University of California

Los Angeles

Connecting Theory and Practice in Modern Cryptography

A dissertation submitted in partial satisfaction

of the requirements for the degree

Doctor of Philosophy in Computer Science

by

Abishek Kumarasubramanian

2014

Abstract of the Dissertation

Connecting Theory and Practice in Modern Cryptography

by

Abishek Kumarasubramanian

Doctor of Philosophy in Computer Science

University of California, Los Angeles, 2014

Professor Amit Sahai, Chair

Cryptography is an active field of theoretical research. It is also a place where many unproven, but time-tested practical ideas exist. This work exposits a few strands that connect these two sides of the Cryptographic coin.

The two main results presented are,

1. The first traitor tracing scheme based on prime order bilinear groups. While prime order bilinear groups are practical, existing schemes for traitor tracing were based on the more structurally rich, but much less practical composite order bilinear groups. Our work brings the rich structure of composite order bilinear groups and the efficiency of prime order groups together.

2. A formal model for CAPTCHAs which captures the intuition that any automaton must request human help to solve CAPTCHA problems. We use this model to obtain positive results in the field of concurrent security. This is the first result that brings CAPTCHA, a widely adopted practical security tool, into main stream cryptography to achieve tasks that are known to be theoretically impossible in the plain model.

The dissertation of Abishek Kumarasubramanian is approved.

Don Blasius

Eli Gafni

Rafail Ostrovsky

Amit Sahai, Committee Chair

University of California, Los Angeles

2014

To Amma, Appa and Bharath…

Table of Contents

1 Introduction **1**

1.1 Efficient traitor tracing 2

1.2 Cryptography using Captcha puzzles 6

2 Efficient Traitor Tracing **12**

2.1 Preliminary Definitions 12

2.1.1 Traitor Tracing 12

2.1.2 Trace & Revoke 13

2.1.3 PLBE 13

2.1.4 AugBE 16

2.1.5 Equivalence of Traitor Tracing and PLBE 18

2.2 Background on Bilinear Maps 18

2.2.1 Bilinear Groups 18

2.2.2 Complexity Assumptions 19

2.3 Key Ideas 20

2.4 Our Construction 22

2.4.1 AugBE using Symmetric Bilinear Groups 23

2.4.2 AugBE using Asymmetric Bilinear Groups 26

2.4.3 PLBE 29

2.5 Security Proof 29

2.5.1 Index Hiding 29

2.5.2 Message Hiding 42

2.6 Implementation 43

2.6.1 Encryption Time 45
2.6.2 Ciphertext Size 46
2.6.3 Decryption Time 48
2.6.4 Comparison with the ElGamal Encryption 48
2.7 Conclusion 49

3 Cryptography using Captcha 51
3.1 Preliminaries 51
3.2 Modeling CAPTCHA Puzzles 53
3.3 A Straight-line Extractable Commitment Scheme 58
3.4 Constructing UC-Puzzles using CAPTCHA 63
3.5 Conclusion 68

4 Concurrent Non-Malleable Zero Knowledge 70
4.1 Proof of Theorem 3.3.2 70
4.2 Proof of UC Puzzle Construction 75
4.3 Brief Review of Protocol Composition 79
4.3.1 Universal Composition 80
4.3.2 Concurrent Self Composition 83
4.4 Concurrent NMZK: Constant Round, Straight-line 84
4.5 Negative Results 86
4.5.1 Universal Composition in the Direct Access Model 86
4.5.2 Concurrent Self-composition of general functionalities 89

References 91

List of Figures

2.1 Illustration: [BSW06] Traitor tracing scheme 24

2.2 Encryption Time (in secs) of traitor tracing schemes 43

2.3 Ciphertext Size (in bytes) of traitor tracing schemes 45

2.4 Decryption time of traitor tracing schemes 45

2.5 Ciphertext Size 47

2.6 Encryption Time 48

3.1 Straightline Extractable Commitment Protocol $\langle \mathsf{C}^H, \mathsf{R} \rangle$ 61

3.2 UC Puzzle $\langle S^H, R \rangle$ using a Captcha puzzle system $\mathcal{C} := (G, H)$ 68

4.1 Straight-Line Concurrent Non-Malleable Zero-Knowledge $(\mathbf{P}, \mathbf{V})$. 86

4.2 A one-time commitment functionality [CF01] $\mathcal{F}_{\text{COM}}$ 87

List of Tables

2.1 Encryption Time of traitor tracing schemes 44

2.2 CipherText Size (in bytes) of traitor tracing schemes 46

2.3 Encryption Time and CipherText size (in bytes) 47

Acknowledgments

I begin with infinite gratitude to my advisors Prof. Rafail Ostrovsky and Prof. Amit Sahai. I vividly remember every part of my journey with Amit and Rafi from being a clueless first-year grad student to the Ph.D candidate that I am today. As excellent researchers, advisors and people; Amit and Rafi have significantly shaped my thought processes and I am indebted to them for that.

Acknowledgements are due to my fellow labmates, seniors and juniors, who made my Ph.D thoroughly enjoyable as colleagues. Nishanth Chandran, Vipul Goyal, Abhishek Jain and Omkant Pandey were great examples that I learnt from. Thank you Shweta Agrawal, Sanjam Garg, Ran Gelles, Hemantha Maji, Alessandra Scafuro, Alan Roytman and Akshay Wadia for being there every day for that coffee, question, discussion and beer. Finally, thanks to the new students and my students from the various classes I taught at UCLA for giving me the satisfication of having passed on a little drop of the little drop that I learnt for myself during my Ph.D. Special thanks to my alma matter, Rajsekar and Ranjit for being the invisible hand that motivated me.

I would like to take this opportunity to thank people in my personal life who have made quite the influence. Thank you Anandan and Kentaro and Pandu for being great inspirations that nudged me towards this direction in my life. Thank you Venkie and Vidya for supporting and infusing energy in me whenever I needed it. Buvana, Satya and DJ, I thank you guys for listening to every little thing that I had to say throughout my Ph.D.

To those whose names I did not mention, I thank you all for making me who I am today. Every one of you in your own unique way.

CHAPTER 1

Introduction

The need for cryptography has existed for many centuries, starting with the substitution ciphers of hieroglyphs [Wik] much before the invention of formal methods and turing machines. Methods and techniques used back then relied on their efficacy rather than formal proofs. This practice, motivated primarily by defense purposes, has resulted in a whole slew of algorithms and developments which we will call loosely as the field of *computer security*. A look at the latest standards for cryptographic algorithms, such as the one published by the N.I.S.T [NI], shows the same principles of efficiency and efficacy guiding today's constructions.

Complimentary to computer security, one could consider the work of Shannon [Sha48] which introduced the one-time pad and information theoretically secure cryptography as the beginning of the field of, which we will call loosely as, *theoretical cryptography* and provable security. Cryptography as a field considered many theoretical problems such as key-exchange, public-key encryption, secure multi party computation, concurrent and universally-composable security amongst many others [DH76, RSA78, Yao82, CLO02]. It is a rich and diverse field with many beautiful results and ideas.

Given this general situation, this work attempts to connect the dots between these two worlds by bringing in features from one world into the other. More specifically, we present two works by the authors on this theme. First, we present constructions and implementation of efficient traitor tracing schemes. The key contribution of this work is the improved efficiency, making schemes that were known to be theoretically feasible more practical. These are introduced in Section 1.1 and our full results are presented in Chapter 2. Then, we provide theoretical modeling of Captcha. Captcha is a widely used security primitive, with the specific purpose of identifying the presence of a human in a protocol. Our work is a first step towards modeling and analyzing its potential as a formal cryptographic assumption deriving theoretical possibilities and impossibilites from it. This work is introduced in Section 1.2 and our full results are presented in Chapter 3 and Chapter 4.

