender with his pair first. If his bit is 1, he returns both pairs to the sender with his pair second.

3. At this point, the sender no longer knows which of the pairs is which (as long as she doesn't open any containers). However, she knows that both pairs contain $(0, 1)$. She now encodes her bits, one on each pair (by leaving the pair alone for a 0 bit or exchanging the containers within the pair for a 1 bit). She returns both pairs to the receiver.

4. the receiver verifies that both his containers are still sealed and then opens them. The bit he learns from the sender can be deduced from the order of the containers in the pair. He randomly shuffles the sender's pair and returns it to the sender.

5. the sender verifies that the containers in the remaining pair are still sealed. Since the receiver shuffled the containers within the pair, the original encoded bit is lost, so the contents of the containers give her no information about the receiver's bit.

Unfortunately, this simple protocol fails when the adversary is not limited to be passive. For example, an active adversary that corrupts the receiver can replace the sender's pair of the containers with his own at stage (2). In stage (3) the sender encodes both her bits on the receiver's containers, while he still has the sender's pair to return at stage (4).

To prevent this attack, we can let the sender start with additional container pairs (say, three). Then, in stage (3), the sender can randomly choose two of her pairs and have the receiver tell her which ones they are. She can then verify that the pairs are sealed and that they are the correct ones. Now she's left with two pairs (one hers and one the receiver's), but the order may not be what the receiver wanted. So in the modified protocol, before the sender encodes her bits, the receiver tells her whether or not to switch the pairs.

If the receiver tampered with any of her pairs (or replaced them), with probability $\frac{2}{3}$ the sender will catch him (since he can't know in advance which pairs the sender will choose to open). However, this modification gives the sender a new way to cheat: She can secretly open one of the pairs at random (before choosing which or her pairs to verify). There are four pairs, and only one is the receiver's, so with probability $\frac{3}{4}$ she chooses one of her pairs. She can then ask the receiver to give her the locations of the other two pairs. Once she knows the location of the receiver's pair, she knows which bit he wants to learn.

To counter this attack, we let the receiver add two additional pairs as well (so that he returns six pairs at stage (2)). After the sender chooses which of her pairs to verify, the receiver randomly chooses two of his pairs to verify. He gives the sender the locations and she returns the pairs to him. Since there are now six

containers, three of which are the receiver's, if the sender decides to open a container she will open one of the receiver's with probability $\frac{1}{2}$ (which is allowed in a $\left(\frac{1}{2}, \frac{1}{3}\right)$-PCWOT protocol).

However, although the receiver will eventually learn that the sender cheated, if he didn't catch her here (he doesn't with probability $\frac{1}{3}$), the sender will learn his bit before he can abort the protocol. We prevent this by having the sender choose a random value r, and encoding $a_0 \oplus r$ and $a_1 \oplus r$ instead a_0 and a_1. At the end of the protocol the receiver asks the sender to send him either r or $a_0 \oplus a_1 \oplus r$, depending on the value of his bit. Learning only one of the values encoded by the sender gives the receiver no information about the sender's bits. Given the additional information from the sender, it allows him to learn the bit he requires, but gain no information about the other bit. As long as the sender doesn't know which of the two encoded values the receiver learns, his request at the end of the protocol doesn't give her any information about his bit.

Similarly, the receiver can gain information about both of the sender's bits by opening her containers as well as his after she encodes them. This can be prevented by having the sender use the same value for both of her containers (i.e., put 1 in both containers). Since the receiver should never open the sender's pair if he follows the protocol, this shouldn't matter. If he hasn't opened the pair previously, however, he now has no information about the bit encoded in the pair (since he doesn't know which container was originally the first in the pair).

There remains a final problem with the protocol: the receiver can cheat by lying to the sender about the locations of his pairs when he asks her to return them, and instead asking for the sender's remaining pair (along with one of his). In this case the sender remains with two of the receiver's pairs, giving the receiver both of her bits. We solve this by having the sender randomly shuffle the pairs she returns to the receiver. If the pairs are indeed the receiver's, he can tell how she shuffled them. For the sender's pair, however, he has to guess (since he doesn't know their original order. This is almost enough, except that the receiver can still cheat successfully with probability $\frac{1}{2}$ by simply guessing the correct answer. To decrease the probability of successfully cheating to $\frac{1}{3}$, we use triplets instead of pairs, and require the receiver to guess the location of the second container in the triplet under the sender's permutation.

The resulting protocol is what we require. As the protocol is fairly complex, we specify separately the sender's side (Protocol 2.5a) and the receiver's side (Protocol 2.5b).

We prove the following theorem in Section 2.10:

Theorem 2.17. *Protocol 2.5 securely realizes* $\mathcal{F}^{(\frac{1}{2},\frac{1}{3}-PCWOT)}$ *in the UC model.*

2.6 Proof of Security for Weakly-Fair Coin Flipping Protocol (Protocol 2.1)

In this section we prove Theorem 2.4. The proof follows the standard scheme for proofs in the UC model (elaborated in Section 2.2.5). We deal separately with the case where $\mathcal{A}$ corrupts Alice and where $\mathcal{A}$ corrupts Bob.

2.6.1 $\mathcal{A}$ Corrupts Bob

We first describe the ideal simulator, then prove that the environment's view in the ideal and real worlds is identically distributed. The ideal simulator, $\mathcal{I}$, proceeds as follows:

1. $\mathcal{I}$ waits until ideal Alice sends a **Value** message to $\mathcal{F}^{(WCF)}$ and it receives the (**Approve**, d) message from $\mathcal{F}^{(WCF)}$. $\mathcal{I}$ now continues running the protocol with $\mathcal{A}$, simulating $\mathcal{F}^{(DWL)}$. $\mathcal{I}$ sends $4n$ **Receipt** messages to $\mathcal{A}$.

2. $\mathcal{I}$ chooses n random quads exactly as Alice would following the protocol. Consider a quad "committed" when the contents of all unopened containers in the quad are identical (i.e., if three containers have already been opened or if two containers have been opened and contained the same value).

3. As long as there is at least one uncommitted quad, $\mathcal{I}$ responds to **Open** messages from $\mathcal{A}$ by returning the values chosen in stage (2).

Protocol 2.5a $\frac{1}{2}, \frac{1}{3}$-PCWOT (Sender)

Input: bits a_0, a_1.

Prepare three triplets of containers. All the containers contain the value 1.
Send all nine containers to the receiver.
Wait to receive 18 containers (six triplets) from the receiver.
Select a random index $i \in_R \{1, 2, 3\}$ and send i to the receiver.
Wait to receive indices (j_1, j_2) and (k_1, k_2) from the receiver {these should be the locations of the sender's triplets (except for triplet i) and the locations of two of the receiver's triplets}.
Opens all the containers in triplets j_1 and j_2 and verify that they are the correct containers.
Choose two random permutations $\pi_1, \pi_2 \in_R S_3$.
Shuffle the triplets k_1 and k_2 using π_1 and π_2, respectively.
Send the shuffled triplets k_1 and k_2 to the receiver. {the remaining unopened triplets should be the original triplet i and one of the receiver's triplets}
Wait to receive indices ℓ_1, ℓ_2 from the receiver.
Verify that $\ell_1 = \pi_1(2)$ and $\ell_2 = \pi_2(2)$. If not, abort.
Choose a random bit $r \in_R \{0, 1\}$.
if $a_0 \oplus r = 1$ **then** {Encode $a_0 \oplus r$ on first remaining triplet}
    Exchange first two containers in the first triplet. {encode a one}
else
    Do nothing. {encode a zero}
end if
if $a_1 \oplus r = 1$ **then** {Encode $a_1 \oplus r$ on second remaining triplet}
    Exchange first two containers in the second triplet. {encode a one}
else
    Do nothing. {encode a zero}
end if
Returns all six remaining containers to the receiver.
Wait to receive a bit b' from the receiver.
if $b' = 0$ **then**
    Set $x' \leftarrow r$.
else
    Set $x' \leftarrow a_0 \oplus a_1 \oplus r$.
end if
Send x' to the receiver

Protocol 2.5b $\frac{1}{2}, \frac{1}{3}$-PCWOT (Receiver)

Input: Choice bit b.

1: Wait to receive nine containers from the sender.
2: Prepare three triplets of containers (we'll call them triplets 4,5 and 6). Each triplet contains the values $(0, 1, 0)$ in that order.
3: Choose a random permutation $\sigma \in S_6$.
4: Shuffle all six triplets using σ. {the three containers in each triplet are not shuffled}
5: Send all 18 containers to the sender.
6: Wait to receive an index i from the sender.
7: Send the indices $\sigma(\{1, 2, 3\} \setminus \{i\})$ and $\sigma(\{5, 6\})$to the sender. {the locations of the sender's triplets except for triplet i and the locations of the last two triplets created by the receiver}.
8: Wait to receive two triplets from the sender.
9: Verifies that all the containers in the received triplets were unopened and that they are from the original triplets 5 and 6.
10: Open the containers. Let ℓ_1, ℓ_2 be the index of the container containing 1 in each triplet. Send ℓ_1, ℓ_2 to the sender. {e.g., ℓ_1 should be $\pi_1(2)$}
11: Wait to receive six containers (two triplets) from the sender.
12: **if** $\sigma(i) > \sigma(4)$ **then**
13: Verify that all the containers in the first triplet are sealed and were originaly from triplet 4. If not, abort.
14: Let $x = 1$ iff the first container in the first triplet contains 1. $\{x = a_0 \oplus r = 1\}$
15: Set $c \leftarrow 0$
16: **else**
17: Verify that all the containers in the second triplet are sealed and were originaly from triplet 4. If not, abort.
18: Let $x = 1$ iff the first container in the second triplet contains 1. $\{x = a_1 \oplus r = 1\}$
19: Set $c \leftarrow 1$
20: **end if**
21: Send $b \oplus c$ to the sender.
22: Wait to receive response x' from the sender.
23: Output $x \oplus x'$. $\{x \oplus x' = a_b\}$

4. When only a single uncommitted quad remains, denote by x the xor of the values for the committed quads. $\mathcal{I}$ will force the last unopened container in the quad to have the value $x \oplus d$, by choosing the responses from the distribution of permutations conditioned on the last container having the forced value.

5. $\mathcal{I}$ waits for $\mathcal{A}$ to return one container from each quad.

6. If $\mathcal{A}$ halts before returning n containers, or if any of the n containers was opened, $\mathcal{I}$ sends a **Halt** command to $\mathcal{F}^{(WCF)}$. Otherwise it sends a **Continue** command.

7. $\mathcal{I}$ simulates the **Unlocked** messages for all the containers still held by $\mathcal{A}$. It continues the simulation until $\mathcal{A}$ halts.

Lemma 2.18. *For any $\mathcal{Z}$ and $\mathcal{A}$, when $\mathcal{A}$ corrupts Bob, $\mathcal{Z}$'s view of the simulated protocol in the ideal world and $\mathcal{Z}$'s view in the real world are identically distributed.*

Proof. $\mathcal{Z}$'s view of the protocol in both worlds is identical, except for the contents of the containers sent by Alice. An inspection of the simulation shows that the distribution of the contents is also identical: in both the real and ideal worlds, the contents of each quad are uniformly random permutations of $(0, 0, 1, 1)$. Also in both cases, the xor of the committed value of all the quads is a uniformly random bit b. If $\mathcal{A}$ does not open more than three containers in any quad, and returns containers according to the protocol, this is the bit output by Alice in both the real and ideal worlds. If $\mathcal{A}$ opens all four containers, or does not return them according to the protocol, Alice will output $\perp$ in both the real and ideal worlds. □

2.6.2 $\mathcal{A}$ Corrupts Alice

As in the previous case, we first describe the ideal simulator, then prove that the environment's view in the ideal and real worlds is identically distributed. The ideal simulator, $\mathcal{I}$, proceeds as follows:

1. $\mathcal{I}$ sends a **Value** message to $\mathcal{F}^{(WCF)}$ and waits to receive the $(\textbf{Approve}, d)$ message from $\mathcal{F}^{(WCF)}$.

2. $\mathcal{I}$ waits for $\mathcal{A}$ to send the $4n$ **Seal** and **Send** messages to $\mathcal{F}^{(WCF)}$.

Case 2.1: If at least one of the quads is proper (i.e., contains two 0s and two 1s), $\mathcal{I}$ chooses which containers to send in the other quads randomly, and then chooses a container to send in the proper quad so that the xor of all the sent containers is d.

Case 2.2: If all the quads are improper, $\mathcal{I}$ chooses the containers to send from the uniform distribution conditioned on the event that at least one quad has three remaining containers that contain identical bits.

3. $\mathcal{I}$ sends the chosen containers to $\mathcal{A}$, and waits for $\mathcal{A}$ to unlock the remaining containers.

4. If $\mathcal{A}$ does not unlock all the containers, or if one of the remaining quads is improper, $\mathcal{I}$ sends a **Halt** command to $\mathcal{F}^{(WCF)}$. Otherwise $\mathcal{I}$ sends a **Continue** command to $\mathcal{F}^{(WCF)}$.

Lemma 2.19. *For any $\epsilon > 0$ there exists $n = O(\log \frac{1}{\epsilon})$, such that for any $\mathcal{Z}$ and $\mathcal{A}$, when $\mathcal{A}$ corrupts Alice the statistical distance between $\mathcal{Z}$'s view of the simulated protocol in the ideal world and $\mathcal{Z}$'s view in the real world is less than ϵ.*

Proof. $\mathcal{Z}$'s view of the protocol in both worlds is identical, except for the choice of containers sent by Bob. In the real world, Bob's choices are always uniformly random. If not all quads are improper, the distribution of Bob's choices in the ideal world is also uniformly random (since d is uniformly random, and the only choice made by $\mathcal{I}$ that is not completely random is to condition on the xor of the quad values being d). If all the quads are improper, the statistical difference between the uniform distribution and $\mathcal{I}$'s choices is exponentially small in n, since each quad has three remaining identical containers with probability at least $\frac{3}{4}$, and the events for each quad are independent (thus the probability that none of the quads was bad is at most $(\frac{3}{4})^n$). □

2.7 Proof of Security for Strongly-Fair Coin Flip Protocol (Protocol 2.3)

In this section we prove Theorem 2.14. The proof follows the standard scheme for proofs in the UC model (elaborated in Section 2.2.5). We deal separately with the case where $\mathcal{A}$ corrupts the sender and where $\mathcal{A}$ corrupts the receiver.

2.7.1 $\mathcal{A}$ Corrupts Alice

1. $\mathcal{I}$ sends a **Value** command to $\mathcal{F}^{(\frac{1}{r}-SCF)}$. If it receives a **ChooseValue** message from $\mathcal{F}^{(\frac{1}{r}-SCF)}$ it randomly chooses a bit d and sends a **Bias** d command. Denote by d the result of the coin flip.
2. $\mathcal{I}$ waits for $\mathcal{A}$ to commit to the bits $a_1, \ldots, a_r$. If $\mathcal{A}$ stops before committing to r bits, $\mathcal{I}$ halts as well.
3. Otherwise, $\mathcal{I}$ simulates Bob sending $b = d \oplus a_1 \oplus \cdots \oplus a_r$ to Alice. $\mathcal{I}$ then continues the protocol with the simulated Bob behaving honestly.

Lemma 2.20. *For any environment machine $\mathcal{Z}$, and any real adversary $\mathcal{A}$ that corrupts only Alice, the output of $\mathcal{Z}$ when communicating with $\mathcal{A}$ in the real world is identically distributed to the output of $\mathcal{Z}$ when communicating with $\mathcal{I}$ in the ideal world.*

Proof. The proof is by inspection. First, note that the output of the ideal Bob always matches the output of the simulated Bob (by the choice of b). Since $\mathcal{I}$ simulates Bob following the protocol precisely, the only difference $\mathcal{Z}$ could notice is the distribution of b. However, this is uniform in both the real and ideal worlds, since in the ideal world d (the result of $\mathcal{F}^{(\frac{1}{r}-SCF)}$'s coin flip) is uniformly distributed, and in the real world Bob chooses b uniformly. Thus, $\mathcal{Z}$'s view is identically distributed in both worlds. □

2.7.2 $\mathcal{A}$ Corrupts Bob

1. $\mathcal{I}$ sends a **Value** command to $\mathcal{F}^{(\frac{1}{r}-SCF)}$. We'll say that $\mathcal{I}$ "has control" if it received a **ChooseValue** message, and that $\mathcal{I}$ "doesn't have control" if it received a '(**Coin**, d) message from $\mathcal{F}^{(\frac{1}{r}-SCF)}$.
2. If $\mathcal{I}$ has control, it chooses a random bit d itself.
3. $\mathcal{I}$ simulates Bob receiving commit messages from $\mathcal{F}_1^{(RIS)}, \ldots, \mathcal{F}_r^{(RIS)}$.
4. $\mathcal{I}$ waits for Bob (controlled by $\mathcal{A}$) to send b to Alice.

Case 1: If $\mathcal{A}$ halts before sending b, $\mathcal{I}$ sends a **Bias** d command to $\mathcal{F}^{(\frac{1}{r}-SCF)}$ and also halts.

Case 2: If $\mathcal{A}$ attempts to open the commitments before sending b, or if $b = 0$, $\mathcal{I}$ sends a **Bias** d command to $\mathcal{F}^{(\frac{1}{r}-SCF)}$ (this is ignored if $\mathcal{I}$ does not have control). $\mathcal{I}$ then randomly chooses $a_2, \ldots, a_r$, sets $a_1 \leftarrow d \bigoplus_{i>1} a_i$ and continues the protocol, proceeding as if Alice sent **Commit** a_i to $\mathcal{F}_i^{(RIS)}$. In this case no matter what Bob does, in the real-world protocol Alice must eventually output d.

Case 3: If $\mathcal{A}$ sends $b = 1$ before opening any commitments:

i. $\mathcal{I}$ begins simulating the protocol rounds, randomly choosing a value for each a_i when $\mathcal{A}$ opens (or simulated Alice verifies) $\mathcal{F}_i^{(RIS)}$. The simulation continues in this manner until the contents of all but one of the commitments have been revealed (either because $\mathcal{A}$ prematurely opened the commitments, or during the verification phase).

ii. Call a round j "good" if the verification stage of round j succeeded and all previous rounds were good. Denote the current round by i, the index of the highest good round so far by j (by definition $j < i$), and by k the smallest index such that the committed bit in instance $\mathcal{F}_k^{(RIS)}$ is not yet known to $\mathcal{A}$ (note that $k \geq i$, since all instances up to i must have been revealed during verification). The actions of $\mathcal{I}$ now depend on i, j, k and whether $\mathcal{I}$ has control:

Case 3.1: If $i < k$, or if $\mathcal{F}_k^{(RIS)}$ is being opened (rather than verified): (this is equivalent to the case where even in the real world $\mathcal{A}$ couldn't bias the result)

- $\mathcal{I}$ sends a **Bias** d command to $\mathcal{F}^{(\frac{1}{r}-SCF)}$.
- $\mathcal{I}$ chooses a random index $i^* \in \{1, \ldots, r\}$.
- If $i^* > j$, $\mathcal{I}$ sets $a_k \leftarrow d \bigoplus_{\ell \neq k} a_\ell$, otherwise $a_k \leftarrow b \oplus d \bigoplus_{\ell \neq k} a_\ell$.
- $\mathcal{I}$ continues the simulation as if Alice had actually chosen the bits $a_1, \ldots, a_r$ to commit and the secret threshold round i^*. Note that if Alice had actually followed the protocol, the choice of a_k ensures that she always outputs d. This is because round i will certainly fail verification (since $\mathcal{F}_i^{(RIS)}$ has already been opened), so round j will remain the last round which passed verification.

Case 3.2: If $i = k$, $\mathcal{F}_k^{(RIS)}$ is being verified and $\mathcal{I}$ does not have control: (this is equivalent to the case where $\mathcal{A}$ did not correctly guess the secret threshold round, but could have cheated successfully if he had)

- $\mathcal{I}$ chooses a random index $i^* \in \{1, \ldots, r\} \setminus \{i\}$.
- If $i^* > j$, $\mathcal{I}$ sets $a_k \leftarrow d \bigoplus_{\ell \neq k} a_\ell$, otherwise $a_k = b \oplus d \bigoplus_{\ell \neq k} a_\ell$.
- $\mathcal{I}$ continues the simulation as if Alice had actually chosen the bits $a_1, \ldots, a_r$ to commit and the secret threshold round i^*. Note that if Alice had actually followed the protocol, the choice of a_k ensures that she always outputs d. This is because, by the choice of i^*, it doesn't matter whether or not round i fails verification (either $i^* > j$, in which case also $i^* > i$, or $i^* \leq j < i$).

Case 3.3: If $i = k$, $\mathcal{F}_k^{(RIS)}$ is being verified and $\mathcal{I}$ has control: (this is equivalent to the case where $\mathcal{A}$ correctly guessed the secret threshold i, and can cheat successfully)

- $\mathcal{I}$ chooses a random bit for a_k and continues the simulation.
- If $\mathcal{A}$ chooses to fail the verification, $\mathcal{I}$ sets $d^* \leftarrow d \bigoplus_\ell a_\ell$, otherwise (the verification succeeds) $\mathcal{I}$ sets $d^* \leftarrow b \oplus d \bigoplus_\ell a_\ell$.
- $\mathcal{I}$ sends a **Bias** d^* command to $\mathcal{F}^{(\frac{1}{r}-SCF)}$.
- $\mathcal{I}$ continues the simulation until $\mathcal{A}$ halts.

Lemma 2.21. *For any environment machine $\mathcal{Z}$, and any real adversary $\mathcal{A}$ that corrupts only Bob, the output of $\mathcal{Z}$ when communicating with $\mathcal{A}$ in the real world is identically distributed to the output of $\mathcal{Z}$ when communicating with $\mathcal{I}$ in the ideal world.*

Proof. $\mathcal{Z}$'s view can consist of $a_1, \ldots, a_r$ (the results of opening the commitments) and of the ideal Alice's output d.

In both the real and ideal worlds, in all cases the first $r-1$ commitments opened by $\mathcal{A}$ are independent and uniformly random (this can be easily seen by inspecting the simulator).

For any adversary that reaches Case 1 or Case 2 in the real world, the final commitment is always the xor of b (the bit sent by $\mathcal{A}$), the first $r-1$ commitments and the output of the real Alice (since the threshold round does not affect the result in this case). This is also the situation in the ideal world.

For an adversary that reaches Case 3.1, the final commitment is the xor of the first $r-1$ commitments and the output of the real Alice with probability $\frac{r-j}{r}$ (this is the probability that the secret threshold round was after the last good round), and the complement of that with probability $\frac{j}{r}$ (the probability that the threshold round is in the first j rounds). By the choice of i^*, the distribution of the last commitment in the ideal model is identical in this case.

Finally, consider the adversary that reaches Case 3.2 or Case 3.3. This adversary is honest until round i, then opens all commitments except $\mathcal{F}_i^{(RIS)}$, whose contents are revealed during verification.

1. In the real world, with probability $\frac{1}{r}$ round i is the threshold round, in which case the final commitment is the xor of the first $r-1$ commitments and d if $\mathcal{A}$ fails the verification and the complement of that if $\mathcal{A}$ does not fail. With the same probability, $\mathcal{I}$ is in control, and therefore executes Case 3.3 (which calculates the final commitment in the same way).

2. With probability $1 - \frac{1}{r}$, round i is not the threshold round. In this case, the final commitment is the xor of the first $r-1$ commitments and d with probability $\frac{r-i}{r-1}$ (the threshold round is after i), and the complement of that with probability $\frac{i-1}{r-1}$ (the threshold round is before i). In the same way, with probability $1 - \frac{1}{r}$, $\mathcal{I}$ is not in control, and executes Case 3.2. The choice of i^* ensures the correct distribution of the final commitment.

Since any adversary must reach one of the cases above, we have shown that for all adversaries $\mathcal{Z}$'s view of the protocol is identical in the real and ideal worlds. □

Together, Lemma 2.20 and Lemma 2.21 imply Theorem 2.14.

2.8 Proof of Security for Remotely Inspectable Seals

Below we prove Theorem 2.15 (in Section 2.8.1) and Thereom 2.2 (in Section 2.8.2).

2.8.1 Proof of Security for $\frac{1}{2}$-RIS Protocol (Protocol 2.4)

The proof of Theorem 2.15 follows the standard scheme for proofs in the UC model (elaborated in Section 2.2.5). We deal separately with the case where $\mathcal{A}$ corrupts the sender and where $\mathcal{A}$ corrupts the receiver.

$\mathcal{A}$ corrupts Alice (the sender)

To simulate the **Commit** command, $\mathcal{I}$ waits until $\mathcal{A}$ sends two envelopes to Bob. Denote the envelopes e_0 and e_1.

Case 1: If $\mathcal{A}$ does not send the envelopes, $\mathcal{I}$ sends the **Halt** command to $\mathcal{F}^{(\frac{1}{2}-RIS)}$ (causing ideal Bob to output $\perp$) and halts.

Case 2: If both envelopes contained the same bit b, $\mathcal{I}$ sends a **Commit** b message to $\mathcal{F}^{(\frac{1}{2}-RIS)}$.

Case 3: If the envelopes contained two different bits, $\mathcal{I}$ randomly selects a bit b and sends **Commit** b to $\mathcal{F}^{(\frac{1}{2}-RIS)}$.

To simulate the **Verify** command:

1. $\mathcal{I}$ waits for $\mathcal{A}$ to initiate a coin flip.
2. If both envelopes sent by $\mathcal{A}$ contained the same bit, $\mathcal{I}$ chooses a random bit r, otherwise it sets r to the index of the envelope containing b.
3. $\mathcal{I}$ sends r as the result of the coin flip to $\mathcal{A}$.
4. $\mathcal{I}$ simulates sending envelope e_r to $\mathcal{A}$.
5. $\mathcal{I}$ sends the **Verify** command to $\mathcal{F}^{(\frac{1}{2}-RIS)}$ and waits for the functionality's response.

   Case 1: If the response is $\perp$, verifying envelope e_r will return a **broken** message.

   Case 2: If the response was **Sealed**, verifying envelope e_r will return a **sealed** message.
6. $\mathcal{I}$ continues the simulation until $\mathcal{A}$ halts.

Note that the **Open** command need not be simulated in this case — in both the ideal and the real worlds this does not involve the sender at all.

Lemma 2.22. *For any environment machine $\mathcal{Z}$, and any real adversary $\mathcal{A}$ that corrupts only Alice, the output of $\mathcal{Z}$ when communicating with $\mathcal{A}$ in the real world is identically distributed to the output of $\mathcal{Z}$ when communicating with $\mathcal{I}$ in the ideal world.*

Proof. The proof is by case analysis. First, consider the view during the commit stage. Any adversary must fall in one of the three cases. In Case 1, in both the real and ideal worlds $\mathcal{Z}$'s view consists of Bob outputting $\perp$ and Alice halting. In Case 2 and Case 3, $\mathcal{Z}$'s view looks the same from $\mathcal{A}$'s point of view, and in both worlds Bob will output **Committed**.

If $\mathcal{Z}$ tells Bob to open the commitment before the verify stage, The output will be identical in the real and ideal worlds (it will be $(\textbf{Opened}, b)$, where b is a uniformly random bit if $\mathcal{A}$ commited two different bits).

During the verification stage, r is always a random uniform bit. There are only two cases to consider: either $\mathcal{Z}$ told Bob to open the commitment earlier, or it did not. If it did, $\mathcal{F}^{(\frac{1}{2}-RIS)}$ will return a failed verification, and $\mathcal{A}$ will also see a failed verification (exactly as would be the case in the real world). If it did not, $\mathcal{A}$ will see a successful verification in both the real and ideal worlds.

Thus, in all cases $\mathcal{Z}$'s view is identically distributed in both worlds. □

$\mathcal{A}$ corrupts Bob (the receiver)

The simulation is in two phases. In the initial phase (corresponding to the **Commit** and **Open** commands):

1. $\mathcal{I}$ waits until it receives **Committed** from $\mathcal{F}^{(\frac{1}{2}-RIS)}$. It then simulates $\mathcal{A}$ receiving two envelopes, e_0 and e_1.
2. If $\mathcal{A}$ requests to open any of the envelopes, $\mathcal{I}$ sends an **Open** command to $\mathcal{F}^{(\frac{1}{2}-RIS)}$ and waits to receive the $(\textbf{Opened}, b)$ response. It then continues the simulation as if both envelopes had contained b.

The second phase begins when $\mathcal{I}$ receives a $(\textbf{Verifying}, x)$ message from $\mathcal{F}^{(\frac{1}{2}-RIS)}$ (signifying that ideal Alice sent a **Verify** command). $\mathcal{I}$ initiates the verification phase with $\mathcal{A}$.

1. $\mathcal{I}$ chooses r in the following way: If, in the verification message, $x \neq \perp$ (that is, $\mathcal{I}$ has a choice about whether the verification will fail), it chooses r randomly from the set of unopened envelopes (if both were opened, it chooses randomly between them). If, in the verification message, $x = \perp$ (that is, the verification will definitely fail), $\mathcal{I}$ chooses r randomly from the set of opened envelopes (note that at least one envelope must be open for this to occur, because otherwise $\mathcal{I}$ would not have sent an **Open** command to $\mathcal{F}^{(\frac{1}{2}-RIS)}$ and would thus always have a choice).
2. $\mathcal{I}$ continues the simulation following the protocol exactly, letting the contents of the envelopes both be b (where $b \leftarrow x$ if $x \neq \perp$, otherwise it is the response to the **Open** command sent in the previous phase.
3. The simulation continues until $\mathcal{A}$ returns an envelope. If that envelope was opened, or its index does not match r, $\mathcal{I}$ fails the verification by sending a **Halt** command to $\mathcal{F}^{(\frac{1}{2}-RIS)}$. If the envelope was not opened and its index does match r, $\mathcal{I}$ sends the **ok** command to $\mathcal{F}^{(\frac{1}{2}-RIS)}$ (note that if $\mathcal{I}$ had no choice, the index r always matches an envelope that was already opened).

Lemma 2.23. *For any environment machine $\mathcal{Z}$, and any real adversary $\mathcal{A}$ that corrupts only Bob, the output of $\mathcal{Z}$ when communicating with $\mathcal{A}$ in the real world is identically distributed to the output of $\mathcal{Z}$ when communicating with $\mathcal{I}$ in the ideal world.*

Proof. Since $\mathcal{I}$ simulates Alice exactly, except for the contents of the envelopes and the result of the coin flip and her response to verification, these are the only things that can differ in $\mathcal{Z}$'s view between the real and ideal worlds.

Simple inspection of the protocol shows that ideal Alice's output and the contents of the envelopes are always consistent with $\mathcal{A}$'s view. It remains to show that the distribution of r is identical in the real and ideal worlds. The only case in the ideal world in which r is not chosen uniformly at random by $\mathcal{I}$ is when exactly one of the envelopes was opened. However, this means $\mathcal{I}$ must have sent an **Open** command to $\mathcal{F}^{(\frac{1}{2}-RIS)}$, and therefore with probability $\frac{1}{2}$ the verification will fail. Thus, r is still distributed uniformly in this case. □

Together, Lemma 2.23 and Lemma 2.22 prove Theorem 2.15.

2.8.2 Amplification for Remotely Inspectable Seals

The following protocol constructs an p^k-RIS using k instances of $\mathcal{F}^{(p-RIS)}$:

Commit b Alice chooses k random values $r_1, \ldots, r_k$ such that $r_1 \oplus \cdots \oplus r_k = b$. She commits to the values in parallel using $\mathcal{F}_1^{(p-RIS)}, \ldots, \mathcal{F}_k^{(p-RIS)}$.

Verify Alice sends **Verify** commands in parallel to all k instances of $\mathcal{F}^{(p-RIS)}$. The verification passes only if all k verifications return **Sealed**.

Open Bob opens all k commitments. The result is the xor of the values returned.

The ideal adversary in this case is fairly simple. The case where the sender is corrupt is trivial, and we omit it (since the sender can't cheat in the basic $\mathcal{F}^{(p-RIS)}$ instance). When $\mathcal{A}$ corrupts the receiver, the simulation works in two phases: In the initial phase (corresponding to **Commit** and **Open**):

1. $\mathcal{I}$ waits to receive the **Committed** command from $\mathcal{F}^{(p^k-RIS)}$.
2. Whenever $\mathcal{A}$ asks to open a commitment for $\mathcal{F}_i^{(p-RIS)}$:

Case 2.1: If at least one additional commitment is still unopened, $\mathcal{I}$ chooses a random bit r_i and returns this as the committed value.

Case 2.2: If $\mathcal{F}_i^{(p-RIS)}$ is the last unopened $\mathcal{F}^{(p-RIS)}$ instance, $\mathcal{I}$ sends an **Open** command to $\mathcal{F}^{(p^k-RIS)}$ and sets the value of the last commitment to be the xor of all the other commitments and the response, b.

The second phase begins when $\mathcal{I}$ receives a (**Verifying**, x) message from $\mathcal{F}^{(p^k-RIS)}$ (signifying that ideal Alice sent a **Verify** command). $\mathcal{I}$ initiates the verification phase with $\mathcal{A}$. Denote the number commitments opened by $\mathcal{A}$ by j.

Case 1: If $j = k$, $\mathcal{I}$ has sent an **Open** command previously to $\mathcal{F}^{(p^k-RIS)}$.

Case 1.1: If it has a choice about verification (occurs with probability p^k), $\mathcal{I}$ sends a (**Verifying**, r_i) message to $\mathcal{A}$ for all instances of $\mathcal{F}^{(p-RIS)}$. If $\mathcal{A}$ decides to fail verification in any of the instances, $\mathcal{I}$ sends a **Halt** command to $\mathcal{F}^{(p^k-RIS)}$. Otherwise $\mathcal{I}$ sends an **ok** response to $\mathcal{F}^{(p^k-RIS)}$.

Case 1.2: Otherwise, $\mathcal{I}$ chooses k bits $q_1, \ldots, q_k$ by sampling from the binomial distribution $B(k, p)$, conditioned on at least one bit being 1 (i.e., equivalent to letting $q_i = 1$ independently with probability p, repeating until not all bits are 0). For each bit where $q_i = 0$ it sends (**Verifying**, r_i), and for the other bits it sends (**Verifying**, $\perp$). $\mathcal{I}$ sends a **Halt** command to $\mathcal{F}^{(p^k-RIS)}$.

Case 2: If $j < k$, no **Open** command was sent, so $\mathcal{I}$ will always have a choice whether to fail verification. $\mathcal{I}$ sends a (**Verifying**, x_i) message to $\mathcal{A}$ for each instance of $\mathcal{F}^{(p-RIS)}$. For instances which were not opened, $x_i = r_i$. For instances that were opened, $\mathcal{I}$ chooses with probability p to send $x_i = r_i$ and with probability $1 - p$ to send $x_i = \perp$. It then waits for $\mathcal{A}$ to respond. If in any of the instances it chose $x_i = \perp$, or if $\mathcal{A}$ decides to fail verification in any of the instances, it sends a **Halt** command to $\mathcal{F}^{(p^k-RIS)}$. Otherwise $\mathcal{I}$ sends an **ok** response to $\mathcal{F}^{(p^k-RIS)}$.

It is easy to see by inspection that the adversary's view is identical in the real and ideal worlds. Setting $k = O(\log \frac{1}{\epsilon})$, the amplification protocol gives us the proof for Theorem 2.2.

2.9 Proof of Security for Bit-Commitment Protocol

In this section we prove Protocol 2.2 realizes the WBC functionality (proving Theorem 2.13) and show how to amplify WBC to get full bit-commitment (proving Theorem 2.1). We begin with the proof of security for Protocol 2.2. The proof follows the standard scheme for proofs in the UC model (elaborated in Section 2.2.5). We deal separately with the case where $\mathcal{A}$ corrupts the sender and where $\mathcal{A}$ corrupts the receiver.

2.9.1 $\mathcal{A}$ corrupts Alice (the sender)

We divide the simulation, like the protocol, into two phases.

Simulation of the Commit Phase

$\mathcal{I}$ starts the simulated commit protocol with $\mathcal{A}$ ($\mathcal{I}$ simulates the honest receiver, Bob, in this protocol). $\mathcal{I}$ sends four (simulated) envelopes to $\mathcal{A}$. $\mathcal{I}$ chooses a random permutation $\sigma \in S_4$. If $\mathcal{A}$ opens any of the envelopes, $\mathcal{I}$ gives results that are consistent with Bob following the protocol (i.e., the envelopes' contents are determined by $\sigma(0,0,1,1)$). $\mathcal{I}$ continues the simulation until Alice (controlled by $\mathcal{A}$) sends a bit, d, to Bob (as required by the protocol). The succeeding actions depend on how many envelopes $\mathcal{A}$ opened:

Case 1: $\mathcal{A}$ did not open any envelopes or opened two envelopes containing different bits. In this case $\mathcal{I}$ chooses a random bit b and sends a **Commit** b command to $\mathcal{F}^{(\frac{3}{4}-WBC)}$.

Case 2: $\mathcal{A}$ opened a single envelope containing x. In this case $\mathcal{I}$ chooses a random bit b to be $d \oplus x$ with probability $\frac{1}{3}$ and $d \oplus (1-x)$ with probability $\frac{2}{3}$. $\mathcal{I}$ sends a **Commit** b command to $\mathcal{F}^{(\frac{3}{4}-WBC)}$.

Case 3: Alice opened two envelopes containing identical bits x. Letting $b = d \oplus (1-x)$, $\mathcal{I}$ sends a **Commit** b command to $\mathcal{F}^{(\frac{3}{4}-WBC)}$.

Case 4: Alice opened three envelopes whose xor is x. Letting $b = d \oplus (1-x)$, $\mathcal{I}$ sends a **Commit** b command to $\mathcal{F}^{(\frac{3}{4}-WBC)}$.

Case 5: Alice opened four envelopes. Letting $b = 0$, $\mathcal{I}$ sends a **Commit** b command to $\mathcal{F}^{(\frac{3}{4}-WBC)}$.

Simulation of the Open Phase

$\mathcal{I}$ begins simulating the *Open* phase of the protocol with $\mathcal{A}$, and waits for $\mathcal{A}$ to send an envelope and a bit b'. If $\mathcal{A}$ asks to open an envelope i before this occurs, $\mathcal{I}$ proceeds in the following way:

Let $P_{consistent}$ be the set of permutations of $(0,0,1,1)$ that are consistent with $\mathcal{A}$'s view so far (i.e., the permutations that map the correct contents to the envelopes $\mathcal{A}$ has already opened), and P_{valid} the set of permutations in which at least one of the envelopes that will remain unopened after opening i contains $b \oplus d$ (where b is the bit to which $\mathcal{I}$ committed in the *Commit* phase). $\mathcal{I}$ randomly chooses a permutation from $P_{consistent} \cap P_{valid}$ and responds to the request to open i as if Bob had chosen this permutation in the *Commit* phase.

Note that $\mathcal{I}$'s choice of d and b ensures that at the end of the *Commit* phase $P_{consistent} \cap P_{valid}$ is not empty. As long as i is not the last unopened envelope, $P_{consistent} \cap P_{valid}$ will remain non-empty. If i is the last unopened envelope, $\mathcal{I}$ responds with the value consistent with the other opened envelopes.

Once $\mathcal{A}$ sends the bit b' and an envelope, $\mathcal{I}$ proceeds as follows: If the envelope is unopened, and $b' = b$, $\mathcal{I}$ sends the **Open** command to $\mathcal{F}^{(\frac{3}{4}-WBC)}$. Otherwise, $\mathcal{I}$ aborts the protocol by sending the **Halt** command to $\mathcal{F}^{(\frac{3}{4}-WBC)}$ (and simulating Bob aborting the protocol to $\mathcal{A}$).

Lemma 2.24. *For any environment machine $\mathcal{Z}$ and any real adversary $\mathcal{A}$ that corrupts only the sender, the output of $\mathcal{Z}$ when communicating with $\mathcal{A}$ in the real world is identically distributed to the output of $\mathcal{Z}$ when communicating with $\mathcal{I}$ in the ideal world.*

Proof. $\mathcal{I}$ simulates Bob (the receiver) exactly following the protocol (apart from the envelope contents), and the simulation ensures that the ideal Bob's output is consistent with $\mathcal{A}$'s view of the protocol. The only possible differences between $\mathcal{Z}$'s view in the real and ideal worlds are the contents of the envelopes sent by Bob. Inspection of the protocol and simulation shows that in both the real and ideal worlds $\mathcal{A}$ always sees a random permutation of $(0,0,1,1)$. □

2.9.2 $\mathcal{A}$ corrupts Bob (the receiver)

As before, the simulation is divided into two phases.

Simulation of the Commit Phase

$\mathcal{I}$ waits until $\mathcal{A}$ sends four envelopes and until the **Committed** message is received from $\mathcal{F}^{(\frac{3}{4}-WBC)}$. $\mathcal{I}$'s actions depend on the contents of the envelopes sent by $\mathcal{A}$:

Case 1: If the envelopes sent by $\mathcal{A}$ are a valid quad (two zeroes and two ones), $\mathcal{I}$ sends a random bit d to $\mathcal{A}$.

Case 2: If the envelopes are all identical (all zeroes or all ones), $\mathcal{I}$ aborts the protocol by sending the **Halt** command to $\mathcal{F}^{(\frac{3}{4}-WBC)}$ (and simulating Alice aborting the protocol to $\mathcal{A}$)

Case 3: If the envelopes contain three ones and a zero, or three zeroes and a one, denote x the singleton bit. $\mathcal{I}$ sends a **Break** message to $\mathcal{F}^{(\frac{3}{4}-WBC)}$. If the response is $\perp$, $\mathcal{I}$ simulates Alice aborting the protocol to $\mathcal{A}$ and halts. If the response is $(\textbf{Broken}, b)$, $\mathcal{I}$ sends $b \oplus (1-x)$ to $\mathcal{A}$.

Simulation of the Open Phase

$\mathcal{I}$ waits to receive the $(\textbf{Opened}, b)$ message from $\mathcal{F}^{(\frac{3}{4}-WBC)}$. It then proceeds depending on $\mathcal{A}$'s actions in the *Commit* phase:

Case 1: If $\mathcal{A}$ sent a valid quad, $\mathcal{I}$ randomly picks one of the two envelopes that contain $d \oplus b$ and returns it to $\mathcal{A}$.

Case 2: If the envelopes sent by $\mathcal{A}$ were not a valid quad, they must be three ones and a zero or three zeroes and a one (otherwise $\mathcal{I}$ would have aborted in the *Commit* phase). In this case $\mathcal{I}$ randomly chooses one of the three identical envelopes and simulates returning it to $\mathcal{A}$.

$\mathcal{I}$ sends the bit b to $\mathcal{A}$ as well. If $\mathcal{A}$ checks whether the envelope returned by Alice is sealed, $\mathcal{I}$ simulates an affirmative reply from $\mathcal{F}^{(DE)}$.

Lemma 2.25. *For any environment machine $\mathcal{Z}$ and any real adversary $\mathcal{A}$ that corrupts only the receiver, the output of $\mathcal{Z}$ when communicating with $\mathcal{A}$ in the real world is identically distributed to the output of $\mathcal{Z}$ when communicating with $\mathcal{I}$ in the ideal world.*

Proof. $\mathcal{I}$'s simulation of Alice (the sender) is always consistent with a real Alice that follows the protocol (from $\mathcal{A}$'s point of view), and it ensures that the ideal Alice's output is also consistent with $\mathcal{A}$'s view. $\mathcal{A}$'s view consists of d, the bit sent by Alice in the commit phase (or Alice halting in the commit phase), and the choice of envelope returned in the open phase. In both the real and ideal worlds, when $\mathcal{A}$ sends a proper quad d is uniformly random When $\mathcal{A}$ sends a quad whose bits are all identical, in both worlds Alice will abort. When $\mathcal{A}$ sends a quad containg three bits with value $1-x$ and one bit with value x, in the real world Alice would abort with probability $\frac{1}{4}$ (if x is the unopened envelope), and send $d = b \oplus (1-x)$ with probability $\frac{3}{4}$. In the ideal world, d is distributed identically, since $\mathcal{F}^{(\frac{3}{4}-WBC)}$ allows cheating with probability $\frac{3}{4}$.

In the real world, if $\mathcal{A}$ sent a proper quad in the commit phase, the envelope returned in the open phase is a random envelope and its value, r, satisfies $r = d \oplus b$. Inspection of the simulation shows that the same holds in the ideal world. if $\mathcal{A}$ sent an improper quad in the commit phase (conditioned on Alice not aborting), the envelope is randomly selected from one of the three containing the same bit, and its value satisfies $(1-r) = d \oplus b$. Again, this holds in the ideal world.

Thus, $\mathcal{Z}$'s views are identically distributed in both worlds, □

Together, Lemmas 2.24 and 2.25 imply Theorem 2.13.

2.9.3 Amplification for Weak Bit Commitment

The following protocol constructs an p^k-WBC using k instances of $\mathcal{F}^{(p-WBC)}$:

Commit b Alice chooses k random values $r_1, \ldots, r_k$ such that $r_1 \oplus \cdots \oplus r_k = b$. She commits to the values in parallel using $\mathcal{F}_1^{(p-WBC)}, \ldots, \mathcal{F}_1^{(k-WBC)}$.

Open Alice opens all k commitments. The result is the xor of the values returned.

The proof that this procotol securely realizes $\mathcal{F}^{(p^k-WBC)}$ is extremely similar to the proof of the RIS amplification protocol (in Section 2.8.2), and we omit it here. Letting $k = O(\log \frac{1}{\epsilon})$, the amplification protocol gives us the proof for Theorem 2.1.

2.10 Proof of Security for Oblivious Transfer Protocol

This section contains the proof of Theorem 2.17. The proof follows the standard scheme for proofs in the UC model (elaborated in Section 2.2.5). We deal separately with the case where $\mathcal{A}$ corrupts the sender and where $\mathcal{A}$ corrupts the receiver. Note that when $\mathcal{A}$ corrupts the sender, $\mathcal{I}$ simulates an honest receiver and references to steps in the protocol refer to Protocol 2.5b, while when $\mathcal{A}$ corrupts the receiver, $\mathcal{I}$ is simulating an honest sender and the steps refer to Protocol 2.5a.

2.10.1 $\mathcal{A}$ corrupts the receiver

Assume $\mathcal{A}$ begins by corrupting the receiver. $\mathcal{I}$ also corrupts the receiver and sends a **CanCheat** command to $\mathcal{F}^{(\frac{1}{2},\frac{1}{3}-PCWOT)}$. $\mathcal{I}$ waits for the sender to send a **Send** command, then begins simulating the real-world protocol by sending nine **Receipt** messages to $\mathcal{A}$ (acting for the receiver). Call a triplet of containers in which all containers are sealed and all belonged to the original triplet *good*. We now describe a decision tree for $\mathcal{I}$. The edges in the tree correspond either to choices made by $\mathcal{A}$ (these are marked by †), or to responses from $\mathcal{F}^{(\frac{1}{2},\frac{1}{3}-PCWOT)}$.

Case $1^\dagger$: All the triplets created by the sender are returned by $\mathcal{A}$ at step (3) and all are good. In this case, $\mathcal{I}$ randomly chooses i as specified in the protocol and continues the simulation until step (5). The protocol continues depending on $\mathcal{A}$'s actions:

Case $1.1^\dagger$: $\mathcal{A}$ sent incorrect locations for the sender's triplets $\sigma(\{j_1, j_2\}) \neq \{1,2,3\} \setminus \{i\}$. In this case the real sender would have aborted, so $\mathcal{I}$ aborts.

Case $1.2^\dagger$: $\mathcal{A}$ sent correct locations for the triplets $\{1,2,3\} \setminus \{i\}$, but one of the triplets he wants the sender to return (k_1 or k_2) is actually triplet i. $\mathcal{I}$ chooses π_1 randomly as required by the protocol (note: below, we always refer to the permutation used shuffle the receiver's triplet as π_1 and the permutation used to shuffle the sender's triplet as π_2). The simulation continues depending on whether $\mathcal{I}$ cheated successfully:

Case 1.2.1: $\mathcal{I}$ cheated successfully and received b_0 and b_1 (this occurs with probability $\frac{1}{3}$). In this case $\mathcal{I}$ continues the simulation until step (10), where $\mathcal{A}$ sends ℓ_2, its guess for $\pi_2(2)$, to the sender. At this point $\mathcal{I}$ always accepts (equivalently, it selects π_2 at random from the set of permutations for which $\pi_2(2) = \ell_2$). $\mathcal{I}$ can now continue simulating a real sender, following the protocol exactly.

Case 1.2.2: $\mathcal{I}$ failed to cheat and did not receive b_0, b_1. $\mathcal{I}$ continues the simulation until the end of step (10), where $\mathcal{A}$ sends ℓ_2, its guess for $\pi_2(2)$. at this point $\mathcal{I}$ always aborts (equivalently, it selects π_2 at random from the set of permutations for which $\pi_2(2) \neq \ell_2$, and continues the simulation for the sender, who will then abort).

Case $1.3^\dagger$: $\mathcal{A}$ sent correct locations for the triplets $\{1,2,3\} \setminus \{i\}$ and both the triplets he asks the sender to return are the receiver's. In this case simulates the sender returning thw two triplets two the receiver. $\mathcal{I}$ chooses a random bit a'. If the receiver asks to open his triplet, $\mathcal{I}$ returns answers consistent with the sender encoding a' on the receiver's triplet. $\mathcal{I}$ continues the simulation until step (24), when the receiver sends the bit b'. Since $\mathcal{I}$ knows σ, given b' $\mathcal{I}$ can compute the unique value, b, that is consistent with the input of an honest receiver using the same permutation σ and the same public messages. $\mathcal{I}$ sends a **Choice** b command to $\mathcal{F}^{(\frac{1}{2},\frac{1}{3}-PCWOT)}$ and receives a_b. $\mathcal{I}$ then simulates the sender responding with $a_b \oplus a'$ to the receiver in stage (30). The simulation then continues until it $\mathcal{A}$ halts.

Case 2†: Of the triplets created by the sender, at most two are good and returned by $\mathcal{A}$ at step (3). Let j be the index of a bad (or missing) triplet (if there is more than one $\mathcal{I}$ chooses randomly between them). The simulation continues depending on whether $\mathcal{I}$ can cheat successfully:

Case 2.1: $\mathcal{I}$ received both b_0 and b_1 (this occurs with probability $\frac{1}{3}$). In this case $\mathcal{I}$ chooses $i = j$. $\mathcal{I}$ then continues the protocol simulating an honest sender and letting the ideal receiver output whatever the $\mathcal{A}$ commands it to output.

Case 2.2: $\mathcal{I}$ cannot cheat successfully. In this case $\mathcal{I}$ chooses i randomly from $\{1, 2, 3\} \setminus \{j\}$. This forces $\mathcal{A}$ to send the simulated sender the location of triplet j at step (5). No matter what he sends the real sender running the protocol in the "real-world" scenario would abort. Hence $\mathcal{I}$ always aborts at step (6).

Lemma 2.26. *For any environment machine $\mathcal{Z}$, and any real adversary $\mathcal{A}$ that corrupts only the receiver, the output of $\mathcal{Z}$ when communicating with $\mathcal{A}$ in the real world is identically distributed to the output of $\mathcal{Z}$ when communicating with $\mathcal{I}$ in the ideal world.*

Proof. The proof is by case analysis. $\mathcal{I}$'s decision tree implicitly groups all possible adversaries by their actions at critical points in the protocol. To show that $\mathcal{Z}$'s view of the protocol is identically distributed in the real and ideal worlds, it is enough to show that the distribution of the view is identical given any specific choice by $\mathcal{Z}$ and $\mathcal{A}$. Since $\mathcal{I}$'s actions are identical for all adversaries in the same group, it is enough to consider the groups implied by $\mathcal{I}$'s decision tree.

Case 1.1 This is the case where $\mathcal{A}$ returned triplets that were all good, but sent incorrect locations for the sender's triplets. $\mathcal{Z}$'s view in this case consists only of **Receipt** messages, the index i that is chosen at random both in the real world and in the ideal world, and the $\perp$ message sent by the sender.

Case 1.2 This is the case where $\mathcal{A}$ returned triplets that were all good, but asked for triplet i instead of his own triplets. $\mathcal{Z}$'s view up to step (10) consists of the **Receipt** messages, the index i, the permutation π_1. All these are chosen identically in both the real and ideal worlds. In the real world, with probability $\frac{1}{3}$ the sender would have chosen π_2 that is inconsistent with $\mathcal{A}$'s guess ℓ_2, in which case the protocol would halt with the sender outputting $\perp$. In the ideal world, $\mathcal{I}$ can cheat with probability $\frac{1}{3}$, so with the same probability the protocol halts and the sender outputs $\perp$. Conditioned on the protocol not halting, the view in both cases is also identically distributed, because in the ideal world $\mathcal{I}$ cheated successfully and can simulate the real sender exactly (since it now knows a_0 and a_1).

Case 1.3 This is the case where the adversary follows the protocol exactly (as far as messages sent to the sender and the $\mathcal{F}^{(IWL)}$ functionality). In this case, $\mathcal{I}$ also simulates an honest sender exactly until step (13). In the real and ideal worlds, the bit encoded on Bob's triplet (a') is uniformly random. The response sent in stage (30) is in both cases completely determined (in the same way) by a', the input bits a_0, a_1 and the receiver's actions.

Case 2 This is the case where the adversary opened (or replaced) containers before returning them in stage (3). The view up to this stage in both the real and ideal world consists of **Receipt** messages and the ids of the opened containers (the contents are always 1 bits). In both the real world and ideal worlds, the index i sent by the sender is uniformly distributed in $\{1, 2, 3\}$ (in the ideal world this is because the probability that $\mathcal{I}$ cheats successfully is $\frac{1}{3}$, so that with probability $\frac{1}{3}$, i is set to some fixed j, and with probability $\frac{2}{3}$ it is set to one of the other values). Also, in both worlds, the probability that the sender picked an index which was opened (replaced) by $\mathcal{A}$ is determined by the number of containers that were opened (and is at least $\frac{1}{3}$). In either case, $\mathcal{I}$ can exactly simulate the sender, since if cheating was unsuccessful the protocol will necessarily halt before $\mathcal{I}$ needs to use the sender's inputs. Thus, in both cases the protocol will be identically distributed.

□

2.10.2 $\mathcal{A}$ corrupts the sender

Assume $\mathcal{A}$ begins by corrupting the sender. $\mathcal{I}$ corrupts the real sender and sends a **CanCheat** command to $\mathcal{F}^{(\frac{1}{2},\frac{1}{3}-PCWOT)}$. $\mathcal{I}$ then runs the simulation until the simulated sender sends nine containers. $\mathcal{I}$ simulates the receiver returning the containers to the sender (note that the steps in the protocol now refer to Protocol 2.5b). The simulation now depends on $\mathcal{A}$'s actions:

Case 1†: $\mathcal{A}$ asks to open one of the containers before sending i to the receiver in step (6). Denote the index of the container $\mathcal{A}$ opens by j. $\mathcal{I}$ continues the simulation based on the response to the **CanCheat** command:

Case 1.1: $\mathcal{I}$ can cheat. In this case $\mathcal{I}$ pretends $\mathcal{A}$ opened one of the sender's containers (chosen randomly); $\mathcal{I}$ selects a random permutation $\sigma \in S_6$ from the set of permutations that map one of the sender's containers to index j. $\mathcal{I}$ then continues the simulation to the end, as if the receiver was honest and had shuffled the containers using σ. If the simulation reaches stage (12) without anyone aborting, $\mathcal{I}$ sends a **Send** $0, 0$ command to $\mathcal{F}^{(\frac{1}{2},\frac{1}{3}-PCWOT)}$ and waits for the real (ideal dummy) receiver to send the **Choice** command. $\mathcal{I}$ then continues the simulation using the the real receiver's bit. After the sender sends the bit in step (22), $\mathcal{I}$ calculates the simulated receiver's output and sends a **Resend** command to $\mathcal{F}^{(\frac{1}{2},\frac{1}{3}-PCWOT)}$ using that bit.

Case 1.2: $\mathcal{I}$ can't cheat. In this case $\mathcal{I}$ selects a random permutation $\sigma \in S_6$ from the set of permutations that map one of the receiver's containers to index j. $\mathcal{I}$ then continues the simulation as if the receiver had shuffled the containers using σ. No matter what $\mathcal{A}$ does, $\mathcal{I}$ will then abort at step (9), (13) or (17), since that is what the real receiver would have done.

Case 2†: $\mathcal{A}$ does not open any container before sending i in stage (6). The simulation continues until stage (9) or until $\mathcal{A}$ asks to open a container that shouldn't be opened according to the protocol:

Case 2.1†: $\mathcal{A}$ does not open any container (except those called for by the protocol, which will always be $\mathcal{A}$'s own containers) until the beginning of stage (9). Note that in this case w.l.o.g., $\mathcal{A}$ can wait to open containers until step (11). $\mathcal{I}$ continues the simulation, randomly choosing σ at stage (10). The simulation can now take the following paths:

Case 2.1.1†: $\mathcal{A}$ does not open any container until step (12). By this stage the sender no longer holds any containers, so $\mathcal{A}$ cannot open containers later either. $\mathcal{I}$ continues the simulation using 0 in place of the receiver's choice bit. Since $\mathcal{I}$ knows what exchanges $\mathcal{A}$ made on each of the triplets, at the end of the protocol it can recover both a_0 and a_1. It sends a **Send** a_0, a_1 command to $\mathcal{F}^{(\frac{1}{2},\frac{1}{3}-PCWOT)}$.

Case 2.1.2†: $\mathcal{A}$ opens one of the containers before step (12).

Case 2.1.2.1: $\mathcal{I}$ can cheat. In this case $\mathcal{I}$ pretends the container $\mathcal{A}$ opens belongs to the sender's triplet. $\mathcal{I}$ sends a **Send** $0, 0$ command to $\mathcal{F}^{(\frac{1}{2},\frac{1}{3}-PCWOT)}$ and waits for the real receiver to send the **Choice** command. $\mathcal{I}$ then continues the simulation using the real receiver's bit. After the corrupt sender sends the bit in stage (22), $\mathcal{I}$ calculates the simulated receiver's output and sends a **Resend** command to $\mathcal{F}^{(\frac{1}{2},\frac{1}{3}-PCWOT)}$ using that bit.

Case 2.1.2.2: $\mathcal{I}$ can't cheat. In this case $\mathcal{I}$ pretends the container $\mathcal{A}$ opens belongs to the receiver's triplet. Whatever $\mathcal{A}$ does, $\mathcal{I}$ will then abort in step (13) or (17).

Case 2.2†: $\mathcal{A}$ asks to open a container not called for by the protocol before stage (9). Denote the index of this container by j. $\mathcal{I}$'s actions depend on whether it can cheat:

Case 2.2.1: $\mathcal{I}$ can cheat. In this case $\mathcal{I}$ selects a random permutation $\sigma \in S_6$ from the set of permutations that map one of the sender's containers to index j. $\mathcal{I}$ then continues the simulation to the end as if the receiver was honest and had shuffled the containers using σ. If the simulation reaches step (11) without anyone aborting, $\mathcal{I}$ sends a **Send** $0, 0$ message to $\mathcal{F}^{(\frac{1}{2},\frac{1}{3}-PCWOT)}$ and waits for the real receiver to send a **Choice** message. $\mathcal{I}$ continues the simulation using the real receiver's bit. At the end of the simulation, $\mathcal{I}$ knows the simulated receiver's output and uses that in a **Resend** command to $\mathcal{F}^{(\frac{1}{2},\frac{1}{3}-PCWOT)}$.

Case 2.2.2: $\mathcal{I}$ can't cheat. In this case $\mathcal{I}$ selects a random permutation $\sigma \in S_6$ from the set of permutations that map one of the receiver's containers to index j. $\mathcal{I}$ then continues the simulation as if the receiver had shuffled the containers using σ. If an opened container is sent to the receiver in step (8), $\mathcal{I}$ will then abort at stage (9), since that is what the real receiver would have done. If the opened container is not sent to the receiver at step (8), $\mathcal{I}$ will abort at step (13) or (17).

Lemma 2.27. *For any environment machine $\mathcal{Z}$, and any real adversary $\mathcal{A}$ that corrupts only the sender, the output of $\mathcal{Z}$ when communicating with $\mathcal{A}$ in the real world is identically distributed to the output of $\mathcal{Z}$ when communicating with $\mathcal{I}$ in the ideal world.*

Proof. The proof is by case analysis. $\mathcal{I}$'s decision tree implicitly groups all possible adversaries by their actions at critical points in the protocol. To show that $\mathcal{Z}$'s view of the protocol is identically distributed in the real and ideal worlds, it is enough to show that the distribution of the view is identical given any specific choice by $\mathcal{Z}$ and $\mathcal{A}$. Since $\mathcal{I}$'s actions are identical for all adversaries in the same group, it is enough to consider the groups implied by $\mathcal{I}$'s decision tree.

Case 1 This is the case where $\mathcal{A}$ first deviates from the protocol by opening one of the containers before sending i in step (6). In the real world, the receiver's choice of σ is uniformly random. Thus, no matter what container $\mathcal{A}$ chooses to open, it will be one of the receiver's containers with probability $\frac{1}{2}$. In the ideal world, $\mathcal{I}$'s choice of σ is also random: with probability $\frac{1}{2}$, $\mathcal{I}$ can cheat, in which case σ is chosen from the half of S_6 permutations that map j to the sender's containers. With probability $\frac{1}{2}$, $\mathcal{I}$ cannot cheat, in which case σ is chosen from the half of S_6 permutations that map j to the receiver's containers. The rest of the simulation in the ideal world is an exact simulation of the real receiver (in the case that $\mathcal{I}$ cannot cheat, it will never need to use the sender's input bits, since it will halt in step (9), (13) or (17). Thus in both cases the protocol view is identically distributed.

Case 2.1.1 This is the case where $\mathcal{A}$ honestly follows the protocol (from the point of view of $\mathcal{I}$). In this case, up to stage (12), $\mathcal{I}$ simulates a real receiver exactly. The only difference between the simulation and the real world is that $\mathcal{I}$ uses the choice bit 0 in the simulation rather than the receiver's input bit. However, the view of $\mathcal{A}$ is identical, since in both cases the bit requested by the receiver in stage (12) is uniformly random (because σ is chosen at random, and $\mathcal{A}$ has no information about the order of the final two triplets). The receiver's output is identical in both worlds, because $\mathcal{I}$ can compute the sender's inputs from $\mathcal{A}$'s actions.

Case 2.1.2 This is the case where $\mathcal{A}$ first deviates from the protocol by opening a container during step (11). Up to the deviation from the protocol, $\mathcal{I}$ simulates the real receiver exactly, so the protocol view up to that point is identical in both worlds. In both worlds $\mathcal{A}$ has no information about the order of the two remaining triplets (this is determined by the choice of σ and i). In the real world, the container $\mathcal{A}$ opens will be the receiver's container with probability $\frac{1}{2}$. In the ideal world, this will also be the case, since $\mathcal{I}$ can cheat with probability $\frac{1}{2}$. If $\mathcal{I}$ can cheat, the rest of the simulation exactly follows the protocol (since $\mathcal{I}$ now knows the real receiver's choice bit). If $\mathcal{I}$ cannot cheat, the choice of σ ensures that the rest of the simulation still follows the protocol exactly, since the receiver will abort before it needs to use its choice bit. Thus, in both worlds the protocol view is identically distributed.

Case 2.2 This is the case where $\mathcal{A}$ first deviates from the protocol by opening a container after sending i in step (6) but before stage (9). As in Case 1 (and for the same reasons), σ is uniformly distributed in both worlds. If $\mathcal{I}$ can cheat, the simulation follows the protocol exactly ($\mathcal{I}$ knows the real receiver's choice), so the view is identical. If $\mathcal{I}$ cannot cheat the choice of σ ensures that $\mathcal{I}$ will never have to use the real receiver's choice, so the view is again distributed identically to the real world.

□

Together, Lemma 2.26 and Lemma 2.27 prove Theorem 2.17.

2.11 Discussion and Open Problems

2.11.1 Zero Knowledge Without Bit Commitment

In the bare model, where bit-commitment is impossible, Zero knowledge proofs exist only for languages in SZK — which is known to be closed under complement and is thus unlikely contain NP. An interesting open question is whether the class of languages that have zero-knowledge proofs in the DWL model (where bit-commitment is impossible; see Section 2.3.3) is strictly greater than SZK (assuming $P \neq NP$).

2.11.2 Actual Human Feasibility

The protocols we describe in this paper can be performed by unaided humans, however they require too many containers to be practical for most uses. It would be useful to construct protocols that can be performed with a smaller number of containers (while retaining security), and with a smaller number of rounds.

Another point worth mentioning is that the protocols we construct in the distinguishable models only require one of the parties to seal and verify containers. Thus, the binding property is only used in one direction, and the tamper-evidence and hiding properties in the other. This property is useful when we want to implement the protocols in a setting where one of the parties may be powerful enough to open the seal undetectably. This may occur, for instance, in the context of voting, where one of the parties could be "the government" while the other is a private citizen.

In both the weakly and strongly-fair CF protocols, only the first round requires envelopes to be created, and their contents do not depend on communication with the other party. This allows the protocols to be implemented using scratch-off cards (which must be printed in advance). In particular, the weakly-fair coin flipping protocol can be implemented with a scratch-off card using only a small number of areas to be scratched.

In the case of bit-commitment, our protocol requires the powerful party to be the receiver. It would be interesting to construct a BC protocol for which the powerful party is the sender (i.e., only the sender is required to to seal and verify envelopes).

Chapter 3

Polling With Physical Envelopes: A Rigorous Analysis of a Human-Centric Protocol

3.1 Introduction

In the past few years, a lot of attention has been given to the design and analysis of electronic voting schemes. Constructing a protocol that meets all (or even most) of the criteria expected from a voting scheme is generally considered to be a tough problem. The complexity of current protocols (in terms of how difficult it is to describe the protocol to a layperson) reflects this fact. A slightly easier problem, which has not been investigated as extensively, is that of polling schemes.

Polling schemes are closely related to voting, but usually have slightly less exacting requirements. In a polling scheme the purpose of the pollster is to get a good statistical profile of the responses, however some degree of error is admissible. Unlike voting, absolute secrecy is generally not a requirement for polling, but some degree of response privacy is often necessary to ensure respondents' cooperation.

The issue of privacy arises because polls often contain questions whose answers may be incriminating or stigmatizing (e.g., questions on immigration status, drug use, religion or political beliefs). Even if promised that the results of the poll will be used anonymously, the accuracy of the poll is strongly linked to the trust responders place in the pollster. A useful rule of thumb for polling sensitive questions is "better privacy implies better data": the more respondents trust that their responses cannot be used against them, the likelier they are to answer truthfully. Using polling techniques that clearly give privacy guarantees can significantly increase the accuracy of a poll.

A well-known method for use in these situations is the "randomized response technique" (RRT), introduced by Warner in 1965 [75]. Roughly, Warner's idea was to tell responders to lie with some fixed, predetermined, probability (e.g., roll a die and lie whenever the die shows one or two). As the probability of a truthful result is known exactly, statistical analysis of the results is still possible[1], but an individual answer is always plausibly deniable (the respondent can always claim the die came up one).

Unfortunately, in some cases this method causes its own problems. In pre-election polls, for example, responders have a strong incentive to always tell the truth, ignoring the die (since the results of the polls are believed to affect the outcome of the elections). In this case, the statistical analysis will give the cheating responders more weight than the honest responders. Ambainis, Jakobsson and Lipmaa [3] proposed the "Cryptographic Randomized Response Technique" to deal with this problem. Their paper contains a

[1] For instance, suppose $p > \frac{1}{2}$ is the probability of a truthful response, n is the total number of responses, x is the number of responders who actually belong in the "yes" category and R is the random variable counting the number of "yes" responses. R is the sum of n independent indicator random varables, so R is a good estimation for $E(R) = px+(1-p)(n-x) = x(2p-1)+n(1-p)$. Therefore, given R, we can accurately estimate the actual number of "yes" responders: $x = \frac{E(R)-n(1-p)}{2p-1}$.

number of different protocols that prevent malicious responders from biasing the results of the poll while preserving the deniability of the randomized response protocol. Unlike Warner's original RRT, however, the CRRT protocols are too complex to be implemented in practice without the aid of computers. Since the main problem with polling is the responders' lack of trust in the pollsters, this limitation makes the protocols of [3] unsuitable in most instances.

The problem of trust in complex protocols is not a new one, and actually exists on two levels. The first is that the protocol itself may be hard to understand, and its security may not be evident to the layman (even though it may be formally proved). The second is that the computers and operating system actually implementing the protocol may not be trusted (even though the protocol itself is). This problem is more acute than the first. Even for an expert, it is very difficult to verify that a computer implementation of a complex protocol is correct.

Ideally, we would like to design protocols that are simple enough to grasp intuitively and can also be implemented transparently (so that the user can follow the steps and verify that they are correct).

3.1.1 Our Results

In this paper we propose two very simple protocols for cryptographic randomized response polls, based on *tamper-evident seals* (introduced in a previous paper by the authors [53]). A tamper-evident seal is a cryptographic primitive that captures the properties of a sealed envelope: while the envelope is sealed, it is impossible to tell what's inside, but if the seal is broken the envelope cannot be resealed (so any tampering is evident). In fact, our CRRT protocols are meant to be implemented using physical envelopes (or scratch-off cards) rather than computers. Since the properties of physical envelopes are intuitively understood, even by a layman, it is easy to verify that the implementation is correct.

The second important contribution of this paper, differentiating it from previous works concerning human-implementable protocols, is that we give a formal definition and a rigorous proof of security for the protocols. The security is unconditional: it relies only on the physical tamper-evidence properties of the envelopes, not on any computational assumption. Furthermore, we show that the protocols are "universally composable" (as defined by Canetti [16]). This is a very strong notion of security that implies, via Canetti's Composition Theorem, that the security guarantees hold even under general concurrent composition

Our protocols implement a relaxed version of CRRT (called *weakly secure* in [3]). We also give an inefficient strong CRRT protocol (that requires a large number of rounds), and give impossibility results and lower bounds for strong CRRT protocols with a certain range of parameters (based on Cleve's lower bound for coin flipping [24]). These suggest that constructing a strong CRRT protocol using scratch-off cards may be difficult (or even impossible if we require a constant number of rounds).

3.1.2 Related Work

Randomized Response Technique. The randomized response technique for polling was first introduced in 1965 [75]. Since then many variations have been proposed (a survey can be found in [18]). Most of these are attempts to improve or change the statistical properties of the poll results (e.g., decreasing the variance), or changing the presentation of the protocol to emphasize the privacy guarantee (e.g., instead of lying, tell the responders to answer a completely unrelated question). A fairly recent example is the "Three Card Method" [36], developed for the United States Government Accountability Office (GAO) in order to estimate the size of the illegal resident population. None of these methods address the case where the responders maliciously attempt to bias the results.

To the best of our knowledge, the first polling protocol dealing explicitly with malicious bias was given by Kikuchi, Akiyama, Nakamura and Gobioff. [49], who proposed to use the protocol for voting (the protocol described is a randomized response technique, although the authors do not appear to have been aware of the previous research on the subject). Their protocol is still subject to malicious bias using a "premature halting" attack (this is equivalent to the attack on the RRT protocol in which the responder rolls a die but refuses to answer if result of the die is not to his liking). A more comprehensive treatment, as well as a formal definition of cryptographic randomized response, was given by Ambainis et al. [3]. In their paper, Ambainis et al. also give a protocol for *Strong* CRRT, in which the premature halting attack is impossible.

In both the papers [49, 3], the protocols are based on cryptographic assumptions and require computers to implement.

Independently of this work, Stamm and Jakobsson show how to implement the protocol of [3] using playing cards [72]. They consider this implementation only as a visualization tool. However, if we substitute envelopes for playing cards (and add a verification step), this protocol gives a Responder-Immune protocol (having some similarities to the one described in Section 3.3.2).

Deniable and Receipt-Free Protocols. The issues of deniability and coercion have been extensively studied in the literature (some of the early papers in this area are [7, 69, 15, 17, 37]). There are a number of different definitions of what it means for a protocol to be *deniable*. Common to all of them is that they protect against an adversary that attacks actively only *after* the protocol execution: in particular, this allows the parties to lie about their random coins. Receipt-Free protocols provide a stronger notion of security: they guarantee that even if a party is actively colluding with the adversary, the adversary should have no verifiable information about which input they used. Our notion of "plausible deniability" is weaker than both "traditional" deniability and receipt-freeness, in that we allow the adversary to gain some information about the input. However, as in receipt-freeness, we consider an adversary that is active before and during the protocol, not just afterwards.

Secure Protocols Using "Real" Objects. The idea of using real objects to provide security predates cryptography: people have been using seals, locks and envelopes for much of history. Traditionally, the objects were constructed directly to solve a given problem (e.g., locks were created specifically to provide access control). Using real objects to implement protocols for tasks that are not obviously related to their original purpose is a newer notion. Fagin, Naor and Winkler [38] propose protocols for comparing secret information that use various objects, from paper cups to the telephone system. In a more jocular tone, Naor, Naor and Reingold [59] propose a protocol that provides a "zero knowledge proof of knowledge" of the correct answer to the children's puzzle "Where's Waldo" using "low-tech devices" (e.g., a large newspaper and scissors). In all these works the security assumptions and definitions are informal or unstated. Crépeau and Kilian [31] show how to use a deck of cards to play "discreet" solitary games (these involve hiding information from yourself). Their model is formally defined, however it is not malicious; the solitary player is assumed to be honest but curious.

A related way of using real objects is as aids in performing a "standard" calculation. Examples in this category include Schneier's "Solitaire" cipher [70] (implemented using a pack of cards), and the "Visual Cryptography" of Naor and Shamir [61] (which uses the human visual system to perform some basic operations on images). The principles of Visual Cryptography form the basis for some more complex protocols, such as the "Visual Authentication" protocol of Naor and Pinkas [60], and Chaum's human verifiable voting system [21].

Tamper-Evident Seals. This work can be viewed as a continuation of a previous work by the authors on tamper-evident seals [53]. In [53], we studied the possibility of implementing basic cryptographic primitives using different variants of physical, tamper-evident seals. In the current work we focus on their use in realistic cryptographic applications, rather than theoretical constructs (for instance, there is a very sharp limit on the number of rounds and the number of envelopes that can be used in a protocol that we expect to be practical for humans). We limit ourselves to the "distinguishable envelope" (DE) model, as this model has a number of intuitive physical embodiments, while at the same time is powerful enough, in theory, to implement many useful protocols[2] (an informal description of this model is given in Section 3.2.3; for a formal definition see [53]).

Overview of Paper. In Section 3.2, we give formal definitions of the functionalities we would like to realize and the assumptions we make about the humans implementing the protocols. Section 3.3 gives an informal description of the CRRT protocols. In Section 3.6, we show how to amplify a weak pollster-immune CRRT protocol in order to construct a strong CRRT protocol, and give some impossibility results and lower bounds for strong CRRT protocols. The formal protocol specifications and proofs of security for our pollster-immune and responder-immune CRRT protocols appear in Sections 3.4 and 3.5, respectively. Finally, a discussion

[2]Although the "indistinguishable envelope model" (also defined in [53]) is stronger (e.g., oblivious transfer *is* possible in this model), it seems to be very hard to devise a secure, physical realization of this functionality.

and some open problems appear in Section 3.7.

3.2 The Model

Ideal Functionalities. Many two-party functionalities are easy to implement using a trusted third party that follows pre-agreed rules. In proving that a two-party protocol is secure, we often want to say that it behaves "as if it were performed using the trusted third party".

The "Universal Composability" framework, defined by Canetti [16], is a formalization of this idea. In the UC model, the trusted third party is called the *ideal functionality*. If every attack against the protocol can also be carried out against the ideal functionality, we say the protocol realizes the functionality. Canetti's Composition Theorem says that *any* protocol that is secure using the ideal functionality, will remain secure if we replace calls to the ideal functionality with executions of the protocol.

Defining the security guarantees of our protocols as ideal functionalities has an additional advantage as well: it is usually easier to understand what it means for a protocol to satisfy a definition in this form than a definition given as a list of properties. Below, we describe the properties we wish to have in a CRRT protocol, and give formal definitions in the form of ideal functionalities.

3.2.1 Cryptographic Randomized Response

A randomized response protocol involves two parties, a pollster and a responder. The responder has a secret input bit b (this is the true response to the poll question). In the ideal case, the pollster learns a bit c, which is equal to b with probability p (p is known to the pollster) and to $1-b$ with probability $1-p$. Since p is known to the pollster, the distribution of responders' secret inputs can be easily estimated from the distribution of the pollster's outputs.

The essential property we require of a Randomized Response protocol is *plausible deniability*: A responder should be able to claim that, with reasonable probability, the bit learned by the pollster is not the secret bit b. This should be the case even if the pollster maliciously deviates from the protocol.

A *Cryptographic* Randomized Response protocol is a Randomized Response protocol that satisfies an additional requirement, *bounded bias*: The probability that $c = b$ must be at most p, even if the responder maliciously deviates from the protocol. The bounded bias requirement ensures that malicious responders cannot bias the results of the poll (other than by changing their own vote). Note that even in the ideal case, a responder can always choose any bias p' between p and $1-p$, by randomly choosing whether to vote b or $1-b$ (with the appropriate probability).

Strong p-CRRT

In a *strong* CRRT protocol, both the deniability and bounded bias requirements are satisfied. Formally, this functionality has a single command:

Vote x The issuer of this command is the responder. On receiving this command the functionality tosses a weighted coin c, such that $c = 0$ with probability p. It then outputs $x \oplus c$ to the pollster and the adversary.

Unfortunately, we do not know how to construct a *practical* strong CRRT protocol that can be implemented by humans. In Section 3.6, we present evidence to suggest that finding such a protocol may be hard (although we do show an *impractical* strong CRRT protocol, that requires a large number of rounds). The protocols we propose satisfy relaxed conditions: The first protocol is immune to malicious pollsters (it is equivalent to strong CRRT if the responder is honest), while the second is immune to malicious responders (it is equivalent to strong CRRT if the pollster is honest).

Pollster-Immune p-CRRT (adapted from Weak CRRT in [3])

This is a weakened version of CRRT, where a malicious pollster cannot learn more than an honest pollster about the responder's secret bit. A malicious responder can bias the result by deviating from the protocol

(halting early). A cheating responder will be caught with fixed probability, however, so the pollster can accurately estimate the number of responders who are cheating (and thus bound the resulting bias). When the pollster catches the responder cheating, it outputs ⊠ instead of its usual output. Formally, the ideal functionality accepts the following commands:

Query The issuer of this command is the pollster, the other party is the responder. The functionality ignores all commands until it receives this one. On receiving this command the functionality chooses a uniformly random bit r (we'll call this the *provisional result* bit) and a bit v, such that $v = 1$ with probability $2p-1$ (we'll call v the *responder control* bit). If the responder is corrupted, the functionality then sends both bits to the adversary.

Vote b On receiving this command from the responder, the functionality checks whether $v = 1$ (i.e., the responder has control). If so, it outputs b to the pollster, otherwise it outputs r (the provisional result chosen in response to the **Query** command) to the pollster.

Halt This command captures the responder's ability to cheat. On receiving this command from a corrupt responder, the functionality outputs ⊠ to the pollster and halts.

The functionality described above is slightly more complex (and a little weaker) than would appear to be necessary, and this requires explanation. Ideally, the functionality should function as follows: the responder casts her vote, and is notified of the actual bit the pollster would receive. The responder then has the option to halt (and prevent the pollster from learning the bit). Our protocol gives the corrupt responder a little more power: the responder first learns the control bit v (determining whether the responder has any control over the bit sent to the pollster), and the provisional response bit r (which will be sent if the responder does not have control). The responder can then plan her actions based on this information. This slightly weaker functionality is the one that is actually realized by our protocol (for $p = \frac{3}{4}$).

Responder-Immune p-CRRT

In this weakened version of CRRT, malicious responders cannot bias the results more than honest responders, but a malicious pollster can learn the responder's secret bit. In this case, however, the responder will discover that the pollster is cheating. When the responder catches the pollster cheating, it outputs ⊠ to signify this. The functionality accepts the following commands:

Vote b The issuer of this command is the responder. On receiving this command the functionality tosses a weighted coin c, such that $c = 0$ with probability p. It then outputs $b \oplus c$ to the pollster and adversary.

Reveal The command may only be sent by a corrupt pollster *after* the Vote command was issued by the responder. On receiving this command, the functionality outputs b to the adversary and ⊠ to the responder.

Test x : The command may only be sent by a corrupt pollster, after the Vote command was issued by the responder. On receiving this command:

- if $x = b$, then with prob. $\frac{1}{2}$ it outputs b to the adversary and ⊠ to the responder, and with prob. $\frac{1}{2}$ it outputs $\perp$ to the adversary (and nothing to the responder).
- if $x = 1 - b$ the functionality outputs $\perp$ to the adversary (and nothing to the responder).

Ideally, we would like to realize responder-immune CRRT without the **Test** command. Our protocol realizes this slightly weaker functionality (for $p = \frac{2}{3}$). It may appear that a corrupt pollster can cheat without being detected using the **Test** command. However, for any corrupt pollster strategy, if we condition on the pollster's cheating remaining undetected, the pollster gains no additional information about the responder's choice (since in that case the response to the **Test** command is always $\perp$).

3.2.2 Modelling Humans

The protocols introduced in this paper are meant to be implemented by humans. To formally prove security properties of the protocols, it is important to make explicit the abilities and limitations we expect from humans.

Following Instructions. The most basic assumption we make about the parties participating in the protocol is that an honest party will be able to follow the instructions of the protocol correctly. While this requirement is clearly reasonable for computers, it may not be so easy to achieve with humans. The ability to follow instructions depends on the complexity of the protocol, as well as the protocol participants and the environment in which the protocol is executed. Our protocols are secure and correct only assuming the honest parties are actually following the protocol. Unfortunately, we do not know how to predict whether this assumption actually holds for a specific protocol without "real" experimental data.

Random Choice. Our protocols require the honest parties to make random choices. Choosing a truly random bit may be very difficult for a human (in fact, even physically tossing a coin has about 0.51 probability of landing on the side it started on [35]). For the purposes of our analysis, we assume that whenever we require a party to make a random choice it is uniformly random. In practice, a random choice may be implemented using simple physical means (e.g., flipping a coin or rolling a die). In practice, the slight bias introduced by physical coin flipping will not have a large effect on the correctness or privacy of our protocols. We note that even if relatively "bad" randomness is used (e.g., just asking the parties to "choose randomly"), the security of the protocol does not break down completely. For example, if a malicious pollster can make a better guess as to the responder's random choices, he will gain additional information about the responder's "real" answer — but as long as he can't guess the random choices with certainty, the responder will still have some measure of deniability.

Non-Requirements. Unlike many protocols involving humans, we do not assume any additional capabilities beyond those described above. We don't require parties to forget information they have learned, or to perform actions obliviously (e.g., shuffle a deck without knowing what the permutation was). Of particular note, we don't require the parties to watch each other during the protocol: this means the protocols can be conducted by physical mail.

3.2.3 Distinguishable Envelopes

Our CRRT protocols require a physicial assumption: tamper-evident envelopes or scratch-off cards. Formally, we model these by an ideal functionality we call "Distinguishable Envelopes" (originally defined in [53]; for completeness, we include the definition in Appendix 3.A). Loosely speaking, a distinguishable envelope is an envelope in which a message can be sealed. Anyone can open the envelope (and read the message), but the broken seal will be evident to anyone looking at the envelope.

3.2.4 Proofs in the UC Model

Below, we outline the requirements for proofs in the UC model (this description is taken from [53]; for an in-depth explanation of the UC model, see [16]). Those who are familiar with the UC model may safely skip this subsection.

Formally, the UC model defines two "worlds", which should be indistinguishable to an outside observer called the "environment machine" (denoted $\mathcal{Z}$).

The "ideal world" contains two "dummy" parties, the "target" ideal functionality, $\mathcal{Z}$ and an "ideal adversary", $\mathcal{I}$. The parties in this world are "dummy" parties because they pass any input they receive directly to the target ideal functionality, and write anything received from the ideal functionality to their local output. $\mathcal{I}$ can communicate with $\mathcal{Z}$ and the ideal functionality, and can corrupt one of the parties. $\mathcal{I}$ sees the input and any communication sent to the corrupted party, and can control the output of that party. The environment machine, $\mathcal{Z}$, can set the inputs to the parties and read their local outputs, but cannot see the communication with the ideal functionality.

The "real world" contains two "real" parties: $\mathcal{Z}$ and the "real adversary", $\mathcal{A}$. In addition it may contain the "service" ideal functionalities (in our case the distinguishable envelope functionality). $\mathcal{A}$ can communicate with $\mathcal{Z}$ and the "service" ideal functionalities, and can corrupt one of the parties. The uncorrupted parties follow the protocol, while corrupted parties are completely controlled by $\mathcal{A}$. As in the ideal world, $\mathcal{Z}$ can set the inputs for the parties and see their outputs, but not internal communication (other than what is known to the adversary).

The protocol securely realizes an ideal functionality in the UC model, if there exists $\mathcal{I}$ such that for any $\mathcal{Z}$ and $\mathcal{A}$, $\mathcal{Z}$ cannot distinguish between the ideal world and the real world. Our proofs of security follow the general outline for a proof typical of the UC model: we describe the ideal adversary, $\mathcal{I}$, that "lives" in the ideal world. Internally, $\mathcal{I}$ simulates the execution of the "real" adversary, $\mathcal{A}$. We can assume w.l.o.g. that $\mathcal{A}$ is simply a proxy for $\mathcal{Z}$, sending any commands received from the environment to the appropriate party and relaying any communication from the parties back to the environment machine. $\mathcal{I}$ simulates the "real world" for $\mathcal{A}$, in such a way that $\mathcal{Z}$ cannot distinguish between the ideal world when it is talking to $\mathcal{I}$ and the real world. In our case we will show that $\mathcal{Z}$'s view of the execution is not only indistinguishable, but actually identical in both cases.

All the ideal adversaries used in our proofs have, roughly, the same idea. They contain a "black-box" simulation of the real adversary, intercepting its communication with the tamper-evident container functionalities and replacing it with a simulated interaction with simulated tamper-evident containers. The main problem in simulating a session that is indistinguishable from the real world is that the ideal adversary does not have access to honest parties' inputs, and so cannot just simulate the honest parties. Instead, the ideal adversary makes use of the fact that in the ideal world the "tamper-evident seals" are simulated, giving it two tools that are not available in the real world:

First, the ideal adversary does not need to commit in advance to the contents of containers (it can decide what the contents are at the time they are opened), since, in the real world, the contents of a container don't affect the view until the moment it is opened.

Second, the ideal adversary knows exactly what the real adversary is doing with the simulated containers *at the time the real adversary performs the action*, since any commands sent by the real adversary to the simulated tamper-evident container functionality are actually received by the ideal adversary. This means the ideal adversary knows when the real adversary is cheating. The target functionalities, when they allow cheating, fail completely if successful cheating gives the corrupt party "illegal" information: in case cheating is successful they give the adversary the entire input of the honest party. Thus, the strategy used by the ideal adversary is to attempt to cheat (by sending a command to the target ideal functionality) when it detects the real adversary cheating. If it succeeds, it can simulate the rest of the protocol identically to a real honest party (since it now has all the information it needs). If it fails to cheat, the ideal adversary uses its "inside" information to cause the real adversary to be "caught" in the simulation.

3.3 An Informal Presentation of the Protocols

It is tempting to try to base a CRRT protocol on oblivious transfer (OT), since if the responder does not learn what the pollster's result is, it may be hard to influence it (in fact, one of the protocols in [3] is based on OT). However, OT is impossible in the DE model [53]. As we show in Section 3.6.1, this proof implies that in any CRRT protocol using distinguishable envelopes, the responder *must* learn a lot about the pollster's result. In both our protocols, the responder gets complete information about the final result.

To make the presentation more concrete, suppose the poll question is "do you eat your veggies?". Clearly, no one would like to admit that they do not have a balanced diet. On the other hand, pressure groups such as the "People for the Ethical Treatment of Salad" have a political interest in biasing the results of the poll, making it a good candidate for CRRT.

3.3.1 Pollster-Immune CRRT

This protocol can be implemented with pre-printed scratch-off cards: The responder is given a scratch-off card with four scratchable "bubbles", arranged in two rows of two bubbles each. In each row, the word

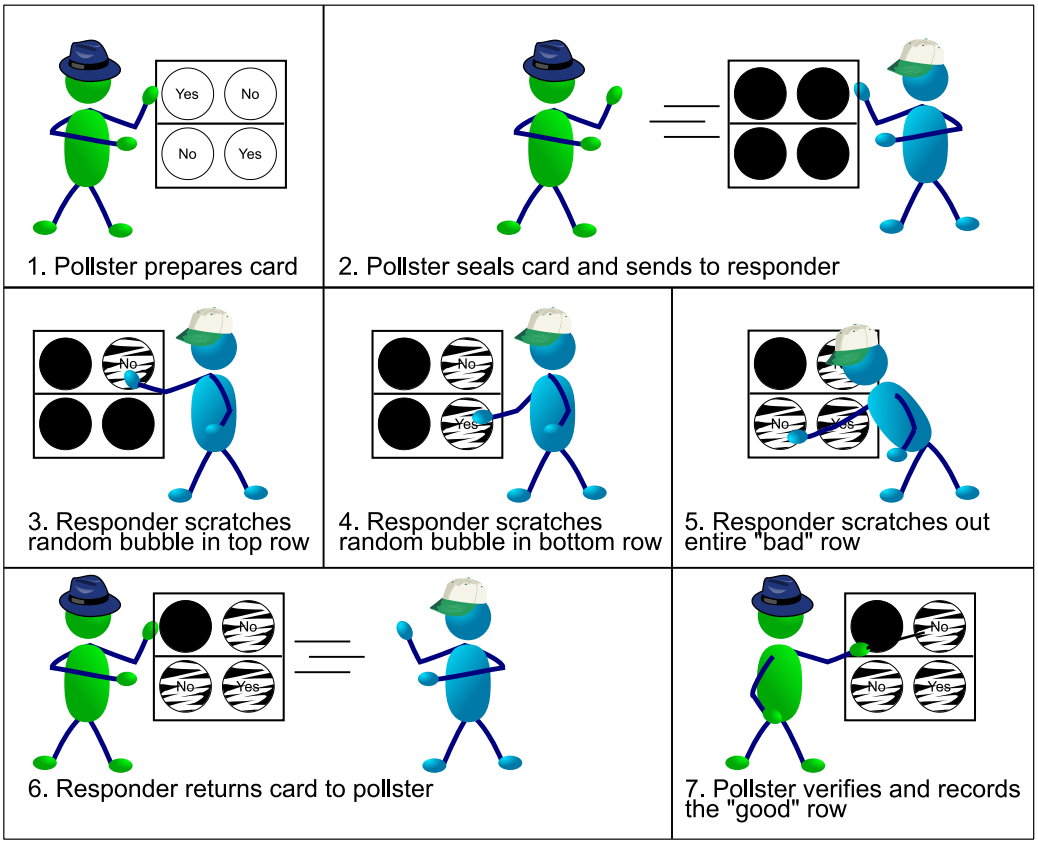

Figure 3.3.1: Sample execution of pollster-immune protocol

"Yes" is hidden under one bubble and the word "No" under the other (the responder doesn't know which is which). The responder scratches the bubbles in two stages: first, she scratches a single random bubble in each row. Suppose the responder doesn't eat her veggies. If one of the rows (or both) show the word "No", she "wins" (and the pollster will count the response as expressing dislike of vegetables). If both bubbles show "Yes", she "loses" (and the pollster will count the response as expressing a taste for salad).

In any case, before returning the card to the pollster, the responder performs the second scratching stage: she "eliminates" the row that shows the unfavored answer by scratching the entire row (she picks one of the rows at random if both rows show the same answer) Thus, as long as the responder follows the protocol, the pollster receives a card that has one "eliminated" (entirely scratched) row and one row showing the result he will count. An example of protocol execution appears in Figure 3.3.1.

Security Intuition. Note that exactly $\frac{3}{4}$ of the time, the counted result will match the responder's intended result. Moreover, without invalidating the entire card, the responder cannot succeed with higher probability. On the other hand, this provides the responder with plausible deniability: she can always claim both rows were "bad", and so the result didn't reflect her wishes. Because the pollster doesn't know which were the two bubbles that were scratched first, he cannot refute this claim. An important point is that plausible deniability is preserved even if the pollster attempts to cheat (this is what allows the responder to answer the poll accurately even when the pollster isn't trusted). Essentially, the only way the pollster can cheat without being unavoidably caught is to put the *same* answer under both bubbles in one of the rows. To get a feeling for why this doesn't help, write out the distribution of responses in all four cases (cheating/honest, Yes/No). It will be evident that the pollster does not get any additional information about the vote from cheating in this way.

On the other hand, the responder learns the result before the pollster, and can decide to quit if it's not to her liking (without revealing the result to the pollster). Since the pollster does not know the responder's outcome, this has the effect of biasing the result of the poll. However, by counting the number of prematurely halted protocol executions, the pollster can accurately estimate the *number* of cheating responders.

The formal protocol specification and proof appear in Section 3.4.

3.3.2 Responder-Immune CRRT

The responder takes three envelopes (e.g., labelled "1", "2" and "3"), and places one card containing either "Yes" or "No" in each of the envelopes. If she would like to answer "No", she places a single "Yes" card in a random envelope, and one "No" card in each of the two remaining envelopes. She then seals the envelopes and gives them to the pollster (remembering which of the envelopes contained the "Yes" card).

The pollster chooses a random envelope and opens it, revealing the card to the responder. He then asks the responder to tell him which of the two remaining envelopes contains a card with the *opposite* answer. He opens that envelope as well. If the envelope does contain a card with the opposite answer, he records the answer on the first card as the response to the poll, and returns the third (unopened) envelope to the responder.

If both opened envelopes contain the same answer, it can only be because the responder cheated. In this case, the pollster opens the third envelope as well. If the third envelope contains the opposite answer, the pollster records the answer on the first card as the response to the poll. If, on the other hand, all three envelopes contain the same answer, the pollster rolls a die: A result of 1 to 4 (probability $\frac{2}{3}$) means he records the answer that appears in the envelopes, and a result of 5 or 6 means she records the opposite answer. An example of protocol execution (where both parties follow the protocol) appears in Figure 3.3.2.

Security Intuition. In this protocol, the responder gets her wish with probability at most $\frac{2}{3}$ no matter what she does. If she follows the protocol when putting the answers in the envelopes, the pollster will choose the envelope containing the other answer with probability $\frac{1}{3}$. If she tries to cheat by putting the same answer in all three envelopes, the pollster will roll a die and choose the opposite answer with probability $\frac{1}{3}$. The pollster, on the other hand, can decide to open all three envelopes and thus discover the real answer favored by the responder. If he does this, however, the responder will see that the seal on the returned envelope was broken and know the pollster was cheating.

The pollster may also be able to cheat in an additional way: he can open two envelopes before telling the

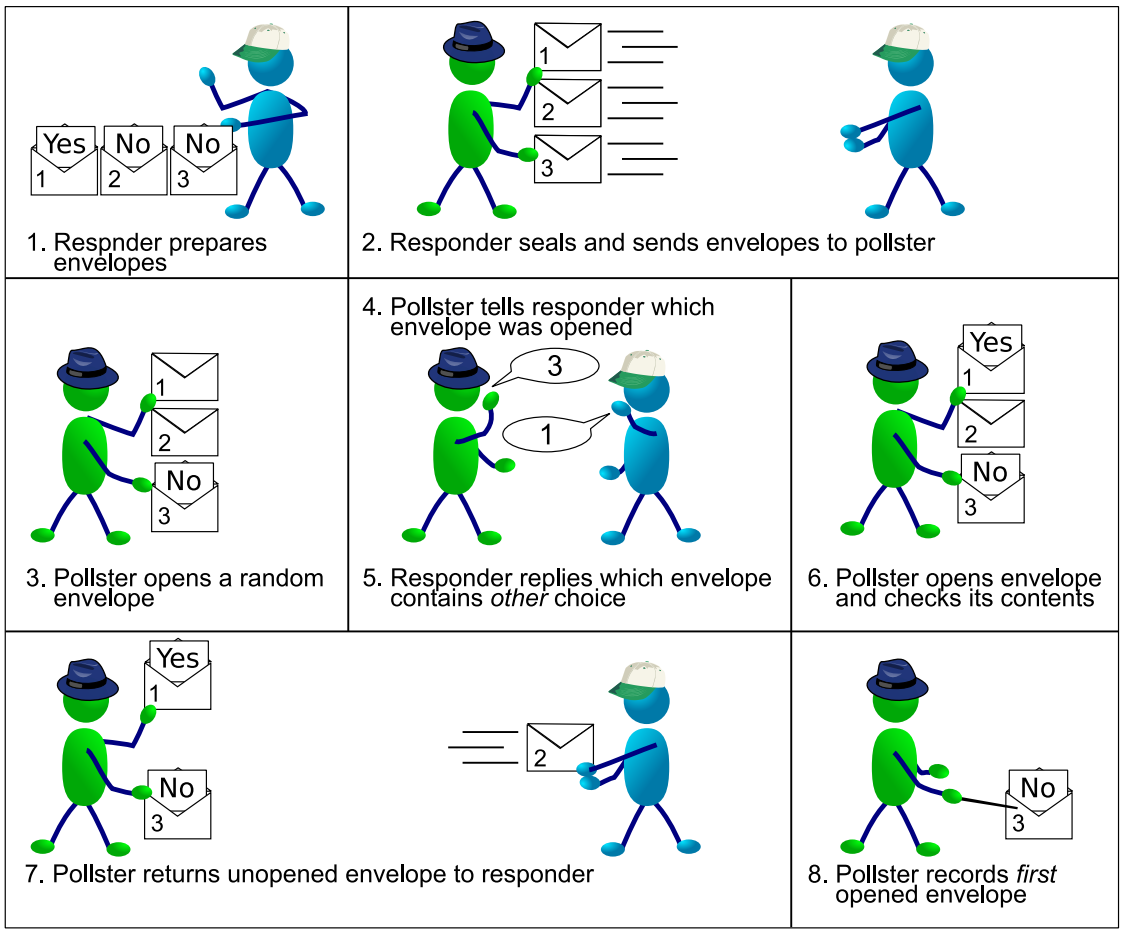

Figure 3.3.2: Sample execution of responder-immune protocol

responder which envelope he opened, and hope that the responder will not require him to return an envelope that was already opened. This attack is what requires us to add the **Test** command to the functionality.

The formal protocol specification and proof appear in Section 3.5.

Implementation Notes. This protocol requires real envelopes (rather than scratch-off cards) to implement, since the responder must choose what to place in the envelopes (and we cannot assume the responder can create a scratch-off card). In general, tamper-evidence for envelopes may be hard to achieve (especially as the envelopes will most likely be provided by the pollster). In this protocol, however, the pollster's actions can be performed in full view of the responder, so any opening of the envelopes will be immediately evident. When this is the case, the responder can tell which envelope the pollster opened first, so the protocol actually realizes the stronger version of the Responder-Immune CRRT functionality (without the **Test** command).

If the penalty for a pollster caught cheating is large enough, the privacy guaranteed by this protocol, may be enough to convince responders to answer accurately in a real-world situation even with the weaker version of the functionality. This is because any pollster cheating that can possibly reveal additional information about the responder's choice carries with it a corresponding risk of detection.

3.4 A Pollster-Immune $\frac{3}{4}$-CRRT Protocol

3.4.1 Formal Specification

A formal specification is given as Protocol 3.1 (see Section 3.3.1 for an informal description). The specification is in two parts: Protocol 3.1a describes the pollster's side of the protocol, while Protocol 3.1b describe's the responder's side. In the description of the protocol, we use the shorthand "generate an envelope" to mean sending a corresponding **Seal** command to $\mathcal{F}^{(DE)}$, and "send an envelope" to mean sending a corresponding **Send** command to $\mathcal{F}^{(DE)}$.

Protocol 3.1a Pollster-Immune $\frac{3}{4}$-CRRT (Pollster executing **Query**)

1: Choose two random bits $p_0, p_1 \in_R \{0,1\}$.
2: Generate two pairs of sealed envelopes: $(E_{0,0}, E_{0,1})$ containing $(p_0, 1-p_0)$, respectively, and $(E_{1,0}, E_{1,1})$ containing $(p_1, 1-p_1)$, respectively.
3: Send all four envelopes to responder.
4: Wait for responder to return four envelopes envelopes.
5: **if** 4 envelopes were returned and exactly 3 were opened **then**
6: **if** there exists $i \in \{0,1\}$ s.t. $E_{i,0}$ is still sealed **then**
7: Output $1-p_i$.
8: **else** {there exists $i \in \{0,1\}$ s.t. $E_{i,1}$ is still sealed}
9: Output p_i.
10: **end if**
11: **else** {Responder is trying to cheat}
12: Output $\boxtimes$.
13: **end if**

3.4.2 Proof of Security

In this section we give the proof that Protocol 3.1 securely realizes Pollster-Immune $\frac{3}{4}$-CRRT in the UC model. The proof follows the standard outline for a UC proof, as described in Secton 3.2.4.

We'll deal separately with the case when $\mathcal{A}$ corrupts the pollster, $\mathcal{P}$, and when it corrupts the responder, $\mathcal{R}$ (we consider only static corruption, where the adversary must decide ahead of time which party to corrupt). The proof that the views of $\mathcal{Z}$ in the real and ideal worlds are identical is by exhaustive case analysis.

Protocol 3.1b Pollster-Immune $\frac{3}{4}$-CRRT (Responder executing **Vote** x)

Input: a bit x

Choose two random bits $r_0, r_1 \in_R \{0,1\}$
Wait to receive envelopes $(E_{0,0}, E_{0,1})$ and $(E_{1,0}, E_{1,1})$ from pollster.
Open envelopes E_{0,r_0} and E_{1,r_1} {A random envelope from each pair}
if there exists $i \in \{0,1\}$ s.t. E_{i,r_i} contains $1-x$ and $E_{1-i,r_{1-i}}$ contains x **then**
    Let $r_2 \leftarrow i$
else {both E_{0,r_0} and E_{1,r_1} contain the same bit}
    Choose a random bit $r_2 \in_R \{0,1\}$
end if
Open envelope $E_{r_2,1-r_{r_2}}$ {the remaining unopened envelope in the pair r_2}.
if $E_{r_2,1-r_{r_2}}$ and $E_{r_2,r_{r_2}}$ contain different bits **then**
    Send $(E_{0,0}, E_{0,1})$ and $(E_{1,0}, E_{1,1})$ back to pollster. {Note that one envelope will remain sealed}
else {Pollster tried to cheat}
    Output $\boxtimes$.
end if

$\mathcal{A}$ corrupts $\mathcal{P}$

The ideal-world simulator $\mathcal{I}$, proceeds as follows:

1. $\mathcal{I}$ waits to receive c, the outcome of the poll from the ideal functionality. $\mathcal{I}$ now begins simulating $\mathcal{F}^{(DE)}$ and $\mathcal{R}$ (the simulated $\mathcal{R}$ follows the protocol exactly as an honest party would). The simulation runs until $\mathcal{P}$ sends four envelopes as required by the protocol (up to this point $\mathcal{R}$ did not participate at all in the protocol).
2. $\mathcal{I}$'s simulation now depends on the values in the envelopes created by $\mathcal{P}$:

Case 2.1: If both pairs of envelopes are valid (contain a 0 and a 1), $\mathcal{I}$ chooses one of the pairs at random and simulates opening the envelope in the pair that contains c and both envelopes in the other pair (there is an assignment to the random coins of $\mathcal{R}$ which would have this result in the real world). It then simulates the return of all four envelopes to $\mathcal{P}$.

Case 2.2: If both pairs of envelopes are invalid, $\mathcal{I}$ simulates $\mathcal{R}$ halting (this would eventually happen in a real execution as well).

Case 2.3: If exactly one pair of envelopes is invalid, denote the value in the invalid pair by z.

Case 2.3.1: If $c = z$, $\mathcal{I}$ simulates opening both envelopes in the valid pair and a random envelope in the invalid pair (depending on the random coins of $\mathcal{R}$, this is a possible result in the real world). It then simulates the return of all four envelopes to $\mathcal{P}$

Case 2.3.2: If $c \neq z$, $\mathcal{I}$ simulates $\mathcal{R}$ halting (depending on the random coins of $\mathcal{R}$, this is also a possible result in the real world).