1.1 Efficient traitor tracing

Consider a scenario in which a content distributor, like a cable/radio broadcaster, wants to broadcast content while making sure that only those users who have paid for the service have access to the content. In such a system, each user will need a decoder with a secret key in order to decrypt the content. A naïve solution to achieve this would be to use an encryption system such that the corresponding secret key is known to all legitimate users. The broadcasting authority can then encrypt the content and broadcast the ciphertext. All legitimate users with the secret key will be able to decrypt the content. But if a dishonest user sells his key, then an attacker could build pirate decoders which it could then distribute, *allowing unauthorized users to decrypt all future broadcast content without ever having to communicate with the attacker again.* A malicious user could also use his own key to build pirate decoders. The problem is that in this system, there is no way to identify *rogue* users. A *traitor tracing* or *trace* & *revoke* system is designed to solve this problem. The purpose of a trace & revoke system, introduced by Chor et al. [CFN94], is to help content distributors identify rogue users and revoke their secret keys. If revocation is not desired, one can have just traitor tracing schemes, which helps the distributor identify the keys used in a pirate decoder. The content distributor can then hold the corresponding rogue user responsible for the loss incurred.

It should be observed that a traitor tracing system is not designed to help to protect any particular content. The problem of traitor tracing is distinct from what is commonly referred to as "Digital Rights Management" (DRM). DRM systems have traditionally been concerned with protecting the widespread distribution of content that is *already* in the hands of the (perceived) attacker. Clearly, there are fundamental obstacles to achieving this goal, since the attacker can simply record what he sees and then retransmit this. In a trace & revoke system, an authority can use the tracing mechanisms to identify all of the key material (actively) used in a pirate box and then disable these keys from being used to access *future* broadcasts. The use of trace & revoke systems best fits application such as satellite radio or other active broadcast services where users are interested in having a device that can access the current broadcast, without having to be in constant communication with a dishonest party.[1] Given a pirate decoder, the challenge in a trace & revoke system is to identify at least one of the users whose key must have been used to construct the pirate decoder and then revoke that key from the system. As such, traitor tracing can be seen as providing a type of cryptographic method for digital forensics – once a decoding box is discovered in the wild, the associated cryptographic tracing algorithm allows one to (provably) associate a particular user's secret key with the box.

A naïve solution to the problem just described (in a system of N users) would be to have N instances of an off-the-shelf encryption system such that the i^{th} secret key is known to the i^{th} user. The broadcasting authority could encrypt the content under each public key and broadcast all the ciphertexts[2]. Each legitimate user will then be able to decrypt the part of ciphertext corresponding to its private key. Given a pirate decoder, it is also possible for this system to identify at least one of the rogue users whose key was used to build it. We could then revoke this key by simply not encrypting under it in future broadcasts. But this system is very inefficient. For this system, the ciphertext size is *linear* in the number of users. We provide an efficient implementation of this naïve solution using a fast Elgamal encryption scheme and compare it with the performance of our scheme in Section 2.6.2.

Previous Work. To overcome this limitation of inefficiency, many results with different levels of security have been proposed. A weak security property that has been the subject of the greatest amount of previous work is the *t-collusion-resistant traitor tracing.* A t-collusion-resistant tracing [BF99,KD98,NP00,KY02b,DF03,MSK02,TSZ03,CPP05] system will work

[1]Traitor tracing systems are not appropriate for systems where "protecting" released content is considered the highest priority.

[2]Note that here, the content itself would be a secret key for a private-key encryption scheme (such as AES), which would then be used to encrypt the actual content.

as long as the pirate uses fewer than t user keys in building the pirate box. Prior to [BSW06], all such schemes required a ciphertext size blow-up at least linear in this parameter t.

A system that allows for traitor tracing regardless of how many users' keys are captured by the attacker is called *fully collusion-resistant*. Boneh, Sahai, and Waters [BSW06] presented the first fully collusion-resistant traitor tracing system with $O(\sqrt{N})$ size ciphertexts and public keys. A fully-collusion-resistant traitor tracing system with constant size ciphertexts [BN08] has also been constructed, but at the cost of enormous private key sizes (quadratic in the number of users).

Another issue of concern in traitor tracing systems is the need for a tracing authority, e.g [BSW06, BN08] which use a secret tracing key to identify rogue users. [CPP05, WHI01, Pfi96, PW97, KY02a] allow for a public tracing algorithm that does not require any secret inputs. Other systems such as the one in [BGW05, BW06] provide security only against a static adversary and achieve $O(1)$ size ciphertext and private key, but need $O(N)$ size public key (which is used in the decryption algorithm).

When considering only broadcast encryption, [BW06] acheive adaptive security with $O(1)$ size ciphertext and private key ($O(N)$ size public key) and also provide a system with $O(\sqrt{N})$ ciphertext and public key. [GW09] obtain adaptively secure broadcast encryption with $O(1)$ cipher-text, $O(N)$ private and public key. The recent work of [Wat09] obtains identical parameters and also provides identity based encryption.

Building on [BSW06], Boneh and Waters [BW06] presented a fully collusion resistant, publicly traceable trace & revoke scheme, representing the "state-of-the-art" prior to this work. However, [BW06] crucially makes use of composite order bilinear groups, which lead to significant losses in efficiency that make the scheme impractical in many settings. The goal of the present work is to build new techniques to achieve order-of-magnitude improvements in efficiency without sacrificing any security.

Our Contribution. We present a new traitor tracing system that achieves the same strong security properties as [BSW06], but avoids the use of composite order bilinear groups. Instead, using new techniques, our scheme is based on prime order bilinear groups, and its security depends on the hardness of the widely believed decisional linear assumption. This allows for shorter group elements and much more efficient schemes (see Section 2.6). We also extend this to build publicly traceable trace & revoke schemes, improving similarly in efficiency over [BW06].

Hardness assumptions in composite order bilinear groups are limited by known attacks on factoring their modulii. Because of sub-exponential attacks against factoring, for appropriate security, large composite order groups must be used. When compared with prime order bilinear groups, for the same level of practical security (see Section 2.6 for details), a simple exponentiation in composite order bilinear groups is about 25 times slower than one in prime order groups. Also, one pairing operation in these larger composite order groups is approximately 30 times costlier than a pairing in prime order groups. The main contribution of this research is to present traitor tracing schemes based on prime order bilinear groups making them practical.

We also implement our protocol using the PBC library [Lyn] (see Section 2.6). We compare the efficiency our traitor tracing scheme with an implementation of [BSW06]. We obtain encryption times up to 6 times better than [BSW06] and ciphertexts that are 50% smaller. Decryption is 10 times faster.

We note that the techniques we use are general and can be used to convert other cryptosystems based on composite order groups to ones based on prime order bilinear groups. In this respect, our work is similar to generic methods described in a very recent concurrent and independent work by Freeman [Fre09]. However, our schemes are different from the work of [Fre09]. His work focuses on generality and while our work is on optimizing and implementing efficient traitor tracing systems. He provides a traitor tracing scheme using asymmetric bilinear groups while we provide schemes based on both symmetric and asymmetric groups. Also, our asymmetric construction is more efficient than his construction, which does not have any known implementation.