3. $\mathcal{I}$ continues the simulation until $\mathcal{A}$ halts.

Note that throughout the simulation, all simulated parties behave in a manner that is feasible in the real world as well. Thus, the only possible difference between the views of $\mathcal{Z}$ in the ideal and real worlds is the behavior of the simulated $\mathcal{R}$, which depends only on the contents of the four envelopes sent by $\mathcal{P}$ and the output of the ideal functionality (which in turn depends only on b). It is easy (albeit tedious) to go over all 32 combinations of envelopes and input, and verify that the distribution of $\mathcal{R}$'s output in both cases (the real and ideal worlds) are identical. We enumerate the basic cases below. All other cases are identical to one of the following by symmetry:

Case 1: $\mathcal{A}$ sends two valid pairs of envelopes. Assume it sends 

(the other combinations follow by symmetry). $\mathcal{I}$ returns the following distribution ("?" denotes a sealed envelope):

(a) With probability $\frac{3}{4}$ ($c = b$) it selects uniformly from
$\left\{ \begin{array}{|c|c|}\hline \text{b} & \text{?} \\ \hline \text{b} & \text{1-b} \\ \hline \end{array} \;,\; \begin{array}{|c|c|}\hline \text{b} & \text{1-b} \\ \hline \text{b} & \text{?} \\ \hline \end{array} \right\}$

(b) With probability $\frac{1}{4}$ ($c \neq b$) it selects uniformly from
$\left\{ \begin{array}{|c|c|}\hline \text{?} & \text{1-b} \\ \hline \text{b} & \text{1-b} \\ \hline \end{array} \;,\; \begin{array}{|c|c|}\hline \text{b} & \text{1-b} \\ \hline \text{?} & \text{1-b} \\ \hline \end{array} \right\}$

In the real world, the order of envelopes opened by $\mathcal{R}$ would be distributed uniformly from one of the following sets (each with probability $\frac{1}{4}$):

(a) $\left\{ \begin{array}{|c|c|}\hline 1^{st} & \text{?} \\ \hline 3^{rd} & 2^{nd} \\ \hline \end{array} \right\}$

(b) $\left\{ \begin{array}{|c|c|}\hline 1^{st} & \text{?} \\ \hline 2^{nd} & 3^{rd} \\ \hline \end{array} \;,\; \begin{array}{|c|c|}\hline 1^{st} & 3^{rd} \\ \hline 2^{nd} & \text{?} \\ \hline \end{array} \right\}$

(c) $\left\{ \begin{array}{|c|c|}\hline 3^{rd} & 1^{st} \\ \hline 2^{nd} & \text{?} \\ \hline \end{array} \right\}$

(d) $\left\{ \begin{array}{|c|c|}\hline 3^{rd} & 1^{st} \\ \hline \text{?} & 2^{nd} \\ \hline \end{array} \;,\; \begin{array}{|c|c|}\hline \text{?} & 1^{st} \\ \hline 3^{rd} & 2^{nd} \\ \hline \end{array} \right\}$

Note that the observed result is distributed identically in both cases.

Case 2: $\mathcal{A}$ sends two invalid pairs of envelopes: in this case, in both the real and ideal worlds the adversary will see the responder halting with probability 1.

Case 3: $\mathcal{A}$ sends one valid and one invalid pair of envelopes:

Case 3.1: $\mathcal{A}$ sends $\begin{array}{|c|c|}\hline \text{b} & \text{b} \\ \hline \text{b} & \text{1-b} \\ \hline \end{array}$ (the cases in which the rows or the columns are transposed are symmetric).

The distribution of the returned envelopes in the ideal world is:

i. With probability $\frac{3}{4}$ ($c = b$) it selects uniformly from
$\left\{ \begin{array}{|c|c|}\hline \text{b} & \text{?} \\ \hline \text{b} & \text{1-b} \\ \hline \end{array} \;,\; \begin{array}{|c|c|}\hline \text{?} & \text{1-b} \\ \hline \text{b} & \text{1-b} \\ \hline \end{array} \right\}$

ii. With probability $\frac{1}{4}$ ($c \neq b$) it halts.

In the real world, the order of envelopes opened by $\mathcal{R}$ would be distributed uniformly from one of the following sets (choosing each set with probability $\frac{1}{4}$, and an element within the set uniformly at random); the elements marked with † lead to $\mathcal{R}$ halting:

i. $\left\{ \begin{array}{|c|c|}\hline 1^{st} & \text{?} \\ \hline 3^{rd} & 2^{nd} \\ \hline \end{array} \right\}$

ii. $\left\{ \begin{array}{|c|c|}\hline 1^{st} & \text{?} \\ \hline 2^{nd} & 3^{rd} \\ \hline \end{array} \;,\; \begin{array}{|c|c|}\hline 1^{st} & 3^{rd} \\ \hline 2^{nd} & \text{?} \\ \hline \end{array}^{\dagger} \right\}$

iii. $\left\{ \begin{array}{|c|c|}\hline \text{?} & 1^{st} \\ \hline 2^{nd} & 3^{rd} \\ \hline \end{array} \;,\; \begin{array}{|c|c|}\hline 3^{rd} & 1^{st} \\ \hline 2^{nd} & \text{?} \\ \hline \end{array}^{\dagger} \right\}$

iv. $\left\{ \begin{array}{|c|c|}\hline \text{?} & 1^{st} \\ \hline 3^{rd} & 2^{nd} \\ \hline \end{array} \right\}$

Note that in both worlds $\mathcal{R}$ halts with probability $\frac{1}{4}$, and otherwise the returned envelopes are identically distributed.

Case 3.2: $\mathcal{A}$ sends $\begin{array}{|c|c|}\hline \text{1-b} & \text{1-b} \\ \hline \text{b} & \text{1-b} \\ \hline \end{array}$ (the cases in which the rows or the columns are transposed are symmetric).

The distribution of the returned envelopes in the ideal world is:

i. With probability $\frac{1}{4}$ ($c \neq b$) it selects uniformly from
$\left\{ \begin{array}{|c|c|}\hline \text{1-b} & ? \\ \hline \text{b} & \text{1-b} \\ \hline \end{array} , \begin{array}{|c|c|}\hline ? & \text{1-b} \\ \hline \text{b} & \text{1-b} \\ \hline \end{array} \right\}$

ii. With probability $\frac{3}{4}$ ($c = b$) it halts.

In the real world, the order of envelopes opened by $\mathcal{R}$ would be distributed uniformly from one of the following sets (choosing each set with probability $\frac{1}{4}$, and an element within the set uniformly at random); the elements marked with † lead to $\mathcal{R}$ halting:

i. $\left\{ \begin{array}{|c|c|}\hline 1^{st} & ? \\ \hline 3^{rd} & 2^{nd} \\ \hline \end{array} , \begin{array}{|c|c|}\hline 1^{st} & 3^{rd} \\ \hline ? & 2^{nd} \\ \hline \end{array}^{\dagger} \right\}$

ii. $\left\{ \begin{array}{|c|c|}\hline 1^{st} & 3^{rd} \\ \hline 2^{nd} & ? \\ \hline \end{array}^{\dagger} \right\}$

iii. $\left\{ \begin{array}{|c|c|}\hline ? & 1^{st} \\ \hline 3^{rd} & 2^{nd} \\ \hline \end{array} , \begin{array}{|c|c|}\hline 3^{rd} & 1^{st} \\ \hline ? & 2^{nd} \\ \hline \end{array}^{\dagger} \right\}$

iv. $\left\{ \begin{array}{|c|c|}\hline 3^{rd} & 1^{st} \\ \hline 2^{nd} & ? \\ \hline \end{array}^{\dagger} \right\}$

Note that in both worlds $\mathcal{R}$ halts with probability $\frac{3}{4}$; if it does not halt the returned envelopes are identically distributed.

$\mathcal{A}$ corrupts $\mathcal{R}$

The ideal-world simulator $\mathcal{I}$, proceeds as follows:

1. $\mathcal{I}$ waits to receive v and r from the ideal functionality (in response to the **Query** command sent by the ideal $\mathcal{P}$).
2. $\mathcal{I}$ simulates $\mathcal{R}$ receiving four envelopes. The remainder of the simulation depends on the values of v and r:

Case 2.1: If $v = 1$, $\mathcal{I}$ chooses a uniformly random bit t. The first envelope $\mathcal{R}$ opens in the first pair will have the value t, and the first envelope opened in the second pair will have the value $1 - t$. The values revealed in the remaining envelopes will always result in a valid pair.

Case 2.2: If $v = 0$, The first envelope $\mathcal{R}$ opens in each pair will have the value r, and the remaining envelopes the value $1 - r$.

3. $\mathcal{I}$ continues the simulation until $\mathcal{R}$ sends all four envelopes back to $\mathcal{P}$. If $\mathcal{R}$ opened exactly three envelopes, $\mathcal{I}$ sends **Vote** b to the ideal functionality, where b is calculated as by the pollster in the protocol description. If $\mathcal{R}$ did not open exactly three envelopes, $\mathcal{I}$ sends the **Halt** command to the ideal functionality.

Note that throughout the simulation, all simulated parties behave in a manner that is feasible in the real world as well. Furthermore, the outputs of the ideal and simulated $\mathcal{P}$ are always identical. Thus, the only possible difference between the views of $\mathcal{Z}$ in the ideal and real worlds is the contents of the envelopes opened by $\mathcal{R}$. In the real world, the envelope contents are random. In the ideal world, v and r are i.i.d.

uniform bits. Therefore the order in which the envelopes are opened does not matter; any envelope in the first pair is independent of any envelope in the second. Hence, the distributions in the ideal and real worlds are identical.

3.5 A Responder-Immune $\frac{2}{3}$-CRRT Protocol

3.5.1 Formal Specification

The formal specification is given as Protocol 3.2 (see Section 3.3.2 for an informal description). In the description of the protocol, we use the shorthand "generate an envelope" to mean sending a corresponding **Seal** command to $\mathcal{F}^{(DE)}$, and "send an envelope" to mean sending a corresponding **Send** command to $\mathcal{F}^{(DE)}$.

Protocol 3.2a Responder-Immune $\frac{2}{3}$-CRRT (Pollster)

Wait to receive envelopes E_1, E_2, E_3 from responder.
Choose a random index $j \in_R \{1, 2, 3\}$
Open envelope E_j. Denote the contents e_j.
Send j to responder.
Wait to receive an index k from responder. Denote ℓ the index of the third envelope ($\ell \notin \{j, k\}$).
Open envelope E_k. Denote the contents e_k.
if $e_j \neq e_k$ **then**
    Return the envelope E_ℓ to responder. {the third, unopened, envelope}
    Output e_j.
else {responder tried to cheat}
    Open envelope E_ℓ. Denote its contents e_ℓ.
    if $e_\ell = e_j$ **then** {all three envelopes contain the same value}
        Choose a bit $x' \in B$ such that $\Pr[x' = e_j] = \frac{2}{3}$.
        Output x'.
    else
        Output e_j.
    end if
end if

Protocol 3.2b Responder-Immune $\frac{2}{3}$-CRRT (Responder)

Input: a bit x
Choose a random index $i \in_R \{1, 2, 3\}$
Create three envelopes E_1, E_2, E_3. Envelope E_i will contain $1 - x$. For $i' \in \{1, 2, 3\} \setminus \{i\}$ envelope $E_{i'}$ will contain x.
Send envelopes E_1, E_2, E_3 to pollster.
Wait to receive an index j from pollster.
if $i = j$ **then**
    Choose a random $k \in_R \{1, 2, 3\} \setminus \{i\}$
else $\{i \neq j\}$
    Let $k \leftarrow i$
end if
Send k to pollster. Denote ℓ the index of the third envelope ($\ell \notin \{j, k\}$).
Wait to receive E_ℓ from pollster
if pollster did not send E_ℓ or E_ℓ was opened **then**
    Output $\boxtimes$. {pollster tried to cheat}
end if

3.5.2 Proof of Security

Denote the pollster $\mathcal{P}$ and the responder $\mathcal{R}$. The proof here also follows the outline given in Section 3.2.4. As in the previous section, we'll deal separately with the case when $\mathcal{A}$ corrupts $\mathcal{P}$ and when it corrupts $\mathcal{R}$.

$\mathcal{A}$ corrupts $\mathcal{P}$

When the pollster is corrupted, $\mathcal{I}$ must simulate the responder and the ideal functionality $\mathcal{F}^{(DE)}$. Since $\mathcal{I}$ does not know the honest responder's input, it cannot simulate the honest responder simply by following the protocol. However, it does receive c from the ideal CRRT functionality (in response to the **Vote** b command sent by the ideal responder).

Note that the only information the pollster receives about the responder's input is in the contents of the envelopes it opens and the index k sent by $\mathcal{R}$ in step 10 of the protocol. Since $\mathcal{I}$ simulates $\mathcal{F}^{(DE)}$, it can "retroactively" set the contents of each envelope at the moment it is opened by $\mathcal{P}$. The idea behind the simulation is that when $\mathcal{P}$ acts honestly, c contains sufficient information for $\mathcal{I}$ to simulate $\mathcal{R}$ exactly. When $\mathcal{P}$ is dishonest, $\mathcal{I}$ can use the **Test** and **Reveal** commands (depending on what $\mathcal{P}$ does) to gain additional information; this will cause the ideal CRRT functionality to halt with some probability, but an honest responder would also halt in the real world with the same probability.

More formally, $\mathcal{I}$ begins the simulation by "sending" $\mathcal{P}$ three envelopes (simulating the corresponding messages from $\mathcal{F}^{(DE)}$ to $\mathcal{P}$). Internally, the contents of all three envelopes are "uncommitted". $\mathcal{I}$ "commits" to the contents of an envelope if $\mathcal{P}$ requests to open an envelope whose value is still uncommitted, or when $\mathcal{P}$ sends an index j (corresponding to step 4 in the protocol). When a committed envelope is opened, $\mathcal{I}$ will always reveal the committed value (note that once an envelope is committed, $\mathcal{I}$ will never change the committed value). It remains to describe how $\mathcal{I}$ determines the contents to which the envelopes will be committed and the response to the index j:

Case 1: $\mathcal{P}$ opens an envelope with index e_1, and all three envelopes are still uncommitted. In this case $\mathcal{I}$ commits the value of envelope e_1 to c.

Case 2: $\mathcal{P}$ opens an uncommitted envelope e_2, and exactly one envelope (e_1) is already committed to the value c. Denote e_3 the index of the third envelope. In this case $\mathcal{I}$ sends **Test** c to the ideal CRRT functionality and waits for the response:

- Case 2.1: The response to the **Test** command is $\perp$. In this case, $\mathcal{I}$ commits the e_2 to contain $1-c$ (envelope e_3 remains uncommitted).
- Case 2.2: The response to the **Test** command is c. In this case, $\mathcal{I}$ commits e_2 to contain c and the envelope e_3 to contain $1-c$ (at this point, all three envelopes are committed).

Case 3: $\mathcal{P}$ opens an uncommitted envelope e_3 and both remaining envelopes (e_1 and e_2) are committed. In this case, $\mathcal{I}$ sends the **Reveal** command to the ideal CRRT functionality and commits e_3 to contain the responder's input.

Case 4: $\mathcal{P}$ sends $j = e_1$ to $\mathcal{R}$ and the other two envelopes are still uncommitted. In this case, $\mathcal{I}$ chooses a uniformly random value $k \in_R \{1,2,3\} \setminus \{j\}$. Let $e_2 = k$ and e_3 be the index of the remaining envelope. $\mathcal{I}$ commits envelope e_2 to $1-c$. If e_1 is not already committed, $\mathcal{I}$ commits the value of envelope e_1 to c. Envelope e_3 still remains uncommitted. $\mathcal{I}$ simulates $\mathcal{R}$ returning the index k.

Case 5: $\mathcal{P}$ sends $j = e_2$ to $\mathcal{R}$, j is uncommitted and exactly one of the other envelopes is committed (denote its index e_1). Let e_3 be the index of the remaining (uncommitted) envelope. Note that the contents of e_1 must be c, since $\mathcal{P}$ must have previously opened an uncommitted envelope when all three envelopes were uncommitted, hence $\mathcal{I}$ would have acted according to case 1 in the simulation. $\mathcal{I}$ sends **Test** c to the ideal CRRT functionality and waits for the response:

- Case 5.1: The response to the **Test** command is $\perp$. In this case, $\mathcal{I}$ commits e_2 to $1-c$. $\mathcal{I}$ now sends a **Test** c command to the ideal CRRT functionality:
  - Case 5.1.1: The response to the second **Test** command is $\perp$. In this case, $\mathcal{I}$ sets $k \leftarrow e_1$. Note that the third envelope is still uncommitted.

Case 5.1.2: The response to the second **Test** command is c. In this case, $\mathcal{I}$ sets $k \leftarrow e_3$ to be the index of the third envelope, and commits the envelope e_3 to c (all three envelopes are now committed).

Case 5.2: The response to the **Test** command is c. In this case, $\mathcal{I}$ commits e_2 to c, sets $k \leftarrow e_3$ and commits the envelope e_3 to $1 - c$ (at this point, all three envelopes are committed).

$\mathcal{I}$ simulates $\mathcal{R}$ responding with the index k.

Case 6: $\mathcal{P}$ sends $j = e_1$ to $\mathcal{R}$, e_1 is committed to a value c and exactly one of the other envelopes is committed (denote its index e_2, and that of the remaining envelope e_3). Note that the value of e_2 must be $1 - c$ (otherwise all three envelopes would already be committed). In this case, $\mathcal{I}$ sends **Test** $1 - c$ to the ideal CRRT functionality and waits for the response:

Case 6.1: The response to the **Test** command is $\perp$. In this case, $\mathcal{I}$ sets $k \leftarrow e_2$. Note that the third envelope is still uncommitted.

Case 6.2: The response to the **Test** command is $1 - c$. In this case, $\mathcal{I}$ sets $k \leftarrow e_3$ and commits envelope e_3 to $1 - c$ (all three envelopes are now committed).

$\mathcal{I}$ simulates $\mathcal{R}$ responding with the index k.

Case 7: $\mathcal{P}$ sends $j = e_3$ to $\mathcal{R}$, e_3 is uncommitted and both of the other envelopes are already committed. Note that they must be committed to the values c and $1 - c$ (otherwise all three envelopes would already be committed). $\mathcal{I}$ sends the **Reveal** command to the ideal CRRT functionality and commits e_3 to be the responder's input, x. $\mathcal{I}$ sets k to be the index of the previously committed envelope whose value was $1 - x$ and simulates $\mathcal{R}$ responding with the index k.

We can assume w.l.o.g. that the environment's view of the protocol consists of its random coins, the messages received from $\mathcal{F}^{(DE)}$ and $\mathcal{R}$ and any output by $\mathcal{R}$ at the end of the protocol ($\mathcal{R}$ outputs nothing if the protocol completes successfully, and $\boxtimes$ if it was aborted prematurely). This is because the messages sent by $\mathcal{P}$ and the responder's input are all deterministic functions of this view.

We claim that for any value of the environment's random coins, its views in the real and ideal worlds are identically distributed (where the distribution is over the random coins of the real-world responder and the ideal-world simulator, respectively).

Fix some value of the random coins for the environment. Denote x the value of the responder's input (determined by the environment's random coins). Note that the decision to open an envelope or to send an index j to the responder is a deterministic function of the environment's coins, the contents of previously opened envelopes and the response to a previously sent index (if there was one). Thus, to show the views are identical, it is enough to show that the contents of the opened envelopes and the response to j are identically distributed.

Note that the only actions $\mathcal{P}$ can perform that may have an effect on the these values are opening a previously unopened envelope and sending the index j to $\mathcal{R}$. We'll call these *relevant* actions. Moreover, each relevant action can be performed only once (if we consider opening each of the three envelopes a separate action). Hence, we can exhaustively consider all possible permutations of the adversary's relevant actions and their subsets. Since, in the ideal-world simulation, opening a committed envelope will always show the previously committed value, while an uncommitted envelope's value is never seen (it will be committed at the moment it is opened), we can ignore the cases in which the pollster opens previously committed envelopes as long as we show that the distribution of committed values is identical to that in the real world (where all three envelopes are committed before sending them to the pollster). Below, we give an exhaustive case analysis. To help clarify the presentation, we tag each case with a pictoral representation: $\boxed{\curvearrowleft}$ signifies an uncommitted envelope being opened, $\boxed{?}$ signifies an envelope that has not yet been committed, $\boxed{c}$ signifies an envelope committed to the bit c and $\boxed{*}$ an envelope committed to any bit. The notation $\overset{j\rightarrow}{\boxed{*}}$ signifies that this is the envelope whose index j is sent to the responder, while $\overset{j}{\boxed{*}}$ and $\overset{k}{\boxed{*}}$ signify that this was the envelope whose index was previously sent as j or received as k (respectively).

Case 1: $\boxed{\curvearrowleft}\ \boxed{?}\ \boxed{?}$ $\mathcal{P}$ opens an envelope e_1, all three envelopes are still unopened and $\mathcal{P}$ has not yet sent an index to $\mathcal{R}$, In the real-world, e_1 will contain x with probability $\frac{2}{3}$ and $1-x$ with probability $\frac{1}{3}$.

In the ideal world, this corresponds to case 1 in the simulation and e_1 will contain the value c sent by the CRRT functionality (which is x with probability $\frac{2}{3}$ and $1-x$ with probability $\frac{1}{3}$). No other envelopes will be committed by $\mathcal{I}$.

Case 2: $\boxed{\text{c}}\ \boxed{\curvearrowleft}\ \boxed{?}$ $\mathcal{P}$ opens an envelope e_2, envelope e_1 has already been opened (a value c was revealed), envelope e_3 has not been openend and $\mathcal{P}$ has not yet sent an index to $\mathcal{R}$. In the ideal world, this situation matches case 2 in the simulation, hence the simulator would send **Test** c to the CRRT functionality.

Case 2.1: $c = x$. In this case, in the real-world envelopes e_2 and e_3 contain c and $1-c$ in a random order, hence $\mathcal{P}$ will see c with probability $\frac{1}{2}$ on opening e_2. If it sees c, then no matter what further actions are taken by $\mathcal{P}$, $\mathcal{R}$ will output $\boxtimes$ (since at least one of the envelopes containing x must be returned unopened to $\mathcal{R}$ in order for the protocol to terminate successfully).

In the ideal world, the response to the **Test** c command would be c with probability $\frac{1}{2}$. Therefore, e_2 will contain c with probability $\frac{1}{2}$. If it does then e_3 will also be committed $1-c$ (as would be the case in the real world) and $\mathcal{R}$ will output $\boxtimes$. If it does not e_3 will remain uncommitted at this point and the output of $\mathcal{R}$ will be determined by the subsequent actions of $\mathcal{P}$ (as it would be in the real world as well).

Case 2.2: $c = 1-x$. In this case, the real-world envelopes e_2 and e_3 both contain $x = 1-c$ (hence $1-c$ will be revealed with probability 1 when opening e_2. In the ideal world, the response to **Test** c will be $\perp$ with probability 1, hence e_2 will also contain $1-c$. e_3 will remain uncommitted at this point.

Case 3: $\boxed{\text{c}}\ \boxed{\text{1-c}}\ \boxed{\curvearrowleft}$ or $\overset{k}{\boxed{\text{c}}}\ \overset{j}{\boxed{*}}\ \boxed{\curvearrowleft}$ or $\overset{j}{\boxed{*}}\ \overset{k}{\boxed{*}}\ \boxed{\curvearrowleft}$

$\mathcal{P}$ opens an envelope e_3; the other two envelopes (e_1 and e_2) satisfy at least one of the following conditions:

- Both e_1 and e_2 have both been opened and contain c and $1-c$, respectively.
- e_1 was opened and contains the value c, then the index $j = e_2$ was sent to $\mathcal{R}$ and the response was $k = e_1$.
- The index $j = e_1$ was sent to $\mathcal{R}$ and the response was $k = e_2$.

In the real-world, envelope e_3 would always contain x. Note that no matter what actions are later taken by $\mathcal{P}$, $\mathcal{R}$ will always output $\boxtimes$ in the real-world since at least one envelope must be returned unopened for the protocol to terminate successfully.

In the ideal world, this corresponds to case 3 in the simulation; $\mathcal{I}$ would send the **Reveal** command to the ideal CRRT functionality and set the value of e_3 to be x (as in the real-world). $\mathcal{R}$ would output $\boxtimes$.

Case 4: $\overset{j\rightarrow}{\boxed{*}}\ \boxed{?}\ \boxed{?}$ $\mathcal{P}$ sends an index j to $\mathcal{R}$ and either $e_1 = j$ is the only opened envelope or none of the envelopes has been opened. In the real world, the response k will be chosen uniformly from $\{1,2,3\} \setminus j$. In the ideal world, this corresponds to case 4 in the simulation, hence the k will be chosen with the same distribution.

In the real world, envelope e_1 contains x with probability $\frac{2}{3}$ and $1-x$ with probability $\frac{1}{3}$, while envelope k will always contain the complement of the bit in e_1. In the ideal world, at this point e_1 is committed to c and e_2 to $1-c$, so the contents of both envelopes are identically distributed in the real and ideal worlds.

Case 5: $\boxed{\text{c}}\ \overset{j\rightarrow}{\boxed{?}}\ \boxed{?}$ $\mathcal{P}$ sends an index j to $\mathcal{R}$ and exactly one envelope ($e_1 \neq j$) has been opened and contains the value c. Denote $e_2 = j$ and e_3 the index of the remaining envelope. In the ideal world, this corresponds to case 5, and $\mathcal{I}$ would send a **Test** c command to the ideal CRRT functionality.

Case 5.1: $c = x$. In the real world, this means the unopened envelopes contain c and $1 - c$ in a random order. With probability $\frac{1}{2}$, j contains c and e_3 contains $1 - c$, hence the responder would always respond with $k = e_3$. With probability $\frac{1}{2}$ envelope j contains $1 - c$, in which case with probability $\frac{1}{2}$ the responder would respond with $k = e_3$ and with probability half $k = e_1$. In summary, $k = e_3$ with probability $\frac{3}{4}$ and $k = e_1$ with probability $\frac{1}{4}$. Note that if $k = e_3$, the responder will output $\boxtimes$ no matter what actions the pollster subsequently takes, since it would expect envelope e_1 to be returned unopened.

In the ideal world, the **Test** c command will return c with probability $\frac{1}{2}$, in which case $\mathcal{I}$ will set $k = e_3$. With probability $\frac{1}{2}$ the **Test** c command will return $\perp$, in which case $\mathcal{I}$ sends a second **Test** c command. With probability $\frac{1}{2}$ the second command will return c, in which case $k = e_3$ and with probability $\frac{1}{2}$ it will return $\perp$, in which case $k = e_1$. The distribution for k is identical to that in the real-world, and the cases in which $\mathcal{R}$ outputs $\boxtimes$ will also cause the responder in the real world to eventually output $\boxtimes$.

Note that in the both the real and ideal worlds, conditioned on $k = e_3$ the contents of envelopes e_2 and e_3 are always complements and e_2 contains c with probability $\frac{1}{3}$. Conditioned on $k = e_1$, e_2 always contains $1 - c$ in both the real and ideal worlds ($\mathcal{I}$ does not yet commit to e_3 in this case).

Case 5.2: $c = 1 - x$. In the real world, this means both unopened envelopes contain $1 - c$. Hence, the response to j will always be $k = e_1$. In the ideal world, the response to both **Test** c commands will always be $\perp$, hence $\mathcal{I}$ will always set $k = e_1$ and commit e_2 to $1 - c$.

Case 6: $\overset{j\rightarrow}{\boxed{\text{c}}}\boxed{\text{1-c}}\boxed{\text{?}}$ $\mathcal{P}$ sends $j = e_1$ to $\mathcal{R}$, envelope j was opened to a value c, envelope e_2 was opened to a value $1 - c$ and envelope e_3 is still unopened. In the ideal world, this corresponds to case 6 in the simulation, hence $\mathcal{I}$ will send a **Test** $1 - c$ to the ideal CRRT functionality.

Case 6.1: $c = x$. In this case, e_3 contains c in the real world, and the responder will always answer with $k = e_2$. In the ideal world, the response to the **Test** $1 - c$ command will always be $\perp$, hence the responder will always answer with $k = e_2$ (leaving e_3 uncommitted).

Case 6.2: $c = 1 - x$. In this case, e_3 contains $1 - c$ in the real world and the responder will answer $k = e_2$ with probability $\frac{1}{2}$ and $k = e_3$ with probability $\frac{1}{2}$. Note that if $k = e_3$, then the responder will always eventually output $\boxtimes$, since it will expect e_2 to be unopened.

In the ideal world, the response to the **Test** $1 - c$ command is $\perp$ with probability $\frac{1}{2}$ (in which case $k = e_2$, and $1 - c$ with probability $\frac{1}{2}$, in which case $k = e_3$, $\mathcal{I}$ commits e_3 to $1 - c$ and and $\mathcal{R}$ outputs $\boxtimes$.

Case 7: $\boxed{\text{c}}\boxed{\text{1-c}}\overset{j\rightarrow}{\boxed{\text{?}}}$ $\mathcal{P}$ sends $j = e_3$ to $\mathcal{R}$, j has not been opened, envelope e_1 was opened and contained c and envelope e_2 was opened and contained $1 - c$. In the real world, envelope e_3 will always contain x, and the responder will always eventually ouput $\boxtimes$ since it will expect either e_1 or e_2 to remain unopened.

In the ideal world, this corresponds to case 6 in the simulation. $\mathcal{I}$ will send the **Reveal** command to the ideal CRRT functionality and simulate the responder in the real world with the revealed envelope contents. Thus, the resulting view will be identically distributed to that in the real world.

$\mathcal{A}$ corrupts $\mathcal{R}$

1. $\mathcal{I}$ runs the simulation until $\mathcal{R}$ sends three envelopes. It then sends the **Vote** b command to the ideal functionality, where b is the majority of the envelope contents.
2. $\mathcal{I}$ waits to receive c, the poll response, from the ideal functionality. It then chooses a random index j from the envelopes whose contents are c (if none of the envelopes contain c, $\mathcal{I}$ chooses j at random).
3. $\mathcal{P}$ sends j to $\mathcal{R}$, and $\mathcal{I}$ continues the simulation (simulating an honest $\mathcal{P}$ exactly following the protocol) until $\mathcal{A}$ terminates.

This simulation is completely identical to the real-life case, except in the way $\mathcal{P}$ chooses the index j. Whereas in real life j is chosen at random, in the ideal-world simulation j is chosen according to the poll result. However, a simple calculation shows that j is distributed uniformly in the ideal world as well.

3.6 Strong CRRT Protocols

Ideally, we would like to have CRRT protocols that cannot be biased at all by malicious responders, while perfectly preserving the responder's deniability, even against malicious pollsters. Unfortunately, the protocols described in Section 3.3 do not quite achieve this. At the expense of increasing the number of rounds, we can get arbitrarily close to the Strong-CRRT functionality defined in Section 3.2.1.

Consider protocol 3.3, in which the pollster and responder use a pollster-immune p-CRRT protocol (whose ideal functionality is denoted $\mathcal{F}^{(pCRRT)}$) r times, one after the other (with the responder using the same input each time). The pollster outputs the majority of the subprotocols' outputs. If the responder halts at any stage, the pollster uses uniformly random bits in place of the remaining outputs (this protocol is a direct adaptation of a classic $\frac{1}{\sqrt{r}}$-strongly-fair coin flipping protocol [24]).

Protocol 3.3a Almost-Strong CRRT (Pollster)

Let $flag \leftarrow$ true
for $1 \leq i \leq r$ **do**
  if $flag$ **then** {Responder hasn't halted so far}
    Send a **Query** command to $\mathcal{F}_i^{(pCRRT)}$ (instance i of pollster-immune p-CRRT functionality).
    Wait to receive output b_i from $\mathcal{F}_i^{(pCRRT)}$.
    if $b_i = \perp$ **then** {Responder halted}
      Let $flag \leftarrow$ false
      Let $b_i \overleftarrow{R} \{0,1\}$
    end if
  else {Responder halted in a previous round}
    Let $b_i \overleftarrow{R} \{0,1\}$
  end if
end for
Output maj $\{b_1, \ldots, b_r\}$

Protocol 3.3b Almost-Strong CRRT (Responder)

Input: a bit x
for $1 \leq i \leq r$ **do**
  Send a **Vote** x command to $\mathcal{F}_i^{(pCRRT)}$ (instance i of pollster-immune p-CRRT functionality).
end for

Corrupt Responder. Protocol 3.3 gives a corrupt responder at most $O(\frac{1}{\sqrt{r}})$ advantage over an honest responder. More formally, let $q^{(\mathcal{A})}$ be the probability that the pollster, running protocol 3.3a, outputs 1 when interacting with a responder $\mathcal{A}$ (running an arbitrary protocol). Denote $q \doteq q^{(\mathcal{R}(1))}$ be the probability the pollster outputs 1 when both the pollster and responder are honest (note that since the entire protocol is symmetric with respect to the responder's input, we can consider w.l.o.g. an adversary that attempts to bias the result towards 1). Then:

Lemma 3.1. *For any adversary $\mathcal{A}^*$ corrupting the responder, the advantage of the adversary is $q^{(\mathcal{A}^*)} - q = O(\frac{1}{\sqrt{r}})$.*

Proof. Intuitively, this is because the only advantage an adversary can gain over the honest user is halting at some round i. However, halting affects the result only if the other $r-1$ rounds were balanced (this is the only case in which the outcome of the i^{th} round affects the majority). This occurs with probability $O(\frac{1}{\sqrt{r}})$.

More formally, we will bound the advantage of a slightly more powerful type of adversary: one that runs Protocol 3.3b honestly (with input 1) until step 3 (the end of the loop), then can choose an arbitrary i and set $b_i \leftarrow 1$. This type of adversary is at least as powerful as a "real" adversary interacting with the honest pollster since, given a "real" adversary $\mathcal{A}$, we can achieve an advantage at least as large by running the protocol with the new adversary $\mathcal{A}'$, setting i to be the round at which $\mathcal{A}$ halted. Let $b_j^{(X)}$ be the value of bit b_j when the pollster interacts with an adversary X. Then for all $1 \leq j < i$, $\Pr[b_j^{(\mathcal{A})} = 1] = p = \Pr[b_j^{(\mathcal{A}')} = 1]$, for $i < j \leq r$, $\Pr[b_j^{(\mathcal{A})} = 1] = \frac{1}{2} < p = \Pr[b_j^{(\mathcal{A}')} = 1]$ while $\Pr[b_i^{(\mathcal{A})} = 1] \leq 1 = \Pr[b_i^{(\mathcal{A}')} = 1]$. Hence, $\Pr[\text{maj}\left\{b_1^{(\mathcal{A})} \ldots, b_r^{(\mathcal{A})}\right\} = 1] \leq \Pr[\text{maj}\left\{b_1^{(\mathcal{A}')} \ldots, b_r^{(\mathcal{A}')}\right\} = 1]$.

Note that changing a single bit from 0 to 1 can only change the result the remaining bits have an equal number of zeroes and ones. Since $b_1, \ldots, b_r$ are i.i.d and $\Pr[b_j = 1] = p$ for all j,

$$\Pr\left[\sum_{j \neq i} b_j = \frac{r-1}{2}\right] = \binom{r-1}{\frac{1}{2}(r-1)} p^{\frac{1}{2}(r-1)}(1-p)^{\frac{1}{2}(r-1)} \tag{3.6.1}$$

$$\leq O\left(\frac{1}{\sqrt{r-1}}\right) 2^{(r-1)H(\frac{1}{2})} p^{\frac{1}{2}(r-1)}(1-p)^{\frac{1}{2}(r-1)} \tag{3.6.2}$$

$$\leq O\left(\frac{1}{\sqrt{r}}\right) \tag{3.6.3}$$

where $H(\alpha) \doteq -\alpha \log \alpha - (1-\alpha)\log(1-\alpha)$ is the entropy function, (3.6.2) uses the approximation $\binom{n}{\alpha n} = (1 \pm O(1/n))C\frac{1}{\sqrt{n}}2^{nH(\alpha)}$ (C is a constant depending on α) and (3.6.3) derives from the fact that $H(\frac{1}{2}) = 1$ and $p^x(1-p)^x$ has a maximum at $p = \frac{1}{2}$ for all $x > 0$.

□

Corrupt Pollster. Since Protcol 3.3 uses a pollster-immune CRRT protocol as a subprotocol, and the pollster does not interact in any other way with the responder, a corrupt pollster cannot actively gain additional information about the responder's input.

We must still show that the pollster does not *passively* gain too much information about the responder's input. In the ideal world, the pollster only sees a single bit of information, corresponding to whether the majority of the p-CRRT subprotocols returned 1. In the real world, the pollster receives additional information: the *number* of p-CRRT subprotocols that returned 1.

However, it turns out that this information does not help the pollster to distinguish between the case where the responder's input is 0 and the case where it is 1. To see this, consider the distributions $\mathcal{X}_0$ and $\mathcal{X}_1$, representing the distributions of the ideal pollster's view when the responder's input is 0 and 1, respectively. The probability that the pollster can guess b, given a sample $X \sim \mathcal{X}_b$, is a measure of the information the pollster is "allowed" to have about b. We denote the corresponding distributions for the real world $\mathcal{Y}_0$ and $\mathcal{Y}_1$, where $\mathcal{Y}_b = B(r, bp + (1-b)(1-p))$ is the binomial distribution.

The largest possible difference between the probabilities that two probability distributions can assign to an event (hence, the maximum probability of correctly determining the source of sample taken from one of them) is the *total variation distance* between the two distributions (denoted $\delta(\mathcal{D}_0, \mathcal{D}_1)$

We show the total variation distance between $\mathcal{X}_0$ and $\mathcal{X}_1$ (the ideal case) is identical to the distance

between $\mathcal{Y}_0$ and $\mathcal{Y}_1$ (the real case). For $b \in \{0,1\}$, let $X_b \sim \mathcal{X}_b$ and $Y_b \sim \mathcal{Y}_b$. Then:

$$\begin{aligned}
\delta(\mathcal{X}_0, \mathcal{X}_1) &= \frac{1}{2}\left(|\Pr[X_0 = 0] - \Pr[X_1 = 0]| + |\Pr[X_1 = 1 - \Pr[X_0 = 1]]|\right) \\
&= \frac{1}{2}\left(\sum_{i=0}^{\lfloor r/2 \rfloor}\left[\binom{r}{i}|p^i(1-p)^{r-i} - p^{r-i}(1-p)^i|\right] + \sum_{i=\lceil r/2 \rceil}^{r}\left[\binom{r}{i}|p^i(1-p)^{r-i} - p^{r-i}(1-p)^i|\right]\right) \\
&= \frac{1}{2}\left(\sum_{i=0}^{r}\left[\binom{r}{i}|p^i(1-p)^{r-i} - p^{r-i}(1-p)^i|\right]\right) \\
&= \frac{1}{2}\sum_{i=0}^{r}|\Pr[Y_1 = i] - \Pr[Y_0 = i]| \\
&= \delta(\mathcal{Y}_0, \mathcal{Y}_1)
\end{aligned}$$

Limitations. The problem with using Protocol 3.3 is that the probability that an honest responder will get the result she wants tends to 1 as the number of rounds grows. In order to obtain a q-Almost-Strong CRRT, we must use a p-CRRT protocol where $p = \frac{1}{2} + O(\frac{1}{\sqrt{r}})$ (we discuss possibilities for construction such a protocol in Section 3.7.1).

This adds further complexity to the protocol (e.g., the protocol in Section 3.7.1 requires $\Omega(\frac{1}{\epsilon})$ bubbles on the scratch-off card for $p = \frac{1}{2} + \epsilon$). Thus, this multi-round protocol is probably not feasible in practice.

3.6.1 Lower Bounds and Impossibility Results

In this section we attempt to show that constructing practical strong CRRT protocols is a difficult task. We do this by giving impossibility results and lower bounds for implementing subclasses of the strong CRRT functionality. We consider a generalization of the strong p-CRRT functionality defined in Section 3.2.1, which we call (p, q)-CRRT. The (p, q)-CRRT functionality can be described as follows:

Vote b The issuer of this command is the responder. On receiving this command the functionality tosses a weighted coin c, such that $c = 0$ with probability p. It then outputs $b \oplus c$ to the pollster. The functionality supplies the responder with exactly enough additional information so that she can guess c with probability $q \geq p$.

In the definition of strong CRRT given in Section 3.2.1, we specify exactly how much information the pollster learns about the responder's choice, but leave completely undefined what a cheating responder can learn about the pollster's result. The (p, q)-CRRT functionality quantifies this information: in a (p, p)-CRRT, the responder does not gain any additional information (beyond her pre-existing knowledge that the pollster's result will equal her choice with probability p). In a $(p, 1)$-CRRT, the responder learns the pollster's result completely. We show that (p, p)-CRRT implies oblivious transfer (and is thus impossible in the DE model), while $(p, 1)$-CRRT implies strong coin-flipping (and thus we can lower-bound the number of rounds required for the protocol). For values of q close to p or close to 1, the same methods can still be used to show lower bounds.

(p, q)-CRRT when q is close to p:

First, note that when $p = q$ we can view the (p, q)-CRRT functionality as a binary symmetric channel (BSC) with error probability $1 - p$. Crépeau and Kilian have shown that a protocol for Oblivious Transfer (OT) can be constructed based on any BSC [30]. However, it is impossible to implement OT in the Distinguishable Envelope (DE) model [53]. Therefore (p, p)-CRRT cannot be implemented in the DE model. It turns out that this is also true for any q close enough to p. This is because, essentially, the (p, q)-CRRT functionality is a $(1 - q, 1 - p)$-Passive Unfair Noisy Channel (PassiveUNC), as defined by Damgård, Kilian and Salvail [34]. A (γ, δ)-PassiveUNC is a BSC with error δ which provides the corrupt sender (or receiver) with additional information that brings his perceived error down to γ; (i.e., a corrupt sender can guess the bit received by

the receiver with probability $1-\gamma$, while an honest sender can guess this bit only with probability $1-\delta$). For γ and δ that are close enough (the exact relation is rather complex), Damgård, Fehr, Morozov and Salvail [32] show that a (γ, δ)-PassiveUNC is sufficient to construct OT. For the same range of parameters, this implies that realizing (p, q)-CRRT is impossible in the DE model.

(p, q)-CRRT when q is close to 1:

When $q = 1$, both the pollster and the responder learn the poll result together. A $(p, 1)$-CRRT can be used as a protocol for strongly fair coin flipping with bias $p - \frac{1}{2}$. In a strongly fair coin flipping protocol with bias ϵ, the bias of an honest party's output is at most ϵ regardless of the other party's actions — even if the other party aborts prematurely. If q is close to 1, we can still construct a coin flipping protocol, albeit without perfect consistency. The protocol works as before, except that the responder outputs his best guess for the pollster's output: both will output the same bit with probability q.

A result by Cleve [24] shows that even if all the adversary can do is halt prematurely (and must otherwise follow the protocol exactly), any r-round protocol in which honest parties agree on the output with probability $\frac{1}{2} + \epsilon$ can be biased by at least $\frac{\epsilon}{4r+1}$. Cleve's proof works by constructing $4r + 1$ adversaries, each of which corresponds to a particular round. An adversary corresponding to round i follows the protocol until it reaches round i. It then halts immediately, or after one extra round. The adversary's decision is based only on what the ourput of an honest player would be in the same situation, should the *other* party halt after this round. Cleve shows that the average bias achieved by these adversaries is $\frac{\epsilon}{4r+1}$, so at least one of them must achieve this bias. The same proof also works in the DE model, since all that is required is that the adversary be able to compute what it would output should the other player stop after it sends the messages (and envelopes) for the current round. This calculation may require a party to open some envelopes (the problem being that this might prevent the adversary from continuing to the next round). However, an honest player would be able to perform the calculation in the next round, after sending this round's envelopes, so it cannot require the adversary to open any envelopes that may be sent in the next round.

Cleve's lower bound shows that a (p, q)-CRRT protocol must have at least $\frac{q-\frac{1}{2}}{4(p-\frac{1}{2})} - \frac{1}{4}$ rounds. Since a protocol with a large number of rounds is impractical for humans to implement, this puts a lower bound on the bias p (finding a CRRT protocol with a small p is important if we want to be able to repeat the poll while still preserving plausible deniability).

This result also implies that it is impossible to construct a $(p, 1)$-CRRT protocol in which there is a clear separation between the responder's choice and the final output. That is, the following functionality, which we call *p-CRRT with confirmation*, is impossible to implement in the DE model:

Vote b The issuer of this command is the responder. On receiving this command the functionality outputs "Ready?" to the pollster. When the pollster answers with "ok" the functionality tosses a weighted coin c, such that $c = 0$ with probability p. It then outputs $b \oplus c$ to the pollster and responder.

p-CRRT with confirmation is identical to $(p, 1)$-CRRT, except that the output isn't sent until the pollster is ready. The reason it is impossible to implement is that this functionality can be amplified by parallel repetition to give a strongly fair coin flipping protocol with arbitrarily small p. Since the amplification is in parallel, it does not increase the number of rounds required by the protocol, and thus contradicts Cleve's lower bound. Briefly, the amplified protocol works as follows: the responder chooses k inputs randomly, and sends each input to a separate (parallel) instance of p-CRRT with confirmation. The pollster waits until all the inputs have been sent (i.e., it receives the "Ready?" message from all the instances), then sends "ok" to all the instances. The final result will be the xor of the outputs of all the instances. Since the different instances act independently, the bias of the final result is exponentially small in k.

3.7 Discussion and Open Problems

3.7.1 p-CRRT for General p

In this paper we give protols for pollster-immune $\frac{3}{4}$-CRRT and responder-immune $\frac{2}{3}$-CRRT. However, in practice we sometimes require p-CRRT protocols for different values of p.

In particular, if we need to repeat the poll (for example, when we use it as a subprotocol in our almost-strong CCRT protocol (Protocol 3.3), we need the basic protocol to have p closer to $\frac{1}{2}$ (in order to maintain the plausible deniability). Below, we outline some (slightly flawed) ideas for protocols that attempt this. Finding completely secure protocols for general values of p remains an open question.

Generalizing our pollster-immune protocol to any rational p. The following protocol will work for any rational $p = \frac{k}{n}$ (assume $k > \frac{1}{2}n$): As in Protocol 3.1, the pollster generates two rows of bubbles. One row contains k "Yes" bubbles and $n - k$ "No" bubbles in random order (this row is the "Yes" row), and the other contains k "No" bubbles and $n - k$ "Yes" bubbles (this row is the "No" row). The rows are also in a random order. The responder's purpose is to find the row matching her choice. She begins by scratching a single bubble in each row. If both bubbles contain the same value, she "eliminates" a random row (by scratching it out completely). Otherwise, she "eliminates" the row that does not correspond to her choice. The pollster's output is the majority value in the row that was not eliminated. The probability that the pollster's output matches the responder's choice is exactly p.

Unfortunately, this protocol is completely secure only for a semi-honest pollster (one that correctly generates the scratch-off cards). A malicious pollster can cheat in two possible ways: he can replace one of the rows with an invalid row (one that does not contain exactly k "Yes" bubbles or exactly k "No" bubbles), or he can use two valid rows that have the same majority value (rather than opposite majority values). In both cases the pollster will gain additional information about the responder's choice. This means the protocol does not realize the ideal pollster-immune CRRT functionality.

If the pollster chooses to use an invalid row, he will be caught with probability at least $\frac{1}{2}(1 - p)$ (since with this probability the responder will scratch identical bubbles in both rows, and choose to eliminate the invalid row). We can add "cheating detection" to the protocol to increase the probability of detecting this attack. In a protocol with cheating detection, the pollster gives the responder ℓ scratch-off cards rather than just one (each generated according to the basic protocol). The responder chooses one card to use as in the basic protocol. On each of the other cards, she scratches off a single row (chosen randomly), and verifies that it contains either exactly k "Yes" bubbles or exactly k "No" bubbles. She then returns all the cards to the pollster (this step is necessary to prevent the responder from increasing her chances by trying multiple cards until one gives the answer she wants). A pollster that cheats by using an invalid row will be caught with probability $1 - \frac{1}{\ell}$.

A malicious pollster can still cheat undetectably by using two valid rows with identical majorities. This gives only a small advantage, however, and in practice the protocol may still be useful when p is close to $\frac{1}{2}$.

Generalizing our responder-immune protocol to any rational p. When the pollster's actions are performed in view of the responder (in particular, when the responder can see exactly which envelopes are opened by the pollster), Protocol 3.2 has a straightforward generalization to any rational $p = \frac{k}{n}$, where $k > \frac{1}{2}n$: the responder uses n (rather than 3) envelopes, of which k contain her choice and $n - k$ contain its opposite. After the pollster chooses an envelope to open, the responder shows him $n - k$ envelopes that contain the opposite value.

Note that when this generalized protocol is performed by mail, it does *not* realize the ideal functionality defined in Section 3.2.1 (the pollster can cheat by opening additional envelopes before sendng an index to the responder).

Efficient Generalization to Arbitrary p. We have efficient p-CRRT protocols for specific values of p: $p = \frac{3}{4}$ in the pollster-immune case, and $p = \frac{2}{3}$ in the responder-immune case. Our generalized protocols are not very efficient: for $p = \frac{1}{2} + \epsilon$ they require $\Omega(\frac{1}{\epsilon})$ envelopes. In a protocol meant to be implemented by humans, the efficiency of the protocol has great importance. It would be useful to find an efficient general protocol to approximate arbitrary values of p (e.g., logarithmic in the approximation error).

3.7.2 Additional Considerations

Polling Protocols by Mail. The pollster-immune CRRT protocol requires only a single round; This makes it convenient to use in polls through the post (it only requires the poll to be sent to the responder, "filled out" and returned). The responder-immune protocol presents additional problems when used through the post.

First, in this case the protocol realizes a slightly weaker functionality than in the face-to-face implementation. Second, it requires two rounds, and begins with the responder. This means, in effect, that it would require an extra half-round for the pollster to notify the responder about the existence of the poll. It would be interesting to find a one-round protocol for the responder-immune functionality as well. It may be useful, in this context, to differentiate between "information-only" communication (which can be conducted by phone or email), and transfer of physical objects such as envelopes (which require "real" mail).

Side-Channel Attacks. The privacy of our protocols relies on the ability of the responder to secretly perform some actions. For instance, in the pollster-immune protocol we assume that the order in which the bubbles on the card were scratched remains secret. In practice, some implementations may be vulnerable to an attack on this assumption. For example, if the pollster uses a light-sensitive dye on the scratch-off cards that begins to darken when the coating is scratched off, he may be able to tell which of the bubbles was scratched first. Side-channel attacks are attacks on the model, not on the CRRT protocols themselves. As these attacks highlight, when implementing CRRT using a physical implementation of Distinguishable Envelopes, it is important to verify that this implementation actually does realize the required functionality.

Dealing With Human Limitations. Our protocols make two assumptions about the humans implementing them: that they can make random choices and that they can follow instructions. The former assumption can be relaxed: if the randomness "close to uniform" the security and privacy will suffer only slightly (furthermore, simple physical aids, such as coins or dice, make generating randomness much easier). The latter assumption is more critical; small deviations from the protocol can result in complete loss of privacy or security. Constructing protocols that are robust to human error could be very useful.

Practical Strong CRRT Protocols. As we discuss in Section 3.6.1, for a range of parameters p, q-CRRT is impossible, and for a different range of parameters it is impractical. For some very reasonable values, such as $\frac{3}{4}$-Strong CRRT, we can approximate the functionality using a large number of rounds, but do not know how to prove any lower bound on the number of rounds required. Closing this gap is an obvious open question. Alternatively, finding a physical model in which efficient Strong CRRT is possible is also an interesting direction.

APPENDIX

3.A Formal Definition of Distinguishable Envelopes

The definition below is a extracted from [53].

Functionality $\mathcal{F}^{(DE)}$ models a tamper-evident "envelope". As long as the envelope is closed, its contents are completely hidden. Any party can open the envelope and learn its contents. However, the creator of the envelope will be able to tell whether the envelope was previously opened.

Functionality $\mathcal{F}^{(DE)}$ contains an internal table that consists of tuples of the form $(id, value, creator, holder, state)$. The table represents the state and location of the tamper-evident envelopes. It contains one entry for each existing envelopes, indexed by the container's id and creator. We denote $value_{id}$, $creator_{id}$, $holder_{id}$ and $state_{id}$ the corresponding values in the table in row id (assuming the row exists). The table is initially empty. The functionality is described as follows, running with parties $P_1, \ldots, P_n$ and ideal adversary $\mathcal{I}$:

Seal $(id, value)$ This command creates and seals an envelope. On receiving this command from party P_i, the functionality verifies that id has the form $(P_i, \{0,1\}^*)$ (this form of id is a technical detail to ensure that ids are local to each party). If this is the first **Seal** message with id id, the functionality stores the tuple $(id, value, P_i, P_i, \textbf{sealed})$ in the table. If this is not the first **Seal** with id id, it verifies that $creator_{id} = holder_{id} = P_i$ and, if so, replaces the entry in the table with $(id, value, P_i, P_i, \textbf{sealed})$.

Send (id, P_j) On receiving this command from party P_i, the functionality verifies that an entry for container id appears in the table and that $holder_{id} = P_i$. If so, it outputs $(\textbf{Receipt}, id, creator_{id}, P_i, P_j)$ to P_j and $\mathcal{I}$ and replaces the entry in the table with $(id, value_{id}, creator_{id}, P_j, state_{id})$.

Open *id* On receiving this command from P_i, the functionality verifies that an entry for container *id* appears in the table and that $holder_{id} = P_i$. It sends $(\textbf{Opened}, id, value_{id}, creator_{id})$ to P_i. It also replaces the entry in the table with $(id, value_{id}, creator_{id}, holder_{id}, \textbf{broken})$.

Verify *id* On receiving this command from P_i, the functionality verifies that an entry for container *id* appears in the table and that $holder_{id} = P_i$. It then considers $state_{id}$. If $state_{id} = \textbf{broken}$ it sends $(\textbf{Verified}, id, \textbf{broken})$ to P_i. Otherwise, it sends $(\textbf{Verified}, id, \textbf{ok})$ to P_i.

A Note About Notation In the interests of readability, we will often refer to parties "preparing" an envelope instead of specifying that they send a **Seal** message to the appropriate functionality. Likewise we say a party "verifies that an envelope is sealed" when the party sends a **Verify** message to the functionality, waits for the response and checks that the resulting **Verified** message specifies an **ok** status. We say a party "opens an envelope" when it sends an **Open** message to the functionality and waits for the **Opened** response.

Chapter 4

Receipt-Free Universally-Verifiable Voting With Everlasting Privacy

4.1 Introduction

4.1.1 Challenges in Designing Voting Protocols

One of the main problems with traditional systems is that the accuracy of the election is entirely dependent on the people who count the votes. In modern systems, this usually consists of fairly small committees: if an entire committee colludes, they can manufacture their own results. Even worse, depending on the exact setup, it may be feasible to stuff ballot boxes, destroy votes or perform other manipulations.

The problems with assuring election integrity were a large factor in the introduction of mechanical voting machines, and more recently, optical scan and "Direct Recording Electronic" (DRE) machines. These perform a function identical to a ballot box and paper ballots, using a different medium: the basic protocol remains the same. While alleviating some of the problems (such as ballot stuffing), in some cases they actually aggravate the main one: instead of relying on a large number of election committees (each of which has a limited potential for harm), their security relies on a much smaller number of programmers. Even worse, a rogue programmer may be able to change the results of the entire election *with virutally no chance of detection*.

There has also been a large amount of more theoretical research, aimed at using cryptographic tools to define and solve the problems inherent in conducting secure elections. The most important advantage of cryptographic voting protocols over their physical counterparts is the potential for *universal verifiability*: the possibility that every voter (and even interested third parties) can verify that the ballot-counting is performed correctly. The challenge, of course, is satisfying this property while still maintaining the secrecy of individual ballots.

A problem that was first introduced with mechanical voting machines, and exacerbated in DRE and many cryptographic systems, is that part of the protocol must be performed by a machine (or computer), whose inner workings are opaque to most voters. This can have a serious impact on the trust a voter places in the results of the election (e.g., "how do I know that when I pushed the button next to candidate A the machine didn't cast a vote for B?"). One of the targets recently identified in the cryptographic literature is to design systems that can be trusted by human voters *even if the election computers are running malicious code*.

Another attack on both traditional and cryptographic voting systems is vote-buying and coercion of voters. To prevent this, we would like a voter to be unable to convince a third party of her vote *even if she wants to do so*. This property, called *receipt-freeness*, is strictly stronger than ballot secrecy, and seems even harder to achieve simultaneously with universal-verifiability. As is the case for election integrity, it is much more difficult to design a receipt-free protocol if the voter is required to perform secret calculations on a computer (e.g., perform an RSA encryption of her vote): the voter may be forced to use an untrusted

computer to perform the calculations (or even one provided by the coercer), in which case the coercer can learn the secret.

There are also problems specific to the cryptographic case. One of these is that cryptographic protocols are often based on computational assumptions (e.g., the infeasibility of solving a particular problem). Unfortunately, some computational assumptions may not hold forever (e.g., Adi Shamir estimated that all existing public-key systems, with key-lengths in use today, will remain secure for less than thirty years [71]).

A voting protocol is said to provide *information-theoretic privacy* if a computationally unbounded adversary does not gain any information about individual votes (apart from the final tally). If the privacy of the votes depends on computational assumptions, we say the protocol provides *computational privacy*. Protocols that provide computational privacy may not be proof against coercion: the voter may fear that her vote will become public some time in the future.

While *integrity* that depends on computational assumptions only requires the assumptions to hold during the election, *privacy* that depends on computational assumptions requires them to hold forever. To borrow a term from Aumann et al. [5], we can say that information-theoretic privacy is *everlasting* privacy.

A related problem is that we would like to base our voting schemes on assumptions that are as weak as possible. Existing voting schemes generally require public-key encryption (or very specific computational assumptions, such as the hardness of computing discrete log in certain groups).

4.1.2 Our Results

In this paper, we present the first universally verifiable voting scheme that can be based on a general assumption (existence of a non-interactive commitment scheme).

Our protocol also satisfies the following properties:

- It has everlasting privacy (provided the commitment scheme is statistically hiding). To the best of our knowledge, only one published protocol has this property [27], and this protocol is not receipt-free.
- The protocol does not require human voters to perform any complex operations (beyond choosing a random string and comparing two strings)
- The integrity of the election is guaranteed even if the DRE is corrupted.
- It is receipt-free. We use a technique from Neff's voting scheme [62] to achieve receipt-freeness without requiring complex computation on the voter's part.

We give a rigorous proof that our protocol is secure in the Universally Composable model (given a universally-composable commitment scheme). This is a very strong notion of security. We also give a slightly more efficient protocol based on Pedersen Commitments (this protocol is secure, but not in the UC model, since Pedersen Commitments are not UC secure[1]).

One of the central contributions of this paper is a formal definition of receipt-freeness in the general multi-party computation setting (we also prove that our protocol satisfies this definition). Our definition is a generalization of Canetti and Gennaro's definition for an incoercible computation [17]. To the best of our knowledge, this is the first definition to capture receipt-freeness in the general case (most previous papers that deal with receipt-freeness do not provide a formal definition at all).

4.1.3 Previous Work on Voting Protocols

The first published electronic voting scheme was proposed by Chaum [19], based on *mixes*. Loosely speaking, a *mix* is a device that hides the correspondence between its inputs and outputs by accepting (encrypted) inputs in large batches and mixing them before output. This can be used to hide the correspondence between voters and their votes, allowing each voter to make sure her vote was counted (ensuring the integrity of the election) while preserving the secrecy of the vote. A strong advantage of this scheme over previous voting systems (e.g., putting paper slips in ballot boxes) is that the integrity of the vote no longer rests in the hands

[1] Although the proof of security in the UC model does not directly imply security of the simpler protocol, the proof is extremely similar and we omit it here.

of a few trustees: every voter can verify that their vote was counted (i.e. it has *individual* verification). The *privacy* of the votes does depend on a small number of trustees (the mixing centers), though. Other advantages are convenience and speed: a voter can vote from any location with network access, and the votes are tabulated by computers immediately after they were all cast.

Many additional protocols were suggested since Chaum's. Almost all use some combination of the following general techniques:

Mixes Some of the earliest cryptographic voting schemes (such as Chaum's original voting scheme [19] and a later scheme by Park et al. [63]) were based on mixes. Sako and Kilian introduced the notion of *universally verifiable* mixes [69], which allow external parties to verify that votes were shuffled correctly without sacrificing privacy (this idea is used in our protocol as well).

Blind Signatures A blind signature (introduced by Chaum in [20]) allows a signer to digitally sign a document without knowing what was signed. In a voting scheme based on blind signatures, the general idea is that the voter has her ballot blindly signed by the voting authority, and later publishes the ballot using an anonymous channel. Although Chaum suggested the use of blind signatures for voting in his original paper, the first published protocol that makes use of blind signatures was by Fujioka et al. [40]. A major problem of blind signature schemes is that they require *anonymous channels* (so that the voter can publish her signed vote linking the vote to the voter).

Homomorphic A function E is homomorphic if for any x and y in its domain it satisfies $E(x)E(y) = E(x+y)$. The general idea of a homomorphic voting scheme is for each voter to encrypt her vote using a public-key homomorphic encryption function, where the public key is published before the election. Each voter must prove that her encrypted vote is an encryption of a *valid* vote (the voting schemes differ on the exact way in which this is done). The encrypted votes are summed using the homomorphic property of the encryption function (without decrypting them). Finally, a set of trustees cooperate to decrypt the final tally (the secret key for the encryption scheme is divided between the trustees). The advantages of using homomorphic schemes are efficiency and verifiability: many operations can be carried out on the encrypted votes, in public, so they are both verifiable and can be performed during the voting process (without interaction between the voting authorities). The first protocol of this type was devised by Cohen (Benaloh) and Fischer [26]. Additional examples of this type of scheme are [7, 27, 28, 45].

Receipt-Free Voting Only a small fraction of the proposed voting schemes satisfy the property of receipt-freeness. Benaloh and Tuinstra [7] were the first to define this concept, and to give a protocol that achieves it (it turned out that their full protocol was not, in fact, receipt free, although their single-authority version was [45]). Their protocol was based on homomorphic encryption rather than mixes. To satisfy receipt-freeness, Benaloh and Tuinstra also required a physical "voting booth": completely untappable channels between the voting authority and the voter. Sako and Kilian showed that a one-way untappable channel between the voting authority and the voter is enough [69], and gave a receipt-free mix-type voting scheme based on this assumption (our protocol makes this assumption as well). Other protocols were also devised, however the minimal assumption required by protocols that do not use a trusted third party device (e.g., a smart card) is the one-way untappable channel.

Human Considerations Almost all the existing protocols require complex computation on the part of the voter (infeasible for an unaided human). Thus, they require the voter to trust that the computer actually casting the ballot on her behalf is accurately reflecting her intentions. Chaum [21], and later Neff [62], proposed universally-verifiable receipt-free voting schemes that overcome this problem. Recently, Reynolds proposed another protocol similar to Neff's [67].

All three schemes are based in the "traditional" setting, in which voters cast their ballots in the privacy of a voting booth. Instead of a ballot box the booth contains a DRE. The voter communicates her choice to the DRE (e.g., using a touch-screen or keyboard). The DRE encrypts her vote and posts the encrypted ballot on a public bulletin board. It then proves to the voter, in the privacy of the voting booth, that the

encrypted ballot is a truly an encryption of her intended vote. After all the votes have been cast, the votes are shuffled and decrypted using mix-type schemes.

Chaum's protocol uses a two-part ballot. Together, both parts define the vote in a manner readable to a human. Either part separately, however, contains only an encrypted ballot. The voter chooses one part at random (after verifying that the ballot matches her intended choice), and this part becomes her receipt. The other part is destroyed. The ballots are constructed so that in an invalid ballot, at least one of the two parts must be invalid (and so with probability at least $\frac{1}{2}$ this will be caught at the tally stage). Chaum's original protocol used Visual Cryptography [61] to enable the human voter to read the complete (two-part) ballot, and so required special printers and transparencies. Bryans and Ryan showed how to simplify this part of the protocol to use a standard printer [13, 68].

Neff's protocol makes ingenious use of zero-knowledge arguments. The idea is that a zero-knowledge argument system has a simulator that can output "fake" proofs indistinguishable from the real ones. The DRE performs an interactive zero-knowledge protocol with the voter to prove that the encrypted ballot corresponds to the correct candidate. The DRE uses the simulator to output a zero-knowledge proof for *every other* candidate. The proofs are of the standard "cut-and-choose" variety. In a "real" proof, the DRE commits, then the voter gives a random challenge and the DRE responds. In the "fake" proofs, the voter first gives the random challenge and then the DRE commits and responds. The voter only has to make sure that she gave the challenge for the real candidate *after* the DRE was committed, and that the challenge printed on the receipt matches what she gave. Everything else can be publicly checked outside the voting booth. Since no one can tell from the receipt in which order the commitments and challenges were made, the zero-knowledge property ensures that they cannot be convinced which of the proofs is the real one.

4.2 The Model

The physical setup of our system is very similar to many existing (non-electronic) voting schemes. Voters cast their ballots at polling stations. The votes are tabulated for each station separately, and the final tally is computed by summing the results for all stations.

4.2.1 Basic Assumptions

Human Capability An important property of our protocol is that its security is maintained even if the computers running the elections are corrupt (and only some of the human voters remain honest). Thus, we must define the operations we expect a human to perform. We make three requirements from human voters:

1. They can send messages to the DRE (e.g., using a keyboard). We require voters to send a few short phrases. This should be simple for most humans (but may be a problem for disabled voters).
2. They can verify that two strings are identical (one of which they chose themselves)
3. They can choose a random string. This is the least obvious of the assumptions we make. Indeed, choosing truly (or even seemingly) uniform random bits is probably not something most humans can do at will. However, all we actually need are strings with high enough (min) entropy. Achieving this does seem feasible, using physical aids (coins, dice, etc.) and techniques for randomness extraction. In our security proofs, in order to clarify the presentation, we will ignore these subtleties and assume the voters can choose uniformly random strings.

Physical Commitment In order to achieve receipt-freeness, our protocol requires a commitment with an extremely strong hiding property: The verifier's view at the end of the commit stage is a deterministic function of her view before the commit stage (i.e., not only can the view not contain any information about the committed string, it cannot even contain randomness added by the committer). Such a commitment is not possible in the "plain" cryptographic model (even with computational assumptions), but can be easily implemented by physical means (for example, by covering part of the printer's output with an opaque shield, so that the voter can see that something has been printed but not what). Note that the integrity of the vote does not depend on physical commitment, only its receipt-freeness.

Random Beacon The DRE in our protocol proves the final tally is correct using an interactive zero-knowledge proof. To make this proof trusted by all the voters, we assumpe the existence of a *random beacon*. The random beacon, originally introduced by Rabin [66], replaces a verifier whose messages all consist of independently distributed, random strings. In practice, the beacon can be implemented in many ways, such as by some physicial source believed to be unpredictable [58] (e.g., cosmic radiation , weather, etc.), or by a distributed computation with multiple verifiers. Note that we can simulate the beacon using a random oracle (this is the Fiat-Shamir heuristic): the entire protocol transcript so far is taken as the index in the random oracle that is used as the next bit to be sent by the beacons.

4.2.2 Participating Parties

In our description, we consider only a single polling booth (there is no interaction between booths in our system, apart from the final, public summation of results). Formally, we consider a few classes of participants in our voting protocol:

Voters There are an arbitrary number of voters participating in the protocol (we will denote the number of voters by n). Each voter has a secret input (the candidate she wants to vote for).

DRE The protocol has only a single DRE party. The DRE models the ballot box: it receives the votes of all the voters and computes the final tally at the end. In the ideal model, the DRE has neither input nor output — its only function is to explicitly model the extra capabilities gained by the adversary when the DRE is corrupted.

Verifier The verifier is a party that helps verify that the voting protocols are being followed correctly. Although there can be many verifiers (and voters can be verifiers as well) the verifiers are deterministic and use only public information, so we model them as a single party. In the ideal model, the verifier is the party that outputs the final tally (or aborts if cheating is detected).

Adversary The adversary attempts to subvert the voting protocol. We detail the adversarial model in Sections 4.2.4 and 4.2.5.

4.2.3 Protocol Structure and Communication Model

As is customary in universally-verifiable voting protocols, we assume the availability of a public *Bulletin Board*: a broadcast channel with memory. All parties can read from the board and all messages from the DRE are sent to the bulletin board.

Our voting protocols consist of three phases:

1. *Casting a Ballot.* In this phase, each voter communicates directly with the DRE over a private, untappable, channel (inside the voting booth). All communication from the DRE to the voter is through the bulletin board.

   In practice, the voter will not be able to access the bulletin board while in the voting booth. Thus, we assume there is a separate channel between the DRE and the voter, also with memory (e.g., a printer). The DRE outputs its messages both to the printer (the printed messages form the voter's receipt), and to the bulletin board. This implementation adds an additional stage in which the voter verifies that the contents of her receipt match the contents on the bulletin board.

   We need the physical commitment (see Sec. 4.2.1) only at one point in the Ballot-Casting phase.

2. *Tallying the results.* This phase begins after all voters have cast their ballots, and in this phase the results of the vote are revealed. The tallying consists of an interactive protocol between the DRE and a random beacon, whose only messages are uniformly random strings. This beacon may be implemented using some public, physical source of unpredictability (e.g. solar flares), by collective coin flipping of the voters, or of a number of trustees, or via some other mechanism. In practice, a very efficient method is the Fiat-Shamir heuristic, replacing the beacon's messages with a secure hash of the entire

transcript to that point (in the analysis the hash function is modelled as a random oracle). The entire transcript of the tally phase is sent to the bulletin board.

3. *Universal Verification.* This phase consists of verifying the consistency of the messages on the bulletin board. This can be performed by any interested party, and is a deterministic function of the information on the bulletin board.

4.2.4 Universal Composability

We consider a number of different adversarial models. In the basic model, the adversary can adaptively corrupt the DRE and the voters (since there are arbitrarily many verifiers, and all are running the same deterministic function of public information, it is reasonably to assume not all of them can be corrupted). We formalize the capabilities of the adversary by defining them in the Universally Composable model, using the ideal voting functionality, $\mathcal{F}^{(V)}$. In the ideal world, The DRE does nothing unless it is corrupted. When the DRE is corrupt, ballots no longer remain secret (note that this must be the case in any protocol where the voter divulges her vote to the DRE). The integrity of the vote is always maintained. The verifiers have no input, but get the final output from the functionality (or $\perp$ if the functionality was halted by the adversary).

The adversary is allowed to intercept every outgoing message from the functionality, and has the choice of either delivering the message or sending the **Halt** command to the functionality (in which case the message will not be received by other parties).

The ideal functionality realized by our voting scheme accepts only one "honest" command and one "cheating" command (beyond the special **Halt** and **Corrupt** commands that can be sent by the adversary):

Vote c On receiving this command from voter v, the functionality verifies that this is the only **Vote** command sent by v. It then:

1. Stores the tuple (v, c) in its internal database.
2. If the DRE is corrupt, the functionality outputs **Voted** v, c to the adversary.
3. Broadcasts the message **Voted** v.
4. If all n voters have sent a **Vote** command, the functionality computes the tally $s_c \doteq |\{(v, c') \mid c' = c\}|$ for all candidates $1 \leq c \leq m$ and outputs the tallies to the verifier.

ChangeVote c, c' This command can only be sent by a corrupt voter v and only if the DRE is also corrupt. On receiving this command, the functionality verifies that (v, c) appears in its internal database. It then replaces this tuple with the tuple (v, c'). This command can be sent after the last voter has voted and before the final tally is output.

Halt On receiving this command from the adversary, the functionality sends the special message $\perp$ to the verifier to signify an aborted execution and halts.

Corrupt DRE On receiving this command from the adversary, the functionality marks the DRE as corrupted and sends the contents of its internal database to the adversary (revealing all the votes cast so far).

Corrupt v On receiving this command from the adversary, the functionality marks voter v as corrupted and sends c_v (voter v's input) to the adversary.

The security, privacy and robustness of our protocol is proven by showing that any attack against the protocol in the real world can be performed against the ideal functionality in the ideal world (where the possible attacks are explicitly defined). The formal description of the protocol appears in Section 4.4, and its proof of security in Section 4.6.

4.2.5 Receipt-Freeness

The property of receipt-freeness is not adequately captured by a standard proof in the UC model. To deal with receipt-freeness, we have to consider an adversary that can *coerce* parties in addition to corrupting them. A coercion attack models the real-life scenario in which voters are bribed or threatened to act according the adversaries wishes. The adversary can interrogate coerced parties and give them commands, but does not completely control them (a formal definition of receipt freeness can be found in Section 4.5). When considering the receipt-freeness of our protocol, we do not allow the adversary to coerce or corrupt the DRE. The latter is because corrupting the DRE reveals the voter's inputs, and so the protocol is trivially coercible. The former is because the DRE is a machine, so it does not make sense to bribe or threaten it.

It may make sense to coerce or corrupt the DRE's *programmers*, however. The difference between this situation and a corrupt DRE is that a corrupt DRE can communicate freely with the adversary, while a "maliciously programmed" DRE can communicate with the adversary only through the public communication channel (in one direction) and the voter's input (in the other direction). We briefly discuss this problem in Section 4.8.

4.2.6 Timing Attacks

Like any cryptographic proof, the security of our protocol is guaranteed only as far as the real-world matches the model on which our proof is based. One point we think is important to mention is the "timing" side-channel. Our model does not specify timing information for messages appearing on the bulletin board — only the order of the messages. However, in a real life implementation it may be possible to time the messages sent by the DRE. If the DRE actually does send messages simultaneously to the bulletin board and the voter, this timing information can be used to determine the voter's input (since the time it takes the voter to respond will be different). To prevent this attack, we specify that the DRE sends its output to the bulletin board only after the voter leaves the booth. One possible implementation (that also guarantees that the DRE can't leak information using timing, is that the DRE is not connected to a network at all. Instead, it prints the output to be sent to the bulletin board. The printout is given by the voter to the election officials on exiting the booth, who can scan it and upload the information to the bulletin board.

4.3 Informal Protocol Description

4.3.1 Overview

At the highest level, our voting scheme is extremely simple: the voter enters the voting booth and selects a candidate. The DRE uses a statistically-hiding commitment scheme to publicly commit to the candidate (e.g., by posting the commitment on a public bulletin board). It then proves privately to the voter that the commitment is really to the voter's candidate. After all voters have cast their ballots, the DRE publishes the final tally. It then proves, using a zero knowledge proof of knowledge, that the tally corresponds to the commitments published for each voter.

Since we know how to construct a ZK proof of knowledge for any NP language, and in particular we can construct such a proof system for any string commitment scheme, it would appear that we could use any such system for the private proof (the one that convices the voter that her ballot is being cast as she intended). The zero-knowledge property would ensure that the voter cannot use the proof to convince any third party of her vote.

The problem is that the voter is human, and the general zero-knowledge proof systems require complex computations that are infeasible to perform without the help of computers. Since the scheme must remain secure even if the DRE is malicious, the voter cannot trust the DRE to make these calculations. Allowing voters to use their own computers is not much better. Most voters do not know how to verify that their computer is actually running the correct code. Even worse, a coercive adversary could require a voter to use a computer supplied by the adversary, in which case it could easily learn the identity of the voter's candidate.

Our solution is that used in Neff's voting scheme: Neff observed that a standard cut-and-choose zero knowledge proof of some statement S has the following structure: the prover commits to two proofs P_0, P_1,

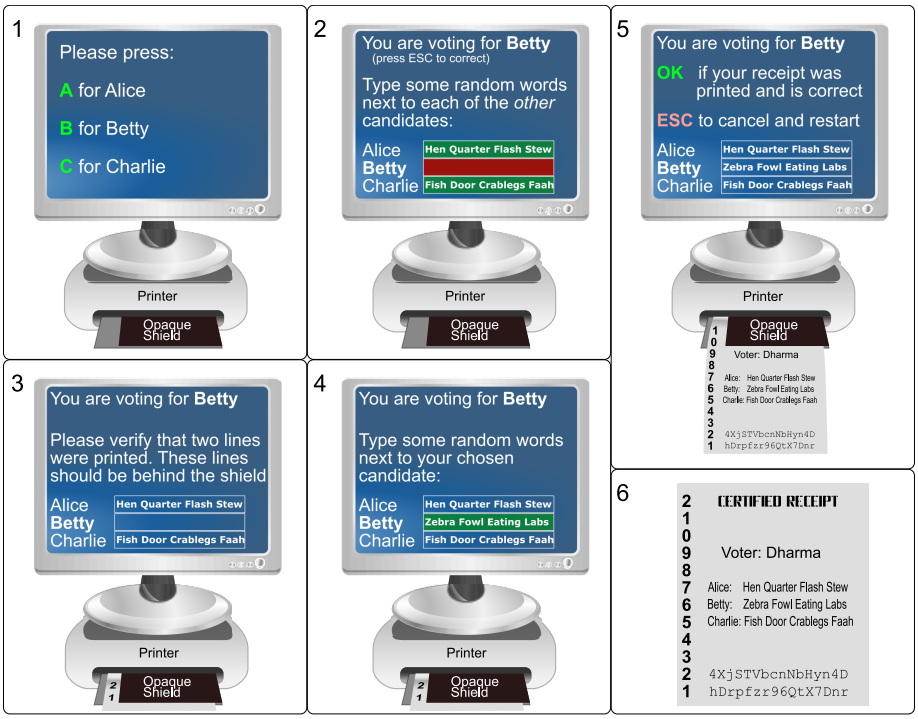

Figure 4.3.1: Ballot for Betty

the verifier makes a choice b, the prover reveals proof P_b and finally the verifier makes sure the revealed proof is correct. The protocol is constructed so that if both P_0 and P_1 are valid proofs, the statement S holds but, given b, anyone can construct a pair P'_0, P'_1 so that P'_b is correct (even if S does not hold). The insight is that a human can easily make the choice b without the aid of a computer. To keep the proof private, the DRE constructs a *dummy* proof for all the other candidates by running the ZK simulator. The only difference between the real and dummy proofs in this case is that in the real proof the DRE first commits, and then the voter chooses b, while in the dummy proof the voter reveals b before the DRE commits. This *temporal* information cannot be inferred from the receipt. However, since the receipt is public, anyone can check (using a computer) that both P_b and P'_b are valid proofs. Even if the voter does not trust her own computer, it is enough that someone with a good implementation of the verification algorithm perform the check.

In the following subsections we present a concrete implementation of our generic protocol. This implementation is based on Pedersen commitments. In Section 4.3.2 we describe an example of a hypothetical voter's interaction with the system. Section 4.3.3 goes behind the scenes, and describes what is actually occurring during this session (as well as giving a brief description of the protocol itself).

4.3.2 A Voter's Perspective

Figure 4.3.1 shows what Dharma, our hypothetical voter, would see while casting her ballot in an election between candidates Alice, Betty and Charlie.

Dharma identifies herself to the election official at the entrance to the polling station and enters the booth. Inside the booth is the DRE: a computer with a screen, keyboard and an ATM-style printer

1. The screen presents the choice of candidates: "Press A for Alice, B for Betty and C for Charlie". Dharma thinks Betty is the best woman for the job, and presses B.

2. The DRE now tells Dharma to enter some random words next to each of the other candidates (Alice

and Charlie). For an even simpler experience, the DRE can prefill the random words, and just give Dharma the option of changing them if she wants. At any time until the final stage, Dharma can change her mind by pressing **ESC**. In that case, the DRE spits out whatever has been printed so far (this can be discarded), and returns to stage 1.

3. The DRE prints two rows. The actual printed information is hidden behind a shield, but Dharma can verify that the two rows were actually printed.

4. Dharma enters a random challenge for her chosen candidate.

5. The DRE prints out the rest of the receipt. Dharma verifies that the challenges printed on the receipt are identical to the challenges she chose. If everything is in order, she presses **OK** to finalize her choice. If something is wrong, or if she changed her mind and wishes to vote for a different candidate, she presses **ESC** and the DRE returns to stage 1.

6. The DRE prints a "Receipt Certified" message on the final line of the receipt. Dharma takes her receipt and leaves the voting booth. At home, she verifies that the public bulletin board has an exact copy of her receipt, including the two lines of "gibberish" (the bulletin board can be viewed from the internet). Alternatively, she can give her receipt to an organization she trusts (e.g., "Betty's Popular People's Front"), who will perform this verification for her.

After all the voters have cast their ballots, the protocol moves to the Final Tally phase. Voters are not required to participate in this phase — it consists of a broadcast from the DRE to the public bulletin board (here we assume we are using a random beacon or the Fiat-Shamir heuristic to make the final tally non-interactive). Note that when the Fiat-Shamir heuristic is used, we do not actually require the DRE to be connected to a network. The DRE can store its output (e.g., on a removable cartridge). After the DRE has written the final tally message, the contents of the cartridge can be uploaded to the internet. Anyone tampering with the cartridge would be detected.

Anyone interested in verifying the election results participates in the Universal Verification phase. This can include voters, candidates and third parties. Voters do not have to participate, as long as they made sure that a copy of their receipt appears on the bulletin board, and they trust that at least one of the verifying parties is honest.

Receipt-Freeness To get an intuitive understanding for why this protocol is receipt-free, suppose Eve tries to bribe Dharma to vote for Alice instead. There are only two things Dharma does differently for Alice and Betty: she presses B in the first step, and she fills in Alice's random words before the DRE prints the first two lines of the receipt, while filling in Betty's afterwards. Eve has no indication of what Dharma pressed (since the receipt looks the same either way). The receipt also gives no indication in what order the candidate's words were filled (since the candidates always appear in alphabetical order). Because the first two lines of the receipt are hidden behind the shield when Dharma enters the challenge for her chosen candidate, she doesn't gain any additional information as a result of filling out the challenges for Alice and Charlie; so whatever Eve asks her to do, she can always pretend she filled out the challenge for Alice after the challenge for Betty.

4.3.3 Behind the Scenes: An Efficient Protocol Based on the Discrete Log Assumption

Pedersen Commitments The concrete protocol we describe in this section is based on Pedersen commitments [64]; statistically hiding commitments whose security is based on the hardness of discrete-log. We briefly describe the Pedersen commitment, assuming discrete log is hard in some cyclic group G of order q, and $h, g \in G$ are generators such that the committer does not know $\log_g h$. To commit to $a \in \mathbb{Z}_q$, the committer chooses a random element $r \in \mathbb{Z}_q$, and sends $h^a g^r$. Note that if the committer can find $a' \neq a$ and r' such that $h^{a'} g^{r'} = h^a g^r$, then $h^{a'-a} = g^{r-r'}$, and so the committer can compute $\log_g h = \frac{r-r'}{a'-a}$ (in contradiction to the assumption that discrete log is hard in G). Therefore the commitment is computationally binding. If r is chosen uniformly at random from $\mathbb{Z}_q$, then g^r is a uniformly random element of G (since

g is a generator), and so for any a the commitment $h^a g^r$ is also a uniformly random element of G. So the commitment is perfectly hiding.

We'll now show what happened behind the scenes, assuming the parameters G, h and g of the Pedersen commitment scheme are decided in advance and known to all parties. For simplicity, we also assume a collision-resistant hash function $H : \{0,1\}^* \mapsto \mathbb{Z}_q$. This allows us to commit to an arbitrary string $a \in \{0,1\}^*$ by committing to $H(a)$ instead (we need this because we require the DRE to commit to some strings that are too long to map trivially into $\mathbb{Z}_q$). Denote $P(a,r) = h^{H(a)} g^r$. To open $P(a,r)$, the committer simply sends a, r.

The security of the scheme depends on a security parameter, k. The probability that the DRE can change a vote without being detected is $2^{-k+1} + O(nk\epsilon)$, where n is the number of voters and ϵ is the probability of successfully breaking the commitment scheme. We will require the DRE to perform a total of $O(mnk)$ commitments (where m is the number of candidates, and n the number of voters).

Casting the Ballot. We'll go over the stages as described above.

1. (Dharma chose Betty). The DRE computes a commitment: $v = P(\text{Betty}, r)$ (where r is chosen randomly), and prepares the first step in a proof that this commitment is really a commitment to Betty. This step consists of computing, for $1 \leq i \leq k$, a "masked" copy of v: $b_i = v g^{r_{B,i}} = P(\text{Betty}, r + r_{B,i})$, where $r_{B,i}$ is chosen randomly.

2. (Dharma enters dummy challenges). The DRE translates each challenge to a k bit string using a predetermined algorithm (e.g., by hashing). Let A_i be the i^{th} bit of the challenge for Alice. For each bit i such that $A_i = 0$ the DRE computes a commitment to Alice: $a_i = P(\text{Alice}, r + r_{A,i})$, while for each bit such that $A_i = 1$ the DRE computes a masked copy of the real commitment to *Betty*: $a_i = v g^{r_{A,i}}$. Note that in this case, $a_i = P(\text{Betty}, r + r_{A,i})$. The set of commitments $a_1, \ldots, a_k$ will form a dummy proof that v is a commitment to Alice (we'll see why we construct them in this way in the description of universal verification phase step 4.3.3). The DRE also computes $c_1, \ldots, c_k$ in the same way for Charlie.

   The DRE now computes a commitment to everything it has calculated so far:

   $$x = P([v, a_1, \ldots, a_k, b_1, \ldots, b_k, c_1, \ldots, c_k], r_x).$$

   It prints x on the receipt (this is what is printed in the first two lines).

3. (Dharma enters the real challenge) The DRE translates this challenge into a k bit string as in the previous step. Denote B_i the i^{th} bit of the real challenge.

4. (The DRE prints out the rest of the receipt). The DRE now computes the answers to the challenges: For every challenge bit i such that is $X_i = 0$ (where $X \in \{A, B, C\}$), the answer to the challenge is $s_{X,i} = r + r_{X,i}$. For $X_i = 1$, the answer is $s_{X,i} = r_{X,i}$. The DRE stores the answers. It then prints the candidates and their corresponding challenges (in alphabetical order), and the voter's name (Dharma).

5. (Dharma accepts the receipt). The DRE prints a "Receipt Certified" message on the final line of the receipt. (the purpose of this message is to prevent voters from changing their minds at the last moment, taking the partial receipt and then claiming the DRE cheated because their receipt does not appear on the bulletin board). It then sends a copy of the receipt to the public bulletin board, along with the answers to the challenges and the information needed to open the commitment x: $(s_{A,1}, \ldots, s_{A,k}, s_{B,1}, \ldots, s_{B,k}, s_{C,1}, \ldots, s_{C,k})$ and $([v, a_1, \ldots, a_k, b_1, \ldots, b_k, c_1, \ldots, c_k], r_x)$.

Final Tally. The DRE begins the tally phase by announcing the final tally: how many voters voted for Alice, Betty and Charlie. Denote the total number of voters by n, and $v_i = P(X_i, r_i)$ the commitment to voter i's choice (X_i) that was sent in the Ballot Phase. The DRE now performs the following proof k times:

1. The DRE chooses a random permutation π of the voters, and n "masking numbers" $m_1, \ldots, m_n$. It then sends the permuted, masked commitments of the voters:
   $v_{\pi(1)} g^{m_{\pi(1)}}, \ldots, v_{\pi(n)} g^{m_{\pi(n)}}$

2. The random beacon sends a challenge bit b

3. If $b = 0$, the DRE sends π and $m_1, \ldots, m_n$ (unmasking the commitments to prove it was really using the same commitments it output in the Ballot-Casting phase). If $b = 1$, the DRE opens the masked commitments (without revealing π, the correspondence to the original commitments). It sends: $(X_{\pi(1)}, r_{\pi(1)} + m_{\pi(1)}), \ldots, (X_{\pi(n)}, r_{\pi(n)} + m_{\pi(n)})$

Universal Verification (and security proof intuition). The purpose of the universal verification stage is to make sure that the DRE sent well-formed messages and correctly opened all the commitments. For the messages from the Ballot-Casting phase, the verifiers check that:

1. $x = P([v, a_1, \ldots, a_k, b_1, \ldots, b_k, c_1, \ldots, c_k], r_x)$ This ensures that the DRE committed to v and $b_1, \ldots, b_k$ (in Dharma's case) before Dharma sent the challenges $B_1, \ldots, B_k$ (because x was printed on the receipt before Dharma sent the challenges).

2. For every commitment x_i (where $x \in \{a, b, c\}$), its corresponding challenge X_i, and the response $s_{X,i}$, the verifiers check that x_i is a good commitment to X when $X_i = 0$ (i.e., $x_i = P(X, s_{X_i})$) and that x_i is a masked version of v if $X_i = 1$ (i.e., $vs_{X_i} = x_i$). Note that if x_i is both a masked version of v *and* a good commitment to X, then v must be a good commitment to X (otherwise the DRE could open v to two different values, contradicting the binding property of the commitment). This means that if v is a commitment to some value other than the voter's choice, the DRE will be caught with probability at least $1 - 2^{-k}$: every commitment x_i can be either a good masked version of v or a good commitment to X, but not both. So for each of the k challenges (which are not known in advance to the DRE), with probability $\frac{1}{2}$. The DRE will not be able to give a valid response.

For the final tally phase, the verifiers also check that all commitments were opened correctly (and according to the challenge bit). As in the Ballot-Casting phase, if the DRE can correctly answer a challenge in both directions (i.e., the commitments are a permutation of good masked versions of commitments to the voter's choices, and also when opened they match the tally), then the tally must be correct. So the DRE has probability at least $\frac{1}{2}$ of getting caught for each challenge if it gave the incorrect tally. If the DRE wants to change the election results, it must either change the final tally, change at least one of the voter's choices or break the commitment scheme. Since the protocol uses $O(nk)$ commitments (note that cheating on the commitments in the dummy proofs doesn't matter), the total probability that it can cheat is bounded by $2 \cdot 2^{-k} + O(nk\epsilon)$.

Protocol Complexity. We can consider both the time and communication complexity of the protocol. In terms of time complexity, the DRE must perform $O(knm)$ commitments in the Ballot Casting phase (where m is the number of candidates), and $O(kn)$ commitments in the Final Tally phase (the constants hidden in the O notation are not large in either case). Verifiers have approximately the same time complexity (they verify that all the commitments were opened).

The total communication complexity is also of the same order. In this case, the important thing is to minimize the DRE's communication to the voter (since this must fit on a printed receipt). Here the situation is much better: the receipt only needs to contain a single commitment and the challenges sent by the voter (each challenge has k bits). Note that we do not use any special properties of the commitment on the receipt (in practice, this can be the output of a secure hash function rather than a Pedersen commitment).

4.3.4 Using Generic Commitment

The protocol we described above makes use of a special property of Pedersen commitment: the fact that we can make a "masked" copy of a commitment. The essence of our zero knowledge proof is that on the one hand, we can prove that a commitment is a masked copy of another without opening the commitment. On the other hand, just by seeing two commitments there is no way to tell that they are copies, so opening one does not give us information about the other.

Our generic protocol uses the same idea, except that we implement "masked copies" using an arbitrary commitment scheme. The trick is to use a nested commitment (commitments to commitments). We create all the "copies" in advance (we can do this since we know how many copies we're going to need); a "copyable" commitment to a, assuming we're going to need k copies, consists of k commitments to commitments to a.

Denote $C(a, r)$ a commitment to a with randomness r. Then the copyable commitment consists of $v = C(C(a, r_1), s_1), \ldots, C(C(a, r_k), s_k)$. The i^{th} copy of the commitment is $C(a, r_i)$. The hiding property of C ensures that there is no way to connect $C(a, r_i)$ to v. On the other hand, the binding property of C ensures that we cannot find any s' such that $C(C(a, r_i), s')$ is in v unless this was the original commitment. A formal specification for constructing copyable commitment from "standard" UC commitment appears in Section 4.7 (the protocol described there is slightly more complex, but the intuition remains the same).

Unlike the case with Pedersen commitments, when using the nested commitment trick an adversary *can* "copy" a commitment to a different value (the adversary can always use different values in the inner commitments). The insight here is that the adversary still has to commit in advance to the locations of the errors. After the DRE commits, we randomly permute the indices (so $v = C(C(a, r_{\sigma(1)}), s_{\sigma(1)}), \ldots, C(C(a, r_{\sigma(k)}), s_{\sigma(k)})$, for some random permutation σ). If the DRE did commit to the same value at every index, this permutation will not matter. If the DRE committed to different values, we show that it will be caught with high probability. Intuitively, we can consider each commitment in the tally phase as a row of a matrix whose columns correspond to the voters. By permuting the indices of the copies, we are effectively permuting the columns of the matrix (since in the i^{th} tally step we force the DRE to use the i^{th} copy. The DRE can pass the test only if all the rows in this matrix have the same tally. But when the columns are not constant, this occurs with small probability. A formal specification of the general protocol appears in Section 4.4. To simplify the proof, the technique we use in practice is slightly different than the one described above, however the idea is the same.

4.4 Abstract Protocol Construction

This section contains a description of the protocol using abstract "ideal functionalities". The abstract specification can be thought of as a high-level view of the protocol construction as well as a formal description. We prove that this construction is secure, and in the following sections give explicit protocols that implement the ideal functionalities.

Our specific protocol constructions explicitly take into account which parts of the protocol can be performed by unaided humans and which require computers. In this high-level description, however, these details are ignored. To prove that the protocol is secure even in the presence of untrusted computers, we first prove it is secure given some basic "ideal" building blocks, and later show that these building blocks can be securely realized by humans.

4.4.1 Building Blocks

The abstract protocol uses a functionality we call "Commit-with-Copy" and denote $\mathcal{F}^{(C\&C)}$. The $\mathcal{F}^{(C\&C)}$ functionality models a string commitment with one committer a verifier and an additional extra property: The committer has the ability to "copy" a commitment, creating a commitment to the same value as the original. The two commitments are "linked", so that the committer can prove that the new commitment is a copy of the old *without revealing its value.*

Any commitment scheme which allows a committer to prove equivalence between commitments has this property: the copy is simply a new commitment to the same value, and proving it is a copy consists of proving the equivalence of the commitments.

For our voting protocol, a weaker version of this functionality suffices: we distinguish between "source" commitments and linked copies (each linked copy has a corresponding source commitment), and fix in advance the number of copies we allow the committer to make. We also allow the commiter to create "fake", unlinked copies of a source commitment; these may not be commitments to the same value as the source commitment, but the committer can only prove a "real" copy is linked to its source. We limit the number of fake commitments as well.

We can tolerate a corrupt committer that makes faulty (linked) copies, as long as the committer decides in advance (at the time of committing) which of the copies will be bad and to what value they can be opened (i.e., a faulty copy is one for which the corrupt committer can "prove" it is a copy, but can open to a value different from the original). We will denote an instance of $\mathcal{F}^{(C\&C)}$ that is limited to k linked copies (some of which may be faulty) by $\mathcal{F}^{(C\&C[k])}$. We show how to implement $\mathcal{F}^{(C\&C[k])}$ in a universally composable manner based on any UC commitment scheme in Section 4.7.

Formally, functionality $\mathcal{F}^{(C\&C[k])}$ allows three forms of commitments: source commitments, linked copies of a source commitment and fake copies of a source commitment. To keep track of existing commitments, $\mathcal{F}^{(C\&C[k])}$ maintains two internal databases: one for source commitments and one for copies (both linked and fake). The source commitment database contains tuples of the form (r, s, F), where r is a unique tag (arbitrary string) identifying the commitment, s the committed string, $F = (s'_1, \ldots, s'_{|F|})$ a vector of potential "fake" copy values (the honest committer is limited to one fake commitment for each value in F, albeit in an arbitrary order). The "copy" database contains tuples of the form (r, s, o, i), where r is a unique tag for the copy, s is the value to which it can be openend, o is the corresponding source commitment, and i is the copy index ($i \in \{1, \ldots, k\}$ for linked copies, and $i = \perp$ for fake copies). We use the shorthand notation "r appears in the source database" to say that there exist s and F such that the tuple (r, s, F) appears in the source commitment database. and "r appears in the copy database" to say that there exist s, o and i such that (r, s, o, i) appears in the copy commitment database. In our constructions, the commitment tags are unique, and we can denote s_r and F_r (or s_r, o_r and i_r) the corresponding elements of the tuple whose first element is r. In the following description, we assume there is a single committer (our protocol only requires the DRE to commit). The functionality accepts commands only from the committer. Output is sent to all parties (as well as the adversary). The commands accepted by the functionality are the following:

SrcCommit r, s, F This command models a source commitment (that can be copied but not opened). r is the unique tag identifying this commitment, s is the commitment value, and F is the vector of possible values for "fake" (unlinked) commitments. On receiving this command, the functionality verifies that r does not appear in the source database, stores (r, s, F) in the database and outputs $(\textbf{SrcCommitted}, r, |F|)$.

BadCommit r, Q, ℓ This command can only be sent by a corrupt committer. From the receiver's point of view, it is identical to the standard **SrcCommit** command. r is the unique identifying tag for the commitment. $Q \doteq (s^{(1)}, \ldots, s^{(k)})$ is a vector of committed values; the i^{th} copy of the commitment will be opened to $s^{(i)}$. ℓ is a bound on the total number of copies (both linked and fake). Note that unlike the honest committer, the corrupt committer can create linked commitments to any value and is not required to commit in advance to the fake commitment values. On receiving this command, the functionality verifies that r does not appear in the source database and that $k \leq |Q| \leq \ell$ and stores $(r, \perp, \perp)$ in the database. For each $i \in |Q|$ it stores $(\perp, s^{(i)}, r, i)$ in the copy database. It then outputs $(\textbf{SrcCommitted}, r, \ell - k)$.

LinkedCopy r, r', i On receiving this command, the functionality verifies that r appears in the source database and that r' does not appear in the copy database. If the committer is corrupt and a **BadCommit** r, Q, ℓ command was previously sent, the functionality verifies that there is an entry of the form $(\perp, s'^{(i)}, r, i)$ in the copy database and replaces it with $(r', s'^{(i)}, r, i)$. Otherwise, the functionality verifies that $1 \leq i \leq k$ and that there is no entry of the form $(*, *, r, i)$ in the copy database (where $*$ matches anything but $\perp$) and stores (r', s_r, r, i) in that database. It then outputs $(\textbf{Committed}, r, r')$

FakeCopy r, r', s This command models an unlinked commitment. To the receiver, it is indistinguishable from a linked copy of the same source, and functions as a standard commitment with regards to opening. Unlike a linked copy, the committer cannot use the **ProveCopy** command to prove a fake copy is linked to the source commitment. On receiving this command, the functionality verifies that r appears in the source database and that r' does not appear in the copy database. If the committer is honest, it also verifies that $s \in F_r$ and that the number of commands of the form **FakeCopy** $r, *, s$ that were previously received is less than the number of occurrences of s in F_r. If these conditions are all met, the functionality stores $(r', s, r, \perp)$ and outputs $(\textbf{Committed}, r, r')$ (note that this output is identical to the output when the committer sends a **LinkedCopy** command).

Open r On receiving this command, the functionality verifies that r is in the copy database. If so, it outputs $(\textbf{Opened}, r, s_r)$.

ProveCopy r On receiving this command, the functionality verifies that r appears in the copy database and that $i_r \neq \perp$ (i.e., r is not a fake copy). It then outputs $(\textbf{CopyOf}, r, o_r, i_r)$. Note that this output also reveals the index of the copy.

4.4.2 Protocol Description

Protocol 4.1a Ballot casting by voter v (Voter)

Input: Chosen candidate x_v, security parameter k

1: Send x_v to DRE
2: Wait to receive $(\textbf{SrcCommitted}, v)$ from $\mathcal{F}^{(C\&C)}$
3: Choose a random subset $R_v \subset [(m+2)k]$ of size $|R| = 2k$.
4: Send R_v to DRE
5: **for** $1 \leq c \leq m$ **do** {Loop over all candidates}
6: **if** $c = x_v$ **then** {Generate "real" proof commitments}
7: Wait to receive $(\textbf{Committed}, v, v_i^{(c)})$ from $\mathcal{F}^{(C\&C)}$ for all $1 \leq i \leq k$
8: Choose "real" challenge bits: $r_v^{(c)} \doteq r_v^{(c,1)}, \ldots, r_v^{(c,k)} \in_R \{0,1\}$
9: Send $r_v^{(c)}$ to DRE
10: **else** {$c \neq x_v$; Generate "dummy" proof commitments}
11: Choose "dummy" challenge bits: $r_v^{(c)} \doteq r_v^{(c,1)}, \ldots, r_v^{(c,k)} \in_R \{0,1\}$
12: Send $r_v^{(c)}$ to DRE
13: Wait to receive $(\textbf{Committed}, v, v_i^{(c)})$ from $\mathcal{F}^{(C\&C)}$ for all $1 \leq i \leq k$
14: **end if**
15: **end for**
16: Broadcast R_v and $r_v^{(1)}, \ldots, r_v^{(m)}$.

Ballot Casting. This part of the protocol is performed for each voter. The ballot casting takes place inside the voting booth, where we assume the voter has an untappable private channel to the DRE. However, while inside the voting booth the voter does not have access to trusted computers (we only require the voter to choose random strings and perform string comparison operations).

The DRE has a public, authenticated broadcast channel. In practice, this would be implemented by a public bulletin board (which can be read by anyone). Since the voter cannot access the bulletin board while inside the voting booth, in practice the DRE will communicate with the voter by printing any messages it sends to the bulletin board (in order to prevent timing attacks, the DRE will send the messages to the bulletin board only at the end of the voting session). On leaving the voting booth, the voter will verify that the messages appearing on the bulletin board match the printed receipt. The DRE also has access to an instance of $\mathcal{F}^{(C\&C[(m+2)k])}$ (we will write simply $\mathcal{F}^{(C\&C)}$), where k is a security parameter and m is the number of candidates. The same instance of $\mathcal{F}^{(C\&C)}$ is used for all voters, and the committer is the DRE in all cases. Outputs of $\mathcal{F}^{(C\&C)}$ meant for the receiver also go to the broadcast channel. Since $\mathcal{F}^{(C\&C)}$ is deterministic, we can assume the DRE has a copy of its internal database.

The ballot casting protocol for voter v with input x_v (i.e., voting for candidate x_v) appears in two parts: Protocol 4.1a (from the voter's point of view) and Protocol 4.1b (from the DRE's point of view).

Note that the difference between the dummy and real proofs, in terms of the voter's actions, is only in the order of the events: in the real proof, the DRE commits first, and then the voter sends a random challenge, while in the dummy proof the voter sends the random challenge first, and only after that the DRE chooses its commitment.

Final Tally. Informally, the final tally protocol is very similar to the standard ZK proof that the commitments appearing on the bulletin board (those made in the Ballot Casting phase) are consistent with the

Protocol 4.1b Ballot casting by voter v (DRE)

Input: Security parameter k

1: Wait to receive x_v from voter v
2: Let $F \doteq (s'_{1,1}, \ldots, s'_{1,k}, \ldots, s'_{m,1}, \ldots, s'_{m,k})$, where $s'_{i,j} \doteq i$.
3: Send **SrcCommit** v, x_v, F to $\mathcal{F}^{(C\&C)}$
4: Wait to receive subset $R_v \subset [(m+2)k]$ from voter. Denote $\bar{R}_v \doteq [(m+2)k] \setminus R_v$. Consider the set $\bar{R}_v$ as a $m \times k$ matrix of indices: $\bar{R}_v \doteq \left\{\bar{R}_v^{(1,1)}, \ldots, \bar{R}_v^{(1,1)}\right\}$
5: **for** $1 \leq c \leq m$ **do** {Loop over all candidates}
6: **if** $c = x_v$ **then** {Generate "real" proof commitments}
7: Send **LinkedCopy** $v, v_i^{(c)}, \bar{R}_v^{(c,i)}$ to $\mathcal{F}^{(C\&C)}$ for all $1 \leq i \leq k$
8: Wait to receive "real" challenge bits: $r_v^{(c)} \doteq r_v^{(c,1)}, \ldots, r_v^{(c,k)}$
9: **else** {$c \neq x_v$; Generate "dummy" proof commitments}
10: Wait to receive "dummy" challenge bits: $r_v^{(c)} \doteq r_v^{(c,1)}, \ldots, r_v^{(c,k)}$
11: **for** $1 \leq i \leq k$ **do**
12: **if** $r_v^{(c,i)} = 0$ **then**
13: Send **FakeCopy** $v_i^{(c)}, c$ to $\mathcal{F}^{(C\&C)}$
14: **else**
15: Send **LinkedCopy** $v, v_i^{(c)}, \bar{R}_v^{(c,i)}$ to $\mathcal{F}^{(C\&C)}$
16: **end if**
17: **end for**
18: **end if**
19: **for** $1 \leq i \leq k$ **do** {Open proof commitments}
20: **if** $r_v^{(c,i)} = 0$ **then**
21: Send **Open** $v_i^{(c)}$ to $\mathcal{F}^{(C\&C)}$
22: **else**
23: Send **ProveCopy** $v_i^{(c)}$ to $\mathcal{F}^{(C\&C)}$
24: **end if**
25: **end for**
26: **end for**

tally announced by the DRE. If the Commit-and-Copy functionality only allows perfect copies, repeating the following interactive protocol would work:

1. The DRE sends a random permutation of copies of the commitments
2. The verifier decides whether the DRE should:
   (a) Open the copies, proving that they match the announced tally
   (b) Disclose the random permutation, and prove that the commitments made by the DRE are actually copies of the original commitments.