1.2 Cryptography using Captcha puzzles

Captcha is an acronym for *Completely Automated Public Turing test to tell Computers and Humans Apart*. These are puzzles that are easy for humans but hard to solve for automated computer programs. They are used to confirm the "presence of a human" in a communication channel. As an illustration of a scenario where such a confirmation is very important, consider the problem of spam. To carry out their nefarious activities, spammers need to create a large number of fake email accounts. Creating a new email account usually requires the filling-in of an online form. If the spammers were to manually fill-in all these forms, then the process would be too slow, and they would not be able to generate a number of fake addresses. However, it is relatively simple to write a script (or an automated *bot*) to quickly fill-in the forms automatically without human intervention. Thus, it is crucial for the email service provider to ensure that the party filling-in the form is an actual human, and not an automated script. This is achieved by asking the party to solve a Captcha, which can only be sovled by a human[3]. A common example of a Captcha puzzle involves the distorted image of a word, and the party is asked to identify the word in the image.

The definition of Captcha stipulates certain limitations on the power of machines, in particular, that they cannot solve Captcha puzzles efficiently. This gives rise to two distinct questions which are interesting from a cryptographic point of view. Firstly, what are the underlying hard problems upon which Captcha puzzles can be based? Von Ahn, Blum, Hopper and Langford [ABH03] study this question formally, and provide constructions based on the conjectured hardness of certain Artificial Intelligence problems.

The second direction of investigation, and the one which we are concerned with in this paper, is to use Captchas as a tool for achieving general cryptographic tasks. There have been only a few examples of use of Captchas in this regard. Von Ahn, Blum, Hopper and Langford [ABH03] use Captchas for image-based steganography. Canetti, Halevi and Steiner [CHS06] construct a scheme to thwart off-line dictionary attacks on encrypted data using Captchas. [DC12] present an encryption protocol using Captcha that is secure

[3]For many more uses of Captcha, see [CAP]

against non-human profiling adversaries. And recently, Dziembowski [Dzi10] constructs a "human" key agreement protocol using only CAPTCHAs. We continue this line of work in the current paper, and investigate the use of CAPTCHAs in zero-knowledge and UC secure protocols. On the face of it, it is unclear how CAPTCHAs may be used for constructing such protocols, or even for constructing building blocks for these protocols, like commitment schemes. However, motivated by current CAPTCHA theory, we define a new *extraction* property of CAPTCHAs that allows us to use them for designing these protocols.

We now give an overview of our contributions. We formally define CAPTCHAs in Section 3.2, but give an informal overview of the model here to make the following discussion cogent. Firstly, modelling CAPTCHA puzzles invariably involves modelling humans who are the key tenets in distinguishing CAPTCHAs from just another one-way function. Following [CHS06] we model the presence of a human entity as an oracle H that is capable of solving CAPTCHA puzzles. A party generates a CAPTCHA puzzle by running a (standard) PPT generation algorithm denoted by G. This algorithm outputs a puzzle-solution pair (z, a). All parties have access to a "human" oracle denoted by H. To "solve" a CAPTCHA puzzle, a party simply queries its oracle with the puzzle and obtains the solution in response. This allows us to distinguish between two classes of machines. Standard PPT machines for which solving CAPTCHAs is a hard problem and oracle PPT machines with oracle access to H which may solve CAPTCHAs efficiently.

The starting point of our work is the observation that if a machine must solve a given CAPTCHA puzzle (called *challenge*), it *must* send one or more CAPTCHA-queries to a human. These queries are likely to be correlated to the challenge puzzle since otherwise they would be of no help in solving the challenge puzzle. Access to these queries, with the help of another human, may therefore provide us with some knowledge about the internal state of a (potentially) malicious machine! This is formulated in our definition of an human extractable CAPTCHA (Definition 3.2.2). Informally, we make the following assumption about CAPTCHA puzzles. Consider two randomly chosen CAPTCHA puzzles (p_0, p_1) of which an adversary obtains only one to solve, say p_b, where the value of b is not known to the challenger. Then by merely looking at his queries to a human oracle H, and with the help of a *human*, a challenger must be able to identify the value of b. More precisely, we augment the human oracle H to possess this added ability. We then model adversaries in our protocols as oracle PPT machines with access to a Captcha solving oracle, but whose internal state can be "extracted" by another oracle PPT machine.

It is clear that this idea, i.e.—the idea of learning something non-trivial about a machine's secret by looking at its Captcha-queries—connects Captcha puzzles with main-stream questions in cryptography much more than ever. This work uses this feature present in Captchas to construct building blocks for zero-knowledge protocols which admit "straight-line" simulation. It is then natural to investigate that if we can get "straight-line" simulation, then perhaps we can answer the following questions as well: construction of *plain-text aware* encryption schemes [BR94], "straight-line" extractable commitment schemes, constant-round fully concurrent zero-knowledge for **NP** [DNS98], fully concurrent two/multi-party computation [Lin03a, PR03, Pas04], universal composition *without* trusted setup assumptions [Can01, CLO02], and so on.

Our Contribution. In section 3.3 (theorem 3.3.2), as the first main result of this work, we construct a commitment scheme which admits "straight-line" extraction. That is, the committed value can be extracted by looking at the Captcha-queries made by the committer to a human oracle.

The starting point (ignoring for a moment an important difficulty) behind our commitment protocol is the following. The receiver R chooses two independent Captcha puzzles (z_0, z_1). To commit to a bit b, the sender C will select z_b using the 1-2-OT protocol and commit to its solution a_b using an ordinary (perfectly-binding) commitment scheme. Since the committer cannot solve the puzzle itself, it must query a human to obtain the solution. By looking at the puzzles C queries to the human, an extractor (with the help of another human oracle) can detect the bit being committed. Since the other puzzle z_{1-b} is computationally hidden from C, this should indeed be possible.

As alluded above, the main difficulty with this approach is that a cheating sender may not query the human on any of the two puzzles, but might still be able to commit to a correct value by obtaining solutions to some related puzzles. This is the issue of malleability that we discuss shortly, and also in section 3.2.

We then use this commitment scheme as a tool to obtain new results in protocol composition. First off, it is straightforward to see that given such a scheme, one can obtain a constant-round concurrent zero-knowledge protocol for all of **NP**. In fact, by using our commitment scheme in place of the "PRS-preamble" [PRS02] in the protocol of Barak, Prabhakaran, and Sahai [BPS06], we obtain a constant-round protocol for *concurrent non-malleable zero-knowledge* [BPS06] (see appendix 4.4).[4]

As a natural extension, we investigate the issue of incorporating CAPTCHA puzzles in the UC framework introduced by Canetti [Can01]. The situation turns out to be very sensitive to the modelling of CAPTCHA puzzles in the UC framework. We discuss two different ways of incorporating CAPTCHA puzzles in the UC framework: [5]

- INDIRECT ACCESS MODEL: In this model, the environment $\mathcal{Z}$ is *not* given direct access to a human H. Instead, the environment is given access to H *only through the adversary* $\mathcal{A}$. This model was proposed in the work of Canetti et. al. [CHS06], who constructed a UC-secure protocol for password-based key-generation functionality. We call this model the *indirect* access model.

- DIRECT ACCESS MODEL: In this model, the environment is given a direct access to H. In particular, the queries made by $\mathcal{Z}$ to H are not visible to the adversary $\mathcal{A}$, in this model.