   Clearly, if the DRE can do both at the same time, the tally must be correct.

An alternative description of this simple protocol is that the DRE sends a matrix of commitment copies, each row containing a random permutation of the commitment copies. The verifier then chooses, independently, a challenge bit for each row. Here, the i^{th} row of the matrix contains the i^{th} copy of each commitment.

When the Commit-and-Copy functionality allows the adversary to make some bad copies, the simple protocol is no longer secure. Our modification is to add an additional "challenge" stage: after the DRE commits (also committing to the bad copies), the challenge consists of a random pairing of the copies; each of the first k copies is paired to a random copy from the second k copies. Each element in the matrix will now consist of a pair of commitments. The DRE then randomly permutes the rows as in the simple protocol. The verifier also makes an additional check: that the two copies of each pair are commitments to the same

candidate. For the DRE to alter the tally results (using the bad copies), it must have at least one bad copy in each row (if a row consists entirely of good copies, the tally for that row matches the voters' intent). Since the bad copies are fixed before the pairs are chosen, the probability that each pair contains exactly two bad copies or two good copies is exponentially small in the number of rows when there is a constant fraction of bad copies (thus, so is the probability that the DRE can successfully pass the verification).

Protocol 4.2 Final Tally (DRE)

1: Choose random permutations $\pi_0, \ldots, \pi_k$, $\pi_i : [n] \mapsto [n]$.
2: Publish $c_{\pi_0(1)}, \ldots, c_{\pi_0(n)}$ {A random shuffle of the votes}
3: Wait to receive permutations $\sigma_1, \ldots, \sigma_n$ from random beacon ($\sigma_i : [k] \mapsto [k]$)
4: **for** $1 \leq i \leq k$ **do**
5: **for** $1 \leq v \leq n$ **do**
6: Send **Copy** $\pi_i(v), \boxed{v, i}, \sigma_{\pi_i(v)}(R^{(i)}_{\pi_i(v)})$ to $\mathcal{F}^{(C\&C)}$
7: Send **Copy** $\pi_i(v), \boxed{v, k+i}, R^{(k+i)}_{\pi_i(v)}$ to $\mathcal{F}^{(C\&C)}$
8: **end for**
9: **end for**
10: Wait to receive challenge bits $b_1, \ldots, b_k$ from random beacon
11: **for** $1 \leq i \leq k$ **do**
12: **if** $b_i = 0$ **then**
13: Send **ProveCopy** $\boxed{v, i}$ and **ProveCopy** $\boxed{v, k+i}$ for all $1 \leq v \leq n$ {This reveals π_i}
14: **else** {$b_i = 1$}
15: Send **Open** $\boxed{v, i}$ and **Open** $\boxed{v, k+i}$ for all $1 \leq v \leq n$ {This reveals a shuffle of the votes (by π_i)}
16: **end if**
17: **end for**

A formal description appears as Protocol 4.2. To help clarify the notation, we give a more intuitive explanation below. Denote x_v the candidate chosen by voter v. Recall that we had $2k$ copies of v "left over" from the Ballot Casting phase. The indices of these copies are the set $R_v \doteq \left\{R_v^{(1)}, \ldots, R_v^{(2k)}\right\}$. To reduce clutter, in the description below when we talk about the i^{th} copy of the commitment to x_v, we actually mean the i^{th} unused copy, whose index is $R_v^{(i)}$. Denote M the $2k \times n$ matrix such that $M_{i,v}$ is the i^{th} copy of the commitment to x_v. We can think of the commitment copies produced in steps 4 to 9 of Protocol 4.2 as a matrix M'', that is produced by permuting the columns and then the rows of M (Figure 4.4.2 depicts an example of the operations for 5 voters and $k = 2$):

1. First a matrix M' is constructed from M by independently permuting each column of M using the permutations $\sigma_1, \ldots, \sigma_n$ (sent by the random beacon). The first k elements of column i are permuted using σ_i, while the last k elements remain in their original positions. (we will consider the elements $M'_{i,v} \doteq M_{\sigma_v(i),v}$ to be paired with the element $M'_{k+i,v} \doteq M_{k+i,v}$).

2. Next, M'' is constructed from M' by independently permuting each row of the M' using the permutations $\pi_1, \ldots, \pi_k$ (chosen by the DRE): in the first k rows, row i is permuted using π_i. In the last k rows, row $k+i$ is again permuted using π_i.

Note that the DRE is not required to perform exactly these steps; they are simply a way to describe the final result. The commitment copy whose tag is $\boxed{v, i}$ corresponds to the element $M''_{i,v} \doteq M'_{i,\pi_i(v)}$.

Universal Verification.

1. (Verify the Ballot Casting) The verifiers check, for every voter, that the responses sent by the DRE to the challenges are correct (i.e., for every $i \in [k]$, $v \in [n]$ and $c \in [m]$, if $r_v^{(c,i)} = 0$ then an $(\textbf{Opened}, v_i^{(c)}, c)$ message was received from $\mathcal{F}^{(C\&C)}$ and if $r_v^{(c,i)} = 1$ a message $(\textbf{CopyOf}, v_i^{(c)}, v, \bar{R}_v^{(c,i)})$ was received from $\mathcal{F}^{(C\&C)}$).

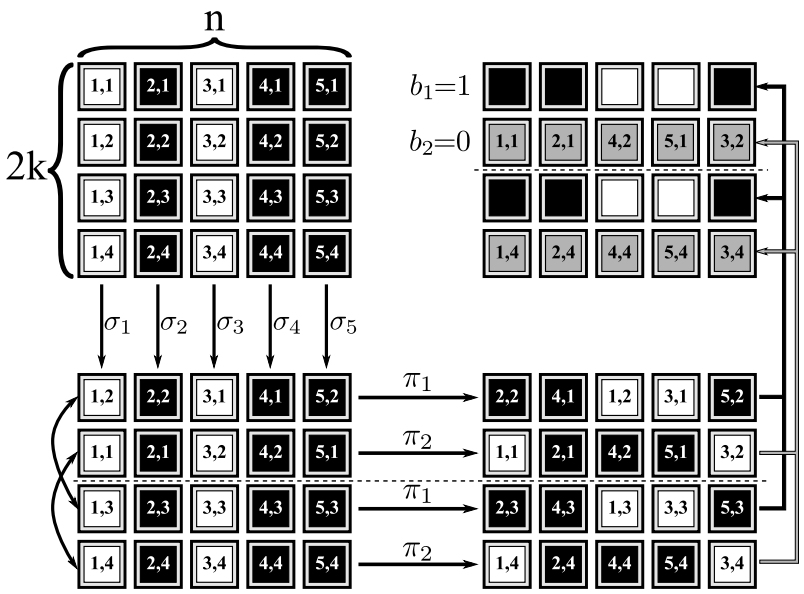

Figure 4.4.1: Example of Final Tally Matrix (for $n = 5$ voters and $k = 2$)

2. (Verify the Final Tally) The verifiers check that for every i such that $b_i = 0$ and every voter v, the responses (**CopyOf**, $\boxed{v, i}$, $\pi_i(v)$, $\sigma_{\pi_i(v)}(R^{(i)}_{\pi_i(v)})$) and (**CopyOf**, $\boxed{v, k+i}$, $\pi_i(v)$, $R^{(k+i)}_{\pi_i(v)}$) were received from $\mathcal{F}^{(C\&C)}$. The verifiers check that for every i such that $b_i = 1$ the responses (**Opened**, $\boxed{v, i}$, x) and (**Opened**, $\boxed{v, k+i}$, y) were received from $\mathcal{F}^{(C\&C)}$, that $x = y$ and that the tally matches the tally announced in step 2 of Protocol 4.2.

If any of the verification steps fail, the verifier outputs $\perp$ and aborts. Otherwise, the verifier outputs the tally.

4.4.3 Protocol Security

We prove the protocol's accuracy and privacy guarantees in the UC framework. Formally. the following theorem is proven in Section 4.6:

Theorem 4.1. *The abstract protocol, using $\mathcal{F}^{(C\&C[(m+2)k])}$, UC-realizes $\mathcal{F}^{(V)}$.*

The protocol is receipt-free under the definition detailed in Section 4.5. We formally prove the following theorem in Section 4.5.4:

Theorem 4.2. *The abstract protocol, using $\mathcal{F}^{(C\&C[(m+2)k])}$, is receipt-free.*

4.5 Incoercibility and Receipt-Freeness

Standard "secure computation" models usually deal with two types of parties: honest parties with a secret input that follow the protocol, and corrupt parties that are completely controlled by the adversary. In voting protocols, we often need to consider a third type of player: a party that has a secret input, but is threatened (or bribed) by the adversary to behave in a different manner. Such a "coerced" player differs from a corrupt party in that she doesn't do what the adversary wishes if she can help it; if she can convince the adversary that she's following its instructions while actually following the protocol using her secret input, she will.

Benaloh and Tuinstra [7] were the first to introduce this concept. Most papers concerning receipt-free voting (including Benaloh and Tuinstra), do not give a rigorous definition of what it means for a protocol to be receipt-free, only the intuitive one: "the voter should not be able to convince anyone else of her vote".

Canetti and Gennaro considered this problem for general multiparty computation (of which voting is a special case), and gave a formal definition of *incoercibility* [17]. Their definition is weaker than receipt-freeness, however: the adversary is only allowed to coerce a player after the protocol is complete (i.e., it cannot require a coerced player to follow an arbitrary strategy, or even specify what randomness to use). To reduce confusion, we refer to the property defined in [17] as *post-factum incoercibility*.

Juels, Catalano and Jakobsson also give a formal definition of *coercion-resistance* [47]. Their definition has a similar flavor, but is specifically tailored to voting in a public-key setting. It is stronger than receipt-freeness in that a coercion-resistant protocol must also prevent an *abstention-attack* (preventing a coerced voter from voting). However, this strong definition requires anonymous channels from voters to tallying authorities, otherwise an abstention-attack is always possible (this is because a coercer that can monitor non-anonymous communication to the tallying authorities can make sure voters do not communicate at all with the authorities, thus forcing them to abstain).

Our formalization of receipt-freeness is a generalization of Canetti and Gennaro's definition (and so can be used for any secure function evaluation), and is strictly stronger (i.e., any protocol that is receipt-free under our definition is post factum incoercible as well). The difference is the adversarial model we consider. Canetti and Gennaro only allow the adversary to query coerced players players after the protocol execution is complete. Each player has a *fake input* in addition to the real one (the one actually used in the protocol). When coerced by the adversary, the parties respond to queries by faking their view of the protocol, so that it will appear to the adversary that they used their fake input instead of their real one.

In our definition, players also have fake inputs in addition to the real ones. In contrast to the post factum incoercible model, the adversary can coerce players at any time during the protocol execution. A coerced player will use the fake input to answer the adversary's queries about the past view (before it was coerced). The adversary is not limited to passive queries, however. Once a player is coerced, the adversary can give it an *arbitrary strategy* (i.e. commands the player should follow instead of the real protocol interactions). We call coerced players that actually follow the adversary's commands "puppets".

A receipt-free protocol, in addition to specifiying what players should do if they are honest, must also specify what players should do if they are coerced; we call this a "coercion-resistance strategy" The coercion-resistance strategy is a generalization of the "faking algorithm" in Canetti and Gennaro's definition — the faking algorithm only supplies an answer to a single query ("what was the randomness used for the protocol"), while the coercion-resistance strategy must tell the party how to react to any command given by the adversary.

Intuitively, a protocol is receipt-free if no adversary can distinguish between a party with real input x that is a puppet and one that has a fake input x (but possibly a different real input) and is running the coercion-resistance strategy. At the same time, the computation's output should not change when we replace coerced parties running the coercion-resistance strategy with parties running the honest protocol (with their real inputs). Note that these conditions must hold even when the coercion-resistance strategy is known to the adversary.

Unfortunately, this "perfect" receipt-freeness is impossible to achieve except for trivial computations. This is because for any non-constant function, there must exist some party P_i and some set of inputs to the other parties such that the output of the function depends on the input used by x_i. If the adversary corrupts all parties except for P_i, it will be able to tell from the output of the function what input what used by P_i, and therefore whether or not P_i was a puppet.

This is the same problem faced by Canetti and Genaro in defining post factum incoercibility. Like theirs, our definition sidesteps the problem by requiring that any "coercion" the adversary can do in the real world it can also do in an ideal world (where the parties only interaction is sending their input to an ideal functionality that computes the function). Thus, before we give the formal definition of receipt-freeness, we must first describe the mechanics of computation in the ideal and real worlds. Below, f denotes the function to be computed.

4.5.1 The Ideal World

The ideal setting is an extension of the model used by Canetti and Genaro (the post factum incoercibility model). As in their model, there are n parties, $P_1, \ldots, P_n$, with inputs $x_1, \ldots, x_n$. Each party also has a "fake" input; they are denoted $x'_1, \ldots, x'_n$. The "ideal" adversary is denoted $\mathcal{I}$.

In our model we add an additional input bit to each party, $c_1, \ldots, c_n$. We call these bits the "coercion-response bits". A trusted party collects the inputs from all the players, computes $f(x_1, \ldots, x_n)$ and broadcasts the result. In this setting, the ideal adversary $\mathcal{I}$ is limited to the following options:

1. Corrupt a subset of the parties. In this case the adversary learns the parties' real inputs and can replace them with inputs of its own choosing.

2. Coerce a subset of the parties. A coercing party's actions depend on its coercion-response bit c_i. Parties for which $c_i = 1$ will respond by sending their real input x_i to the adversary (we'll call these "puppet" parties). Parties for which $c_i = 0$ will respond by sending the fake input x'_i to the adversary.

   At any time after coercing a party, the adversary can provide it with an alternate input x''_i. If $c_i = 1$, the coerced party will use the alternate input instead of its real one (exactly as if it were corrupted). If $c_i = 0$, the party will ignore the alternate input (so the output of the computation will be the same as if that party were honest). There is one exception to this rule, and that is if the alternate input is the special value $\perp$, signifying a forced abstention. In this case the party will use the input $\perp$ regardless of the value of c_i.

$\mathcal{I}$ can perform these actions iteratively (i.e., adaptively corrupt or coerce parties based on information gained from previous actions), and when it is done the ideal functionality computes the function. $\mathcal{I}$'s view in the ideal case consists its own random coins, the inputs of the corrupted parties, the inputs (or fake inputs) of the coerced parties and the output of the ideal functionality $f(x_1, \ldots, x_n)$ (where for corrupted and puppet parties x_i is the input chosen by the adversary).

Note that in the ideal world, the only way the adversary can tell if a coerced party is a puppet or not is by using the output of the computation – the adversary has no other information about the coercion-response bits.

4.5.2 The Real World

Our real-world computation setting is also an extension of the real-world setting in the post factum incoercibility model. We have n players, $P_1, \ldots, P_n$, with inputs $x_1, \ldots, x_n$ and fake inputs $x'_1, \ldots, x'_n$. The adversary in the real-world is denoted $\mathcal{A}$ (the "real" adversary).

The parties are specified by interactive Turing machines restricted to probabilistic polynomial time. Communication is performed by having special communication tapes: party P_i sends a message to party P_j by writing it on the (i, j) communication tape (we can also consider different models of communication, such as a broadcast tape which is shared by all parties). Our model does not allow erasure; communication tapes may only be appended to, not overwritten. The communication is synchronous and atomic: any message sent by a party will be received in full by intended recipients before the beginning of the next round.

We extend the post-factum incoercibility model by giving each party a private communication channel with the adversary and a special read-only register that specifies its corruption state. This register is initialized to the value "honest", and can be set by the adversary to "coerced" or "corrupted". In addition, each party receives the coercion response bit c_i. We can think of the ITM corresponding to each party as three separate ITMs (sharing the same tapes), where the ITM that is actually "running" is determined by the value of the corruption-state register. Thus, the protocol specifies for party P_i a pair of ITMs (H_i, C_i), corresponding to the honest and coerced states (the corrupt state ITM is the same for all protocols and all parties).

The computation proceeds in steps: In each step $\mathcal{A}$ can:

1. Corrupt a subset of the parties by setting their corresponding corruption-state register to "corrupted". When its corruption-state register is set to "corrupted", the party ouputs to the adversary the last

state it had before becoming corrupted, and the contents of any messages previously received. It then waits for commands from the adversary and executes them. The possible commands are:

- Copy to the adversary a portion of one of its tapes (input, random, working or communication tapes).
- Send a message specified by the adversary to some subset of the other parties.

These commands allow the adversary to learn the entire past view of the party and completely control its actions from that point on. We refer to parties behaving in this manner as executing a "puppet strategy".

2. Coerce a subset of the parties by setting their corresponding corruption-state register to "coerced". From this point on $\mathcal{A}$ can interactively query and send commands to the coerced party as it can to corrupted parties. The coerced party's response depends on its coercion-response bit c_i. If $c_i = 1$, the party executes the puppet strategy, exactly as if it were corrupted. If $c_i = 0$, it runs the coercion-resistance strategy C_i instead. The coercion-resistance strategy specifies how to respond to $\mathcal{A}$'s queries and commands.

3. Send commands to corrupted and coerced parties (and receive responses).

$\mathcal{A}$ performs these actions iteratively, adaptively coercing, corrupting and interacting with the parties. $\mathcal{A}$'s view in the real-world consists of its own randomness, the inputs, randomness and all communication of corrupted parties, its communications with the coerced parties and all public communication.

4.5.3 A Formal Definition of Receipt-Freeness

Definition 4.3. A protocol is receipt-free if, for every real adversary $\mathcal{A}$, there exists an ideal adversary $\mathcal{I}$, such that for any input vector $x_1, \ldots, x_n$, fake input vector $x'_1, \ldots, x'_n$ and any coercion-response vector $c_1, \ldots, c_n$:

1. $\mathcal{I}$'s output in the ideal world is indistinguishable from $\mathcal{A}$'s view of the protocol in the real world with the same input and coercion-response vectors (where the distributions are over the random coins of $\mathcal{I}$, $\mathcal{A}$ and the parties).

2. Only parties that have been corrupted or coerced by $\mathcal{A}$ in the real world are corrupted or coerced (respectively) by $\mathcal{I}$ in the ideal world.

It is important to note that even though a protocol is receipt-free by our definition, it may still be possible to coerce players (a trivial example is if the function f consists of the player's inputs). What the definition does promise is that if it is possible to coerce a party in the real world, it is also possible to coerce that party in the ideal world (i.e. just by looking at the output of f).

4.5.4 Receipt-Freeness of Our Voting Protocol

Coercion-Resistance Strategy. The voter begins by running the honest protocol. If at any point during the protocol the voter is coerced with fake input c', the voter simulates a protocol execution up to that point, using the c' as the input and same random choices made in the real execution. Any queries made by $\mathcal{A}$ about past history will be answered using the simulated transcript. $\mathcal{A}$ must require a coerced voter to send the following messages to the DRE, in the proper order (starting from the stage at which the voter was corrupted):

1. A candidate c'. In this case the voter sends her real input, c. The simulated transcript will be identical to the real one, except for the voter sending c' instead of c.

2. A subset $R_v \subset [(m+2)k]$. The voter sends R_v. The simulated transcript is identical to the real one.

3. For every possible candidate c'', a k bit string $b_{c''}$.

Case 1 $c'' = c$ (the voter's real choice). Here what really happens is that the DRE sends k (**Committed**, $v, v_i^{(c)}$) messages. The voter then sends b_c and receives b_c back from the DRE. In the simulated transcript, however, the voter first sends b_c and only then receives the k (**Committed**, $v, v_i^{(c)}$) messages and b_c.

Case 2 $c'' = c'$ (the voter's fake choice). What really happens is that the voter first sends $b_{c'}$ and only then receives the k (**Committed**, $v, v_i^{(c')}$) messages and $b_{c'}$. In the simulated transcript, however, the voter first receives k (**Committed**, $v, v_i^{(c')}$) messages, then sends $b_{c'}$, then receives $b_{c'}$ back from the DRE. Note that the simulated transcript will contain the receipt of the k (**Committed**, $v, v_i^{(c')}$) messages *before* $\mathcal{A}$ asks the voter to send anything — this will be a simulated response what $\mathcal{A}$ required the voter to transmit in the previous step.

Case 3 $c'' \neq c' \wedge c'' \neq c$. Here the voter can simply follow the instructions of $\mathcal{A}$.

If $\mathcal{A}$ diverges from the sequence outlined above, the voter halts (abstains). Since we do not assume voters have access to secure computers, our coercion-resistance strategy must be simple enough to be executed by an unaided human. Note that while we formally require the voter to "simulate" the fake transcript, it is almost identical to the real transcript, except at the points where the voter uses her real input instead of the one provided by the adversary.

Assumptions. Note that while Canetti's composition theorem allows us to prove UC security in the hybrid model (replacing a protocol that UC-realizes Commit-with-Copy by its ideal functionality), the theorem does not apply to receipt-freeness. To prove that our protocol is receipt-free, we must "open the box" and consider the implementation of $\mathcal{F}^{(C\&C)}$.

The receipt-freeness of our protocol relies on the fact that the voter's view of the protocol between steps 7 and 9 of the "real proof" stage in the ballot casting phase (Protocol 4.1a) is a deterministic function of her previous view (i.e., the DRE's commitment cannot add any entropy to the voter's view). While the UC-definition of $\mathcal{F}^{(C\&C)}$ does indeed meet this criterion (the messages received are all of the form (**Committed**, r, r'), where r, r' can be computed in advance by the voter), a real implementation of $\mathcal{F}^{(C\&C)}$ will probably not work this way (since the commitment will contain at least some random-looking string). To prevent this, we add a "physical commitment" to the protocol. This commitment can be implemented, for example, by placing an opaque shield on part of the DRE printer's output, so that the voter can see that something has been printed, but not what. The physical commitment is only required between the steps 7 and 9 of the "real proof" stage. We require that the implementation of $\mathcal{F}^{(C\&C)}$ work even with such a shield in place (i.e., if the **LinkedCopy** or **FakeCopy** commands are interactive, the voter's messages cannot depend on the DRE's messages.

We also require that, for any a, a', i, s, s', the transcripts of **FakeCopy** a, a', s, **FakeCopy** a, a', s' and **LinkedCopy** a, a', i commands be indistinguishable and that the transcripts of **SrcCommit** a, b, F and **SrcCommit** a, c, F' commands be indistinguishable for any a, b, c, F, F'. This is already guaranteed by the fact that a protocol UC-realizes $\mathcal{F}^{(C\&C)}$.

Ideal Simulator. For the Ballot Casting phase, $\mathcal{I}$ simulates the DRE as well as any honest or coerced voters. It uses a fixed value q as the input for all honest and coerced voters (it doesn't know the real inputs). When $\mathcal{A}$ corrupts or coerces a voter, $\mathcal{I}$ also corrupts or coerces the ideal voter. and internally reruns the simulation of that voter's session, using the same random choices it previously made and the voter's real input (or the supplied fake input in case of a coerced voter). When $\mathcal{A}$ sends the fake candidate c' in step 1 of the coercion-resistance strategy, $\mathcal{I}$ gives c' as an alternate input to the corresponding coerced ideal voter. If $\mathcal{A}$ causes one of the coerced voters to abort, $\mathcal{I}$ changes the corresponding ideal voter's input to $\perp$. Note that $\mathcal{I}$ is behaving exactly as would the ideal adversary in the UC security proof for honest and corrupt voters.

We only allow the adversary to coerce or corrupt voters (once the DRE is corrupted, we no longer guarantee secrecy of the votes, much less receipt-freeness). Therefore, given the tally (which $\mathcal{I}$ received from $\mathcal{F}^{(V)}$), $\mathcal{I}$ can simulate the Final-Tally and Universal Verification phases completely ($\mathcal{A}$ cannot do anything in these stages, since voters do not participate).

Proof of Indistinguishability. We make use of the fact that our protocol UC-realizes $\mathcal{F}^{(V)}$ in proving that the output of the simulation is indistinguishable from the real adversary's view. A key point is that the visible transcript of a voter's coercion-resistance strategy is always identical to the real transcript *no matter what the voter's real input is.*

In the ideal world, the adversary cannot, by manipulating corrupt voters, cause $\mathcal{F}^{(V)}$ to output something different from the case where the corrupt voters just vote. Thus, these two cases must also be indistinguishable in the real world (where we run the protocol). Also, note that coerced voters run a coercion-resistance strategy that can be simulated by an adversary using only public information, *and they do not change their actual input.* Therefore, the transcript of communication with both coerced and corrupted voters cannot allow the adversary to distinguish between the real-world and simulated transcripts in the Final-Tally and Universal Verification phases, or transcripts for honest voters in the Ballot-Casting phase.

It remains to show that the adversary cannot distinguish between the real-world and simulated transcripts of coerced voters in the Ballot Casting phase (corrupted voters are simulated exactly, so the transcripts are identically distributed). Since $\mathcal{I}$ simulates the DRE and coerced voter exactly, any difference between the simulated and real world must be due to the difference in the simulated party's input. However, the public transcript of the coercion-resistance strategy is indistinguishable for any two inputs (since the only messages appearing in the public transcript that may contain information about the input are the commitment messages sent by the DRE, and by the hiding property of $\mathcal{F}^{(C\&C)}$ they must be indistinguishable). Furthermore, the simulated transcript is identical to the real transcript had the voter actually used the input given by the adversary.

4.6 Proof of Accuracy and Privacy (Theorem 4.1)

We prove our voting scheme securely realizes the voting functionality $\mathcal{F}^{(V)}$. Since the $\mathcal{F}^{(V)}$ functionality provides both accuracy and privacy, we can conclude our voting scheme has these properties as well. The proof is in the UC framework.

Briefly, the UC framework considers two "worlds": an "ideal world", in which the ideal functionality exists, the honest parties send their inputs to the ideal functionality and output whatever they receive from it, and the "real world", where the $\mathcal{F}^{(C\&C)}$ functionality exists and parties behave according to the protocol. In both worlds, there exists an adversarial *environment machine*, $\mathcal{Z}$, that specifies the inputs for the honest parties and receives their outputs. In the real world, there exists a *real adversary*, $\mathcal{A}$, that is controlled by $\mathcal{Z}$ (it sends any messages it receives to $\mathcal{Z}$, and performs any actions specified by $\mathcal{Z}$).

To prove a protocol realizes a given ideal functionality, we must describe an *ideal simulator*, $\mathcal{I}$, for the ideal world, that simulates the view of $\mathcal{A}$, and show that the view of any environment machine running with $\mathcal{I}$ in the ideal world is indistinguishable from its view when running with $\mathcal{A}$ in the real world.

The idea behind the simulation run by $\mathcal{I}$ is very simple. $\mathcal{I}$ begins by "guessing" the inputs for all of the honest parties. It then simulates $\mathcal{A}$, $\mathcal{F}^{(C\&C)}$ and the honest parties exactly according to the protocol, maintaining a "provisional view" for the simulated honest parties. Whenever $\mathcal{I}$ learns information that contradicts its guesses (e.g., if an honest party is corrupted), it "rewrites" the provisional view of all the affected parties using this new information. The protocol is constructed so that this rewriting can always be done in a way that is consistent with $\mathcal{Z}$'s view of the protocol up to that point.

The indistinguishability follows from the fact that the environment's view contains no information about honest parties' votes (beyond the final tally), and the fact that a cheating DRE will remain undetected with negligible probability (in the security parameter k).

4.6.1 The Ideal World Simulation

In the more detailed description of the simulation below, we focus on the points at which $\mathcal{I}$ deviates from just simulating the parties according to the protocol. We assume the following throughout the simulation:

- Whenever a corrupt voter is instructed by $\mathcal{A}$ to send "bad" messages (syntactically incorrect) to an honest DRE, $\mathcal{I}$ treats this as an abstention.

- If the DRE is corrupt and instructed to send bad messages to an honest voter or to the bulletin board, $\mathcal{I}$ sends the **Halt** command to $\mathcal{F}^{(V)}$ and halts the simulation.

We now describe $\mathcal{I}$'s actions in each phase of the protocol. These actions depend in each stage on which of the parties are corrupt at that point.

Ballot Casting

Below we describe the simulation for voter v. $\mathcal{I}$ begins simulation of this phase if voter v is corrupt and $\mathcal{A}$ instructs the voter to cast a ballot, or if voter v is honest and a $(\textbf{Voted}, v)$ or $(\textbf{Voted}, v, x_v)$ message is received from $\mathcal{F}^{(V)}$.

Case 1: Both v and the DRE are honest ($\mathcal{F}^{(V)}$ broadcast $(\textbf{Voted}, v)$). In this case, $\mathcal{A}$ only receives the output from the broadcast channel, which depends only on v's random challenges (and so will be consistent with any replay of the simulation as long as $\mathcal{I}$ uses the same random challenges). The output on the broadcast channel consists of:

(a) a $(\textbf{SrcCommitted}, v, \ell)$ message from $\mathcal{F}^{(C\&C)}$

(b) the random subset R_v

(c) The "proofs" that v is a commitment to the correct candidate, for each candidate c:

i. $(\textbf{Committed}, v, v_i^{(c)})$ for $1 \leq i \leq k$

ii. The challenge $r_v^{(c)}$

iii. k messages from $\mathcal{F}^{(C\&C)}$ of the form $(\textbf{Opened}, v_i^{(c)}, c)$ or $(\textbf{CopyOf}, v_i^{(c)}, v\bar{R}_v^{(c,i)})$ depending on the bits of $r_v^{(c)}$.

Case 2: v is corrupt and the DRE is honest ($\mathcal{A}$ instructed v to begin the voting phase and send x_v to the DRE). In this case, $\mathcal{I}$ sends a **Vote** x_v command to $\mathcal{F}^{(V)}$ on behalf of v, and simulates an honest DRE exactly according to protocol.

Case 3: v is honest and the DRE is corrupt ($\mathcal{F}^{(V)}$ sent $(\textbf{Voted}, v, x_v)$ to $\mathcal{I}$). If v's input in the provisional view is not x_v, $\mathcal{I}$ rewrites the provisional view to make it so. Note that since the simulated v has sent no messages yet, this does not change the environment's view of the protocol so far. $\mathcal{I}$ can then simulate the honest voter exactly according to protocol.

Case 4: Both v and the DRE are corrupt. In this case $\mathcal{I}$ already knows x_v, the real input of voter v, and sends a **Vote** x_v command to $\mathcal{F}^{(V)}$ on behalf of v. $\mathcal{I}$ simulates both parties acting according to $\mathcal{A}$'s directions. Note that $\mathcal{A}$ may instruct the DRE to send a **SrcCommit** command to $\mathcal{F}^{(C\&C)}$ for a value other than x_v or to send a **BadCommit** command instead. In this case, $\mathcal{I}$ may be required to change v's vote at the beginning of the tally phase.

Since we allow the adversary to be adaptive, we must also consider corruptions that occur during the ballot casting phase:

1. $\mathcal{A}$ corrupts the DRE. In this case $\mathcal{I}$ also corrupts the DRE in the ideal world and receives from $\mathcal{F}^{(V)}$ all the votes cast so far. If the votes cast by honest voters are different than $\mathcal{I}$'s guesses, $\mathcal{I}$ rewrites the provisional view using the new information. Note that this does not change any message previously seen by the environment, since the only messages affected are those between honest voters, the honest DRE and the commitment functionality.
2. $\mathcal{A}$ corrupts voter v. In this case, $\mathcal{I}$ also corrupts voter v and learns its input x_v. If the DRE is honest, v has already voted and x_v differs from $\mathcal{I}$'s guess, $\mathcal{I}$ rewrites its provisional view to take into account the new information; since both v and the DRE were previously honest, this rewritten view will still be consistent with the environment's view. If the DRE was already corrupt, then either v has not yet voted, in which case $\mathcal{I}$'s simulation so far did not depend on x_v, or v has already voted, in which case $\mathcal{I}$ already learned x_v. Hence, $\mathcal{I}$ will not have to rewrite the provisional view at this point.

Final Tally

This phase begins after the voters have all cast their ballots (or abstained). If the DRE is corrupt before or during publication of the final tally (step 2 in Protocol 4.2), $\mathcal{I}$ may have to alter some of the votes cast by previously corrupted voters. If the corrupt voters' votes can be changed so that the tally matches the one announced by the DRE, $\mathcal{I}$ sends the correponding **ChangeVote** commands to $\mathcal{F}^{(V)}$ on behalf of the corrupt voters.

If not, $\mathcal{I}$ continues the simulation without sending **ChangeVote** commands. In this case, the output of the verifier in the ideal world may differ from its output in the real world. However, we show that both the simulated and real-world verifiers would abort in this case with overwhelming probability (in Claim 4.5).

If the DRE is corrupted during the final tally phase (or if the DRE is honest, and one or more honest voters are corrupted), $\mathcal{I}$ learns the real inputs of all the honest voters (in the former case) or the newly corrupted voters (in the latter case). $\mathcal{I}$ rewrites the provisional view to be consistent with the new information, and continues with the simulation exactly following the protocol. The rewriting in this case may require $\mathcal{I}$ to change the values of its secret random coins, depending on what the simulated (honest) DRE has already published:

Case 1: If the shuffled votes have not yet been published (step 2 in Protocol 4.2), $\mathcal{I}$ is only required to rewrite messages between the honest voters and the DRE, and between the DRE and $\mathcal{F}^{(C\&C)}$ (it can leave its secret coins unchanged).

Case 2: If the shuffled votes have been published, but the beacon's random challenge bits have not yet been received (step 10 in Protocol 4.2), $\mathcal{I}$ rewrites the permutation π_0 so that the shuffled votes are consistent with the inputs (note that there must exist such a valid permutation, since the DRE was honest up to this point).

Case 3: If the beacon's random challenge bits have already been received, $\mathcal{I}$ rewrites π_i for all i such that $b_i = 1$ (i.e., those permutations that will remain secret), in addition to π_0.

Universal Verification

In this stage only the verifier participates, and cannot be corrupted. The verifier runs a deterministic algorithm on the information from the public broadcast channel. $\mathcal{I}$ simulates this algorithm. If verification fails $\mathcal{I}$ sends the **Halt** command to $\mathcal{F}^{(V)}$.

4.6.2 Indistinguishability of Views

It remains to show that no environment machine can distinguish between the the real world and the ideal world in which $\mathcal{I}$ runs the simulation detailed above.

The views of the different parties in the real world are:

Verifier: The verifier's view consists of all the published information:

1. all the messages sent by $\mathcal{F}^{(C\&C)}$
2. all the messages sent by the random beacon (challenge bits $b_1, \ldots, b_k$ and challenge permutations $\sigma_1, \ldots, \sigma_n$).
3. the voters' random coins (consisting of the set R_v and the challenges $r_v^{(1)}, \ldots, r_v^{(m)}$)
4. The messages published by the DRE.

Voter v: v's view consists of:

1. The input x_v.
2. The voter's random coins
3. The verifier's view

DRE: The DRE's view consists of:

1. The DRE's random coins (consisting of the permutations $\pi_0, \ldots, \pi_k$)
2. The views of all the voters (inputs and random coins).
3. The verifier's view

The environment's view in the real-world consists of the inputs of all the voters, the verifier's view and the views of all the corrupted parties.

Up to the final tally phase the environment's view of $\mathcal{I}$'s simulation is identical to its view in the real world. This is because the only difference between $\mathcal{I}$'s simulation and the real world is the inputs of the honest voters that $\mathcal{I}$ guesses incorrectly (note that $\mathcal{I}$ maintains, throughout the simulation, a provisional view that is consistent with the real world up to the unknown inputs of the honest voters at every point in the simulation).

If the DRE is honest, the environment's view consists only of the views of the verifier and the corrupt voters – these views are independent of the honest voters' inputs (since they can be deterministically computed from the random coins of the voters and the inputs of the corrupt voters). If the DRE is corrupt, $\mathcal{I}$ knows the inputs of the honest voters, hence its simulation is perfect.

In the final tally phase, $\mathcal{I}$'s simulation is also perfect unless the output of $\mathcal{F}^{(V)}$ does not match the tally output by the simulated verifier. If the DRE is honest, then this output is always identical. $\mathcal{I}$ sends a **Halt** command to $\mathcal{F}^{(V)}$ when the simulated verifier would detect the DRE cheating, so in these cases the output is identical as well. The only way the output can be different is if $\mathcal{F}^{(V)}$ outputs one tally while the simulated DRE outputs a different tally. The simulated DRE must send a **SrcCommit** v, c, F or **BadCommit** v, Q, ℓ command for each voter in the Ballot Casting phase (otherwise verification will always fail in the universal verification phase, in which case $\mathcal{I}$ will send a **Halt** command to $\mathcal{F}^{(V)}$ as well). If the tally output in the simulation differs from that output by $\mathcal{F}^{(V)}$, the DRE must have been corrupt at the time it announced the final tally, and there cannot exist an assignment of votes to the corrupt voters which could account for the tally announced by the DRE (otherwise $\mathcal{I}$ would have changed the votes sent to $\mathcal{F}^{(V)}$). Thus, one of the following must be the case:

Case 1: *For at least one honest voter v, the DRE sent a **SrcCommit** v, x', F such that $x' \neq x_v$, or a **BadCommit** v, Q, ℓ command in which less than $\frac{7}{8}$ of values in Q are x_v.* By Claim 4.4, the probability that the DRE does this but is not detected by the verifier is less than $2^{-\Omega(k)}$. Since $\mathcal{I}$ simulates an honest voter and the verifier exactly, both the simulated and real verifiers would abort with probability $1 - 2^{-\Omega(k)}$ in this case.

Case 2: *For all honest voters, at least $\frac{7}{8}$ of the commitments in the Ballot Casting phase are for their chosen candidate.* Denote M'' the matrix to which the DRE commits in the final tally phase (as defined in Section 4.4.2; this would be the lower right-hand matrix in Figure 4.4.2). For clarity, we will say "the i^{th} copy of v" rather than "the copy whose index is $R_v^{(i)}$".

Call the pair of rows $i, k+i$ in matrix M'' *valid* if they are constructed as required by the protocol: there exists a permutation π_i such that the j^{th} element of row i in M'' is the $\sigma_{\pi_i(j)}(i)$ copy of $\pi_i(j)$ and the j^{th} element of row $k+i$ is the $k+i$ copy of $\pi_i(j)$ (the element $\boxed{j,i}$ is considered the t^{th} copy of v if the response of $\mathcal{F}^{(C\&C)}$ to a **ProveCopy** $\boxed{j,i}$ command would be a (**CopyOf**, $\boxed{j,i}$, v, t) message; this means rows containing "bad" copies created using the **BadCommit** command are also considered valid). Again we have two possible cases:

Case 2.1: *The matrix committed to by the DRE contains more than $\frac{1}{8}k$ invalid row pairs.* The beacon chooses a random subset of the row pairs to test for validity: for every i such that $b_i = 0$, the DRE must prove that the row pair $i, k+i$ is valid by issuing the appropriate commands to $\mathcal{F}^{(C\&C)}$. If an invalid row pair is chosen for verification of validity, it will be detected with probability 1. Since $b_1, \ldots, b_k$ are i.i.d. variables and $\Pr[b_i = 0] = \frac{1}{2}$, the Chernoff bound implies that $\frac{1}{8}k$ invalid row pairs will escape detection with probability at most $2^{-\Omega(k)}$.

Case 2.2: *The matrix committed to by the DRE contains at least $\frac{7}{8}k$ valid row pairs.* By Claim 4.5, the probability that there is a mismatch between the simulated and ideal worlds is bounded by $2^{-\Omega(k)}$.

The case analysis above shows that the probability of each of the possible "bad events" (those in which there is an inconsistency between the real and ideal worlds) is bounded by $2^{-\Omega(k)}$. Since there are a constant number of such events, by the union bound, the statistical difference between the environment's view in the real and ideal world is at most $2^{-\Omega(k)}$.

In order to complete the proof of Theorem 4.1, it remains to prove the claims below.

Claim 4.4 (Probability of catching a DRE making bad commitments to honest voters)**.** *If, in the Ballot Casting phase, the DRE makes more than $\frac{1}{4}$ of the copies bad (i.e., that open to something other than the voter's choice), it will be detected with probability at least $1 - 2^{-\Omega(k)}$ in the universal verification phase.*

Proof. In the proof stage, with probability $1 - 2^{-\Omega(k)}$ the honest voter requires the DRE to prove for a random $\frac{1}{4}(m+1)k$ of the commitments that they are correct copies (since the voter decides independently with probability $\frac{1}{2}$ whether to require a copy or an open test for each of the $(m+1)k$ commitments, by the Chernoff bound the probability that the voter performs the copy test on less than $\frac{1}{4}(m+1)k$ is bounded by $2^{-\Omega(k)}$). One of the following must be the case:

Case 1: There are more than $\frac{1}{8}(m+1)k$ elements that are fake copies (they were generated by a **FakeCopy** command and will fail the copy proof test). The probability that none will be selected for the copy proof test is bounded by $(\frac{1}{2})^{\frac{1}{8}(m+1)k} = 2^{-\Omega(k)}$.

Case 2: At least $\frac{7}{8}$ of the commitments are linked copies. In this case, since more than $\frac{1}{4}$ of the copies are bad, at least $\frac{1}{8}$ of the linked commitments must be bad copies. The voter selects a random subset of size $\frac{1}{4}k$ for the open test with probability at least $1-2^{-\Omega(k)}$ (again by the Chernoff bound). Each commitment selected is a bad copy with probability at least $\frac{1}{8}$ (the events are not independent, but each "good" selection only increases the probability that the next selection will be bad, so they dominate a series of independent events). Thus, the probability that none of the selected commitments is bad is at most $(\frac{7}{8})^{\frac{1}{4}k}$.

By the union, bound, the probability that the DRE will not be caught is bounded by $2^{-\Omega(k)}$). □

Claim 4.5 (Probability of catching a cheating DRE announcing an incorrect tally)**.** *Denote M the matrix committed to by the DRE before any permutations have been applied (i.e., column v of M consists of $2k$ commitments to x_v if the DRE issued a **SrcCommit** v, x_v, F command in the ballot casting phase, or of commitments to the values specified in Q if it issued a **BadCommit** v, Q, ℓ command).*

For any adversary $\mathcal{A}$ in the real world, if for every honest voter v, at least $\frac{7}{8}$ of the commitments in column v of M are commitments to x_v ("good commitments") and at least $\frac{7}{8}$ of the row pairs in M'' are valid (M'' is the actual matrix committed to by the DRE, after rows and columns of M are permuted), then the probability that the DRE outputs a "bad" tally and is not detected by the verifier is bounded by $2^{-\Omega(k)}$ (a tally is "bad" if there does not exist an assignment of inputs to the corrupt voters that would result in that tally).

Proof. For any k, n and $2k \times n$ matrix X, let $top(X)$ be the $k \times n$ matrix consisting of the first k rows of X, and $bot(X)$ the $k \times n$ matrix consisting of the last k rows of X (i.e., row i of $bot(X)$ is row $k+i$ of X).

We can think of the adversary, $\mathcal{A}$, as playing the following game:

1. $\mathcal{A}$ selects the matrix M, under the sole constraint that each column v corresponding to an honest voter contains at least $\frac{7}{8}$ "good" commitments (to x_v).
2. $\mathcal{A}$ receives the permutations $\sigma_1, \ldots, \sigma_n$ from the random beacon.
3. Possibly depending on $\sigma_1, \ldots, \sigma_n$, $\mathcal{A}$ selects at least to $\frac{7}{8}k$ row pairs that will be marked valid.
4. $\mathcal{A}$ receives the bits $b_1, \ldots, b_k$ from the random beacon.

The adversary "wins" if all the valid row pairs for which $b_i = 1$ (i.e. that the adversary will be required to open) have the same tally (after the columns in $top(M)$ are permuted using $\sigma_1, \ldots, \sigma_n$). Note that the permutations $\pi_1, \ldots, \pi_k$ that are chosen by the adversary in Protocol 4.2 are used only to maintain voter privacy and have no effect on the adversary's cheating probability.

We can assume w.l.o.g. that every row of $bot(M)$ for which $b_i = 1$ has at least one "bad" commitment for an honest voter. If not, then either the opened tallies do not match each other (in which case both the real and simulated verifiers will abort), or the tally announced by the DRE is consistent with the one output by $\mathcal{F}^{(V)}$, since all honest voters have good commitments. Denote $j_i \in [n]$ the column index of the first bad commitment in row i. We'll call these commitments the "token" bad commitments.

Since the b_is are i.i.d. with expectation $\frac{1}{2}$, by the Chernoff bound $\Pr\left[\sum_{i=1}^{k} b_i < \frac{1}{4}k\right] < 2^{-\Omega k}$. Assume that $\sum_{i=1}^{k} b_i \geq \frac{1}{4}k$ (i.e., there are at least $\frac{1}{4}k$ indices i such that $b_i = 1$). Denote $B \doteq \{i \mid b_i = 1\}$ the set of opened rows.

In order for $\mathcal{A}$ to "win", the commitments $bot(M)_{i,j_i}$ and $top(M)_{\sigma_{j_i}^{-1}(i),j_i}$ must be to the same value for every $i \in B$. In particular, this means the commitment $top(M)_{\sigma_{j_i}^{-1}(i),j_i}$ must be bad for $\mathcal{A}$ to win. For all $i \in B$, let I_i be a random variable indicating that this is not the case (i.e. $I_1 = 1$ if $top(M)_{\sigma_{j_i}^{-1}(i),j_i}$ is a commitment to x_v and $I_i = 0$ otherwise). If $\sum_{i\in B} I_i > \frac{1}{8}k$, then both the real and simulated verifier will always abort. This is because, no matter how $\mathcal{A}$ chooses the $\frac{1}{8}k$ invalid row pairs, there will remain at least one valid row pair for which $I_i = 1$, i.e., there is a valid row pair with a mismatch between the two rows that will be detected by the verifier.

We claim that $\Pr\left[\sum_{i\in B} I_i \leq \frac{1}{8}k\right] < 2^{-\Omega(k)}$ (where the probability is over the choice of $\sigma_1, \ldots, \sigma_k$).

We assume that for every honest voter v at least $\frac{7}{8}$ of the entries in column v of M are commitments to x_v ("good" commitments). Hence, we can assume w.l.o.g. that $top(M)$ has at least $\frac{7}{8}k$ good commitments in every column that corresponds to an honest voter. If this is not the case then $bot(M)$ has this property; then we can consider the equivalent game in which we exchange $bot(M)$ and $top(M)$ and the permutations $\sigma_1, \ldots, \sigma_n$ by their inverses.

For every honest voter v consider the following algorithm for producing the random permutation σ_v:

1: Initialize $S \leftarrow \{1, \ldots, k\}$
2: **for** $1 \leq i \leq k$ **do**
3: Choose a random index $j \in_R S$
4: Let $S \leftarrow S \setminus \{j\}$ {Remove j from S}
5: Set $\sigma_v(i) \leftarrow j$
6: **end for**

Let $\{I'_i\}_{i\in B}$ be i.i.d. binary random variables such that $\Pr[I'_i = 1] = \frac{5}{8}$. Then $\Pr[\sum I_i \leq \frac{1}{8}k] \leq \Pr\left[\sum I'_i \leq \frac{1}{8}k\right]$. This is because even if we remove the indices of $\frac{1}{4}k$ good commitments from the set S, there still remain at least $\frac{7}{8}k - \frac{1}{4}k = \frac{5}{8}k$ indices of good commitments in S, hence for each of the first $\frac{1}{4}k$ rows of every column, the element mapped to that row is a good commitment with probability at least $\frac{5}{8}$ — even conditioned on the elements mapped to all previous rows.

By the Chernoff bound, $\Pr\left[\sum_{i\in B} I'_i \leq \frac{1}{8}k\right] \leq 2^{-\Omega(k)}$, hence we can conclude that $\Pr\left[\sum_{i\in B} I_i \leq \frac{1}{8}k\right] < 2^{-\Omega(k)}$. □

4.7 Basing Commit-and-Copy on Standard Commitment

In this section we give a protocol that implements the Commit-and-Copy functionality using any UC commitment scheme.

4.7.1 Protocol Description

The protocol implements $\mathcal{F}^{(C\&C[k])}$ between two players, the committer and the receiver, based on standard UC commitment (we denote the standard commitment functionality $\mathcal{F}^{(C)}$).

To clarify the presentation, we informally describe the construction in three stages, first giving a simpler (but flawed) construction that works for non-interactive commitments, then fixing the flaw, and finally describing the extension to any UC commitment. A formal specification is given in Protocol 4.3, composed of Protocols 4.3a (from the committer's side) and Protocol 4.3b (from the receiver's side)

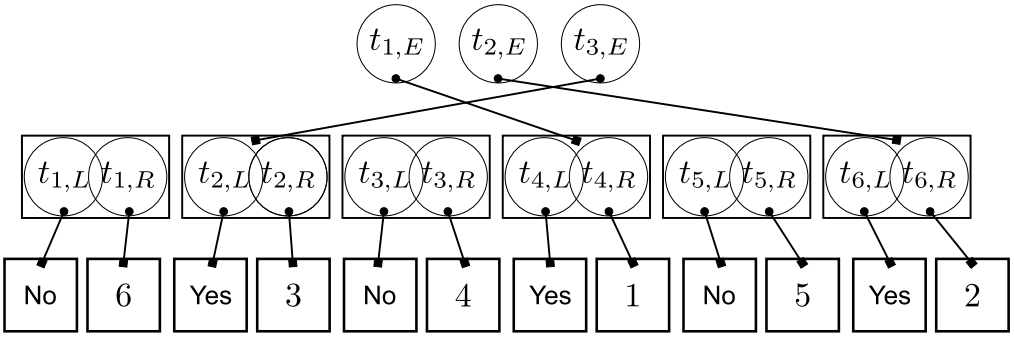

Figure 4.7.1: **SrcCommit** r, **"Yes"**, (**"No"**, **"No"**, **"No"**)

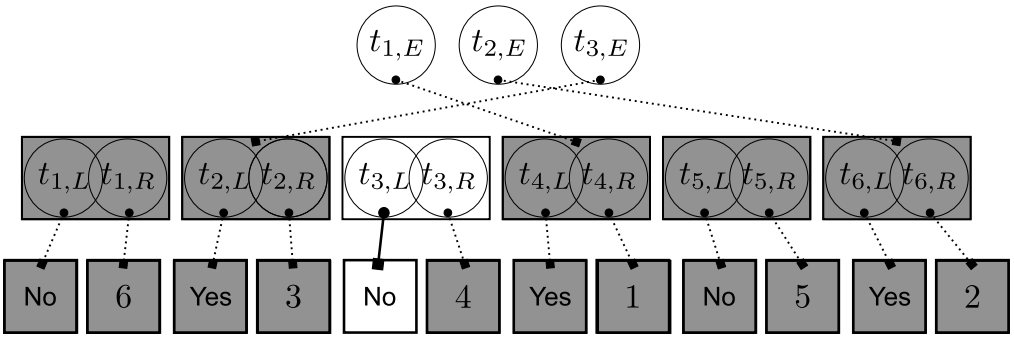

Figure 4.7.2: **Open**

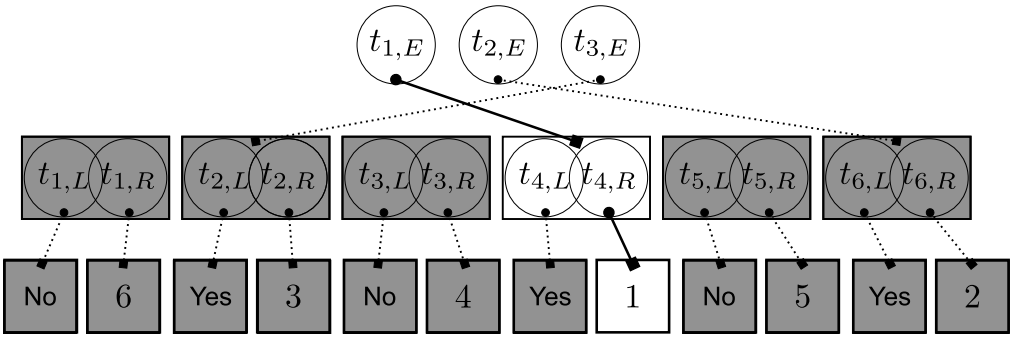

Figure 4.7.3: **ProveCopy**

The main idea behind the construction is to use nested commitments: commitments to commitments. For non-interactive commitments, a commitment to a value is simply a string. A commitment to this string is a nested commitment to the value. In our simple (but flawed) construction, the "source" commitment to a value v is a vector of k nested commitments to the value v. Call the actual commitments to v the "internal" commitments, and the nested commitments the "external" commitments. A **LinkedCopy** r, r', i command for this construction consists of sending the i^{th} internal commitment; Since the receiver only sees the external commitments, this does not reveal the index i. In fact, the receiver can't tell that a given commitment is really an internal commitment (unless the committer opens the corresponding external commitment). Thus, the **FakeCopy** r, r', s command consists of sending a new commitment to s'. Opening a copy (linked or fake) is done by opening the internal commitment; this reveals the value, but not whether the copy was linked or fake. To execute the **ProveCopy** r' command, the committer opens the external commitment corresponding to the internal commitment with tag r'.

Given non-interactive commitments, this construction almost works, but does not fully realize the functionality $\mathcal{F}^{(C\&C[k])}$. The flaw is that an adversary can generate multiple external commitments to the same internal commitment. This allows the adversary to choose, when executing the **ProveCopy** command, which index to reveal (rather than being committed to a specific index for each copy at the time of the **SrcCommit** command, as required by $\mathcal{F}^{(C\&C[k])}$). To overcome this flaw, we modify the protocol slightly: each internal commitment is replaced by a commitment *pair*: a "left" internal commitment, which is a commitment to the actual value, and a "right" internal commitment, which is a commitment to the index. The external commitment is now a commitment to the left/right pair. To execute the **Open** command, the committer opens only the left internal commitment. To execute the **ProveCopy** command, the committer opens both the external and the right internal commitment (but not the left internal commitment). The receiver can now verify that the index of the external commitment is consistent with the right internal commitment.

The final modification of the construction is to allow it to work with any (possibly interactive) UC commitment. In this case, we can no longer think of the commitment as a string. There are two main differences between the notion of non-interactive commitment required to make the above construction work and the one guaranteed by the ideal commitment functionality, $\mathcal{F}^{(C)}$: one is that $\mathcal{F}^{(C)}$ notifies the receiver of the tag for any commitment made (so we can't keep the internal commitments completely private), and the second is that the commitment tag can be chosen arbitrarily by the committer (so we can't postpone the internal commitments to when we need to make a copy). We overcome this by having the committer commit ahead of time to all possible internal commitments (including fake commitments) in a random order. Thus, the internal commitment tags don't give any information about the type of copy or its index. Figure 4.7.1 shows an example of the commitments generated for a **SrcCommit** r, **"Yes"**, (**"No"**, **"No"**, **"No"**) command (with $k = 3$). The circles denote commitment tags, and the squares surround the corresponding commitment values. Figure 4.7.2 shows how an **Open** command would look to a receiver. The dashed lines signify commitments that haven't been opened (the receiver doesn't "see" those). Figure 4.7.3 shows the same for a **ProveCopy** command (**LinkedCopy** and **FakeCopy** commands consist of sending the index of one of the internal commitment pairs).

In the remainder of the section we prove:

Theorem 4.6. *Protocol 4.3 UC-realizes* $\mathcal{F}^{(C\&C[k])}$

The proof of the theorem is in two parts: a description of the ideal-world simulator, $\mathcal{I}$, and a proof that the environment's view in the real-world is indistinguishable from its view in the ideal world.

4.7.2 The Ideal-World Simulation

The ideal simulator for this protocol uses the same basic framework as the simulator for the voting protocol (in Section 4.6.1): $\mathcal{I}$ maintains a "provisional view" for the honest parties, and rewrites the view if it learns new, conflicting, information. $\mathcal{I}$ simulates $\mathcal{F}^{(C)}$ and honest parties exactly except for rewriting the provisional view (in that case, the rewritten view will also be the result of a correct simulation, but with possibly different inputs and randomness). If both parties are honest, $\mathcal{I}$ does not perform any action until one of the parties is corrupted by the adversary (the adversary cannot see messages passing between the two honest parties). When a party is corrupted by $\mathcal{A}$, $\mathcal{I}$ corrupts the corresponding ideal party. The corrupt

Protocol 4.3a $\mathcal{F}^{(C\&C[k])}$ from $\mathcal{F}^{(C)}$ (Committer)

SrcCommit r, s, F Let $\ell = |F| + k$ and denote $F = (s'_1, \ldots, s'_{\ell-k})$. Note: the commitment tags $t_{i,X}$ (for $X \in \{L, R, E\}$) consist of the string $[r; i, X]$.

1: Send a (**SrcCommitting**, r, ℓ) message to the receiver.
2: Choose a random permutation $\pi_r \colon [\ell] \mapsto [\ell]$.
3: Let $x_1, \ldots, x_\ell$ be the vector consisting of k copies of s followed by the elements of F. (i.e., $x_i = s$ for $1 \le i \le k$ and $x_i = s'_{i-k}$ for $k < i \le \ell$).
4: **for** $1 \le i \le \ell$ **do** {Generate the "internal" commitments in a random order}
5: Send a **Commit** $t_{i,L}, x_{\pi_r^{-1}(i)}$ command to $\mathcal{F}^{(C)}$. {"Left" internal commitment: to value}
6: Send a **Commit** $t_{i,R}, \pi_r^{-1}(i)$ command to $\mathcal{F}^{(C)}$. {"Right" internal commitment: to index}
7: **end for**
8: **for** $1 \le i \le k$ **do** {Generate the "external" commitments}
9: Send a **Commit** $t_{i,E}, [t_{\pi_r^{-1}(i),L}; t_{\pi_r^{-1}(i),R}]$ command to $\mathcal{F}^{(C)}$.
10: **end for**

LinkedCopy r, r', i $(1 \le i \le k)$

1: Send (**Copy**, $r, r', \pi_r(i)$) to the receiver.

FakeCopy r, r', s (where $s = s'_i$)

1: Send a message (**Copy**$r, r', \pi_r(k+i)$) to the receiver.

Open r'

1: Send (**Opening**, r') to the receiver.
2: Let i be the index of the copy in $x_1, \ldots, x_\ell$ (i.e., the committer previously sent a (**Copy**$r, r', \pi_r(i)$) message to the receiver during execution of the corresponding **LinkedCopy** or **FakeCopy** command. Send **Open** $t_{\pi_r(i),L}$ to $\mathcal{F}^{(C)}$. {This is the "value" part of the commitment to x_i}

ProveCopy r' Previously a **LinkedCopy** r, r', i command must have been executed.

1: Send (**Proving copy**, r', r, i) to the receiver.
2: Send **Open** $t_{i,E}$ to $\mathcal{F}^{(C)}$.
3: Send **Open** $t_{\pi_r(i),R}$ to $\mathcal{F}^{(C)}$.

Protocol 4.3b $\mathcal{F}^{(C\&C[k])}$ from $\mathcal{F}^{(C)}$ (Receiver)

The receiver maintains three databases to keep track of the commitments: an "internal" commitment tag database, "external" commitment tag database, and a "mapping" datatabase that maps the tags of commitment copies to internal commitments.

SrcCommit r, s, F Let $\ell = |F| + k$ and denote $F = (s'_1, \ldots, s'_{\ell-k})$.

1: Wait to receive the $(\textbf{SrcCommitting}, r, \ell)$ message from the commiter
2: **for** $1 \le i \le \ell$ **do** {Receive internal commitment pairs}
3: Wait to receive a $(\textbf{Committed}, t'_L)$ message from $\mathcal{F}^{(C)}$.
4: Wait to receive a $(\textbf{Committed}, t'_R)$ message from $\mathcal{F}^{(C)}$.
5: Store (r, i, t'_L, t'_R) in the internal commitment database.
6: **end for**
7: **for** $1 \le i \le k$ **do** {Receive external commitments}
8: Wait to receive a $(\textbf{Committed}, t'_E)$ message from $\mathcal{F}^{(C)}$.
9: Store (r, i, t'_E) in the external commitment database.
10: **end for**
11: Output $(\textbf{SrcCommitted}, r, \ell - k)$

LinkedCopy r, r', i $(1 \le i \le k)$

1: Wait to receive $(\textbf{Copy}, r, r', i')$ from the committer.
2: Verify that r' does not appear in the mapping database (i.e., no message of the form $(\textbf{Copy}*, r', *)$ was previously received from the committer).
3: Store (r', r, i') in the mapping database.
4: Output $(\textbf{Committed}, r, r')$.

FakeCopy r, r', s (where $s = s'_i$)

1: Wait to receive $(\textbf{Copy}r, r', i')$ from the committer.
2: Verify that r' does not appear in the mapping database (i.e., no message of the form $(\textbf{Copy}*, r', *)$ was previously received from the committer).
3: Store (r', r, i') in the mapping database.
4: Output $(\textbf{Committed}, r, r')$.

Open r'

1: Wait to receive $(\textbf{Opening}, r')$ from committer and $(\textbf{Opened}, t', s)$ message from $\mathcal{F}^{(C)}$.
2: Verify a tuple of the form (r', r, i') appears in the mapping database
3: Verify that a tuple of the form (r, i', t'_L, t'_R) appears in the internal commitment database.
4: Verify that $t' = t'_L$.
5: Output $(\textbf{Opened}, r', s)$.

ProveCopy r' Previously a **LinkedCopy** r, r', i command must have been executed.

1: Wait for $(\textbf{Proving copy}, r', r, i)$ from committer.
2: Wait for $(\textbf{Opened}, t'_E, x)$ message from $\mathcal{F}^{(C)}$.
3: Wait for $(\textbf{Opened}, t'_R, y)$ message from $\mathcal{F}^{(C)}$.
4: Verify the tuple (r, i, t'_E) appears in the external commitment database.
5: Verify a tuple of the form (r', r, i') appears in the mapping database
6: Verify that a tuple of the form (r, i', t'_L, t'_R) appears in the internal commitment database.
7: Verify that $x = [t'_L; t'_R]$.
8: Verify that $y = [r; i]$
9: Output $(\textbf{CopyOf}, r', r, i)$.

party is simulated correctly (i.e., it will follow all of $\mathcal{A}$'s instructions exactly, and respond to queries about its view using the provisional view maintained by $\mathcal{I}$). If the simulated (honest) receiver outputs $\perp$ at any time, $\mathcal{I}$ sends the **Halt** command to $\mathcal{F}^{(C\&C)}$.

Note that the receiver has no input and sends no messages to the committer or to $\mathcal{F}^{(C)}$, so $\mathcal{I}$'s simulation of the honest receiver is always perfect. Its simulated view consists of the messages sent by the committer and by $\mathcal{F}^{(C)}$— these are never changed when $\mathcal{I}$ rewrites its provisional view, so the receiver's view is always identically distributed in the real and ideal worlds.

The committer's input consists of a sequence of commands to $\mathcal{F}^{(C\&C)}$ (along with their parameters). When the committer is honest, $\mathcal{I}$ guesses the input in order to run the simulation (initially, it guesses the committer's input is empty). Note that the honest committer's view in the real world consists entirely of its input and the random coins used to construct the permutations π_r when executing the **SrcCommit** command. When the receiver is corrupted and $\mathcal{I}$ receives messages from $\mathcal{F}^{(C\&C)}$ to the receiver, it gains new information about the committer's input that may conflict with the guesses used to construct the provisional view:

(**SrcCommitted**, r, ℓ) $\mathcal{I}$ now knows the committer's input contains a **SrcCommit** r, s, F command (it initially guesses $s = 0$ and F a vector of zeroes of size $\ell - k$). $\mathcal{I}$ simulates an honest committer executing **SrcCommit** r, s, F (note that the corrupt receiver only learns r, ℓ; the vector of internal and external commitment tags is deterministic).

(**Committed**, r, r') $\mathcal{I}$ now knows the committer's input contains a **LinkedCopy** r, r', i or a **FakeCopy** r, r', s command. It makes a guess consistent with its provisional view (choosing a random index i that is still unused according to the provisional view) and simulates the honest committer executing the command. Note that the (**Copy**r, r', i') message sent to the corrupt receiver always contains random index i' (chosen uniformly from the set of unused indices), since $\mathcal{I}$ choses π_r randomly as required by the protocol for the honest committer.

(**Opened**, r, s) $\mathcal{I}$ learns that the committer's input contains an **Open** r command, and also the value of the commitment r. $\mathcal{I}$ rewrites its provisional view so that the value of the commitment r is now s (it can do this by rewriting the internal database of the simulated $\mathcal{F}^{(C)}$ for the corresponding internal commitments). $\mathcal{I}$ then simulates the **Open** r command following the protocol exactly. Note that rewriting the provisional view in this case does not change anything in the corrupt receiver's view.

(**CopyOf**, r', r, i) $\mathcal{I}$ learns that the committer's input contains a **ProveCopy** r' command, and also that the commitment with tag r' was generated by a **LinkedCopy** r', i command. $\mathcal{I}$ rewrites the the provisional to make it consistent with this new information. If $\mathcal{I}$ previously sent a (**Copy**r, r', j) message to the receiver, for $i \neq \pi_r(j)$, $\mathcal{I}$ rewrites π_r by choosing a new permutation uniformly at random from the set of permutations that is consistent with the **ProveCopy** commands executed so far. Otherwise, $\mathcal{I}$ rewrites its provisional view by rewriting the messages sent to the simulated $\mathcal{F}^{(C)}$. Note that the rewriting does not affect the corrupt receiver's view in any way; the receiver's view of a **LinkedCopy** and **FakeCopy** command are identical and changing the value of **Commit** commands to $\mathcal{F}^{(C)}$ does not change the receiver's view at all. $\mathcal{I}$ then simulates the honest committer executing a **ProveCopy** r' command.

When a previously honest committer is corrupted, $\mathcal{I}$ learns the committer's entire input. If the receiver is honest, $\mathcal{I}$ simply runs internally the simulation of the honest committer with the new information. If the receiver is corrupt, $\mathcal{I}$ already has a provisional view for the committer; it rewrites the provisional view using the new information by rewriting the random permutations and the values of the commitments sent to $\mathcal{F}^{(C)}$ (note that $\mathcal{I}$ never has to rewrite the values of commitments that have already been opened or permutation indices that have already been revealed, since those will have been rewritten during the **Open** and **ProveCopy** commands).

To complete the description of the simulation, we specify what commands $\mathcal{I}$ sends to $\mathcal{F}^{(C\&C)}$ on behalf of a corrupt committer (the receiver never sends commands to $\mathcal{F}^{(C\&C)}$):

SrcCommit If the committer is corrupt and sent a message of the form (**SrcCommitting**, r, ℓ) to the simulated honest receiver, $\mathcal{I}$ does the following:

1: **for** $1 \leq i \leq \ell$ **do** {Wait for committer to send internal commitment pairs}
2: Wait for commiter to send a **Commit** $t_{i,L}, x_i$ command to $\mathcal{F}^{(C)}$.
3: Wait for commiter to send a **Commit** $t_{i,R}, y_i$ command to $\mathcal{F}^{(C)}$.
4: **end for**
5: **for** $1 \leq i \leq k$ **do** {Wait for committer to send external commitments}
6: Wait for commiter to send a **Commit** $t_{i,E}, e_i$ command to $\mathcal{F}^{(C)}$.
7: **if** $e_i = [t_{j,L}, t';]$ **then** {$t_{j,L}$ is a valid "left" commitment to x_j}
8: $s^{(i)} \leftarrow x_j$
9: **else**
10: $s^{(i)} \leftarrow 0$
11: **end if**
12: **end for**
13: Set $Q \leftarrow (s^{(1)}, \ldots, s^{(k)})$
14: Send a **BadCommit** r, Q, ℓ command to $\mathcal{F}^{(C\&C)}$.

LinkedCopy or FakeCopy If the committer is corrupt and sent a message of the form $(\textbf{Copy}, r, r', j)$ to the receiver, $\mathcal{I}$ simulates the receiver until the end of the command execution. $\mathcal{I}$'s message to $\mathcal{F}^{(C\&C)}$ depends on the committer's previous actions:

Case 1: The committer was previously honest and sent a **SrcCommit** r, s, F to $\mathcal{F}^{(C\&C)}$. $\mathcal{I}$ must have previously run the simulation of the **SrcCommit** command and chosen the permutation π_r. If $\pi_r(j) \leq k$, $\mathcal{I}$ sends a **LinkedCopy** $r, r', \pi_r(j)$ command to $\mathcal{F}^{(C\&C)}$. If $\pi_r(j) > k$, $\mathcal{I}$ sends a **FakeCopy** $r, r', s'_{\pi_r(j)-k}$ to $\mathcal{F}^{(C\&C)}$ (where s'_i is the i^{th} element of F).

Case 2: The committer was corrupt when executing the **SrcCommit** command (in which case $\mathcal{I}$ sent a **BadCommit** r, Q, ℓ command to $\mathcal{F}^{(C\&C)}$). Let s' be the committed value corresponding to $t_{j,L}$ in the execution of the **SrcCommit** command (i.e., the committer sent a **Commit** $t_{j,L}, s$ command to $\mathcal{F}^{(C)}$ during execution of the **SrcCommit** command; note that if the committer did not send such a command, the honest receiver would abort at this stage, so $\mathcal{I}$ would abort as well). If, for some $1 \leq i \leq k$, the committer sent a **Commit** $t_{i,E}, [t_{j,L}; t_{j,R}]$ command and the committed value corresponding to $t_{j,R}$ is i (i.e., this is a valid linked commitment), $\mathcal{I}$ sends a **LinkedCopy** r, r', i command to $\mathcal{F}^{(C\&C)}$. If this is not the case, $\mathcal{I}$ sends a **FakeCopy** r, r', s command to $\mathcal{F}^{(C\&C)}$.

Open If the committer is corrupt and sent a message of the form $(\textbf{Opening}, r')$ to the simulated honest receiver, $\mathcal{I}$ simulates the receiver until the end of the command execution. Note that if the receiver did not abort, $\mathcal{I}$ must have previously sent a **LinkedCopy** r, r', i or **FakeCopy** r, r', s command to $\mathcal{F}^{(C\&C)}$. $\mathcal{I}$ then sends an **Open** r' command to $\mathcal{F}^{(C\&C)}$.

ProveCopy If the committer is corrupt and sent a message of the form $(\textbf{Proving copy}, r', r, i)$ to the simulated honest receiver, $\mathcal{I}$ simulates the receiver until the end of the command execution. If the receiver did not abort, the committer must have sent a **Commit** $t_{i,E}, [t_{j,L}; t_{j,R}]$ command for some j in the **SrcCommit** phase, and the committed value corresponding to $t_{j,R}$ must be i. In this case, $\mathcal{I}$ would have sent a **LinkedCopy** r, r', i command in the execution of the corresponding **Copy** command. $\mathcal{I}$ sends **ProveCopy** r' to $\mathcal{F}^{(C\&C)}$.

4.7.3 Indistinguishability of Views

The final step of the proof of Theorem 4.6 is to show that for any environment machine $\mathcal{Z}$, the view of a protocol execution in the ideal world with the simulator $\mathcal{I}$ is indistinguishable from the view of the protocol execution in the real world with real adversary $\mathcal{A}$.

In the real world, $\mathcal{Z}$'s view consists of the committer's inputs (which $\mathcal{Z}$ chooses), the receiver's output, and the views of corrupted parties (these include the parties' random coins). In the ideal world, $\mathcal{Z}$'s view consists of the committer's inputs, the receiver's output and the simulated views of the corrupted parties. The simulated view of a corrupted party consists of $\mathcal{I}$'s provisional view for the party at the moment it was

corrupted (after the final rewriting of the provisional view using the information $\mathcal{I}$ learns by corrupting the corresponding ideal party) and the messages received from the simulated honest party after it was corrupted (once both parties are corrupted, any subsequent messages sent are determined entirely by $\mathcal{Z}$, hence they are a deterministic function of $\mathcal{Z}$'s view up to that point).

We claim the views are identically distributed. By examining the simulation, it is easy to verify that the simulated receiver's output is always consistent with its output in the ideal world. Note that the only randomness in the protocol is the random permutations π_r chosen by the committer when executing the **SrcCommit** command. Thus, to show the views are identically distributed, it remains to show that the random permutations π_r are identically distributed.

When the committer is corrupt during execution of a **SrcCommit** command, the permutation is chosen by $\mathcal{A}$ in both the real and ideal worlds (hence, it is identically distributed in both). When the committer is honest, it chooses π_r uniformly at random in the real world. In the ideal world, $\mathcal{I}$ also initializes the provisional view by choosing π_r uniformly at random. $\mathcal{I}$ may later be forced to rewrite the provisional view and change the value of π_r. However, we can think of this as choosing a permutation in the following way: adversarially pick a set of indices $i_1, \ldots, i_k$ (these correspond to the indices for which a **LinkedCopy** command will be sent). Sequentially, for $1 \leq j \leq k$, $\mathcal{I}$ chooses a uniformly random value for $\pi_r(i_j)$ (this corresponds to $\mathcal{I}$ revealing the choice in a **ProveCopy** command). $\mathcal{I}$ then chooses the rest of the values for π_r uniformly at random (from the remaining possibilities). The resulting permutation π_r is uniformly distributed. This completes the proof of Theorem 4.6.

4.8 Discussion

Robustness of the Voting Scheme. The *robustness* of a voting scheme is its ability to recover from attacks without requiring the election to be canceled. Because the DRE is the sole tallying authority, a corrupt DRE can clearly disrupt the elections (e.g., by failing to output the tally). The UC proof shows that voters cannot disrupt the elections just by interacting with the DRE (the only thing a voter can do in the ideal model is either vote or abstain). However, our model does not prevent *false* accusations against the DRE. For example, a corrupt voter that is able to fake a receipt can argue that the DRE failed to publish it on the bulletin board. Using special paper or ink for the receipts may help [23], but preventing this and similar attacks remains an open problem. A mitigating factor is that, in the event of such an attack, it will be immediately evident that *something* is wrong (although it might not be clear who is at fault). Moreover, the perpetrator of such an attack cannot do so anonymously, as both the accusation and the identity of the accuser are public.

Traditional Paper Trail as Backup. Many critics of non-cryptographic DRE systems are pushing for a "voter verified paper-trail": requiring the DRE to print a plaintext ballot that the voter can inspect, which is then placed in a real ballot box. In a non-cryptographic system, the paper trail can help verify that a DRE is behaving honestly, and act as a recovery mechanism when it is not. In our system, a paper trail can be used for the recovery property alone: if the DRE is misbehaving, our protocol ensures it will be detected with high probability (without requiring random audits of the paper trail). In that case, we can perform a recount using the paper ballots.

Splitting the Vote. Our scheme suffers a large drawback compared to many of the published universally-verifiable schemes: we do not know how to distribute the vote between multiple authorities. In our protocol, the DRE acts as the only tallying authority. Thus, a corrupt DRE can reveal the voters' ballots. This is also the case with Chaum and Neff's schemes, however (as well as traditional DRE schemes in use today). In a subsequent work [56], we construct a voting scheme that can split the ballot between two authorities. However, this scheme can no longer be based on general commitment, as it requires the commitment to have specific homomorphic properties. Combining the good properties from both schemes remains an open problem.

Randomness and Covert channels. One problem that also plagues Neff's scheme, and possibly Chaum's [48], is that a corrupt DRE can convey information to an adversary using subliminal channels. In this case, an adversary only needs access to the bulletin board in order to learn all the voters choices. The source of the

problem is that the DRE uses a lot of randomness (e.g., the masking factors for the commitments and the random permutations in the final tally phase).

In our scheme's instantiation based on Pedersen commitments, we have a partial solution to this problem. It requires the output of the DRE to be sent through a series of "filters" before reaching the printer or bulletin board. The idea is that for each Pedersen commitment of the form $x = h^a g^r$ received by a filter, it will choose a random masking factor s, and output xg^s. If the DRE opens x by sending (a, r), the filter will send instead $(a, r + s)$. In a similar way the filter can mask the permutations used in the final-tally phase by choosing its own random permutations and composing them. This only requires one-way communication between the filters (i.e., each filter receives data from the filter before it in the series, and sends data only to the filter after it in the series). Note that the filters do not need to know the value of the commitments or the original permutations in order to perform its operation. If the DRE is honest, the filter receives no information and so cannot covertly send any. If the filter is honest, any randomness sent by the DRE is masked, so no information embedded in that randomness can leak. By using a series of filters, each from a different trustee, we can ensure that the DRE does not utilize covert channels (as long as at least one of the filters is honest). This solution requires special homomorphic properties from the underlying commitment scheme. Finding a general solution to this problem is an interesting open problem.

An even stronger adversary may be able to both coerce voters and maliciously program the DRE. Our protocol is vulnerable in this case. For example, a coercer can require a voter to use a special challenge, which is recognized by the DRE (a coercer can verify that the voter used the required challenge, since it appears on the public bulletin board). Once it knows a voter is coerced, the DRE can change the vote as it wishes (since the coerced voter will not be able to complain). Possibly mitigating the severity of this attack is the fact that, in order to significantly influence the outcome of an election, the adversary must coerce many voters. This makes it much harder to keep the corruption secret.

Separate Tallying. Our scheme requires the tally to be performed separately for each DRE. This reveals additional information about voter's choices (in many real elections this information is also available, however). An open problem is to allow a complete tally without sacrificing any of the other properties of our scheme (such as receipt-freeness and everlasting privacy).

Chapter 5

Split-Ballot Voting: Everlasting Privacy With Distributed Trust

5.1 Introduction

Recent years have seen increased interest in voting systems, with a focus on improving their integrity and trustworthiness. This focus has given an impetus to cryptographic research into voting protocols. Embracing cryptography allows us to achieve high levels of verifiability, and hence trustworthiness (every voter can check that her vote was counted correctly), without sacrificing the basic requirements of ballot secrecy and resistance to coercion.

A "perfect" voting protocol must satisfy a long list of requirements. Among the most important are:

Accuracy The final tally must reflect the voters' wishes.

Privacy A voter's vote must not be revealed to other parties.

Receipt-Freeness A voter should not be able to prove for whom she voted (this is important in order to prevent vote-buying and coercion).

Universal Verifiability Voters should be able to verify that their own votes were "cast as intended", and any interested party should be able to verify that all the votes were "counted as cast".

Surprisingly, all four of these seemingly contradictory properties can be satisfied simultaneously using cryptographic techniques. Unfortunately, applying cryptographic techniques introduces new problems. One of these is that cryptographic protocols are often based on computational assumptions (e.g., the infeasibility of solving a particular problem). Some computational assumptions, however, may have a built-in time limit (e.g., Adi Shamir estimated that all existing public-key systems, with key-lengths in use today, will remain secure for less than thirty years [71]).

A voting protocol is said to provide *information-theoretic privacy* if a computationally unbounded adversary does not gain any information about individual votes (apart from the final tally). If the privacy of the votes depends on computational assumptions, we say the protocol provides *computational privacy*. Note that to coerce a voter, it is enough that the voter *believe* there is a good chance of her privacy being violated, whether or not it is actually the case (so even if Shamir's estimate is unduly pessimistic, the fact that such an estimate was made by an expert may be enough to allow voter coercion). Therefore, protocols that provide computational privacy may not be proof against coercion: the voter may fear that her vote will become public some time in the future.

While integrity that depends on computational assumptions only requires the assumptions to hold during the election, privacy that depends on computational assumptions requires them to hold forever. To borrow a term from Aumann, Ding and Rabin [5], we can say that information-theoretic privacy is *everlasting* privacy.

A second problem that cryptographic voting protocols must consider is that most cryptographic techniques require complex computations that unaided humans are unable to perform. However, voters may

not trust voting computers to do these calculations for them. This mistrust is quite reasonable, because there is no way for them to tell if a computer is actually doing what it is supposed to be doing (as a trivial example consider a voting program that lets a voter choose a candidate, and then claims to cast a vote for that candidate; it could just as easily be casting a vote for a different candidate).

Finally, a problem that is applicable to all voting protocols is the problem of concentrating trust. We would like to construct protocols that don't have a "single point of failure" with respect to their security guarantees. Many protocols involve a "voting authority". In some protocols, this authority is a single-point of failure with respect to privacy (or, in extreme cases, integrity). Protocols that require the voter to input their votes to a computer automatically have a single point of failure: the computer is a single entity that "knows" the vote. This is not an idle concern: many ways exist for a corrupt computer to undetectably output information to an outside party (in some cases, the protocol itself provides such "subliminal channels").

5.1.1 Our Contributions

In this paper we introduce the first universally-verifiable voting protocol with everlasting privacy that can be performed by unaided humans and distributes trust across more than one voting authority. This protocol has reasonable complexity ($O(m)$ exponentiations per voter, where m is the number of candidates) and is efficient enough to be used in practice.

We formally prove our protocol is secure in the Universal Composability (UC) framework, which provides very strong notions of security. Loosely speaking, we show that running our protocol is as secure as running the election using an absolutely trustworthy third party (the "ideal functionality"), to whom all the voters secretly communicate their choices, and who then announces the final tally (a formal definition of this functionality appears in Section 5.4).

Surprisingly, we can attain this level of security even though we base the voting protocol on commitment and encryption schemes that are not, themselves, universally composable (we propose using a modification of the Pedersen commitment scheme together with Paillier encryption; see Appendix 5.A for details).

As part of the formal proof of security, we can specify precisely what assumptions we make when we claim the protocol is secure (this is not the case for most existing voting protocols, that lack formal proofs completely).

In addition, we formally prove that our protocol is receipt-free (voters cannot prove for whom they voted, even if they want to), using a simulation-based definition of receipt-freeness previously introduced by the authors [55].

Our insistence on rigorous proofs of correctness is not just "formalism for the sake of formalism". We believe that formal proofs of security provide several very significant practical advantages. First, a precondition for proving security is providing a formal definition of what we are trying to prove. This definition is useful in itself: it gives us a better understanding of what our protocol achieves, where it can be used and what its failure modes are. This is especially evident for definitions in simulation-based models (such as universal composability), since the definition of an ideal functionality is usually very intuitive.

Secondly, even fairly simple protocols may have hard to find weaknesses. Without a formal proof, we can never be certain that we have considered all possible avenues of attack. A formal proof lists a small number of assumptions that imply the security of the protocol. This means that to verify that a particular implementation is secure, we can concentrate on checking only these assumptions: as long as they are all satisfied, we can be certain an attack will not come from an unexpected direction. To illustrate this point, we demonstrate a subtle attack against the receipt-freeness of the Punchscan voting system [22] (see Section 5.2.4).

Finally, even though formal proofs are not "foolproof" — our definitions may not capture the "correct" notion of security, or the proof itself may contain errors — they give us a basis and a common language for meaningful discussions about protocols' security.

5.1.2 Related Work

Voting Protocols. Chaum proposed the first published electronic voting scheme in 1981 [19]. Many additional protocols were suggested since Chaum's. Among the more notable are [40, 26, 7, 27, 28, 45].

Most of the proposed voting schemes satisfy the accuracy, privacy and universal-verifiability properties. However, only a small fraction satisfy, in addition, the property of of receipt-freeness. Benaloh and Tuinstra [7] were the first to define this concept, and to give a protocol that achieves it (it turned out that their full protocol was not, in fact, receipt free, although their single-authority version was [45]). To satisfy receipt-freeness, Benaloh and Tuinstra also required a "voting booth": physically untappable channels between the voting authority and the voter.

Human Considerations. Almost all the existing protocols require complex computation on the part of the voter (infeasible for an unaided human). Thus, they require the voter to trust that the computer casting the ballot on her behalf is accurately reflecting her intentions. Chaum [21], and later Neff [62], proposed universally-verifiable receipt-free voting schemes that overcome this problem. Reynolds [67] proposed another protocol similar to Neff's.

All three schemes are based in the "traditional" setting, in which voters cast their ballots in the privacy of a voting booth. Instead of a ballot box, the booth contains a "Direct Recording Electronic" (DRE) voting machine. The voter communicates her choice to the DRE (e.g., using a touch-screen or keyboard). The DRE encrypts her vote and posts the encrypted ballot on a public bulletin board. It then proves to the voter, in the privacy of the voting booth, that the encrypted ballot is truly an encryption of her intended vote.

Chaum's original protocol used Visual Cryptography [61] to enable the human voter to read a complete (two-part) ballot that was later separated into two encrypted parts, and so his scheme required special printers and transparencies. Bryans and Ryan showed how to simplify this part of the protocol to use a standard printer [13, 68]. A newer idea of Chaum's is the Punchscan voting system [22], which we describe in more detail in Section 5.2.4.

Previously, the authors proposed a voting protocol, based on statistically-hiding commitments, that combines everlasting security and a human-centric interface [55]. This protocol requires a DRE, and inherently makes use of the fact that there is a single authority (the DRE plays the part of the voting authority).

Adida and Rivest [1] suggest the "Scratch&Vote" system, which makes use of *scratch-off cards* to provide receipt-freeness and "instant" verifiability (at the polling place). Their scheme publishes encryptions of the votes, and is therefore only computationally private.

Our new scheme follows the trend of basing protocols on physical assumptions in the traditional voting-booth setting. Unlike most of the previous schemes we also provide a rigorous proof that our scheme actually meets its security goals.

5.2 Informal Overview of the Split-Ballot Protocol

Our voting scheme uses two independent voting authorities that are responsible for preparing the paper ballots, counting the votes and proving that the announced tally is correct.

If both authorities are honest, the election is guaranteed to be accurate, information-theoretically private and receipt-free. If at least one of the authorities is honest, the election is guaranteed to be accurate and private (but now has only computational privacy, and may no longer be receipt-free). If both authorities are corrupt, the tally is still guaranteed to be accurate, but privacy is no longer guaranteed.

An election consists of four phases:

1. Setup: In this stage the keys for the commitment and encryption schemes are set up and ballots are prepared.
2. Voting: Voters cast their ballots. This stage is designed to be performed using pencil and paper, although computers may be used to improve the user experience.

   A vote consists of four ballots, two from each voting authority. The voter selects one ballot from each authority for auditing (they will not be used for voting). The remaining two ballots are used to vote. The voter's choices on both ballots, taken together, uniquely define the vote. A partial copy of each ballot is retained by the voter as a "verification receipt" (a more detailed description appears in Section 5.2.2).

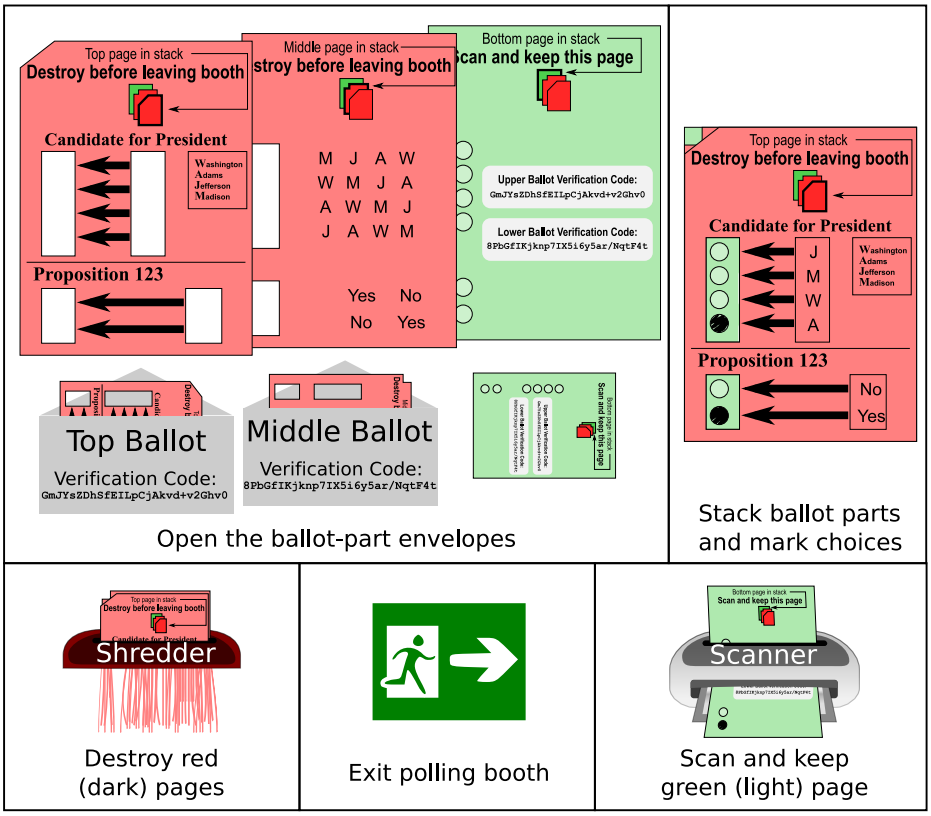

Figure 5.2.1: Illustrated Sample Vote

3. Tally: The two authorities publish all of the ballots. Voters may verify that their receipts appear correctly in the published tally. The two authorities then cooperate to tally the votes. The final result is a public proof that the tally is correct.
4. Universal Verification: In this phase any interested party can download the contents of the public bulletin board and verify that the authorities correctly tallied the votes.

5.2.1 Shuffling Commitments

One of the main contributions of this paper is achieving "everlasting privacy" with more than one voting authority. At first glance, this seems paradoxical: if a voting authority publishes any information at all about the votes (even encrypted), the scheme can no longer be information-theoretically private. On the other hand, without publishing information about the votes, how can two voting authorities combine their information?

We overcome this apparent contradiction by introducing the "oblivious commitment shuffle": a way for independent authorities to verifiably shuffle perfectly-hiding commitments (which will give us information-theoretic privacy).

The problem of verifiably shuffling a vector of *encrypted* values has been well studied. The most commonly used scheme involves multiple authorities who successively shuffle the encrypted vector, each using a secret permutation, and then prove that the resulting vector of encrypted values is valid. Finally, the authorities cooperate to decrypt the ultimate output of the chain. If even one of the authorities is honest (and keeps its permutation secret), the remaining authorities gain no information beyond the final tally.

This type of scheme breaks down when we try to apply it to perfectly-hiding commitments rather than encryptions. The problem is that in a perfectly-hiding commitment, the committed value cannot be determined from the commitment itself. Thus, the standard method of opening the commitments after shuffling cannot be used.

The way we bypass the problem is to allow the authorities to communicate privately using a homomorphic encryption scheme. This private communication is not perfectly hiding (in fact, the encryptions are perfectly binding commitments), but the voting scheme itself can remain information-theoretically private because the encryptions are never published. The trick is to encrypt separately both the message *and the randomness* used in the commitments. We use a homomorphic encryption scheme over the same group as the corresponding commitment. When the first authority shuffles the commitments, it simultaneously shuffles the encryptions (which were generated by the other authority). By opening the shuffled encryptions, the second authority learns the contents and randomness of the shuffled commitments (without learning anything about their original order). The second authority can now perform a traditional commitment shuffle.

5.2.2 Human Capability

Our protocol makes two questionable assumptions about human voters: that they can randomly select a bit (to determine which ballots to audit), and that they perform modular addition (to split their vote between the two authorities). The first is a fairly standard assumption (in fact, we do not require uniform randomness, only high min-entropy). The second seems highly suspect. However, it is possible to implement the voting protocol so that the modular addition occurs implicitly as a result of a more "natural" action for humans.

We propose an interface that borrows heavily from Punchscan's in order to make the voting task more intuitive. The basic idea is to form the ballot from three separate pages. The first page contains the list of candidates, along with a letter or symbol denoting each (this letter can be fixed before the election). The second page contains a table of letters: each column of the table is a permutation of the candidates. The third page is the one used to record the vote; it contains a scannable bubble for each row of the table in the middle page.

Holes are cut in the top page and middle pages, so that when all three are stacked a single random column of the table on the middle page is visible, as are the bubbles on the bottom page. The voter selects a candidate by marking the bubble corresponding to her choice. Since one authority randomly selects the table (on the middle page) and the other authority randomly selects which of its columns is used (determined

by the position of the hole in the top page), the position of the bubble marked by the voter does not give information about her choice unless both the middle and top pages are also known.

5.2.3 Vote Casting Example

To help clarify the voting process, we give a concrete example, describing a typical voter's view of an election (this view is illustrated in Figure 5.2.1). The election is for the office of president, and also includes a poll on "Proposition 123". The presidential candidates are Washington, Adams, Jefferson and Madison.

Sarah, the voter, enters the polling place and receives two pairs of ballot pages in sealed envelopes, each pair consisting of a "Top" ballot page and a "Middle" ballot page (we can think of the two voting authorities as the "Top" authority and the "Middle" authority). Each envelope is marked either "Top" or "Middle", and has a printed "verification code" (this code is actually a commitment to the public section of the ballot, as described in Section 5.5.1). Sarah first chooses a pair of ballot pages to audit. This pair is immediately opened, and the "red" (dark) ballot pages inside the envelopes are scanned, as are the verification codes on the envelopes. Sarah is allowed to keep all parts of the audited ballots.

The election officials give Sarah a green (light) "bottom page" that is printed with the verification codes from both the remaining (unopened) envelopes (alternatively, the verification codes could be printed on a sticker that is affixed to the green page before handing it to Sarah). She enters the polling booth with the green page and both unopened envelopes.

Inside the polling booth, Sarah opens the envelopes and takes out the red pages. The middle page is printed with a table of letters representing the candidates (the letters were chosen in advance to be the first letter of the candidate's surname). The order of the letters in the table is chosen randomly by the Middle authority (different ballot pages may have different orders). Similarly, the order of the "Yes" and "No" responses to Proposition 123 is random. The top page has a hole cut out that reveals a single column of the table — which column is randomly chosen by the Top authority. Sarah stacks all three pages (the top ballot page, the middle ballot part, and the green "bottom page"). Taken together, these pages form a complete ballot. Sarah wants to vote for Adams and to vote Yes on Proposition 123. She finds her candidate's letter on the ballot, and marks the corresponding bubble (the marks themselves are made on the green, bottom page that can be seen through the holes in the middle and top pages). She also finds the "Yes" choice for Proposition 123, and marks its corresponding bubble.

Sarah then separates the pages. She shreds the red pages that were inside the envelopes. To prevent vote-selling and coercion attacks, Sarah must be *forced* to destroy the red pages (e.g., perhaps the output of the shredder is visible to election officials outside the voting booth).

Sarah exits the voting booth holding only the marked, green page. This sheet of paper is then scanned (with the help of the election officials). The scanner can give immediate output so Sarah can verify that she filled the bubbles correctly, and that the scanner correctly identified her marks. Note that Sarah doesn't have to *trust* the scanner (or its software) in any way: The green page and the audited ballots will be kept by Sarah as receipts which she can use to prove that her vote was not correctly tabulated (if this does occur). At home Sarah will make sure that the verification code printed on the pages, together with the positions of the marked bubbles, are published on the bulletin board by the voting authorities. Alternatively, she can hand the receipts over to a helper organization that will perform the verification on her behalf.

5.2.4 The Importance of Rigorous Proofs of Security for Voting Protocols

To demonstrate why formal proofs of security are important, we describe a vote-buying attack against a previous version of the Punchscan voting protocol. The purpose of this section is not to disparage Punchscan; on the contrary, we use Punchscan as an example because it is one of the simplest protocols to understand and has been used in practice. A closer look at other voting protocols may reveal similar problems. Our aim is to encourage the use of formal security analysis to detect (and prevent) such vulnerabilities.

We very briefly describe the voter's view of the Punchscan protocol, using as an example an election race between Alice and Bob. The ballot consists of two pages, one on top of the other. The top page contains the candidates' names, and assigns each a random letter (either A or B). There are two holes in the top page through which the bottom page can be seen. On the bottom page, the letters A and B appear in a random

order (so that one letter can be seen through each hole in the top page). Thus, the voter is presented with one of the four possible ballot configurations (shown in Figure 5.2.2).

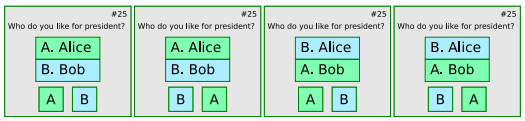

Figure 5.2.2: Punchscan Ballot Configurations

To vote, the voter marks the letter corresponding to her candidate using a wide marker: this marks both the top and bottom pages simultaneously. The two pages are then separated. The voter chooses one of the pages to scan (and keep as a receipt), while the other is shredded (these steps are shown in Figure 5.2.3).

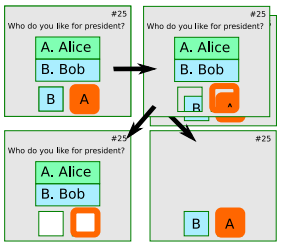

Figure 5.2.3: Punchscan Ballot

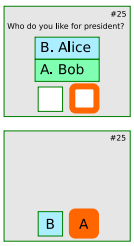

Figure 5.2.4: "Bad" Receipts

Each pair of pages has a short id, which a voting authority can use to determine what was printed on each of the pages (this allows the authority to determine the voter's vote even though it only receives a single page). For someone who does not know the contents of the shredded page, the receipt does not give any information about the voter's choice.

Giving each voter a receipt for her vote is extremely problematic in traditional voting systems, since the receipt can be used to coerce voters or to buy votes. Punchscan attempts to prevent vote-buying by making sure that the receipt does not contain any information about the voter's choice. At first glance, this idea seems to work: if an adversary just asks a voter to vote for a particular candidate (by following the Punchscan protocol honestly), there is no way the adversary can tell, just by looking at the receipt, whether the voter followed his instructions or not.

Below, we show that for a slightly more sophisticated adversary, a vote-buying attack *is* possible against Punchscan.

A Vote Buying Attack. To demonstrate the attack, we continue to use the Alice/Bob election example. Suppose the coercer wants to bias the vote towards Alice. In this case, he publishes that he will pay for any receipt *except* those shown in Figure 5.2.4 (i.e., everything except a "B,A" bottom page on which "A" was marked, and a "B,A" top page on which the right hole was marked).

This attack will force one fourth of the voters to vote for Alice in order to get paid. To see why, consider the four possible ballot configurations (in Figure 5.2.2). Since the coercer will accept any marking on an "A,B" top page or an "A,B" bottom page, in three of the four configurations the voter can vote as she wishes. However, if both the top and the bottom pages are "B,A" pages (this occurs in one fourth of the cases), the voter is forced to vote for Alice if she wants to return an acceptable receipt.

Although three-fourths of the voters can vote for any candidate, this attack is still entirely practical. When a race is close, only a small number of votes must be changed to tip the result in one direction. Compared to the "worst possible" system in which an adversary can buy votes directly, Punchscan requires the attacker to spend only four times as much to buy the same number of votes. Since the receipts are published, this attack can be performed remotely (e.g., over the internet), making it much worse than a "standard" vote-buying attack (such as chain-voting) that must be performed in person.

We note that the current version of Punchscan (as described in [65]) instructs the voter to commit to the layer she will take before entering the voting booth. This requirement does suffice to foil the attack described above. Similar attacks, however, may still be possible. The point we hope to make is that, lacking a formal proof of security, it is very hard to be certain.

5.3 Underlying Assumptions

One of the important advantages of formally analyzing voting protocols is that we can state the specific assumptions under which our security guarantees hold. Our protocol uses a combination of physical and cryptographic assumptions. Below, we define the assumptions and give a brief justification for each.

5.3.1 Physical Assumptions

Undeniable Ballots. To allow voters to complain convincingly about invalid ballots, they must be *undeniable*: a voter should be able to prove that the ballot was created by the voting authority. This type of requirement is standard for many physical objects: money, lottery-tickets, etc.

Forced Private Erasure. In order to preserve the receipt-freeness of the protocol, we require voters to physically erase information from the ballots they used. The erasure assumption is made by a number of existing voting schemes that require the voter to choose some part of the ballot to securely discard (e.g., Punchscan [22], Scratch&Vote [1]). In practice, this can be done by shredding, by chemical solvent, etc.

At first glance, it might appear that simply spoiling a ballot that was not correctly erased is sufficient. However, this is not the case; the voter must be *forced* to erase the designated content. Otherwise, a coercer can mount a vote-buying attack similar to the one described in section 5.2.4, where some voters are told to invalidate their ballots by refusing to erase them (and showing the complete ballot to the coercer).

Since only the voter should be able to see the contents of the erased part of the ballot, finding a good mechanism to enforce erasure may be difficult (e.g., handing it to an official to shred won't work). However, a large-scale attack that relies on circumventing this assumption may be detected by counting the number of spoiled ballots.

Voting Booth. In order to preserve privacy and receipt-freeness, the voter must be able to perform some actions privately. The actions the voter performs in the voting booth are opening sealed ballots, reading their contents and erasing part of the ballot.

Untappable Channels. We use untappable channels in two different ways. First, in order to guarantee everlasting privacy and receipt-freeness, ballots must be delivered from the voting authorities to the voter without any information about their contents leaking to a third party. The amount of data each voter must receive is small, however, and the untappable channel may be implmented, for example, using sealed envelopes.

Second, for the same reason, communication between the two voting authorities is also assumed to take place using untappable private channels. The amount of information exchanged is larger in this case, but this is a fairly reasonable assumption: the voting authorities can be physically close and connected by direct physical channels.

The untappable channel can also be replaced by encryption using a one-time pad (since this is also information-theoretically private). However, to simplify the proof we consider only an ideal untappable channel in this paper.

Public Bulletin Board. The public bulletin board is a common assumption in universally-verifiable voting protocols. This is usually modeled as a broadcast channel, or as append-only storage with read-access for all parties. A possible implementation is a web-site that is constantly monitored by multiple verifiers to ensure that nothing is erased or modified.

Random Beacon. The random beacon, originally introduced by Rabin [66], is a source of independently distributed, uniformly random strings. The main assumption about the beacon is that it is unpredictable.

In practice, the beacon can be implemented in many ways, such as by some physical source believed to be unpredictable (e.g., cosmic radiation, weather, etc.), or by a distributed computation with multiple verifiers.

We use the beacon for choosing the public-key of our commitment scheme and to replace the verifier in zero-knowledge proofs. For the zero-knowledge proofs, we can replace the beacon assumption by a random oracle (this is the Fiat-Shamir heuristic): the entire protocol transcript so far is taken as the index in the random oracle that is used as the next bit to be sent by the beacon.

5.3.2 Cryptographic Assumptions

Our protocol is based on two cryptographic primitives: perfectly-hiding homomorphic commitment and homomorphic encryption. The homomorphic commitment requires some special properties.

Homomorphic Commitment. A homomorphic commitment scheme consists of a tuple of algorithms: K, C, P_K, and V_K. $K\colon \{0,1\}^{\ell} \times \{0,1\}^{\ell} \mapsto \mathcal{K}$ accepts a public random bit-string and a private auxiliary and generates a commitment public key $cpk \in \mathcal{K}$. C is the commitment function, parametrized by the public key, mapping from a message group $(\mathcal{M}, +)$ and a randomizer group $(\mathcal{R}, +)$ to the group of commitments $(\mathcal{C}, \cdot)$.

P_K and V_K are a zero-knowledge "prover" and "verifier" for the key generation: these are both interactive machines. The prover receives the same input as the key generator, while the verifier receives only the public random string and the public key. To allow the verification to be performed publicly (using a random beacon), we require that all of the messages sent by V_K to P_K are uniformly distributed random strings.