In the indirect access model, we show how to construct UC-secure protocols for all functionalities. In section 3.4, as the second main result of this work, we construct a constant-round *UC-puzzle* protocol as defined by Lin, Pass, and Venkitasubramaniam [LPV09]. By

[4]For readers familiar with concurrent non-malleability, our protocol admits "straight-line" simulation, but the extraction of witnesses from a man-in-the-middle is not straight-line. Also, another modification is needed to the protocol of [BPS06]: we need to use a constant round non-malleable commitment scheme and not that of [DDN00]. We can use any of the schemes presented in [PR05, PPV08, LP11, Goy11].

[5]We assume basic familiarity with the model of universal composition, and briefly recall it in appendix 4.3.1 .

the results of [LPV09], UC-puzzles are sufficient to obtain UC-secure protocols for general functionalities. Our protocol for UC-puzzles is obtained by combining our commitment scheme with a "cut-and-choose" protocol and (standard) zero-knowledge proofs for **NP** [GMW86, Blu87].

In contrast, in the direct access model, it is easy to show that UC-secure computation is impossible for most functionalities. A formal statement is obtained by essentially reproducing the Canetti-Fischlin impossibility result for UC-commitments [CF01] (details reproduced in appendix 4.5.1). The situation turns out to be the same for concurrent self-composition of two-party protocols: by reproducing the steps of Lindell's impossibility results [Lin03b, Lin08], concurrent self-composition in this model can be shown equivalent to universal composition. This means that secure computation of (most) functionalities in the concurrent self-composition model is impossible even with Captcha puzzles.

On modelling Captcha puzzles in the UC framework. The fact that UC-computation is possible in the indirect access model but concurrent self-composition is impossible raises the question whether indirect access model is the "right" model. What does a positive result in this model mean? To understand this, let us compare the indirect access model to the other "trusted setup" models such as the Common-Random-String (CRS) model [BSM91]. In the CRS-model, the simulator $\mathcal{S}$ is in control of generating the CRS in the ideal world—this enables $\mathcal{S}$ to have a "trapdoor" to continue its actions without having to "rewind" the environment. We can view the indirect access model as some sort of a setup (i.e., access to H) controlled by the simulator in the ideal world. The fact that $\mathcal{S}$ can see the queries made by $\mathcal{Z}$ to H in the indirect-access-model, is then analogous to $\mathcal{S}$ controlling the CRS in the CRS-model. The only difference between these two settings is that the indirect-access-model does *not* require any trusted third party. viewed this way, the indirect-access-model can be seen as a "hybrid" model that stands somewhere between a trusted setup (such as the CRS model) and the plain model.

Beyond Conservative Adversaries. An inherent difficulty when dealing with Captcha puzzles, is that of *malleability*. Informally, this means that given a challenge puzzle z, it might be possible for an algorithm $\mathcal{A}$ to efficiently generate a new puzzle z' such that given the solution of z', $\mathcal{A}$ can efficiently solve z. Such a malleability attack makes it difficult to reduce the security of a cryptographic scheme to the "hardness" of solving Captcha puzzles.

To overcome this, previous works [CHS06, Dzi10] only prove security against a very restricted class of adversaries called *conservative* adversaries. Such adversaries are essentially those who do not launch the 'malleability' attack: that is, they only query H on Captcha instances that are provided to them by the system. In both of these works, it is possible that a PPT adversary, on input a puzzle z may produce a puzzle z' such that the solutions of z and z' are related. But both works consider only restricted adversaries which are prohibited from querying H with such a mauled puzzle z'. As noted in [CHS06, Dzi10], this an unreasonable restriction, especially knowing that Captcha puzzles are in fact easily malleable.

In contrast, in this work, we prove the security of our schemes against the standard class of all probabilistic polynomial time (ppt) adversaries. The key-idea that enables us to go beyond the class of conservative adversaries is the formulation of the notion of an *human-extractable* Captcha puzzle. Informally speaking, an human-extractable Captcha puzzle, has the following property: suppose that a ppt algorithm $\mathcal{A}$ can solve a challenge puzzle z, and makes queries $\bar{q}$ to the human H during this process; then there is a ppt algorithm which on input the queries $\bar{q}$, can distinguish with the help of the human that $\bar{q}$ are correlated to z and not to some other randomly generated puzzle, say z''.

We discuss this notion at length in section 3.2, and many other issues related to formalizing Captcha puzzles. This section essentially builds and improves upon previous works of [ABH03, CHS06, Dzi10] to give a unified framework for working with Captcha puzzles. We view the notion of human-extractable Captcha puzzles as an important contribution to prove security beyond the class of conservative adversaries.

CHAPTER 2

Efficient Traitor Tracing

2.1 Preliminary Definitions

2.1.1 Traitor Tracing

A traitor tracing system provides protection for a broadcast encrypter. It consists of four algorithms: *Setup, Encrypt, Decrypt* and *Trace.* The *Setup* algorithm generates the secret keys for all the users in the system and the public parameters for the system. By using these public parameters and the algorithm *Encrypt*, any user can encrypt a message to all the users in the system. A recipient can use his secret key and the *Decrypt* algorithm to decrypt a ciphertext.

In case an authority discovers a pirate decoder, it can then use the $Trace$ algorithm to identify at least one of the users whose private key must have been used in the construction of the pirate decoder. A publicly traceable scheme is one where the $Trace$ algorithm has no secret inputs, i.e there are no tracing secret keys.

The desired security properties of a traitor tracing system are the following:

- **Semantic Security**: An adversary that does not have access to the secret key of any user should not be able to distinguish between encryptions of two messages of its choice.

- **Traceability Against Arbitrary Collusion**: Consider a case where an adversary has access to an arbitrary number of keys of its choice and generates a pirate decoder. Then the tracing algorithm should be able to use the pirate decoder and detect at least one of the users whose key must have been used to construct the pirate decoder.

2.1.2 Trace & Revoke

A Trace & Revoke system is a traitor tracing system that provides an additional property of user revocation. Once a set of rogue users are identified, the system allows for all honest parties to encrypt to the rest of the honest users securely. The system consists of four algorithms *Setup*, *Encrypt*, *Decrypt* and *Trace*. The *Setup* algorithm generates the secret keys for all the users in the system and the public parameters for the system. The *Encrypt* algorithm can be used to encrypt a message to *any* subset of users of the system. *Decrypt* is used to decrypt a valid ciphertext. In a secure Trace & Revoke system, the *Decrypt* algorithm of a user succeeds if and only if the encryption was intended for him (he belongs to the set of users that the message was encrypted to). The *Trace* algorithm is used to identify the key used inside a pirate decoder.

Boneh et al. [BSW06] introduce a new primitive, *Private Linear Broadcast Encryption* (PLBE) and showed that a PLBE is sufficient for implementing a fully collusion-resistant traitor tracing scheme. In this paper, we give an informal treatment (see [BSW06] for details) of traitor tracing systems and their relation to PLBE and present an improved PLBE scheme. However, we recall details on PLBE definitions and its security properties.

Boneh and Waters [BW06] introduce a new primitive, *Augmented Broadcast Encryption* (AugBE) and use an AugBE scheme (based on composite order bilinear groups) to implement a fully collusion-resistant trace & revoke scheme, secure against adaptive adversaries. We present an improved AugBE scheme based only on prime order groups.

2.1.3 PLBE

A *Private Linear Broadcast Encryption* (PLBE) system consists of four algorithms: $Setup_{PLBE}$, $Encrypt_{PLBE}$, $Decrypt_{PLBE}$, $TrEncrypt_{PLBE}$. The algorithms described below are similar to the BSW PLBE system [BSW06] except that our system *does not* need a tracing key.