For any probabilistic polynomial time turing machines (PPTs) K^*, P_K^* (corresponding to an adversarial key-generating algorithm and prover), when $cpk \leftarrow K^*(r_K)$, $r_K \in_R \{0,1\}^{\ell}$ is chosen uniformly at random then, with all but negligible probability (the probability is over the choice of r_K and the random coins of K^*, P_K^* and V_K), either the output of $V_K(r_K, cpk)$ when interacting with P_K^* is 0 (i.e., the verification of the public-key fails) or the following properties must hold:

1. Perfectly Hiding: For any $m_1, m_2 \in \mathcal{M}$, the random variables $C(m_1, r)$ and $C(m_2, r)$ must be identically distributed when r is taken uniformly at random from $\mathcal{R}$. (Note that we can replace this property with *statistically* hiding commitment, but for simplicity of the proof we require the stronger notion).
2. Computationally Binding: For any PPT $\mathcal{A}$ (with access to the private coins of K^*), the probability that $\mathcal{A}(cpk)$ can output $(m_1, r_1) \neq (m_2, r_2) \in \mathcal{M} \times \mathcal{R}$ such that $C_{cpk}(m_1, r_1) = C_{cpk}(m_2, r_2)$ must be negligible. The probability is over the random coins of K^*, $\mathcal{A}$ and r_K.
3. Homomorphic in both $\mathcal{M}$ and $\mathcal{R}$: for all $(m_1, r_1), (m_2, r_2) \in \mathcal{M} \times \mathcal{R}$, and all but a negligible fraction of keys, $C(m_1, r_1) \cdot C(m_2, r_2) = C(m_1 + m_2, r_1 + r_2)$.
4. Symmetry: The tuple (K, C'), where $C'(m, r) \doteq C(r, m)$ should also be a commitment scheme satisfying the hiding and binding properties (i.e., it should be possible to use $C(m, r)$ as a commitment to r).

Finally we also require the interaction between P_K and V_K to be zero-knowledge: there should exist an efficient simulator that, for every r_K and $K(r_k, aux)$, produces a simulated transcript of the interaction that is computationally-indistinguishable from a real one — even though it is not given aux (the secret auxiliary input to K).

Simulated Equivocability. For achieving UC security, we require the commitment scheme to have two additional algorithms: $K'\colon \{0,1\}^{\ell'} \mapsto \{0,1\}^{\ell}$, $C'\colon \{0,1\}^{\ell'} \times \mathcal{C} \times \mathcal{M} \mapsto \mathcal{R}$, such that the output of K' is uniformly random. The scheme must satisfy an additional property when we replace r_K with $K'(l)$, where $l \in_R \{0,1\}^{\ell}$:

5. Perfect Equivocability: For every $m \in \mathcal{M}$ and $c \in \mathcal{C}$, $C_{K^*(K'(l))}(m, C'(l, c, m)) = c$.

That is, it is possible to generate a public-key that is identical to a normal public key, but with additional side information that can be used to efficiently open every commitment to any value

Homomorphic Public-Key Encryption. The second cryptographic building block we use is a homomorphic public-key encryption scheme. We actually need two encryption schemes, one whose message space is $\mathcal{M}$

and the other whose message space is $\mathcal{R}$ (where $\mathcal{M}$ and $\mathcal{R}$ are as defined for the commitment scheme). The schemes are specified by the algorithm triplets $(KG^{(\mathcal{M})}, E^{(\mathcal{M})}, D^{(\mathcal{M})})$ and $(KG^{(\mathcal{R})}, E^{(\mathcal{R})}, D^{(\mathcal{R})})$, where KG is the key-generation algorithm, $E^{(\mathcal{X})}\colon \mathcal{X} \times \mathcal{T} \mapsto \mathcal{E}^{(\mathcal{X})}$ the encryption algorithm and $D^{(\mathcal{X})}\colon \mathcal{E}^{(\mathcal{X})} \mapsto \mathcal{X}$ the decryption algorithm. We require the encryption schemes to be semantically secure and homomorphic in their message spaces: for every $x_1, x_1 \in \mathcal{X}$ and any $r_1, r_2 \in \mathcal{T}$, there must exist $r' \in \mathcal{T}$ such that $E^{(\mathcal{X})}(x_1, r_1) \cdot E^{(\mathcal{X})}(x_2, r_2) = E^{(\mathcal{X})}(x_1 + x_2, r')$. .

We do not require the encryption scheme to be homomorphic in its randomness, but we do require, for every x_1, r_1, x_2, that r' is uniformly distributed in $\mathcal{T}$ when r_2 is chosen uniformly.

To reduce clutter, when it is obvious from context we omit the key parameter for the commitment scheme (e.g., we write $C(m, r)$ instead of $C_{cpk}(m, r)$), and the randomness and superscript for the encryption schemes (e.g., we write $E(m)$ to describe an encryption of m).

Below, we use only the abstract properties of the encryption and commitment schemes. For an actual implementation, we propose using the Paillier encryption scheme (where messages are in $\mathbb{Z}_n$ for a composite n, together with a modified version of Pedersen Commitment (where both messages and randomness are also in $\mathbb{Z}_n$). More details can be found in Appendix 5.A.

5.4 Threat Model and Security

We define and prove the security properties of our protocol using a simulation paradigm. The protocol's functionality is defined by describing how it would work in an "ideal world", in which there exists a completely trusted third party. Informally, our security claim is that any attack an adversary can perform on the protocol in the real world can be transformed into an attack on the functionality in the ideal world. This approach has the advantage of allowing us to gain a better intuitive understanding of the protocol's security guarantees, when compared to the game-based or property-based approach for defining security.

The basic functionality is defined and proved in Canetti's Universal Composability framework [16]. This provides extremely strong guarantees of security, including security under arbitrary composition with other protocols. The ideal voting functionality, described below, explicitly specifies what abilities the adversary gains by corrupting the different parties involved.

We also guarantee receipt-freeness, a property that is not captured by the standard UC definitions, using a similar simulation-based definition (see Appendix 5.C).

5.4.1 Ideal Voting Functionality

The voting functionality defines a number of different parties: n voters, two voting authorities A_1 and A_2, a verifier and an adversary. The voting authorities' only action is to specify the end of the voting phase. Also, there are some actions the adversary can perform only after corrupting one (or both) of the voting authorities. The verifier is the only party with output. If the protocol terminates successfully, the verifier outputs the tally, otherwise it outputs $\perp$ (this corresponds to cheating being detected).

When one (or both) of the voting authorities are corrupt, we allow the adversary to change the final tally, as long as the total number of votes changed is less than the security parameter k (we consider 2^{-k} negligible).[1] This is modeled by giving the tally privately to the adversary, and letting the adversary announce an arbitrary tally using the **Announce** command (described below). If one of the authorities is corrupt, we also allow the adversary to retroactively change the votes of corrupt voters, as a function of the tally (if we were to use a universally-composable encryption scheme, rather than one that is just semantically secure, we could do away with this requirement).

If neither of the voting authorities is corrupt, the adversary cannot cause the functionality to halt. The formal specification for the voting functionality, $\mathcal{F}^{(V)}$, follows:

Vote v, x_v On receiving this command from voter v, the functionality stores the tuple (v, x_v) in the vote database S and outputs "v has voted" to the adversary. The functionality then ignores further messages from voter v. The functionality will also accept this message from the adversary if v was previously

[1] This is a fairly common assumption in cryptographic voting protocols (appearing in [21, 13, 68, 22], among others).

corrupted (in this case an existing (v, x_v) tuple can be replaced). If one of the authorities was corrupted before the first **Vote** command was sent, the functionality will also accept this message from the adversary after the **Tally** command has been received (to change the vote of voters that were corrupted before the tally).

Vote $v, *$ This command signifies a forced random vote. It is accepted from the adversary only if voter v is coerced or corrupted. In that case, the functionality chooses a new random value $x_v \in_R \mathbb{Z}_m$, and stores the tuple (v, x_v) in the database.

Vote $v, \perp$ This command signifies a forced abstention. It is accepted from the adversary only if voter v is coerced or corrupted. In that case, the functionality deletes the tuple (v, x_v) from the database.

Tally On receiving this command from an authority, the functionality computes $\tau_i = |\{(v, x_v) \in S \mid x_v = i\}|$ for all $i \in \mathbb{Z}_m$. If none of the voting authorities are corrupt, the functionality sends the tally $\tau_0, \ldots, \tau_{m-1}$ to the verifier and halts (this is a successful termination). Otherwise (if at least one of the voting authorities is corrupt), it sends the tally, $\tau_0, \ldots, \tau_{m-1}$, to the adversary.

Announce $\tau'_0, \ldots, \tau'_{m-1}$ On receiving this command from the adversary, the functionality verifies that the **Tally** command was previously received. It then computes $d = \sum_{i=0}^{m-1} |\tau_i - \tau'_i|$ (if one of the authorities is corrupt and the adversary changed corrupt voters' choices after the **Tally** command was received, the functionality recomputes $\tau_0, \ldots, \tau_{m-1}$ before computing d). If $d < k$ (where k is the security parameter) it outputs the tally $\tau'_0, \ldots, \tau'_{m-1}$ to the verifier and halts (this is considered a successful termination).

Corrupt v On receiving this command from the adversary, the functionality sends x_v to the adversary (if there exists a tuple $(v, x_v) \in S$).

Corrupt A_a On receiving this command from the adversary, the functionality marks the voting authority A_a as corrupted.

RevealVotes On receiving this command from the adversary, the functionality verifies that both of the voting authorities A_1 and A_2 are corrupt. If this is the case, it sends the vote database S to the adversary.

Halt On receiving this command from the adversary, the functionality verifies that at least one of the voting authorities is corrupt. If so, it outputs $\perp$ to the verifier and halts.

Our main result is a protocol that realizes the ideal functionality $\mathcal{F}^{(V)}$ in the universal composability model. A formal statement of this is given in Theorem 5.1, with a proof in Section 5.6.

5.4.2 Receipt-Freeness

As previously discussed, in a voting protocol assuring privacy is not enough. In order to prevent vote-buying and coercion, we must ensure *receipt-freeness*: a voter shouldn't be able to prove how she voted even if she wants to. We use the definition of receipt-freeness from [55], an extension of Canetti and Gennaro's *incoercible computation* [17]. This definition of receipt-freeness is also simulation based, in the spirit of our other security definitions.

Parties all receive a fake input, in addition to their real one. A coerced player will use the fake input to answer the adversary's queries about the past view (before it was coerced). The adversary is not limited to passive queries, however. Once a player is coerced, the adversary can give it an *arbitrary strategy* (i.e. commands the player should follow instead of the real protocol interactions). We call coerced players that actually follow the adversary's commands "puppets".

A receipt-free protocol, in addition to specifying what players should do if they are honest, must also specify what players should do if they are coerced; we call this a "coercion-resistance strategy" The coercion-resistance strategy is a generalization of the "faking algorithm" in Canetti and Gennaro's definition — the faking algorithm only supplies an answer to a single query ("what was the randomness used for the protocol"), while the coercion-resistance strategy must tell the party how to react to any command given by the adversary.

Intuitively, a protocol is receipt-free if no adversary can distinguish between a party with real input x that is a puppet and one that has a fake input x (but a different real input) and is running the coercion-resistance strategy. At the same time, the computation's output should not change when we replace coerced parties running the coercion-resistance strategy with parties running the honest protocol (with their real inputs). Note that these conditions must hold even when the coercion-resistance strategy is known to the adversary.

In our original definition [55], a protocol is considered receipt-free even if the adversary can force a party to abstain. We weaken this definition slightly, and also allow the adversary to force a party to vote randomly. The intuition is that a uniformly random vote has the same effect, in expectation, as simply abstaining[2]. Our protocol is receipt-free under this definition (Theorem 5.2 gives a more precise statement of this fact).

Note that the intuition for why this is acceptable is not entirely correct: in some situations, the new definition can be significantly weaker. For example, when voting is compulsory, "buying" a random vote may be much cheaper than "buying" an abstention (the price would have to include the fine for not voting). Another situation where forcing randomization may be more powerful than forcing an abstention is if the margin of victory is important (such as in proportional elections). In many cases, however, the difference is not considered substantial enough to matter; we note that Punchscan and Prêt à Voter, two of the most widely-known universally-verifiable voting schemes, are also vulnerable to a forced randomization attack.

5.5 Split-Ballot Voting Protocol

In this section we give an abstract description of the split-ballot voting protocol (by abstract, we mean we that we describe the logical operations performed by the parties without describing a physical implementation). In the interest of clarity, we restrict ourselves to two voting authorities A_1,A_2, n voters and a single poll question with answers in the group $\mathbb{Z}_m$. We assume the existence of a homomorphic commitment scheme (K, C) (with the properties defined in Section 5.3.2) whose message space is a group $(\mathcal{M}, +)$, randomizer space a group $(\mathcal{R}, +)$, and commitment space a group $(\mathcal{C}, \cdot)$. Our protocol requires $\mathcal{M}$ to be cyclic and have a large order: $|\mathcal{M}| \geq 2^{2k+2}$, and we assume $m < 2^k$ (k is the security parameter defined in Section 5.4.1). Furthermore, we assume the existence of homomorphic encryption schemes with the corresponding message spaces.

5.5.1 Setup

The initial setup involves:

1. Choosing the system parameters (these consist of the commitment scheme public key and the encryption scheme public/private key pair). Authority A_1 runs $KG^{(\mathcal{M})}$ and $KG^{(\mathcal{R})}$, producing $(pk^{(\mathcal{M})}, sk^{(\mathcal{M})})$ and $(pk^{(\mathcal{R})}, sk^{(\mathcal{R})})$. A_1 sends the public keys over the private channel to authority A_2. It also runs K using the output of the random beacon as the public random string, and the private coins used in running $KG^{(\mathcal{M})}$ and $KG^{(\mathcal{R})}$ as the auxiliary. This produces the commitment public key, cpk. Authority A_1 now runs P_K using the random beacon in place of the verifier (this produces a public proof that the commitment key was generated correctly).
2. Ballot preparation. Each voting authority prepares at least $2n$ ballot parts (the complete ballots are a combination of one part from each authority). We identify a ballot part by the tuple $\vec{w} = (a, i, b) \in \{1, 2\} \times [n] \times \{0, 1\}$, where A_a is the voting authority that generated the ballot part, i is the index of the voter to whom it will be sent and b a ballot part serial number. Each ballot part has a "public" section that is published and a "private" section that is shown only to the voter. The private section for ballot part $B_{\vec{w}}$ is a random value $t_{\vec{w}} \in_R \mathbb{Z}_m$. For $\vec{w} = (2, i, b)$ (i.e., ballot parts generated by authority A_2), the public section of $B_{\vec{w}}$ consists of a commitment to that value: $c_{\vec{w}} \doteq C(t_{\vec{w}}, r_{\vec{w}})$, where $r_{\vec{w}} \in_R \mathcal{R}$. For $\vec{w} = (1, i, b)$ (ballot parts generated by A_1), the public section contains a vector of commitments: $c_{\vec{w},0}, \ldots, c_{\vec{w},m-1}$, where $c_{\vec{w},j} \doteq C(t_{\vec{w}} + j \pmod m, r_{\vec{w},j})$, and $r_{\vec{w},j} \in_R \mathcal{R}$ (i.e., the commitments are to the numbers 0 through $m-1$ shifted by the value $t_{\vec{w}}$). The authorities publish the public parts of all the ballots to the bulletin board.

[2]Note that the attack we describe in Section 5.2.4 is not equivalent to forcing a random vote: the coercer forces voters to choose the desired candidate with higher probability than the competitor.

5.5.2 Voting

The voter receives two ballot parts from each of the voting authorities, one set is used for voting, and the other to audit the authorities. The private parts of the ballot are hidden under a tamper-evident seal (e.g., an opaque envelope). Denote the voter's response to the poll question by $x_v \in \mathbb{Z}_m$. Informally, the voter uses a trivial secret sharing scheme to mask her vote: she splits it into two random shares whose sum is x_v. The second share is chosen ahead of time by A_2, while the first is selected from the ballot part received from A_1 by choosing the corresponding commitment. A more formal description appears as Protocol 5.1.

Protocol 5.1 Ballot casting by voter v

1: Wait to receive ballots parts $B_{\vec{w}}$, for all $\vec{w} \in \{1,2\} \times \{v\} \times \{0,1\}$ from the authorities.
2: Choose a random bit: $b_v \in_R \{0,1\}$
3: Open and publish ballot parts $B_{(1,v,1-b_v)}$ and $B_{(2,v,1-b_v)}$. {these will be used for auditing the voting authorities}
4: Verify that the remaining ballot parts are still sealed, then enter the voting booth with them.
5: Open the ballot parts $B_{(1,v,b_v)}$ snd $B_{(2,v,b_v)}$.
6: Compute $s_v \doteq x_v - t_{(1,v,b_v)} - t_{(2,v,b_v)} \pmod m$. To reduce clutter, below we omit the subscripts b_v and s_v, denoting $c_{(1,v)} \doteq c_{(1,v,b_v),s_v}$, $r_{(1,v)} \doteq r_{(1,v,b_v),s_v}$, $c_{(2,v)} \doteq c_{(2,v,b_v)}$, $r_{(2,v)} \doteq r_{(2,v,b_v)}$ and $t_{(a,v)} \doteq t_{(a,v,b_v)}$. {The computation can be perfomed implictly by the voting mechanism, e.g., the method described in Section 5.2.2}.
7: Physically erase the private values $t_{\vec{w}}$ from all the received ballot parts. {This step is the "forced ballot erasure"}
8: Leave the voting booth.
9: Publish s_v {recall that $c_{(1,v)}$ and $c_{(2,v)}$ were already published by the authorities}.

Coercion-Resistance Strategy. We assume the adversary cannot observe the voter between steps 4 and 8 of the voting phase (i.e., while the voter is in the voting booth).

If the voter is coerced before step 4, the voter follows the adversary's strategy precisely, but uses random $t_{(a,v)}$ values instead of those revealed on the opened ballots. Because of the forced erasure, the adversary will not be able to tell whether the voter used the correct values or not. By using random values, the end result is that the voter votes randomly (coercing a voter to vote randomly is an attack we explicitly allow).

If the voter is coerced at step 4 or later (after entering the voting booth), she follows the regular voting protocol in steps 4 through 7. Even if she is coerced before step 7, she lies to the adversary and pretends the coercion occurred at step 7 (the adversary cannot tell which step in the protocol the voter is executing while the voter is in the booth). In this case, the adversary cannot give the voter a voting strategy, except one that will invalidate the ballot (since the voter has no more "legal" choices left). The voter must still convince the adversary that her vote was for the "fake input" provided by the adversary rather than her real input. To do this, she pretends the $t_{(2,v)}$ value she received was one that is consistent with the fake input and her real s_v. Using the example in Figure 5.2.1, if Sarah was trying to convince a coercer that she actually voted for Jefferson (instead of Adams), she would claim that the upper ballot part had the hole in the leftmost position (rather than the second position), so that her choice on the lower ballot part corresponds to Jefferson.

Note that the adversary can force the voter to cast a random ballot, for example by telling her to always fill the top bubble on the bottom page. However, forcing a random vote is something we explicitly do not prevent.

5.5.3 Tally

The tally stage is performed by the voting authorities and does not require voter participation (for the intuition behind it, see Section 5.2.1). Before the start of the tally stage, all authorities know, for every voter v: s_v, $c_{(1,v)}$ and $c_{(2,v)}$ (this was published on the public bulletin board). Each authority A_a also knows the private values $t_{(a,v)}$ and $r_{(a,v)}$ (the voter's choice is $x_v = s_v + t_{(1,v)} + t_{(2,v)} \pmod m$).

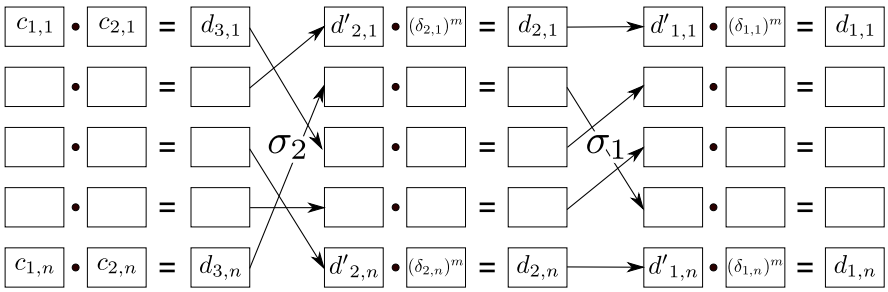

Figure 5.5.1: Tally Phase 1

For all $1 \leq v \leq n$, denote $d_{3,v} \doteq c_{(1,v)}c_{(2,v)} = C(x_v \pmod m, r_{(1,v)} + r_{(2,v)})$. The value $d_{3,v}$ is a commitment to voter v's choice, up to multiples of m; both the value and the randomness for the commitment are shared among the two authorities (note that $c_{(1,v)} = c_{(1,v,b_v),s_v}$ is a commitment to $x_v - t_{(2,v)}$, while $c_{(2,v)}$ is a commitment to $t_{(2,v)}$, so their product, the homomorphic addition of the committed values, gives us a commitment to x_v as required).

The tally stage uses as subprotocols some zero-knowledge proofs. In particular, we use a proof that one vector of commitments "is a valid shuffle" of another, and a proof that "a committed number is in a specific range". Since we use perfectly-hiding commitments, the "value of a commitment" is not well defined. What we actually use is a *zero-knowledge proof of knowledge*: the prover proves that it "knows" how to open the commitment to a value in the range, or how to shuffle and randomize the first vector of commitments to construct the second. These zero-knowledge protocols are based on standard techniques; simple protocols that meet our requirements appear in Appendix 5.B.

The tally is performed in two phases. The first phase (cf. Protocol 5.2) is a public "mix-net"-like shuffle of the commitments, while the second is a private protocol between the two authorities, at the end of which the first authority learns how to open the shuffled commitments.

In the first phase, the authorities, in sequence, privately shuffle the vector of committed votes and publish a new vector of commitments in a random order (we denote the new vector published by authority A_a: $d_{a,1}, \ldots, d_{a,n}$). The new commitments are rerandomized (so they cannot be connected to the original voters) by homomorphically adding to each a commitment to a random multiple of m. To prevent A_2 from using the rerandomization step to "tag" commitments and thus gain information about specific voters, A_1's randomization value is taken from a much larger range (A_2 chooses a multiple of m in $\mathbb{Z}_{2^k}$, while A_1 chooses one in $\mathbb{Z}_{2^{2k}}$; we require $|\mathcal{M}| > 2^{2k+2}$ so that the rerandomization doesn't cause the commitment message to "roll over"). Each authority also proves in zero-knowledge that the published commitments are a valid shuffle of the previous set of commitments (where by "valid shuffle", we mean that if it can open one vector of commitments, then it can open the other vector to a permutation of the values). A graphic representation of this phase of the tally appears in Figure 5.5.1.

The output of the final shuffle at the end of the first phase is a vector of commitments to the voters' choices that neither of the authorities can connect to the corresponding voters. However, neither of the authorities can open the commitments, since the secret randomness is shared among both of them. In the second phase of the tally, the authorities perform the same shuffles on encrypted, rather than committed values (the encryptions all use the public key generated by A_1). At the end of the second phase, A_1 learns the information required to open the final commitment vector, and publishes this, revealing the tally. The communication between the authorities is over an untappable channel between them, so that the public information remains unconditionally private. This phase of the tally is specified by Protocols 5.3 (for authority A_1) and 5.4 (for authority A_2).

5.5.4 Universal Verification and Output

The verification can be performed by anyone with access to the public bulletin board. This phase consists of verifiying all the published zero-knowledge proofs by running the zero-knowledge verifier for each proof,

Protocol 5.2 Tally Phase 1: Authority A_a

Choose a random permutation $\sigma_a \colon [n] \mapsto [n]$,
Choose random values $u_{a,1}, \ldots, u_{a,n} \in_R \mathcal{R}$
Choose random values $z_{a,1}, \ldots, z_{a,n} \in_R \mathbb{Z}_{2^{(3-a)k}}$
If $a = 1$, wait for $d_{2,1}, \ldots, d_{2,n}$ to be published.
for $1 \leq i \leq n$ **do**
  Choose a random value $u'_{a,i} \in_R \mathcal{R}$.
  Publish $d'_{a,i} \doteq d_{a+1,\sigma_a(i)} \cdot C(0, u'_{a,i} - mu'_{a,i})$.
  Publish $\delta_{a,i} \doteq C(z_{a,i}, u'_{a,i})$
  Publicly prove in zero-knowledge (using the random beacon) that $\delta_{a,i}$ is a commitment to a value in $\mathbb{Z}_{2^{(3-a)k}}$ (i.e., that $z_{a,i} < 2^{(3-a)k}$)
  Publicly prove in zero-knowledge (using the random beacon) that $c_{a,i}$ is a commitment to a value in $\mathbb{Z}_{2^{\lceil \log m \rceil}}$
  Denote $d_{a,i} \doteq d'_{a,i}\delta^m_{a,i} = d_{a+1,\sigma_a(i)}C(z_{a,i}m, u_{a,i})$
end for
Publicly prove in zero-knowledge that $d'_{a,1}, \ldots, d'_{a,n}$ is a valid shuffle of $d_{a+1,1}, \ldots, d_{a+1,n}$ (using the random beacon).

Protocol 5.3 Tally Phase 2: Authority A_1

Wait for Tally Phase 1 to terminate.
Verify all the published zero-knowledge proofs of shuffling.
for $1 \leq i \leq n$ **do**
  Send to Authority A_2: $e^{(\mathcal{M})}_{1,i} \doteq E^{(\mathcal{M})}(s_i + t_{(1,i)}) = E^{(\mathcal{M})}(x_i - t_{(2,i)})$
  Send to Authority A_2: $e^{(\mathcal{R})}_{1,i} \doteq E^{(\mathcal{R})}(r_{(1,i)})$
end for
Prove in (interactive) zero-knowledge to A_2 that $e^{(\mathcal{M})}_{1,1}, \ldots, e^{(\mathcal{M})}_{1,n}$ and $e^{(\mathcal{R})}_{1,1}, \ldots, e^{(\mathcal{R})}_{1,n}$ are encryptions of the message (resp. randomness) corresponding to $c_{(1,1)}, \ldots, c_{(1,n)}$.
Wait to receive $e^{(\mathcal{M})}_{2,1}, \ldots, e^{(\mathcal{M})}_{2,n}$ and $e^{(\mathcal{R})}_{2,1}, \ldots, e^{(\mathcal{R})}_{2,n}$ over the untappable channel from A_2.
for $1 \leq i \leq n$ **do**
  Compute $\xi_i \doteq D^{(\mathcal{M})}(e^{(\mathcal{M})}_{2,\sigma_1(i)}) + z_{1,i}m$
  Compute $\rho_i \doteq D^{(\mathcal{R})}(e^{(\mathcal{R})}_{2,\sigma_1(i)}) + u_{1,i}$
  Verify that $d_{1,i} = C(\xi_i, \rho_i)$
end for
Publish $\xi_1, \ldots, \xi_n$ and $\rho_1, \ldots, \rho_n$.

Protocol 5.4 Tally Phase 2: Authority A_2

Wait for Tally Phase 1 to complete.
Verify all zero-knowledge proofs published in phase 1.
Wait to receive $e^{(\mathcal{M})}_{1,1}, \ldots, e^{(\mathcal{M})}_{1,n}$ and $e^{(\mathcal{R})}_{1,1}, \ldots, e^{(\mathcal{R})}_{1,n}$ over the untappable channel from A_1.
Verify the zero-knowledge proof that the encryptions correspond to the committed values and randomness
for $1 \leq i \leq n$ **do**
  Send to Authority A_1: $e^{(\mathcal{M})}_{2,i} \doteq e^{(\mathcal{M})}_{1,\sigma_2(i)}E^{(\mathcal{M})}(t_{(2,\sigma_2(i))} + z_{2,i}m) = E^{(\mathcal{M})}(x_{\sigma_2(i)} + z_{2,i}m)$
  Send to Authority A_1: $e^{(\mathcal{R})}_{2,i} \doteq e^{(\mathcal{R})}_{1,\sigma_2(i)}E^{(\mathcal{R})}(r_{(2,\sigma_2(i))} + u_{2,i})$
end for

using the corresponding output of the random beacon for the random coins. The verifier then computes and outputs the final tally. The verification protocol appears as Protocol 5.5.

Protocol 5.5 Verification

1: Verify the proof that the commitment key was generated correctly {the proof generated in step 1, Section 5.5.1}
2: **for** $1 \leq i \leq n$ **do** {Verify opened ballots chosen for audit}
3: Verify for all $j \in \mathbb{Z}_m$ that $c_{(1,i,1-b_i),j} = C(t_{(1,i,1-b_i)} + j \pmod m, r_{(1,i,1-b_i),j})$
4: Verify that $c_{(2,i,1-b_i)} = C(t_{(2,i,1-b_i)}, r_{(2,i,1-b_i)})$
5: **end for**
6: Verify the shuffle proofs (produced in step 13 of Protocol 5.2)
7: **for** $1 \leq i \leq n$ **do** {Verify opening of final, shuffled commitments}
8: Verify that $d_{1,i} = C(\xi_i, \rho_i)$
9: **end for**
10: **for** $i \in \mathbb{Z}_m$ **do** {Compute and output the tally}
11: Compute $\tau_i \doteq |\{j \in [n] \mid \xi_j \equiv i \pmod m\}|$
12: Output τ_i {The tally for candidate i}
13: **end for**

5.5.5 Security Guarantees

We give two different formal security guarantees for this protocol, formally specified by the theorems below. The first is a guarantee for privacy and accuracy of the tally (Theorem 5.1), and the second a guarantee against vote-buying and coercion (Theorem 5.2).

Theorem 5.1. *The Split-Ballot Voting Protocol UC-realizes functionality $\mathcal{F}^{(V)}$, for an adversary that is fully adaptive up to the end of the voting phase, but then statically decides which of the voting authorities to corrupt (it can still adaptively corrupt voters).*

The reason for the restriction on the adversary's adaptiveness is that the homomorphic encryption scheme we use is *committing*.

Note that this limitation on adaptiveness only holds with respect to the *privacy* of the votes *under composition*, since an adversary whose only goal is to change the final tally can only gain by corrupting both voting authorities at the beginning of the protocol.

The proof of Theorem 5.1 appears in Section 5.6

Theorem 5.2. *The Split-Ballot voting protocol is receipt-free, for any adversary that does not corrupt any of the voting authorities.*

The formal proof of this theorem appears in Section 5.7. The intuition behind it is apparent from the coercion-resistance strategy (described in Section 5.5.2).

5.6 Proof of Accuracy and Privacy Guarantee (Theorem 5.1)

In order to prove the security of our protocol in the UC model, we must show that there exists a simulator $\mathcal{I}$ (the "ideal adversary") that internally runs a black-box straight-line (without rewinding) simulation of the real-world adversary, $\mathcal{A}$, simulating its view of the honest parties and the functionalities used by the real protocol. At the same time, the simulator interacts with the ideal voting functionality to allow it to keep the real-world adversary's simulated view consistent with the input and output of the parties in the ideal world. The protocol UC-realizes the ideal functionality if no environment machine (which sets the inputs for the parties and controls the real-world adversary) can distinguish between a real execution of the protocol and a simulated execution in the ideal world.

The general idea of the simulation is that $\mathcal{I}$ creates, internally, a "provisional view" of the world for all honest parties. Throughout the simulation, it updates this view so that it is consistent with the information $\mathcal{I}$ learns. At any point, if the adversary corrupts a party then $\mathcal{I}$ also corrupts that party (learning its input, and updating the provisional view to be consistent with that information). If the adversary corrupts *both* of the election authorities, A_1 and A_2, $\mathcal{I}$ sends the **RevealVotes** command to $\mathcal{F}^{(V)}$ (and learns the votes of all honest parties). Because the simulator can choose the commitment key in such a way that commitments are equivocable, it can make sure the "visible" parts of the provisional view (those that can be seen by the adversary) are also consistent with what the environment knows at all times (by changing retroactively the contents of the commitments). Thus, $\mathcal{I}$ is able to simulate the honest parties exactly according to the real protocol, creating a view that is statistically close to the view in the real world.

There are four main sticking points in this approach:

1. The adversary can prepare "bad ballots" in the setup phase (which it does not know how to open correctly). Since the commitment is perfectly hiding, the simulator cannot tell at that point which of the ballots is bad. We deal with this by allowing $\mathcal{I}$ to change the tally by a small number of votes (the number depends on the security parameter). The idea is that there are only two ways for the adversary to change the tally: either it can equivocate on commitments (in this case, we can use the environment/adversary pair to break the commitment scheme), or it prepares bad ballots. If it prepares many bad ballots, it will be caught with high probability, since $\mathcal{I}$ simulates the honest voters correctly, and they choose to audit a bad ballot pair with probability $\frac{1}{2}$. So if we simply ignore the cases in which it was not caught, the distributions of the environment's view in the real and ideal world will still be statistically close.
2. Unlike perfectly-hiding commitments, the encryptions send in the second tally phase cannot be retroactively changed in light of new information. This problem is solved by the restriction on the adaptiveness of the adversary (either it doesn't get to see the encryptions at all before it sees their contents, or it doesn't get to see the contents of the encryptions).
3. The revealed values of the shuffled commitments aren't exactly the voters' choices — they are the voters' choices only up to multiples of m. For example, If $x_v = 0$, this value can be shared between the authorities as $0, 0$ (in which case the commitment $d_{3,v} = 0$), but also as $m - 1, 1$, (in which case the commitment $d_{3,v} = m$). Since each authority knows its share of x_v, the revealed commitment can leak information about a specific voter. To prevent this, we would like to "rerandomize" the value by adding a uniformly random multiple of m. However, since it is likely that $m \nmid |\mathcal{M}|$, adding large multiples of m could change the value. Instead, authority A_2 (who shuffles first) adds multiples of m from a large range, but one that is guaranteed not to "overflow". To prevent A_2 from using the randomization step itself to gain information about specific voters, A_1 does the same thing with an even larger range. Thus, the revealed tallies leak only a negligible amount of information about the specific voters.
4. Finally, we use the semantic security of the encryption scheme to show that the environment/adversary pair cannot use the encryptions sent in the tally phase to gain any advantage in differentiating the real and ideal worlds. If it can, we can use them to break the encryption scheme.

Below we describe the protocol for $\mathcal{I}$. In order to make the proof readable, we do not formally specify the entire simulation protocol. Instead, we focus on the points where $\mathcal{I}$ cannot simply follow the real protocol for its simulated parties (either because it lacks information the real parties have, or because $\mathcal{A}$ deviated from the protocol). We also omit the global mechanics of the simulation: whenever $\mathcal{A}$ corrupts a party, $\mathcal{I}$ also corrupts the corresponding ideal party; whenever $\mathcal{A}$ instructs a corrupted party to output a message, $\mathcal{I}$ instructs the corresponding ideal party to output the message as well. If $\mathcal{A}$ deviates from the protocol in a way that is evident to honest real parties (e.g., refuses to send messages, or sends syntactically incorrect messages to honest parties), $\mathcal{I}$ halts the simulation (and the parties output $\perp$). An exception is when corrupt voters deviate from the protocol in this way: in this case the voter would be ignored by honest authorities (rather than stopping the election), and $\mathcal{I}$ forces an abstention for that voter. Except for the noted cases, we explictly specify when $\mathcal{I}$ sends commands to $\mathcal{F}^{(V)}$; the simulation is self-contained and only visible to $\mathcal{A}$ and the environment through the interactions of the simulated parties with the adversary.

5.6.1 Setup Phase

$\mathcal{I}$ chooses a random seed $s \in \{0,1\}^{\ell'}$ and simulates the random beacon, using $K'(s)$ to generate the part of the beacon used as input to the commitment-key generation algorithm, K (this will allow $\mathcal{I}$ to equivocate commitments). It then runs the ballot preparation stage by simulating A_1 and A_2 exactly according to protocol (or according to the real-world adversary's instructions, if one or both are corrupted). At the end of this stage, the adversary is committed to the contents of any ballots it sent.

5.6.2 Voting Phase

Honest Voter. When it receives a "v has voted" message from the voting functionality for an honest voter v, $\mathcal{I}$ begins simulating voter v running Protocol 5.1. The actions of $\mathcal{I}$ depend on whether the adversary has corrupted both A_1 and A_2:

Case 1: If both A_1 and A_2 are corrupt, $\mathcal{I}$ learns x_v from the voting functionality. It can then exactly simulate an honest voter voting x_v

Case 2: If at least one of the authorities is honest, $\mathcal{I}$ simulates a voter using a random value x'_v for the voter's choice value. This involves choosing a random commitment from the set received from A_1.

Corrupt Voter. Throughout this phase, $\mathcal{I}$ simulates a corrupt voter v by following $\mathcal{A}$'s instructions exactly. If both authorities were honest at the beginning of the voting stage (in particular, if the ballots were generated by $\mathcal{I}$), $\mathcal{I}$ computes $x_v = s_v + t_{(1,v)} + t_{(2,v)}$ and sends a **Vote** (v, x_v) command to $\mathcal{F}^{(V)}$. If at least one authority was corrupt at the beginning ot the voting stage, $\mathcal{I}$ sends nothing to the ideal functionality (but in this case it will be able to cast a vote in the tally phase). Denote by W the number of voters for which $\mathcal{I}$ did *not* cast a vote during the voting phase.

Voter Corrupted During Protocol. If a voter v is honest at the beginning of the phase, and is corrupted during the protocol, $\mathcal{I}$ learns the real choice x_v when the voter is corrupted. If both authorities were already corrupt, $\mathcal{I}$ already knows x_v so this information will not have changed any of the views in the simulation. If at least one authority was honest and $x'_v \neq x_v$, $\mathcal{I}$ has to rewrite its history (both that of the honest authority and that of voter v) to be consistent with the new information. In this case, it uses its ability to equivocate commitments to rewrite the value of $t_{(a,v)}$, where a is the index of an honest authority (this will also change the randomness of the commitments). The new value will satisfy $x_v = s_v + t_{(1,v)} + t_{(2,v)}$.

Authority Corrupted During Protocol. If, at the beginning of the voting phase, both authorities are honest and later one is corrupted, $\mathcal{I}$ does not need to rewrite its history, since it is consistent with the real world simulation.

If, at the beginning the voting phase, at least one authority is honest and sometime later both of the authorities become corrupted, $\mathcal{I}$ learns the choices of all previous voters at that point. In this case, when the last authority is corrupted, $\mathcal{I}$ may be forced to rewrite its history as well as that of the honest voters whose choices didn't match the guesses made by $\mathcal{I}$. It does this by using its ability to equivocate commitments in order to change only the private parts of the ballots which were not seen by $\mathcal{A}$ before the corruption.

Case 1: If A_1 is corrupted last, $\mathcal{I}$ can rewrite the value $t_{(1,v)}$ to match the voter's choice by equivocating on the commitments $c_{(1,v),1}, \ldots, c_{(1,v),m}$

Case 2: If A_2 is corrupted last, $\mathcal{I}$ can rewrite the value $t_{(2,v)}$ to match the voter's choice by equivocating on the commitment $c_{(2,v)}$

5.6.3 Tally Phase

This phase begins when a voting authority sends the **Tally** command to $\mathcal{F}^{(V)}$, or the adversary has corrupted an authority and decides to end the voting phase. $\mathcal{I}$ waits for $\mathcal{F}^{(V)}$ to announce the tally. The simulator's strategy now depends on which of the voting authorities is corrupt (note that from this stage on the adversary is static with regard to corrupting the voting authorities). We can assume w.l.o.g. that $\mathcal{A}$ corrupted the

relevant authorities at the start of the Setup phase: if only one authority is corrupted during the protocol, $\mathcal{I}$'s simulation is identical to the case where the authority was corrupt from the start, but chose to follow the protocol correctly. If both authorities are corrupted, $\mathcal{I}$ was required to rewrite its view at the time of the second corruption; however, the rewritten history is identical to a simulation in which both authorities were corrupted from the start and chose to follow the protocol honestly.

Case 1: **Neither voting authority is corrupt.** $\mathcal{I}$ generates vectors of random commitments for $d_{1,1},\ldots,d_{1,n}$, $d_{2,1},\ldots,d_{2,n}$, $d'_{1,1},\ldots,d'_{1,n}$ and $d'_{2,1},\ldots,d'_{2,n}$.

Using its ability to equivocate, $\mathcal{I}$ can open the commitments to any value. In particular, it can pass all the zero-knowledge proofs. and open the commitments $d_{1,1},\ldots,d_{1,n}$ to values that match the announced tally and are identically distributed to the outcome in the real world protocol.

Case 2: **Exactly one authority is corrupt.** $\mathcal{I}$ rewrites its provisional view for the honest authority to make it consistent with the announced tally, using a random permutation for the honest voters. It then simulates the honest authority according to protocol, using its provisional view. At the end of Protocol 5.3, $\mathcal{I}$ simulates the verifier running Protocol 5.5. If verification fails, $\mathcal{I}$ halts in the ideal world as well. Otherwise, $\mathcal{I}$ now knows the real-world tally: $\tau_0,\ldots,\tau_{m-1}$. This may be different from the ideal-world tally announced by $\mathcal{F}^{(V)}$; If the ideal-world tally can be changed to the real-world one by adding W votes and changing k votes (where W is the number of corrupt voters for whom $\mathcal{I}$ did not cast a vote in the voting phase), $\mathcal{I}$ sends the appropriate **Vote** commands to $\mathcal{F}^{(V)}$, then sends an **Announce** command with the updated tally. Otherwise, $\mathcal{I}$ outputs "tally failure" and halts. In this case the real and ideal worlds are very different; however, we prove that this happens with negligible probability (see Claim 5.5).

If an honest voter v is corrupted during the tally phase, $\mathcal{I}$ learns x_v and must provide the voter's provisional history to $\mathcal{A}$. If $x'_v \neq x_v$, $\mathcal{I}$ chooses one of the remaining honest voters v' for which $x'_{v'} = x_v$ and rewrites both voters' views by equivocating on the commitments generated by the honest authority. In the new views $x'_{v'} \leftarrow x'_v$ and $x'_v \leftarrow x_v$. Note that there will always be such an honest voter, since the provisional views of the simulated honest voters are consistent with the ideal-world tally, which is consistent with the inputs of the ideal voters.

Case 3: **Both authorities are corrupt.** In this case, $\mathcal{I}$ knows the inputs of all voters, so it can create a completely consistent view. $\mathcal{I}$ follows the instructions of the real-world adversary until $\xi_1,\ldots,\xi_n$ and $\rho_1,\ldots,\rho_n$ are published. It then simulates the verifier running Protocol 5.5. If verification fails, $\mathcal{I}$ halts in the ideal world as well. Otherwise, $\mathcal{I}$ now knows the real-world tally: $\tau_0,\ldots,\tau_{m-1}$. As in the case of a single corrupt authority, if the ideal-world tally can be changed to the real-world one by adding W votes and changing k votes, $\mathcal{I}$ sends the appropriate **Vote** commands to $\mathcal{F}^{(V)}$, then sends a **Announce** command with the real-world tally. Otherwise, $\mathcal{I}$ outputs "tally failure" and halts.

5.6.4 Indistinguishability of the Real and Ideal Worlds

To complete the proof of Theorem 5.1, we must show that the environment machine cannot distinguish between the real protocol executed in the real world, and $\mathcal{I}$'s simulation executed in the ideal world. This is equivalent to showing that the views of the environment in both cases are computationally indistinguishable.

To define the environment's view, it is helpful to explicitly describe the views of all the parties in the real world:

Verifier: The verifier's view consists of all the public information (and only that):

1. The output of the random beacon: R.
2. The commitment public key, $cpk = K(R)$, generated by A_1 in the Setup phase
3. The proof of correctness for the commitment public key (the output of P_K)
4. The audit bits of the voters: b_v for all $v \in [n]$.
5. The public and private parts of all the audit ballots: $B_{\vec{w}}$ for all $\vec{w} = (a, v, i)$ such that $i = 1 - b_v$.

6. Randomness for all the audit ballots: $r_{\vec{w}}$ for all $\vec{w} = (2, v, i)$ such that $i = 1 - b_v$ and $r_{\vec{w},j}$ for all $v \in [n]$, $j \in \mathbb{Z}_m$ and $\vec{w} = (1, v, 1 - b_v)$.
7. Public part of the ballots for cast ballots: s_v,$c_{(1,v),j}$ and $c_{(2,v)}$ for all $v \in [n]$ and $j \in \mathbb{Z}_m$.
8. Public proofs of tally correctness: $d'_{a,1}, \ldots, d'_{a,n}$, $\delta_{a,1}, \ldots, \delta_{a,n}$ for $a \in \{1, 2\}$ and the transcripts of the zero-knowledge proofs that these were generated correctly.
9. The Opening of the shuffled commitments: $\xi_1, \ldots, \xi_n$ and $\rho_1, \ldots, \rho_n$.

Voter v: The voter's view includes all the public information (the verifier's view) in addition to:

1. The voter's input: x_v
2. The private part of the voter's ballots: $t_{(1,v)}$ and $t_{(2,v)}$

Authority A_1: The view of this authority consists of the verifier's view, and in addition:

1. The public keys for the encryption schemes: $pk^{(\mathcal{M})}$ and $pk^{(\mathcal{R})}$.
2. The secret keys for the encryption schemes: $sk^{(\mathcal{M})}$ and $sk^{(\mathcal{R})}$.
3. The private parts of the voters' ballots: $t_{(1,v)}$ for all $v \in [n]$
4. The randomness for the commitments in the voters' ballots: $r_{\vec{w},j}$ for all $v \in [n]$, $j \in \mathbb{Z}_m$ and $\vec{w} = (1, v, b_v)$
5. The permutation σ_1 and the values $u_{1,1}, \ldots, u_{1,n}$, $u'_{1,1}, \ldots, u'_{1,n}$ and $z_{1,1}, \ldots, z_{1,n}$
6. The secret random coins for the encryptions sent to A_2.
7. The encryptions received from A_2: $e^{(\mathcal{M})}_{2,1}, \ldots, e^{(\mathcal{M})}_{2,n}$ and $e^{(\mathcal{R})}_{2,1}, \ldots, e^{(\mathcal{R})}_{2,n}$ and their contents.
8. The secret random coins used in the zero-knowledge proofs in which A_1 was the prover.

Authority A_2: The view of this authority consists of the verifier's view, and in addition:

1. The public keys for the encryption schemes: $pk^{(\mathcal{M})}$ and $pk^{(\mathcal{R})}$
2. The private parts of the voters' ballots: $t_{(2,v)}$ for all $v \in [n]$
3. The randomness for the commitments in the voters' ballots: $r_{\vec{w}}$ for all $v \in [n]$ and $\vec{w} = (2, v, b_v)$
4. The permutation σ_2 and the values $u_{2,1}, \ldots, u_{2,n}$, $u'_{2,1}, \ldots, u'_{2,n}$ and $z_{2,1}, \ldots, z_{2,n}$
5. The encryptions received from A_1: $e^{(\mathcal{M})}_{1,1}, \ldots, e^{(\mathcal{M})}_{1,n}$ and $e^{(\mathcal{R})}_{1,1}, \ldots, e^{(\mathcal{R})}_{1,n}$.
6. The secret random coins for the encryptions sent to A_1.
7. The secret random coins used in the zero-knowledge proofs in which A_2 was the prover.

$\mathcal{A}$: The real-world adversary's view consists of the verifier's view, any messages it receives from the environment and the view of any party it corrupts.

Environment: The environment's view consists of $\mathcal{A}$'s view, and in addition, the inputs of all the voters: $x_1, \ldots, x_n$ and its own random coins.

In the ideal world, the environment's view is the same, except that the views of the real-world parties are those simulated by $\mathcal{I}$. The other difference is that in the ideal world, the output of the verifier seen by the environment is the tally produced by $\mathcal{F}^{(V)}$ (none of the other parties have output).

The following lemma completes the proof of Theorem 5.1:

Lemma 5.3. *The enviroment's view in the real world is computationally indistinguishable from its view in the ideal world.*

Proof. Setup and Voting Phases. First, note that in the Setup and Voting phases of the protocol, as long as $\mathcal{I}$ can perfectly equivocate commitments, the views are identically distributed. This is because the views of the simulated parties are always kept in a perfectly consistent state, given the knowledge $\mathcal{I}$ has about the voters' inputs. Whenever a simulated party is corrupted, $\mathcal{I}$ gains enough information to perfectly rewrite its view so that it is consistent with the previous view of the environment.

Thus, the only possible way the views could differ is if $\mathcal{I}$ cannot equivocate commitments. This can occur only if the commitment key published in the Setup phase is not the one chosen by $\mathcal{I}$. But by the definition of the commitment scheme, the probability that A_1 can generate a key that passes verification but does not allow equivocation is negligible.

Tally Phase. In the Tally phase, $\mathcal{I}$ deviates from a perfect simulation only when at least one of the authorities is corrupt. The possible reasons $\mathcal{I}$'s simulation may be imperfect are:

1. The **Vote** commands sent to $\mathcal{F}^{(V)}$ by the simulator are not consistent with the tally in the simulation. This would cause the output of the verifier in the ideal world to differ from its output in the real world. Note that the inconsistency will be noticable only if the ideal-world tally is "far" from the real-world tally: If it can be modified by adding W arbitrary votes and changing up to k votes, $\mathcal{I}$ will be able to "fix" the ideal-world tally. The probability that $\mathcal{I}$ fails in this way is negligible, as we prove in Claim 5.5.
2. If A_1 is honest and A_2 is corrupt:
   (a) The encryptions $e_{1,1}^{(\mathcal{M})}, \ldots, e_{1,n}^{(\mathcal{M})}$ and $e_{1,1}^{(\mathcal{R})}, \ldots, e_{1,n}^{(\mathcal{R})}$ may be inconsistent with the environment's view in the ideal world: the encrypted values contain $\mathcal{I}$'s simulated inputs for the honest voters (which may differ from the real inputs of the voters). The environment, however, knows the real inputs of the voters. Although the environment's views may be statistically far, the semantic security of the encryption scheme implies that the encryptions of two different messages are computationally indistinguishable, even when the messages are known. Hence, the environment's views of the encryptions in the real and ideal world are computationally indistinguishable.
   (b) The distribution of $\xi_1 \ldots, \xi_n$ may be inconsistent with the environment's view in the ideal world. This can happen because $\xi_1, \ldots, \xi_n$ is a permutation of $x_1, \ldots, x_n$ only modulo m; the actual values depend on the permutations σ_1, σ_2 the values $t_{(1,1)}, \ldots, t_{(1,n)}$, $t_{(2,1)}, \ldots, t_{(2,n)}$, $z_{(1,1)}, \ldots, z_{(1,n)}$, $z_{(2,1)}, \ldots, z_{(2,n)}$ and on the order of the simulated votes chosen by $\mathcal{I}$ (which may differ from the actual order). However, the statistical distance between the views of $\xi_1, \ldots, \xi_n$ in the real and ideal worlds is negligible. Intuitively, this is because A_2 can only add multiples of m up to 2^k. No matter what it does, once A_1 adds a multiple of m uniformly chosen up to 2^{2k}, the output distribution will be nearly uniform (only an exponentially small fraction of the output values will have different probabilities). We prove this formally in Claim 5.4.
3. If A_1 is corrupt and A_2 is honest, the distribution of $\xi_1 \ldots, \xi_n$ may be inconsistent with the environment's view in the ideal world. This can happen for exactly the same reason as it does when A_2 is corrupt and A_1 honest: the extra multiples of m in $\xi_1 \ldots, \xi_n$ may contain information about the order of the votes. As in the previous case, the statistical distance between the views of $\xi_1, \ldots, \xi_n$ in the real and ideal worlds is negligible (here the intuition is that A_1 can only set t_v to be at most $2m$, while A_2 will add a multiple of m uniformly chosen up to 2^k. We omit the formal proof, as it is almost identical to the proof of Claim 5.4.

Verification Phase. Since the verifier cannot be corrupted and uses only public information, the views of the environment in the verification phase are identical in the real and ideal world, except in case of a tally failure (which we prove occurs with negligible probability). □

The proof of Lemma 5.3 is completed by the claims below:

Claim 5.4. *Fix any permutation σ_2 and values $t_{(1,1)}, \ldots, t_{(1,n)} \in \mathbb{Z}_{2m}$, $t_{(2,1)}, \ldots, t_{(2,n)} \in \mathbb{Z}_{2m}$, $z_{(2,1)}, \ldots, z_{(2,n)} \in \mathbb{Z}_{2^k}$ and $x_1, \ldots, x_n \in \mathbb{Z}_m$. For a permutation π, let $\Xi_\pi = \xi_1 \ldots, \xi_n$ be the output of the protocol when it is run with the fixed values, permuted voter inputs $x_{\pi(1)}, \ldots, x_{\pi(n)}$ and when σ_1, $z_{(1,1)}, \ldots, z_{(1,n)}$ are chosen randomly (following the protocol specification).*

Then for any two permutations π_1 and π_2, the statistical difference between Ξ_{π_1} and Ξ_{π_2} is at most $n2^{-k+1}$.