- $(PK, K_1, K_2 \ldots K_N) \xleftarrow{\$} Setup_{PLBE}(\lambda)$: $Setup_{PLBE}$ algorithm takes as input the security parameter λ and sets up the public parameters PK for the system along with generating the secret keys $(K_1, K_2 \dots K_N)$ for all the users in the system. N is the number of users in the system.

- $C \overset{\$}{\leftarrow} \mathit{Encrypt}_{PLBE}(PK, M)$: Any user can encrypt a message M using just the public key PK, and any user that possess one of the secret keys can decrypt the ciphertext.
- $M \leftarrow \mathit{Decrypt}_{PLBE}(C, K_i, i)$: Any user i having access to the private key K_i can decrypt a ciphertext C and obtain the corresponding message M.
- $C \overset{\$}{\leftarrow} \mathit{TrEncrypt}_{PLBE}(PK, i, M)$: The $\mathit{TrEncrypt}_{PLBE}$ algorithm takes in a message M and encrypts it to ciphertext C such that only users $\{i \dots N\}$ with secret keys $(K_i, K_{i+1} \dots K_N)$ can decrypt the message. This algorithm is used only for tracing.

2.1.3.1 Desired Security Properties.

A PLBE system is considered secure if no adversary has significant advantage in the following games:

- **Indistinguishability:** This property requires that the ciphertexts generated by $\mathit{Encrypt}_{PLBE}(PK, M$ and $\mathit{TrEncrypt}_{PLBE}(PK, 1, M)$ are indistinguishable. The game between the adversary and the challenger proceeds as follows.
  - **Setup:** The challenger runs the Setup_{PLBE} algorithm and sends the generated public key PK and the secret keys $K_1, K_2 \dots K_N$ to the adversary.
  - **Challenge:** The adversary sends a message M to the challenger. The challenger flips an unbiased coin and obtains a random $\beta \in \{0, 1\}$. If $\beta = 0$, it then sets the ciphertext as $C \overset{\$}{\leftarrow} \mathit{Encrypt}_{PLBE}(PK, M)$, and as $C \overset{\$}{\leftarrow} \mathit{TrEncrypt}_{PLBE}(PK, 1, M)$ otherwise. It sends C to the adversary.
  - **Guess:** The adversary returns a guess $\beta' \in \{0, 1\}$ of β.

  The advantage of the adversary is $Adv_{ID} = |Pr[\beta' = \beta] - \frac{1}{2}|$.

- **Index Hiding:** This property prevents an adversary from distinguishing between $\mathit{TrEncrypt}_{PLBE}(PK, i, M)$ and $\mathit{TrEncrypt}_{PLBE}(PK, i + 1, M)$ when the adversary knows all the secret keys except the i^{th} secret key. The game between the adversary and the challenger proceeds as follows. The game takes the index i as input which is given as input to both the challenger and the adversary.

- **Setup:** The challenger runs the $Setup_{PLBE}$ algorithm and sends the generated public key PK and the secret keys $K_1, K_2 \ldots K_{i-1}, K_{i+1} \ldots K_N$ to the adversary. The adversary does not know K_i.
- **Challenge:** The adversary sends a message M to the challenger. The challenger flips an unbiased coin and obtains a random $\beta \in \{0, 1\}$. It sets the ciphertext as $C \xleftarrow{\$} TrEncrypt_{PLBE}(PK, i + \beta, M)$ and sends it to the adversary.
- **Guess:** The adversary returns a guess $\beta' \in \{0, 1\}$ of β.

The advantage of the adversary is $Adv_{IH}[i] = |Pr[\beta' = \beta] - \frac{1}{2}|$.

- **Message Hiding:** This property requires that an adversary can not break semantic security when encryption is performed on input $i = N + 1$. The game between the adversary and the challenger proceeds as follows.

  - **Setup:** The challenger runs the $Setup_{PLBE}$ algorithm and sends the generated public key PK and the secret keys $K_1, K_2 \ldots K_N$ to the adversary.
  - **Challenge:** The adversary sends messages M_0, M_1 to the challenger. The challenger flips an unbiased coin and obtains a random $\beta \in \{0, 1\}$. It sets the ciphertext as $C \xleftarrow{\$} TrEncrypt_{PLBE}(PK, N + 1, M_\beta)$ and sends it to the adversary.
  - **Guess:** The adversary returns a guess $\beta' \in \{0, 1\}$ of β.

  The advantage of the adversary is $Adv_{MH} = |Pr[\beta' = \beta] - \frac{1}{2}|$.

Definition 2.1.1. *An N-user PLBE system is considered secure if for all polynomial time adversaries Adv_{ID}, $Adv_{IH}[i]$ for all $i \in \{1 \ldots N\}$ and Adv_{MH} are negligible in the security parameter λ.*

2.1.4 AugBE

An *Augmented Broadcast Encryption* (AugBE) [BW06] system consists of three algorithms: $Setup_{AugBE}$, $Encrypt_{AugBE}$, $Decrypt_{AugBE}$.

- $(PK, K_1, K_2 \dots K_N) \overset{\$}{\leftarrow} Setup_{AugBE}(\lambda)$: $Setup_{AugBE}$ algorithm takes as input the security parameter λ and sets up the public parameters PK for the system along with generating the secret keys $(K_1, K_2 \dots K_N)$ for all the users in the system. N is the number of users in the system.

- $C \overset{\$}{\leftarrow} Encrypt_{AugBE}(S, PK, i, M)$: This algorithm takes as input a subset $S \subseteq \{1, \dots, N\}$ of users, the public key PK, and an index $1 \leq i \leq N+1$, and a message M. The algorithms outputs a ciphertext which can be decrypted by any user belonging to the set $S \cap \{i, \dots, N\}$. the ciphertext.

- $M \leftarrow Decrypt_{AugBE}(S, j, K_j, C, PK)$: A user j having access to the private key K_j can decrypt a ciphertext C and obtain the corresponding message M. If he is not able to decrypt he outputs $\perp$.

AugBE and PLBE system consists of similar algorithms. The only difference between the AugBE and PLBE systems is that PLBE algorithms do not take set S as input. The set of all users is implied each time set S is referred to. We refer the reader to [BSW06] for further details.

2.1.4.1 Desired Security Properties.

We now describe the security properties required of an AugBE system. The security properties required of a PLBE system are implied by the ones for an AugBE system under the condition that the set S is the set of all users. An AugBE system is considered secure if no adversary has significant advantage in the following games:

- **Index Hiding:** This property prevents an adversary from distinguishing between $Encrypt_{AugBE}(S, PK, i, M)$ and $Encrypt_{AugBE}(S, i+1, PK, M)$ when the adversary knows all the secret keys except the i^{th} secret key. Also when $i \notin S$, an adversary with access to all the private keys in the system, should not be able to tell if the encryption has been done to index i or $i+1$. The game between the adversary and the challenger proceeds as follows. The game takes the index i as input which is given as input to both the challenger and the adversary.

- **Setup:** The challenger runs the $Setup_{AugBE}$ algorithm and sends the generated public key PK and the secret keys $K_1, K_2 \ldots K_{i-1}, K_{i+1} \ldots K_N$ to the adversary. The adversary does not know K_i.
- **Query:** The adversary outputs a bit $s' \in \{0,1\}$. If $s' = 1$, the challenger sends the adversary K_i, else he does nothing.
- **Challenge:** The adversary sends a message M and a set $S \subseteq \{1, \ldots, N\}$ to the challenger. The only restriction is if $s' = 1$ then $i \notin S$. The challenger flips an unbiased coin and obtains a random $\beta \in \{0,1\}$. It sets the ciphertext as $C \xleftarrow{\$} Encrypt_{AugBE}(S, PK, i+\beta, M)$ and sends it to the adversary.
- **Guess:** The adversary returns a guess $\beta' \in \{0,1\}$ of β.

The advantage of the adversary is $Adv_{IH}[i] = |Pr[\beta' = \beta] - \frac{1}{2}|$.

- **Message Hiding:** This property requires that an adversary can not break semantic security when encryption is performed on input $i = N+1$. The game between the adversary and the challenger proceeds as follows.
  - **Setup:** The challenger runs the $Setup_{AugBE}$ algorithm and sends the generated public key PK and the secret keys $K_1, K_2 \ldots K_N$ to the adversary.
  - **Challenge:** The adversary sends messages M_0, M_1 and a set $S \subset \{1, \ldots, N\}$ to the challenger. The challenger flips an unbiased coin and obtains a random $\beta \in \{0,1\}$. It sets the ciphertext as $C \xleftarrow{\$} Encrypt_{AugBE}(S, PK, N+1, M_\beta)$ and sends it to the adversary.
  - **Guess:** The adversary returns a guess $\beta' \in \{0,1\}$ of β.

The advantage of the adversary is $Adv_{MH} = |Pr[\beta' = \beta] - \frac{1}{2}|$.

Definition 2.1.2. *An N-user AugBE system is considered secure if for all polynomial time adversaries Adv_{ID}, $Adv_{IH}[i]$ for all $i \in \{1 \dots N\}$ and Adv_{MH} are negligible in the security parameter λ.*

2.1.5 Equivalence of Traitor Tracing and PLBE

We have presented an intuition behind the argument. A more formal argument appears in [BSW06]. The tracing algorithm will be given a pirate decoder that is able to decrypt messages encrypted using $TrEncrypt(PK, 1, M)$ with significant probability. The probability of success of this pirate decoder, when encryption is done to user $N+1$, should be negligible because of the message hiding game. The tracing algorithm of the traitor tracing scheme estimates the probability of success of the adversary when the ciphertext is generated using $TrEncrypt(PK, i, M)$ for every $i \in \{1 \dots N+1\}$. Since the probability is being reduced from significant to negligible between encryptions to $TrEncrypt(PK, 1, M)$ and $TrEncrypt(PK, N+1, M)$, the probability must fall significantly for some $i \in \{1 \dots N+1\}$. We argue that the given pirate decoder could not have done this without the knowledge of the i^{th} key. If it didn't know the i^{th} key, then we could use this pirate decoder as an adversary in the Index Hiding game with parameter i and distinguish between $TrEncrypt(PK, i, M)$ and $TrEncrypt(PK, i+1, M)$ with significant probability. But this can not be true for a secure PLBE. Hence, we can use a secure PLBE to construct a traitor tracing scheme.

2.2 Background on Bilinear Maps

2.2.1 Bilinear Groups

Symmetric and Asymmetric Bilinear Groups of Prime Order. Consider three multiplicative cyclic groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ of prime orders (possibly different). Let g_1 be a generator of $\mathbb{G}_1$ and g_2 a generator of $\mathbb{G}_2$. Let r be the order of G_1, the smaller of the two groups. We define an efficiently computable bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ with the properties: (1) e is non-degenerate: $e(g_1, g_2)$ should not evaluate to the identity element of $\mathbb{G}_T$. (2) The map is bilinear: $\forall u \in \mathbb{G}_1$, $\forall v \in \mathbb{G}_2$ and $a, b \in \mathbb{Z}_r$ we should have $e(u^a, v^b) = e(u, v)^{ab}$. Such groups are refereed to as *Asymmetric Bilinear Groups of Prime Order.* Bilinear groups in which $\mathbb{G}_1 = \mathbb{G}_2 \equiv \mathbb{G}$ are called *Symmetric Bilinear Groups of Prime Order.* It can be seen that for such groups the bilinear map is symmetric since $e(g^a, g^b) = e(g, g)^{ab} = e(g^b, g^a)$.

Bilinear Groups of Composite Order. Bilinear groups of composite order are similar to the ones of prime order. The key difference is that the order of the groups $\mathbb{G}$ and $\mathbb{G}_T$ is composite. Lets say the order is n, where $n = pq$, p and q are large primes depending on the security parameter. We will use $\mathbb{G}_p$ and $\mathbb{G}_q$ to denote the order p and q subgroups of $\mathbb{G}$, respectively.

2.2.2 Complexity Assumptions

Let $\mathcal{G}$ be an algorithm that takes the security parameter λ as input and generates the tuple $(r, \mathbb{G}, \mathbb{G}_T, e)$.

Decision 3-party Diffie Hellman. This assumption is popular and has been used previously in a number of schemes including the PLBE scheme [BSW06]. A challenger generates a bilinear group $\mathbb{G}$ using $(r, \mathbb{G}, \mathbb{G}_T, e) \xleftarrow{\$} \mathcal{G}(\lambda)$. It generates a random generator g for the group $\mathbb{G}$. It chooses $a, b, c \xleftarrow{\$} \mathbb{Z}_r$.

An algorithm $\mathcal{A}$, solving the Decision 3-party Diffie Hellman problem is given $Z = (r, \mathbb{G}, \mathbb{G}_T, e, g, g^a, g^b, g^c)$. The challenger flips an unbiased coin and obtains a random $\beta \in \{0, 1\}$. If $\beta = 0$, it then sets $T = g^{abc}$ and $T = R$ otherwise, where $R \xleftarrow{\$} \mathbb{G}$. It then sends T to $\mathcal{A}$. The adversary returns a guess $\beta' \in \{0, 1\}$ of β. The advantage of $\mathcal{A}$ in this game is $Adv_{D3DH} = |Pr[\beta = \beta'] - \frac{1}{2}|$. The Decision 3-party Diffie Hellman assumption states that this advantage is negligible in the security parameter.

Decisional Linear Assumption. This is a simple extension of the Decisional Diffie Hellman (DDH) Assumption introduced in [BBS04a] for bilinear groups in which the DDH assumption is actually easy. A challenger generates a bilinear group $\mathbb{G}$ using $(r, \mathbb{G}, \mathbb{G}_T, e) \xleftarrow{\$} \mathcal{G}(\lambda)$. It generates a random generator g for the group $\mathbb{G}$. It chooses $a, b, c, x, y \xleftarrow{\$} \mathbb{Z}_r$.

An algorithm $\mathcal{A}$, solving the Decisional Linear Assumption problem is given

$$Z = (r, \mathbb{G}, \mathbb{G}_T, e, g, g^a, g^b, g^c, g^{ax}, g^{by}).$$

The challenger flips an unbiased coin and obtains a random $\beta \in \{0,1\}$. If $\beta = 0$, it then sets $T = g^{c(x+y)}$ and $T = R$ otherwise, where $R \xleftarrow{\$} \mathbb{G}$. It then sends T to $\mathcal{A}$. The adversary returns a guess $\beta' \in \{0,1\}$ of β. The advantage of $\mathcal{A}$ in this game is $Adv_{DLN} = |Pr[\beta = \beta'] - \frac{1}{2}|$. Decisional Linear Assumption states that this advantage is negligible in the security parameter.