Proof. Note that $\xi_i = x_{\pi(\sigma_2(\sigma_1(i)))} + (f_{\sigma_2(\sigma_1(i))} + z_{2,\sigma_1(i)} + z_{1,i})m$, where $f_i \in \mathbb{Z}_4$. First, we define a new random variable $\Xi'_\pi = (\xi'_1 \ldots, \xi'_n)$, whose value is similar to Ξ_π, except without the fixed multiples of m: $\xi'_i = x_{\pi(\sigma_2(\sigma_1(i)))} + z_{1,i}m$. Note that $\pi \circ \sigma_2 \circ \sigma_1$ is a random permution for any fixed π and σ_2, and $\{z_{1,i}\}$ are identically and independently distributed. Hence, for any two permutations π_1 and π_2, Ξ'_{π_1} and Ξ'_{π_2} are identically distributed. Next, we will show that for any permutation π, the statistical difference between Ξ'_π and Ξ_π is at most $n2^{-k}$. Thus, by the triangle inequality, the statistical difference between Ξ_{π_1} and Ξ_{π_2} is at most $n2^{-k+1}$. Denote $\vec{\xi} \doteq (\xi_1 \ldots, \xi_n)$. By definition, the statistical difference between the two distributions is:

$$\begin{aligned}
\Delta(\Xi'_\pi, \Xi_\pi) &= \frac{1}{2}\sum_{\vec{\xi}} \left|\Pr[\Xi'_\pi = \vec{\xi}] - \Pr[\Xi_\pi = \vec{\xi}]\right| \\
&= \frac{1}{2n!}\sum_{\vec{\xi}} \left|\sum_{\sigma_1} \left[\Pr[\Xi'_\pi = \vec{\xi} \mid \sigma_1] - \Pr[\Xi_\pi = \vec{\xi} \mid \sigma_1]\right]\right| \\
&\leq \frac{1}{2n!}\sum_{\vec{\xi},\sigma_1} \left|\Pr[\Xi'_\pi = \vec{\xi} \mid \sigma_1] - \Pr[\Xi_\pi = \vec{\xi} \mid \sigma_1]\right|.
\end{aligned}$$

Denote $p \doteq \Pr[\Xi_\pi = \vec{\xi} \mid \sigma_1]$ and $p' \doteq \Pr[\Xi'_\pi = \vec{\xi} \mid \sigma_1]$. Let $\zeta_i \doteq f_{\sigma_2(\sigma_1(i))} + z_{2,\sigma_1(i)}$ (the fixed multiples of m) and $\theta_i \doteq x_{\pi(\sigma_2(\sigma_1(i)))}$ (the permuted inputs). Note that since $f_{\sigma_2(\sigma_1(i))} < 4 \leq 2^k$ we have $0 \leq \zeta_i \leq 2^{k+1}$. By our definition of p:

$$p = \Pr[\bigwedge_{i=1}^{n} (\theta_i + \zeta_i m + z_{1,i}m = \xi_i) \mid \sigma_1]$$

(where the probability is only over $\{z_{1,i}\}$). Hence, $p \neq 0$ only if for all $1 \leq i \leq n$, it holds that $\theta_i \equiv \xi_i \pmod m$ and $z_{1,i} = \frac{1}{m}(\xi_i - \theta_i) - \zeta_i$.

Since $z_{1,i}$ is uniformly chosen in $\mathbb{Z}_{2^{2k}}$ (and in particular $0 \leq z_{1,i} \leq 2^{2k}$), it follows that $p \neq 0$ only if for all $1 \leq i \leq n$:

$$\zeta_i \leq \frac{1}{m}(\xi_i - \theta_i) \leq 2^{2k} + \zeta_i.$$

When $p > 0$, then $p = 2^{-2kn}$ (since for every i there is exactly one "good" choice for $z_{1,i}$). Similarly, $p' \neq 0$ only if for all $1 \leq i \leq n$, it holds that $\theta_i \equiv \xi_i \pmod m$ and $\frac{1}{m}(\xi_i - \theta_i) \leq 2^{2k}$. Hence, if $p \neq p'$ then for all $1 \leq i \leq n$: $\theta_i \equiv \xi_i \pmod m$ and either

Case 1: $p \neq 0, p' = 0$**:** for all $1 \leq i \leq n$, $\zeta_i \leq \frac{1}{m}(\xi_i - \theta_i) \leq 2^{2k} + \zeta_i$ and there exists i such that $2^{2k} < \frac{1}{m}(\xi_i - \theta_i) \leq 2^{2k} + \zeta_i$ or

Case 2: $p = 0, p' \neq 0$**:** for all $1 \leq i \leq n$, $\frac{1}{m}(\xi_i - \theta_i) \leq 2^{2k}$ and there exists i such that $\frac{1}{m}(\xi_i - \theta_i) \leq \zeta_i$.

Note that both p and p' depend only on $\vec{\xi}$ (and not on σ_1). So we can write

$$\Delta(\Xi'_\pi, \Xi_\pi) \leq \frac{1}{2}\sum_{\vec{\xi}} |p(\vec{\xi}) - p'(\vec{\xi})| = 2^{-2kn-1}\left|\left\{\vec{\xi} \mid p(\vec{\xi}) \neq p'(\vec{\xi})\right\}\right| \tag{5.6.1}$$

We can list all such $\vec{\xi}$ (possibly overcounting) in the following way:

1. Choose $i \in [n]$ (n possibilities)
2. Choose one of the two cases above for which $p \neq p'$ (2 possibilities):
3. Depending on which is chosen, either:

Case 1: choose ξ_i such that $2^{2k} + \zeta_i \geq \frac{1}{m}(\xi_i - \theta_i) > 2^{2k}$ or
Case 2: choose ξ_i such that $\frac{1}{m}(\xi_i - \theta_i) \leq \zeta_i$

(in either case, since $m \geq 2$, there are $\frac{1}{m}\zeta_i \leq 2^k$ possibilities)

4. For all $1 \leq j \leq n, j \neq i$:

Case 1: choose ξ_j such that $\zeta_j \leq \frac{1}{m}(\xi_j - \theta_j) \leq 2^{2k} + \zeta_j$ or
Case 2: choose ξ_j such that $\frac{1}{m}(\xi_j - \theta_j) \leq 2^{2k}$

(in either case, there are $\prod_j \frac{1}{m} 2^{2k} \leq 2^{2(k-1)(n-1)} \leq 2^{2kn-k}$ possibilities)

Thus, the number of $\vec{\xi}$s that satisfy $p(\vec{\xi}) \neq p'(\vec{\xi})$ is bounded by $n2^{2kn-k+1}$. Plugging this in to inequality 5.6.1, we get $\Delta(\Xi'_\pi, \Xi_\pi) \leq 2^{-2kn-1} \cdot nn2^{2kn-k+1} \leq n2^{-k}$. □

Claim 5.5. *The probability that the ideal simulator terminates the simulation by outputting "tally failure" is a negligible function of k.*

Proof. $\mathcal{I}$ outputs "tally failure" only if the simulated verifier did not abort, and the ideal tally (computed by $\mathcal{F}^{(V)}$) cannot be changed to the real tally (the one output by the simulated verifier) by adding W votes and changing k votes (where $W = 0$ if both authorities were honest at the beginning of the voting phase, and otherwise W is the number of voters who were corrupt during the voting phase).

Assume, in contradiction, that the environment/real-world adversary pair can cause a tally failure with probability ϵ. We will show how to use such an environment/adversary pair to break the binding property of the commitment scheme.

That is, there exists a machine M that can produce, with probability polynomial in ϵ, a commitment c and m_1, r_1, m_2, r_2, $m_1 \neq m_2$ such that $c = C(m_1, r_1) = C(m_2, r_2)$.

M works by simulating the entire ideal world (including the enviroment machine, $\mathcal{F}^{(V)}$ and the ideal-world adversary, $\mathcal{I}$). After each of the public zero-knowledge proofs of knowledge in the Tally phase (steps 9, 10 and 13 in Protocol 5.2), M rewinds the environment/real-adversary pair and runs their corresponding knowledge extractors. Let ϵ' be the probability that the knowledge extractors succeed for all the proofs. Since each knowledge extractor succeeds with probability polynomial in the probability that the verifier accepts, and the verifier accepts all the proofs with probability at least ϵ (otherwise it would not cause a tally failure), $\epsilon' = poly(\epsilon)$.

The proofs of knowledge allow M to extract the permutations for the shuffles performed by the authorities, along with the randomizing values (and ensure that the extracted values are in the correct ranges). Since the final opened commitments are consistent with the real tally, the extracted value of the original commitment vector (before both shuffles and randomization) must also be consistent with the real tally. Let $(x''_v, (r'')_v)$ denote the opening of the commitment $d_{3,v}$ that is output by M.

For any execution of the protocol in the ideal world, the ideal tally consists of the tally of all voters (if both authorities were honest at the beginning of the Voting phase) or only of the honest voters (otherwise) This is because $\mathcal{I}$ does not send a **Vote** command on behalf of corrupt voters if one of the authorities was corrupt at the beginning of the Voting phase). A tally failure means that even after adding W votes to the ideal tally, it differs from the real tally by more than k votes.

Case 1: **All the ballots were generated by honest authorities.** Since the honest authorities are simulated by $\mathcal{I}$ (which is, in turn, simulated by M), this means all the ballots were generated by M. In particular, there exists a voter v such that $x_v \neq x''_v$, and for which M knows $r_{(1,v)}, r_{(2,v)}$. Since $d_{3,v} = C(x_v, r_{(1,v)} + r_{(2,v)})$ by the construction of d_3, and $d_{3,v} = C(x''_v, r''_v)$ by the knowledge extraction, M can open $d_{3,v}$ in two different ways.

Case 2: **At least some of the ballots were generated by a corrupt authority.** In this case, there must still be at least k honest voters $v_1, \ldots, v_k$ such that $x_{v_i} \neq x''_{v_i}$. (since W is the number of corrupt voters). To break the commitment scheme, M rewinds the ideal-world to the end of the setup phase (after all the ballots have been committed), then reruns the simulation with new randomness for $\mathcal{I}$. In particular, the honest voters' audit bits are new random bits. The probability that both simulations end with tally failures is at least ϵ'^2. In particular, when this occurs, the authorities correctly opened all the audited commitments in the second simulation. The probability that all k of the voters will have the same audit bits in both simulations is 2^{-k}. If this does not occur, the corrupt authorities will publish $r_{(1,v_i)}, r_{(2,v_i)}$ such that $d_{3,v_i} = C(x_{v_i}, r_{(1,v_i)} + r_{(2,v_i)})$ for at least one of the honest voters. Again, this will allow M to open d_{3,v_i} in two different ways.

Taking a union bound, in the worst case the probability that M succeeds is at least $\epsilon'^2 - 2^{-k} = poly(\epsilon) - 2^{-k}$. If ϵ is non-negligible, then M can break the binding property of the commitment with non-negligible probablity. □

5.7 Proof of Receipt-Freeness (Theorem 5.2)

The definition of receipt-freeness shares many elements with the security definitions of the universal composability framework, hence the proofs will also be very similar. In both cases, we must show that the view of an adversary in the real world is indistinguishable from its view of a simulated protocol execution in an ideal world, where there exists an ideal simulator, $\mathcal{I}$. There is one main difference: the adversaries (in both the real and ideal worlds) can perform an additional action: coercing a party. Each party has, in addition to their input, a "coercion-response" bit and a fake input (both also hidden from the adversary). The coercion-response bit determines how they will respond to coercion. When the bit is 1, a coerced party behaves exactly as if it were corrupted. When the bit is 0, however, instead it executes a "coercion-resistance strategy". In the ideal world, the strategy is to send its real input to $\mathcal{F}^{(V)}$, but lie to the adversary and claim its input was the fake input. In the real world, the coercion-resistance strategy is specified as part of the protocol (see Section 5.5.2).

Like the proof of security in the UC model, the proof of Theorem 5.2 has two parts: the description of the simulation run by $\mathcal{I}$ in the ideal world, and a proof that the adversary's views in the real and ideal worlds are indistinguishable. The simulation is almost identical to the simulation in the proof of Theorem 5.1. The difference is what happens when $\mathcal{A}$ coerces a voter (this does not occur in the original simulation). $\mathcal{I}$ handles coercions of voters exactly like corruptions (i.e., they do not run the coercion-resistance strategy), with the following modifications:

1. $\mathcal{I}$ coerces rather than corrupts the ideal voter.
2. If voter v was coerced before step 4 of the Voting phase (i.e., before entering the voting booth), $\mathcal{I}$ sends a **Vote** $v, *$ command to $\mathcal{F}^{(V)}$ (signifying a forced random vote), instead of a standard vote command.
3. If the adversary causes the voter to behave in a way that would invalidate her vote (e.g., send syntactically incorrect commands, or abort before casting a ballot), $\mathcal{I}$ sends a **Vote** $v, \perp$ command to $\mathcal{F}^{(V)}$ (signifying a forced abstention).

5.7.1 Indistinguishability of the Real and Ideal Worlds

To complete the proof of Theorem 5.2, we prove the following lemma:

Lemma 5.6. *The adversary's view in the real world is identically distributed to its view in the ideal world.*

Proof. Note that we only need to consider the case where both voting authorities are honest. Hence, the adversary's view consists only of the verifier's view (the random beacon and the information on the bulletin board) and the views of corrupted voters and coerced voters (which, in the real world, may be fake, depending on the value of their coercion-response bit).

We can assume the adversary also determines the inputs, fake inputs, and coercion-response bits of all the voters (these are not given to $\mathcal{I}$).

First, note that the generated ballots always consist of uniformly random values, independet of the voters' inputs, and the private part of the ballot is independent (as a random variable) of the public part of the ballot (since the commitments are perfectly hiding). Thus, the view generated by the coercion-resistance strategy is identically distributed to the view of an honest voter in the real world, which in turn is identically distributed to the view simulated by $\mathcal{I}$.

Since, when authorities are honest, corrupt voters can have no effect on other voters (except by changing the final tally), we can assume w.l.o.g. that the adversary does not corrupt voters (it can simply determine the inputs for honest voters). The only remaining possibility for a difference between the adversary's views is the joint distribution of the inputs, fake inputs, coercion-resistance bits and the final tally. However, since $\mathcal{I}$ can perfectly equivocate on commitments, the simulated tally it produces will always be consistent with the ideal tally, and distributed identically to the tally in the real world. □

5.8 Discussion and Open Problems

Multiple Questions on a Ballot. As shown in the "illustrated example", our voting protocol can be easily adapted to use multiple questions on the same ballot. If there are many questions, the pattern of votes on a single ballot may uniquely identify a voter, hence tallying the questions together may violate voter privacy. In this case, the tally protocol should be performed separately for each question (or for each small group).

More than Two Authorities. We described the protocol using two authorities. The abstract protocol can be extended to an arbitrary number of authorities (although this may require finding a threshold version of the encryption scheme). However, a major stumbling block is the human element: even for two authorities this protocol may be difficult for some users. Dividing a vote into three parts will probably be too complex without additional ideas in the area of human interface.

Receipt-Freeness with a Corrupt Authority. The current protocol is not receipt-free if even one of the authorities is corrupt. Note that this is not a problem in the proof, but in the protocol itself (if the voter does not know which authority is corrupt): the voter can't tell which of the ballots the coercer will have access to, so she risks getting caught if she lies about the value she erased from the ballot. It is an interesting open question whether this type of attack can be prevented.

Better Human Interface. Probably the largest hurdle to implementing this protocol is the human interface. Devising a simple human interface for modular addition could prove useful in other areas as well.

APPENDIX

5.A Homomorphic Commitment and Encryption Schemes Over Identical Groups

Our voting scheme requires a perfectly private commitment scheme with "matching" semantically-secure encryption schemes. The commitment scheme's message and randomizer spaces must both be groups, and the commitment scheme must be homomorphic (separately) in each of the groups. There must be a matching encryption scheme for each group, such that the encryption scheme's message space is homomorphic over that group.

To meet these requirements, we propose using the standard Paillier encryption scheme. The Paillier encryption public key consists of an integer $N = p_1 p_2$, where p_1 and p_2 are safe primes, and an element $e \in \mathbb{Z}^*_{N^2}$. The private key is the factorization of N. The encryption plaintext is in the group $\mathbb{Z}_n$

For the commitment scheme, we propose a modified version of the Pedersen commitment scheme where both messages and randomness are also in the group $\mathbb{Z}_N$. The commitment public key consists of N (the same value as the encryption public key) along with random generators g, h in the order N subgroup of $\mathbb{Z}^*_{4N+1}$. Below we give the details of this construction.

5.A.1 Modified Pedersen

The abstract version of Pedersen commitment has a public key consisting of a cyclic group G and two random generators $g, h \in G$ such that $\log_g h$ is not known to the committer. The cryptographic assumption is that $\log_g h$ is infeasible to compute.

The message and randomizer spaces for this scheme are both $\mathbb{Z}_{|G|}$. $C(m, r) \doteq g^m h^r$. Since g and h are both generators of the group, for any m, when r is chosen at random $g^m h^r$ is a random group element. Therefore, this scheme is perfectly hiding. If an adversary can find $(m_1, r_1) \neq (m_2, r_2)$ such that $g^{m_1} h^{r_1} = g^{m_2} h^{r_2}$, then it can compute $\log_g h = \frac{m_2 - m_1}{r_1 - r_2}$, violating the cryptographic assumption. Hence the scheme is computationally binding. It is easy to see that the scheme is homomorphic.

Finally, if we choose $g, h = g^x$, where g is chosen randomly and x is chosen randomly such that g^x is a generator, we get an identically distributed public key, but knowing x it is easy to equivocate.

In the "standard" implementation of Pedersen, G is taken to be the order q subgroup of $\mathbb{Z}_p^*$, where $p = 2q+1$ and both p and q are prime (i.e., p is a safe prime). g and h are randomly chosen elements in this group. The discrete logarithm problem in G is believed to be hard when p is a safe prime chosen randomly in $(2^n, 2^{n+1})$.

Our modified version of Pedersen takes G to be the order $N = p_1p_2$ subgroup of $\mathbb{Z}_{4n+1}^*$, where p_1 and p_2 are safe primes and $4n+1$ is also prime (we can't use $2n+1$, since that is always divisible by 3 when p_1 and p_2 are safe primes). The computational assumption underlying the security of the commitment scheme is that, when p_1 is a random safe prime and g and h are random generators of G, computing $\log_g h$ is infeasible. Note that it is not necessary to keep the factorization of N secret (in terms of the security of the commitment scheme), but knowing the factorization is not required for commitment.

5.A.2 Choosing the Parameters

The connection between the keys for the commitment and encryption schemes makes generating them slightly tricky. On one hand, only one of the authorities can know the private key for the encryption scheme (since its purpose is to hide information from the other authority). On the other hand, the security of the commitment must be publicly verifiable (even if both authorities are corrupt), hence we cannot allow the authorities to choose the parameters themselves. Moreover, for the commitment to be binding, N must have a large *random* prime factor, and g and h must be chosen randomly.

Below, we sketch our proposed protocol for verifiably generating the system parameters. Protocol 5.6 generates the parameters for the Paillier encryption, and Protocol 5.7 the parameters for the modified Pedersen commitment. The basic idea is that A_1 can use zero-knowledge proofs to show that the modulus $N = p_1p_2$ is product of two safe primes, and to prove that p_1 is a *random* safe prime: basically, that it is the outcome of a coin-flipping protocol that A_1 conducts with the random beacon (this is accomplished by the loop at step 1 in Protocol 5.6). Technically, the proofs should be output by Protocol 5.7 rather than 5.6 (since they are required to prove the security of the commitment scheme). However, to clarify the presentation we have included them in the encryption key-generation protocol.

The generators g, h for the order N subgroup of $\mathbb{Z}_{4N+1}^*$, needed for the Pedersen scheme, are simply random elements of $\mathbb{Z}_{4N+1}$ (chosen using the random beacon). This works because a random element $g \in_R \mathbb{Z}_{4N+1}$ will be an element of $\mathbb{Z}_{4N+1}^*$ with order $o(g) \in \{N, 2N, 4N\}$ except with negligible probability $(O(1/\sqrt{N}))$, assuming p_1 and p_2 are of order $O(\sqrt{N})$). If the order of g is $2N$ (resp. $4N$), then g^2 (resp. g^4) will have order N (this computation can be replicated by the verifiers).

The protocols require integer commitments that have an efficent zero-knowledge proof of multiplication. We need an integer commitment scheme whose setup can be performed using a random beacon (rather than a trusted party). One such possibility is the Damgård-Fujisaki scheme [33], when instantiated using class groups rather than an RSA modulus. For the zero-knowledge proofs that p_1 and p_2 are safe primes, we can use the techniques of Camenisch and Michels [14].

Note that if we had a trusted third party to help with setup, we could significantly simplify it. Even a third party that is only trusted *during the setup* could help (for instance, by allowing us to use Damgård-Fujisaki with an RSA modulus generated by the third party).

5.B Zero-Knowledge Proofs of Knowledge

Our protocols require proving statements in zero-knowledge about committed values. Since we use perfectly-hiding commitments, proving that there exists an opening of a commitment with some property is meaningless: there exist openings of the commitment to every value. Instead, we use zero-knowledge proofs of knowledge [6]. Roughly, there exists an efficient "knowledge extractor" that, given oracle access to a prover that succeeds with with some non-negligible probability, can output a value consistent with what the prover claims to know.

In this section we briefly describe the zero-knowledge proof subprotocols. These are all honest-verifier, public-coin, zero-knowledge proofs of knowledge, using standard cut-and-choose techniques. When they are used publicly (i.e., on the bulletin board), the verifier's coins are taken from the random beacon, hence

Protocol 5.6 Key Generation for Encryption (KG)

Input: Security parameter k

repeat {Generate a verifiable commitment C_1 to a random safe prime: p_1}
  Choose a random $p' \in_R \mathbb{Z}_{2^k}$
  Publish a commitment C' to p'
  Interpret the next output of the random beacon as a number $p'' \in_R \mathbb{Z}_{2^k}$
  if $p_1 = p' + p'' \pmod{2^k}$ is a safe prime **then**
    Publish a commitment C_1 to p_1
    Prove in zero-knowledge (using the random beacon) that C_1 is a commitment to a safe prime, and that it is the sum of p'' and the committed value of C'. {if the commitment is statistically hiding, this will be a zero-knowledge proof of knowledge}.
  else
    Publicly open the commitment C', revealing p'.
  end if
until p_1 is a safe prime
Privately choose a random k-bit safe prime p_2, such that $4p_1p_2 + 1$ is prime.
Publish $N = p_1p_2$
Publish a commitment C_2 to p_2
Prove in zero-knowledge that C_2 is a commitment to a safe prime, and that the product of the values committed to in C_1 and C_2 is N.
Run the standard Paillier key generation using N as the modulus.

Protocol 5.7 Key Generation for Commitment (K)

Input: Modulus N output by KG (Protocol 5.6)

Interpret the next output of the random beacon as elements $g, h \in Z^*_{4N+1}$.
for $x \in \{g, h\}$ **do**
  while $x^N \not\equiv 1 \pmod{N^2}$ **do**
    $x \leftarrow x^2$
  end while
end for
Output g, h and N.

the honest-verifier assumption makes sense. Although more efficient protocols exist for these applications [11, 44], for the purpose of this paper we concentrate on simplicity and ease of understanding.

5.B.1 Proof That Two Commitments Are Equivalent

In step 7 of Protocol 5.3, authority A_1 must prove that an encryption it generated has the same value as a previously published commitment. The following subprotocol works for any two homomorphic commitment schemes (in this case we can consider the encryption a commitment scheme), as long as their message groups are isomorphic. Since our commitment scheme is symmetric (we can consider it a commitment to the randomness), this protocol works for that case as well.

We will assume two commitment schemes C_1 and C_2, with message space $\mathcal{M}$, commitment spaces $\mathcal{C}_1, \mathcal{C}_2$ (resp.) and randomness groups $\mathcal{R}_1, \mathcal{R}_2$ (resp.).

Let $c_1 \in \mathcal{C}_1$ and $c_2 \in \mathcal{C}_2$ be the commitments for which we are proving "equivalence". Formally, what the protocol proves is that the prover knows a value $x \in \mathcal{M}$ and values $r_1 \in \mathcal{R}_1, r_2 \in \mathcal{R}_2$ such that $c_1 = C_1(x, r_1)$ and $c_2 = C_2(x, r_2)$. The complete protocol consists of k repetitions of Protocol 5.8; the probability that the prover can cheat successfully is exponentially small in k.

Protocol 5.8 Zero-Knowledge Proof That Two Committments Are Equivalent

Input: Verifier receives $c_1 \in \mathcal{C}_1$ and $c_2 \in \mathcal{C}_2$, Prover receives $x \in \mathcal{M}$ and $r_1 \in \mathcal{R}_1, r_2 \in \mathcal{R}_2$ such that $c_1 = C_1(x, u_1)$ and $c_2 = C_2(x, u_2)$

1: Prover chooses values $s \in_R \mathcal{M}$ and $u_1 \in_R \mathcal{R}_1, u_2 \in \mathcal{R}_2$
2: Prover sends to verifier: $d_1 \doteq C_1(x + s, r_1 + u_1)$ and $d_2 \doteq C_2(x + s, r_2 + u_2)$
3: Verifier sends to the prover a random bit $b \in_R \{0, 1\}$
4: **if** $b = 0$ **then**
5: Prover sends to the verifier: s, u_1 and u_2.
6: Verifier checks that $d_1 = c_1 C(s, u_1)$ and $d_2 = c_2 C(s, u_2)$
7: **else**
8: Prover sends to the verifier: $x + s$, $r_1 + u_1$ and $r_2 + u_2$
9: Verifier checks that $d_1 = C(x + s, r_1 + u_1)$ and $d_2 = C(x + s, r_2 + u_2)$
10: **end if**

5.B.2 Proof of Commitment Shuffle

We say a vector of commitments is "a valid shuffle" of a second vector if, whenever the prover can open one vector, it can open both vectors to permutations of the same set of values. Note that this property is an equivalence relation (with respect a single prover).

An important point is that we do not require the prover to show that it can *open* either of the vectors of commitments. This property is necessary, because our voting protocol requires a voting authority to shuffle commitments that it does not know how to open.

To construct a zero-knowledge proof, we use a standard cut-and-choose technique. Roughly, the prover publishes a *third* vector of commitments, then, according to the verifier's choice, it either shows that this third vector is a valid shuffle of the first, or that third vector is a valid shuffle of the second. If it is both, the first vector must be a valid shuffle of the second (and vice-versa).

Formally, let $c_1, \ldots, c_n \in \mathcal{C}$ and $c'_1, \ldots, c'_n \in \mathcal{C}$ be commitments. The prover must show that it knows a permutation $\sigma\colon [n] \mapsto [n]$ and values $r_1, \ldots, r_n \in \mathcal{R}$ such that for all $i \in [n]$: $c'_i = c_{\sigma(i)} \cdot C(0, r_i)$.

The protocol consists of k repetitions of Protocol 5.9 (where k is the security parameter).

5.B.3 Proof that a Committed Value is in $\mathbb{Z}_{2^k}$

In steps 9 and 10 of Protocol 5.2, each authority must prove that a committed value "is in an appropriate range".

Protocol 5.9 Zero-Knowledge Proof of Valid Shuffle

Prover chooses a random permutation $\pi\colon [n] \mapsto [n]$
Prover chooses values ${r'}_1, \ldots, {r'}_n \in_R \mathcal{R}$.
for $1 \leq i \leq n$ **do**
  Prover sends to the verifier: $d_i \doteq {c'}_{\pi(i)} \cdot C(0, r'_i)$
end for
Verifier sends to the prover a random bit $b \in_R \{0, 1\}$
if $b = 0$ **then**
  Prover sends to the verifier: π
  Prover sends to the verifier: ${r'}_1, \ldots, {r'}_n$
  for $1 \leq i \leq n$ **do**
    Verifier checks that $d_i = {c'}_{\pi(i)} \cdot C(0, r'_i)$
  end for
else
  Prover sends to the verifier: $\sigma \circ \pi$
  for $1 \leq i \leq n$ **do**
    Prover sends to the verifier: $s_i \doteq r_{\pi(i)} + r'_i$
    Verifier checks that $d_i = c_{\sigma\circ\pi(i)} \cdot C(0, s_i)$
  end for
end if

Formally, $z \in \mathcal{C}$ be a commitment. The prover must show that it knows values $x \in \mathcal{M}$ and $u \in \mathcal{R}$ such that $z = C(x, u)$ and $x \in \mathbb{Z}_{2^k}$ (i.e., that it knows how to open the commitment to a value in the range).

Roughly, the idea behind the protocol is to show that the binary representation of z has only k bits, by homomorphically constructing an equivalent commitment from k commitments to binary values. The protocol itself appears as Protocol 5.10.

5.C A Formal Definition of Receipt-Freeness

This section is taken almost verbatim from [55]. This formalization of receipt-freeness is a generalization of Canetti and Gennaro's definition (and so can be used for any secure function evaluation), and is strictly stronger (i.e., any protocol that is receipt-free under this definition is post-factum incoercible as well). The difference is the adversarial model we consider. Canetti and Gennaro only allow the adversary to query coerced players players after the protocol execution is complete.

Unfortunately, this "perfect" receipt-freeness is impossible to achieve except for trivial computations. This is because for any non-constant function, there must exist some party P_i and some set of inputs to the other parties such that the output of the function depends on the input used by x_i. If the adversary corrupts all parties except for P_i, it will be able to tell from the output of the function what input what used by P_i, and therefore whether or not P_i was a puppet.

This is the same problem faced by Canetti and Genaro in defining post-factum incoercibility. Like theirs, this definition sidesteps the problem by requiring that any "coercion" the adversary can do in the real world it can also do in an ideal world (where the parties' only interaction is sending their input to an ideal functionality that computes the function). Thus, before we give the formal definition of receipt-freeness, we must first describe the mechanics of computation in the ideal and real worlds. Below, f denotes the function to be computed.

5.C.1 The Ideal World

The ideal setting is an extension of the model used by Canetti and Genaro (the post-factum incoercibility model). As in their model, there are n parties, $P_1, \ldots, P_n$, with inputs $x_1, \ldots, x_n$. Each party also has a "fake" input; they are denoted $x'_1, \ldots, x'_n$. The "ideal" adversary is denoted $\mathcal{I}$.

Protocol 5.10 Zero-Knowledge Proof that a Committed Value is in $\mathbb{Z}_{2^k}$

Input: Verifier receives $z \in \mathcal{C}$, Prover receives $x \in \mathcal{M}$ and $u \in \mathcal{R}$ such that $z = C(x,u)$, $x < 2^k$.

1: Denote: $c_0 \doteq C(0,0)$ and $c_1 \doteq C(1,0)$
2: Denote: $b_0, \ldots, b_{k-1}$ the binary representation of x.
3: Prover chooses values $r_1, \ldots, r_{2k} \in_R \mathcal{R}$.
4: **for** $1 \leq i \leq 2k$ **do**
5: Prover computes and sends to verifier:

$$d_{i-1} \doteq \begin{cases} C(b_{i-1}, r_i) & \text{if } i \leq k \\ C(1 - b_{i-k-1}, r_i) & \text{if } i > k \end{cases}$$

6: **end for**
7: Prover proves to verifier (using Protocol 5.9) that $d_0, \ldots, d_{2k-1}$ is a valid shuffle of $\underbrace{c_0, \ldots, c_0}_{\times k}, \underbrace{c_1, \ldots, c_1}_{\times k}$
{note that this is indeed the case, since there are exactly k commitments to 0 and k commitments to 1}
8: Prover and verifier both compute:

$$z' \doteq \prod_{i=0}^{k-1} d_i^{2^i} = C\left(\sum_{i=0}^{k-1} 2^i b_i, \sum_{i=0}^{k-1} 2^i r_i\right)$$

9: Prover proves to verifier (using Protocol 5.8) that z' and z are commitments to the same value. {Note that this is the case, since by the definition of $b_0, \ldots, b_{k-1}$, $x = \sum_{i=0}^{k-1} 2^i b_i$}

In our model we add an additional input bit to each party, $c_1, \ldots, c_n$. We call these bits the "coercion-response bits". A trusted party collects the inputs from all the players, computes $f(x_1, \ldots, x_n)$ and broadcasts the result. In this setting, the ideal adversary $\mathcal{I}$ is limited to the following options:

1. Corrupt a subset of the parties. In this case the adversary learns the parties' real inputs and can replace them with inputs of its own choosing.

2. Coerce a subset of the parties. A coerced party's actions depend on its coercion-response bit c_i. Parties for which $c_i = 1$ will respond by sending their real input x_i to the adversary (we'll call these "puppet" parties). Parties for which $c_i = 0$ will respond by sending the fake input x_i' to the adversary.

   At any time after coercing a party, the adversary can provide it with an alternate input x_i''. If $c_i = 1$, the coerced party will use the alternate input instead of its real one (exactly as if it were corrupted). If $c_i = 0$, the party will ignore the alternate input (so the output of the computation will be the same as if that party were honest). There is one exception to this rule, and that is if the alternate input is one of the special values $\perp$ or $*$, signifying a forced abstention or forced random vote, respectively. In this case the party will use the input $\perp$, or choose a new, random, input regardless of the value of c_i.

$\mathcal{I}$ can perform these actions iteratively (i.e., adaptively corrupt or coerce parties based on information gained from previous actions), and when it is done the ideal functionality computes the function. $\mathcal{I}$'s view in the ideal case consists its own random coins, the inputs of the corrupted parties, the inputs (or fake inputs) of the coerced parties and the output of the ideal functionality $f(x_1, \ldots, x_n)$ (where for corrupted and puppet parties x_i is the input chosen by the adversary).

Note that in the ideal world, the only way the adversary can tell if a coerced party is a puppet or not is by using the output of the computation – the adversary has no other information about the coercion-response bits.

5.C.2 The Real World

Our real-world computation setting is also an extension of the real-world setting in the post-factum incoercibility model. We have n players, $P_1, \ldots, P_n$, with inputs $x_1, \ldots, x_n$ and fake inputs $x'_1, \ldots, x'_n$. The adversary in the real-world is denoted $\mathcal{A}$ (the "real" adversary).

The parties are specified by interactive Turing machines restricted to probabilistic polynomial time. Communication is performed by having special communication tapes: party P_i sends a message to party P_j by writing it on the (i, j) communication tape (we can also consider different models of communication, such as a broadcast tape which is shared by all parties). Our model does not allow erasure; communication tapes may only be appended to, not overwritten. The communication is synchronous and atomic: any message sent by a party will be received in full by intended recipients before the beginning of the next round.

We extend the post-factum incoercibility model by giving each party a private communication channel with the adversary and a special read-only register that specifies its corruption state. This register is initialized to the value "honest", and can be set by the adversary to "coerced" or "corrupted". In addition, each party receives the coercion response bit c_i. We can think of the ITM corresponding to each party as three separate ITMs (sharing the same tapes), where the ITM that is actually "running" is determined by the value of the corruption-state register. Thus, the protocol specifies for party P_i a pair of ITMs (H_i, C_i), corresponding to the honest and coerced states (the corrupt state ITM is the same for all protocols and all parties).

The computation proceeds in steps: In each step $\mathcal{A}$ can:

1. Corrupt a subset of the parties by setting their corresponding corruption-state register to "corrupted". When its corruption-state register is set to "corrupted", the party outputs to the adversary the last state it had before becoming corrupted, and the contents of any messages previously received. It then waits for commands from the adversary and executes them. The possible commands are:

   - Copy to the adversary a portion of one of its tapes (input, random, working or communication tapes).
   - Send a message specified by the adversary to some subset of the other parties.

   These commands allow the adversary to learn the entire past view of the party and completely control its actions from that point on. We refer to parties behaving in this manner as executing a "puppet strategy".
2. Coerce a subset of the parties by setting their corresponding corruption-state register to "coerced". From this point on $\mathcal{A}$ can interactively query and send commands to the coerced party as it can to corrupted parties. The coerced party's response depends on its coercion-response bit c_i. If $c_i = 1$, the party executes the puppet strategy, exactly as if it were corrupted. If $c_i = 0$, it runs the coercion-resistance strategy C_i instead. The coercion-resistance strategy specifies how to respond to $\mathcal{A}$'s queries and commands.
3. Send commands to corrupted and coerced parties (and receive responses).

$\mathcal{A}$ performs these actions iteratively, adaptively coercing, corrupting and interacting with the parties. $\mathcal{A}$'s view in the real-world consists of its own randomness, the inputs, randomness and all communication of corrupted parties, its communications with the coerced parties and all public communication.

5.C.3 A Formal Definition of Receipt-Freeness

Definition 5.7. A protocol is receipt-free if, for every real adversary $\mathcal{A}$, there exists an ideal adversary $\mathcal{I}$, such that for any input vector $x_1, \ldots, x_n$, fake input vector $x'_1, \ldots, x'_n$ and any coercion-response vector $c_1, \ldots, c_n$:

1. $\mathcal{I}$'s output in the ideal world is indistinguishable from $\mathcal{A}$'s view of the protocol in the real world with the same input and coercion-response vectors (where the distributions are over the random coins of $\mathcal{I}$, $\mathcal{A}$ and the parties).

2. Only parties that have been corrupted or coerced by $\mathcal{A}$ in the real world are corrupted or coerced (respectively) by $\mathcal{I}$ in the ideal world.

It is important to note that even though a protocol is receipt-free by this definition, it may still be possible to coerce players (a trivial example is if the function f consists of the player's inputs). What the definition does promise is that if it is possible to coerce a party in the real world, it is also possible to coerce that party in the ideal world (i.e. just by looking at the output of f).

Bibliography

[1] Ben Adida and Ronald L. Rivest. Scratch & vote: self-contained paper-based cryptographic voting. In J. Stern, editor, *Proceedings of WPES '06, the 5th ACM workshop on Privacy in electronic society*, pages 29–40, New York, NY, USA, October 2006. ACM Press.

[2] Dorit Aharonov, Amnon Ta-Shma, Umesh V. Vazirani, and Andrew C. Yao. Quantum bit escrow. In *STOC '00*, pages 705–714, 2000.

[3] Andris Ambainis, Markus Jakobsson, and Helger Lipmaa. Cryptographic randomized response techniques. In *PKC '04*, volume 2947 of *LNCS*, pages 425–438, 2004.

[4] Ross J. Anderson. *Security Engineering: A Guide to Building Dependable Distributed Systems*. John Wiley & Sons, Inc., 2001.

[5] Yonatan Aumann, Yan Zong Ding, and Michael O. Rabin. Everlasting security in the bounded storage model. *IEEE Transactions on Information Theory*, 48(6):1668–1680, 2002.

[6] Mihir Bellare and Oded Goldreich. On defining proofs of knowledge. In Ernest F. Brickell, editor, *Proceedings of CRYPTO 1992, 12th Annual International Cryptology Conference*, volume 740 of *LNCS*, pages 390–420, New York, NY, USA, August 1992. Springer-Verlag Inc.

[7] J. Benaloh and D. Tuinstra. Receipt-free secret-ballot elections. In *Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing*, pages 544–553, New York, NY, USA, May 1994. ACM Press.

[8] Matt Blaze. Cryptology and physical security: Rights amplification in master-keyed mechanical locks. *IEEE Security and Privacy*, March 2003.

[9] Matt Blaze. Safecracking for the computer scientist. *U. Penn CIS Department Technical Report*, December 2004. http://www.crypto.com/papers/safelocks.pdf.

[10] Manuel Blum. Coin flipping over the telephone. In *Proceedings of IEEE COMPCON '82*, pages 133–137, 1982.

[11] Fabrice Boudot. Efficient proofs that a committed number lies in an interval. In Bart Preneel, editor, *Proceedings of EUROCRYPT 2000, International Conference on the Theory and Application of Cryptographic Techniques*, volume 1807 of *LNCS*, pages 431–444, New York, NY, USA, May 2000. Springer-Verlag Inc.

[12] Debra Bowen. California secretary of state: Voting systems top-to-bottom review, August 2007. http://www.sos.ca.gov/elections/elections_vsr.htm.

[13] Jeremy W. Bryans and Peter Y. A. Ryan. A simplified version of the Chaum voting scheme. Technical Report CS-TR 843, University of Newcastle, 2004.

[14] Jan Camenisch and Markus Michels. Proving in zero-knowledge that a number is the product of two safe primes. In Stern [73], pages 107–122.

[15] R. Canetti, C. Dwork, M. Naor, and R. Ostrovsky. Deniable encryption. In *CRYPTO '97*, volume 1294 of *LNCS*, pages 90–104, 1997.

[16] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS '01*, pages 136–145, 2001.

[17] Ran Canetti and Rosario Gennaro. Incoercible multiparty computation. In *FOCS '96*, pages 504–513, 1996.

[18] Arijit Chaudhuri and Rahul Mukerjee. *Randomized Response: Theory and Techniques*, volume 85. Marcel Dekker, 1988.

[19] David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–88, 1981.

[20] David Chaum. Blind signature systems. In David Chaum, editor, *Proceedings of CRYPTO 1983*, page 153, August 1983.

[21] David Chaum. E-voting: Secret-ballot receipts: True voter-verifiable elections. *IEEE Security & Privacy*, 2(1):38–47, January/February 2004.

[22] David Chaum, 2006. http://punchscan.org/.

[23] David Chaum, Richard Carback, Jeremy Clark, Aleksander Essex, Stefan Popoveniuc, Ronald L. Rivest, Peter Y. A. Ryan, Emily Shen, and Alan T. Sherman. Scantegrity ii: End-to-end verifiability for optical scan election systems using invisible ink confirmation codes. In *USENIX/EVT '08*, 2008. http://www.usenix.org/event/evt08/tech/full_papers/chaum/chaum.pdf.

[24] Richard Cleve. Limits on the security of coin flips when half the processors are faulty. In *STOC '86*, pages 364–369, 1986.

[25] Richard Cleve and Russell Impagliazzo. Martingales, collective coin flipping and discrete control processes. http://www.cpsc.ucalgary.ca/∼cleve/pubs/martingales.ps, 1993.

[26] Josh D. Cohen(Benaloh) and Michael J. Fischer. A robust and verifiable cryptographically secure election scheme. In *FOCS '85*, pages 372–382, 1985.

[27] Ronald Cramer, Matthew Franklin, Berry Schoenmakers, and Moti Yung. Multi-authority secret-ballot elections with linear work. In Ueli Maurer, editor, *Proceedings of EUROCRYPT 1996, International Conference on the Theory and Application of Cryptographic Techniques*, volume 1070 of *LNCS*, pages 72–83, New York, NY, USA, May 1996. Springer-Verlag Inc.

[28] Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers. A secure and optimally efficient multi-authority election scheme. In Fumy [41], pages 103–118.

[29] Claude Crépeau. Efficient cryptographic protocols based on noisy channels. In Fumy [41], pages 306–317.

[30] Claude Crépeau and Joe Kilian. Achieving oblivious transfer using weakened security assumptions. In *FOCS '88*, pages 42–52, 1988.

[31] Claude Crépeau and Joe Kilian. Discreet solitary games. In *CRYPTO '93*, volume 773 of *LNCS*, pages 319–330, 1994.

[32] Ivan B. Damgård, Serge Fehr, Kiril Morozov, and Louis Salvail. Unfair noisy channels and oblivious transfer. In *TCC '04*, volume 2951 of *LNCS*, pages 355–373, 2004.

[33] Ivan B. Damgård and Eiichiro Fujisaki. A statistically-hiding integer commitment scheme based on groups with hidden order. In Yulian Zheng, editor, *Proceedings of ASIACRYPT 2002, International Conference on the Theory and Application of Cryptology and Information Security*, volume 2501 of *LNCS*, pages 125–142, New York, NY, USA, December 2002. Springer-Verlag Inc.

[34] Ivan B. Damgård, Joe Kilian, and Louis Salvail. On the (im)possibility of basing oblivious transfer and bit commitment on weakened security assumptions. In Stern [73], pages 56–73.

[35] Persi Diaconis, Susan Holmes, and Richard Montgomery. Dynamical bias in the coin toss, 2004. http://www-stat.stanford.edu/∼cgates/PERSI/papers/headswithJ.pdf.

[36] Judith A. Droitcour, Eric M. Larson, and Fritz J. Scheuren. The three card method: Estimating sensitive survey items–with permanent anonymity of response. In *Proceedings of the American Statistical Association, Social Statistics Section [CD-ROM]*, 2001.

[37] Cynthia Dwork, Moni Naor, and Amit Sahai. Concurrent zero knowledge. In *STOC '98*, pages 409–418, New York, NY, USA, 1998. ACM Press.

[38] Ronald Fagin, Moni Naor, and Peter Winkler. Comparing information without leaking it. *Commun. ACM*, 39(5):77–85, 1996.

[39] Sarah Flannery and David Flannery. *In Code: A Mathematical Journey.* Algonquin Books of Chapel Hill, 2002.

[40] Atsushi Fujioka, Tatsuaki Okamoto, and Kazui Ohta. A practical secret voting scheme for large scale elections. In *AUSCRYPT '92*, volume 718 of *LNCS*, pages 244–251, 1993.

[41] Walter Fumy, editor. *Advances in cryptology — EUROCRYPT '97: International Conference on the Theory and Application of Cryptographic Techniques, Konstanz, Germany, May 11–15, 1997: proceedings*, volume 1233 of *LNCS*, 1997.

[42] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. *J. of the ACM*, 38(3):691–729, July 1991.

[43] Oded Goldreich and Ronen Vainish. How to solve any protocol problem - an efficiency improvement. In Andrew Michael Odlyzko, editor, *Proceedings of CRYPTO 1986*, volume 263 of *LNCS*, pages 73–86, New York, NY, USA, August 1986. Springer-Verlag Inc.

[44] Jens Groth. A verifiable secret shuffle of homomorphic encryptions. In Yvo Desmedt, editor, *Proceedings of PKC 2003, 6th International Workshop on Theory and Practice in Public Key Cryptography*, volume 2567 of *Lecture Notes in Computer Science*, pages 145–160, New York, NY, USA, January 2002. Springer-Verlag Inc.

[45] Martin Hirt and Kazue Sako. Efficient receipt-free voting based on homomorphic encryption. In *Eurocrypt 2000*, volume 1807 of *LNCS*, pages 539+, 2000.

[46] Russell Impagliazzo and Michael Luby. One-way functions are essential for complexity based cryptography. In *FOCS '89*, pages 230–235, 1989.

[47] Ari Juels, Dario Catalano, and Markus Jakobsson. Coercion-resistant electronic elections. In Sabrina De Capitani di Vimercati and Roger Dingledine, editors, *Proceedings of WPES '05, 2005 ACM workshop on Privacy in the electronic society*, pages 61–70, New York, NY, USA, November 2005. ACM Press.

[48] Chris Karlof, Naveen Sastry, and David Wagner. Cryptographic voting protocols: A systems perspective. In *USENIX Security '05*, pages 33–50, 2005.

[49] Hiroaki Kikuchi, Jin Akiyama, Gisaku Nakamura, and Howard Gobioff. Stochastic voting protocol to protect voters privacy. In *IEEE Workshop on Internet Applications*, pages 102–111, July 1999.

[50] Joe Kilian. Founding crytpography on oblivious transfer. In *STOC '88*, pages 20–31, 1988.

[51] Hoi-Kwong Lo and H. F. Chau. Why quantum bit commitment and ideal quantum coin tossing are impossible. In *PhysComp '98*, pages 177–187, 1998.

[52] Dominic Mayers. Unconditionally secure quantum bit commitment is impossible. *Phys. Rev. Lett.*, 78:3414–3417, 1997.

[53] Tal Moran and Moni Naor. Basing cryptographic protocols on tamper-evident seals. In Luís Caires, Giuseppe F. Italiano, Luís Monteiro, Catuscia Palamidessi, and Moti Yung, editors, *Proceedings of ICALP 2005, 32nd International Colloquium on Automata, Languages and Programming*, volume 3580 of *LNCS*, pages 285–297, New York, NY, USA, July 2005. Springer-Verlag Inc.

[54] Tal Moran and Moni Naor. Polling with physical envelopes: A rigorous analysis of a human-centric protocol. In Serge Vaudenay, editor, *EUROCRYPT 2006*, pages 88–108, 2006. http://www.wisdom.weizmann.ac.il/~talm/papers/MN06-crrt.pdf.

[55] Tal Moran and Moni Naor. Receipt-free universally-verifiable voting with everlasting privacy. In Cynthia Dwork, editor, *Proceedings of CRYPTO 2006, 26th Annual International Cryptology Conference*, volume 4117 of *LNCS*, pages 373–392, New York, NY, USA, August 2006. Springer-Verlag Inc. http://www.wisdom.weizmann.ac.il/~talm/papers/MN06-voting.pdf.

[56] Tal Moran and Moni Naor. Split-ballot voting: Everlasting privacy with distributed trust. In *CCS 2007*, pages 246–255, 2007. http://www.wisdom.weizmann.ac.il/~talm/papers/MN07-split-ballot.pdf.

[57] Tal Moran, Moni Naor, and Gil Segev. An optimally fair coin toss: Cleve's bound is tight, 2008. In Submission.

[58] Elchanan Mossel and Ryan O'Donnell. Coin flipping from a cosmic source: On error correction of truly random bits. *Random Struct. Algorithms*, 26(4):418–436, 2005.

[59] Moni Naor, Yael Naor, and Omer Reingold. Applied kid cryptography, March 1999. http://www.wisdom.weizmann.ac.il/∼naor/PAPERS/waldo.ps.

[60] Moni Naor and Benny Pinkas. Visual authentication and identification. In *CRYPTO '97*, volume 1294 of *LNCS*, pages 322–336, 1997.

[61] Moni Naor and Adi Shamir. Visual cryptography. In Alfredo De Santis, editor, *Proceedings of EUROCRYPT 1994, Workshop on the Theory and Application of Cryptographic Techniques*, volume 950 of *LNCS*, pages 1–12, New York, NY, USA, May 1994. Springer-Verlag Inc.

[62] C. Andrew Neff. Practical high certainty intent verification for encrypted votes, October 2004. http://www.votehere.net/vhti/documentation/vsv-2.0.3638.pdf.

[63] Choonsik Park, Kazutomo Itoh, and Kaoru Kurosawa. Efficient anonymous channel and all/nothing election scheme. In *Eurocrypt '93*, volume 765 of *LNCS*, pages 248–259. Springer, 1994.

[64] T. P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO '91*, volume 576 of *LNCS*, pages 129–140, 1991.

[65] Stefan Popoveniuc and Ben Hosp. An introduction to punchscan, 2006. http://punchscan.org/papers/popoveniuc_hosp_punchscan_introduction.pdf.

[66] Michael O. Rabin. Transaction protection by beacons. *J.Computer and System Sciences*, 27(2):256–267, 1983.

[67] David J. Reynolds. A method for electronic voting with coercion-free receipt, 2005. Presentation: http://www.win.tue.nl/~berry/fee2005/presentations/reynolds.ppt.

[68] Peter Y. A. Ryan. A variant of the Chaum voter-verifiable scheme. In *Proceedings of WITS '05, 2005 Workshop on Issues in the Theory of Security*, pages 81–88, New York, NY, USA, 2005. ACM Press.

[69] K. Sako and J. Kilian. Receipt-free mix-type voting schemes. In *EUROCRYPT '95*, volume 921 of *LNCS*, pages 393–403, 1995.

[70] Bruce Schneier. The solitaire encryption algorithm, 1999. http://www.schneier.com/solitaire.html.

[71] Adi Shamir. Cryptographers panel, RSA conference, 2006. Webcast: http://media.omediaweb.com/rsa2006/1_5/1_5_High.asx.

[72] Sid Stamm and Markus Jakobsson. Privacy-preserving polling using playing cards. Cryptology ePrint Archive, Report 2005/444, December 2005.

[73] J. Stern, editor. *Advances in cryptology — EUROCRYPT '99: International Conference on the theory and application of cryptographic techniques, Prague, Czech Republic, May 2–6, 1999: Proceedings*, volume 1592 of *LNCS*, 1999.

[74] Clive Thompson. Can you count on voting machines? New York Times Magazine, January 6 2008. http://www.nytimes.com/2008/01/06/magazine/06Vote-t.html.

[75] Stanley Warner. Randomized response: a survey technique for eliminating evasive answer bias. *Journal of the American Statistical Association*, pages 63–69, 1965.