External Diffie Hellman Assumption. The External Diffie Hellman (XDH) assumption states that the Decisional Diffie Hellman (DDH) assumption is hard in the group G_1. (Not necessarily hard in G_2). This assumption is believed to be true in asymmetric pairings generated using special MNT curves [MNT00, BBS04b].

Subgroup Decision Assumption. This problem was introduced by Boneh et al. [BGN05] and states that for a bilinear group $\mathbb{G}$ of composite order $n = pq$, any algorithm $\mathcal{A}$, given a random element $g \in \mathbb{G}$ and a random element $g_q \in \mathbb{G}_q$, can not distinguish between a random element in $\mathbb{G}$ and a random element in $\mathbb{G}_q$. This assumption is for composite order groups. We do not use this assumption in this work.

2.3 Key Ideas

We now present the intuition behind the working of [BSW06] for composite order bilinear groups and provide a generic construction to achieve the same properties using prime order bilinear groups. Consider a composite order bilinear group $\mathbb{G}_n$ of order n, where $n = pq$ and p, q are primes. Let us denote elements belonging to the p-order subgroup (called $\mathbb{G}_p$) and the q-order subgroup (called $\mathbb{G}_q$) of $\mathbb{G}_n$ by subscripts p and q, respectively. The BSW scheme [BSW06] (and most other composite order bilinear group based schemes) relies on the fact that if $g_p \in \mathbb{G}_p$ and $g_q \in \mathbb{G}_q$, then $e(g_p, g_q) = 1$. The same effect can be obtained in a prime order group by using vector spaces. For a group $\mathbb{G}$ of prime order r, with generator g, consider tuples of elements (g^a, g^b) (analogous to g_q) and (g^{-b}, g^a) (analogous to g_p) belonging to the vector space $V = \mathbb{G}^2$ (analogous to $\mathbb{G}_n$), where a, b are random in $\mathbb{Z}_r$. Define vectors $\vec{\boldsymbol{v_1}} = (a, b)$ and $\vec{\boldsymbol{v_2}} = (-b, a)$. Note that they are orthogonal vectors. The subspace V_p (analogous to $\mathbb{G}_p$) corresponds to the set of elements $(g^{a\tilde{p}}, g^{b\tilde{p}})$ such that $\tilde{p} \in \mathbb{Z}_r$; and similarly subspace, V_q (analogous to $\mathbb{G}_q$) corresponds to the set of elements $(g^{-b\tilde{q}}, g^{a\tilde{q}})$ such that $\tilde{q} \in \mathbb{Z}_r$. It is easy to see that pairing an element of V_p with an element of V_q computed[1] as $e(g^a, g^{-b}) \cdot e(g^b, g^a)$ yields the identity element (analogous to $e(g_p, g_q) = 1$).

Now we need to build on an analog of the subgroup decision assumption (SDH). SDH informally states that given an element of $\mathbb{G}$ and an element of $\mathbb{G}_q$, it is hard to distinguish a random element in $\mathbb{G}_q$ from a random element in $\mathbb{G}$. But this assumption does not hold with V_p and V_q. Given an element $(u, v) \in V_q$, we can construct $(v^{-1}, u) \in V_p$. Using these two elements, it is trivial to distinguish an element in V_q from an element in V.

To fix this problem we consider a 3-dimensional vector space, $V = \mathbb{G}^3$. Consider $\vec{\boldsymbol{v_1}} = (a, 0, c)$, $\vec{\boldsymbol{v_2}} = (0, b, c)$ and $\vec{\boldsymbol{v_3}} = \vec{\boldsymbol{v_1}} \times \vec{\boldsymbol{v_2}}$, where a, b, c are random elements in $\mathbb{Z}_r$. Now let us define the subspace V_q by all elements $(g^{a\tilde{q}}, g^{b\tilde{q}'}, g^{c(\tilde{q}+\tilde{q}')})$ such that $\tilde{q}, \tilde{q}' \in \mathbb{Z}_r$, and let the subspace V_p be defined by elements $(g^{-bc\tilde{p}}, g^{-ac\tilde{p}}, g^{ab\tilde{p}})$ such that $\tilde{p} \in \mathbb{Z}_r$. For this system, also pairing an element of V_q with an element of V_p yields the identity element. This system also has an analog of the subgroup decision assumption. Given (g^a, g^b, g^c), we want it to be hard to distinguish a random element $(g^{a\tilde{q}}, g^{b\tilde{q}'}, g^{c(\tilde{q}+\tilde{q}')}) \in V_q$ from an element $(g^{x_1}, g^{x_2}, g^{x_3}) \in V$, where x_1, x_2, x_3 are random. This follows directly from the decisional linear assumption [BBS04a].

The main difference between the subspaces defined using composite order bilinear groups and subspaces defined using prime order bilinear groups is the flexibility in the way elements from the sub-spaces can be manipulated. In the case of composite order bilinear groups, it is easy to randomize elements from the sub-space V_q; but on the other hand, for prime order groups similar randomization is hard. This prevents the transformation from being applicable in general.

A direct compilation of the BSW traitor tracing scheme with the new ideas presented

[1] $e((g^x, g^y), (g^{x'}, g^{y'}))$ is evaluated as $e(g^x, g^{x'}) \cdot e(g^y, g^{y'})$.

earlier doesn't work because of the reasons mentioned in the previous paragraph. But this can be fixed by allowing the encrypter to define the subspaces at the time of encryption. This was not possible in the BSW traitor tracing scheme [BSW06] because the construction was dependent on the primes p, q. More generally, this trick allows, and in fact, necessitates a *late binding* of the parameters that define the subspaces. Other schemes satisfying this property should also be easy to simplify using our trick. Another crucial difference between our scheme and the BSW scheme is that our scheme does not have subspaces in the target group. Even some of the elements in the base group are not moved to the vector space.

2.4 Our Construction

In this paper we present two new traitor tracing schemes and corresponding trace & revoke systems. As already pointed out in section 2.2 a PLBE scheme is sufficient to construct a traitor tracing system and an AugBE scheme is sufficient to construct a trace & revoke system. In this section we present our PLBE and AugBE improving on the previous schemes [BSW06, BW06]. The schemes in the symmetric and the asymmetric prime order bilinear groups are fundamentally different. It should be noted that all our schemes allow for public traceability. The PLBE schemes can be obtained by dropping certain terms from the AugBE scheme which we describe towards the end of the section.

The number of users in the system, N, is assumed to be equal to m^2 for some m. If the number of users is not a perfect square, then we add some *dummy* users to pad N to the next perfect square. These *dummy* users do not take part in the system in any way. We arrange the users in an $m \times m$ matrix. The user $u : 1 \leq u \leq N$ in the system is identified by the (x, y) entry of the matrix, where $1 \leq x, y \leq m$ and $u = (x-1) \cdot m + y$.

The ciphertext generated by $Encrypt_{AugBE}$ consists of a ciphertext component for every row and a component for every column. For each row x the ciphertext consists of (A_x, B_x, $\vec{\boldsymbol{R}}_{\boldsymbol{x}}$, $\tilde{\vec{\boldsymbol{R}}}_{\boldsymbol{x}}$) and for every column y the ciphertext consists of ($\vec{\boldsymbol{C}}_{\boldsymbol{y}}$, $\tilde{\vec{\boldsymbol{C}}}_{\boldsymbol{y}}$).

Fully collusion resistant traitor tracing (or trace & revoke) is hard because we need to garble parts of the ciphertext making sure that it only impacts a certain subset of the users.

This is made possible by having a ciphertext term have components along different subspaces. For the purposes of this paper, we use the notation V to represent the space of ciphertext elements. The elements in this space can have orthogonal components along V_q and V_p. The information about the sub-space V_q is public while the information for V_p is private.

An encryption to position (i, j) means that only users (x, y) with $x > i$ or $x = i$ & $y \geq j$ can decrypt the message. An encryption to position (i, j) is obtained in the following way. (It is further illustrated in Figure 2.1.)

- **Column Ciphertext Components:** Column ciphertext components for columns $y \geq j$ are *well formed* in both subspaces V_p and V_q, while for columns $y < j$ are *well formed* in V_q but are random in V_p.

- **Row Ciphertext Components:** Row ciphertext components for rows $x < i$ are completely random, and these recipients can not obtain the message information theoretically. For row $x = i$, the row ciphertext is *well formed* in both V_p and V_q. And for rows $x > i$ they are *well formed* in V_q and have no component in V_p.

  A user in row i will be able to decrypt if the column ciphertext is also well formed in both V_p and V_q. However a user in rows $x > i$, will always be able to decrypt because the row ciphertexts for $x > i$ do not have any component in V_p, and the component of column ciphertexts in V_p will simply cancel out with the row ciphertexts.

In the AugBE scheme in addition to the above properties there is a set S that specifies the set of users to which encryption is done. In other words only users in that set can decrypt.

2.4.1 AugBE using Symmetric Bilinear Groups

We introduce some notation before we go further and describe the scheme. For a given vector $\vec{\boldsymbol{v}} = (v_1, \ldots v_i)$, by $g^{\vec{\boldsymbol{v}}}$ we mean the vector $(g^{v_1} \ldots g^{v_i})$. A pairing e on two vectors $\vec{\boldsymbol{R}}$ and $\vec{\boldsymbol{C}}$ is defined by multiplication after the componentwise pairing operation, i.e. $e(\vec{\boldsymbol{R}}, \vec{\boldsymbol{C}}) = \prod_{k=1}^{i} e(R_k, C_k)$, where e is the pairing operation on the underlying group elements. Given a set S of users to which encryption is to be done let $S_x = \{y : (x, y) \in S\}$.

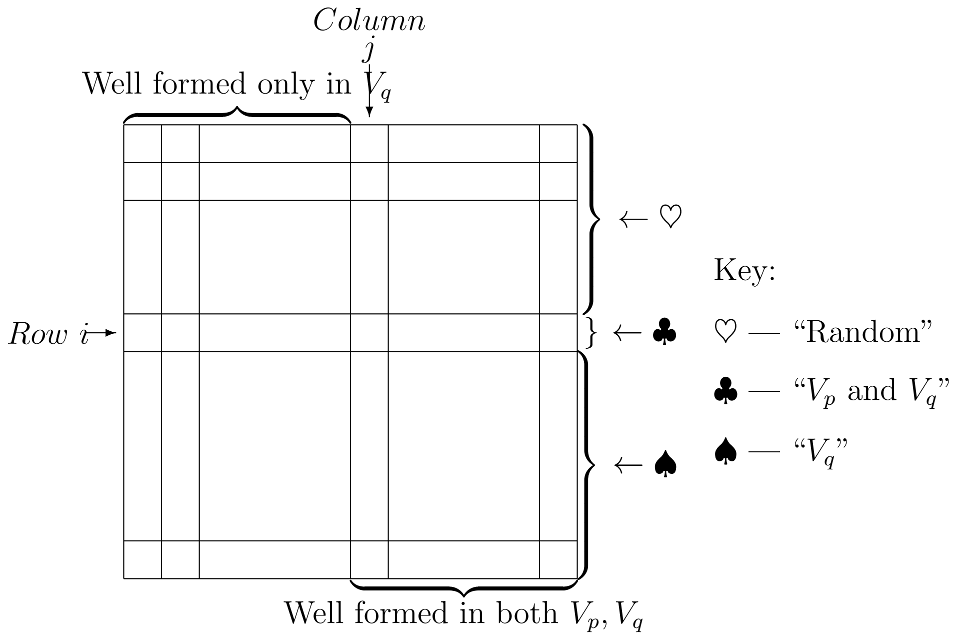


Figure 2.1: Illustration: [BSW06] Traitor tracing scheme

The AugBE scheme consists of the algorithms: $Setup_{AugBE}$, $Encrypt_{AugBE}$, $Decrypt_{AugBE}$.

- $(PK, K_{(1,1)}, \cdots K_{(1,m)}, K_{(2,1)} \cdots K_{(m,m)}) \leftarrow Setup_{AugBE}\ (1^{\lambda}, N = m^2)$

  The $Setup_{AugBE}$ algorithm takes as input the security parameter λ and the number of users N in the system. The algorithm generates a prime order groups $\mathbb{G}$ with a pairing $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_{\mathbb{T}}$. It outputs, g the generator of $\mathbb{G}$ and let r (depends on the security parameter) denote the size of $\mathbb{G}$. It then chooses random $r_1, r_2, r_3, \ldots r_m$, $c_1, c_2 \ldots c_m$, $\alpha_1, \alpha_2 \ldots \alpha_m \in \mathbb{Z}_r$. The public key PK of the AugBE system (along with the group description) is set to:

$$g, E_1 = g^{r_1}, E_2 = g^{r_2}, \ldots, E_m = g^{r_m},$$

$$G_1 = e(g,g)^{\alpha_1}, G_2 = e(g,g)^{\alpha_2}, \ldots, G_m = e(g,g)^{\alpha_m},$$

$$g, H_1 = g^{c_1}, H_2 = g^{c_2}, \ldots, H_m = g^{c_m}$$

$$u_1, u_2, \ldots, u_m \in \mathbb{G}.$$

  The secret key of each user (x, y) is $K_{(x,y)} = \{g^{\alpha_x} \cdot g^{r_x c_y} \cdot u_y^{\sigma_{x,y}}, g^{\sigma_{x,y}}, \forall i, (i \neq y), u_y^{\sigma_{x,y}}\}$.

- $C \leftarrow Encrypt_{AugBE}(PK, S, (i,j), M)$

  This algorithm allows the tracing party to encrypt a message to the recipients who have row value greater than i or those who have row value equal to i and column value greater than or equal to j and belonging to the set S. The algorithm chooses random $t, \eta,\ s_1, s_2 \dots s_m \in \mathbb{Z}_r$. It also chooses random $a, b, c \in \mathbb{Z}_r$ and sets $\vec{\boldsymbol{v_1}} = (a, 0, c)$, $\vec{\boldsymbol{v_2}} = (0, b, c)$ and $\vec{\boldsymbol{v_3}} = \vec{\boldsymbol{v_1}} \times \vec{\boldsymbol{v_2}}$. All elements $g^{\vec{\boldsymbol{v}}}$ when $\vec{\boldsymbol{v}}$ is a linear combination of $\vec{\boldsymbol{v_1}}$ and $\vec{\boldsymbol{v_2}}$ define the V_q space. These elements define the space V_p when the vector $\vec{\boldsymbol{v}}$ is parallel to $\vec{\boldsymbol{v_3}}$. Choose $\vec{\boldsymbol{w_1}}, \vec{\boldsymbol{w_2}}, \dots, \vec{\boldsymbol{w_m}}, \vec{\boldsymbol{v_c}} \in \mathbb{Z}_r^3$. Let $\vec{\boldsymbol{v'_c}} = \vec{\boldsymbol{v_c}} + v_{cr} \cdot \vec{\boldsymbol{v_3}}$ be another vector, with v_{cr} randomly chosen from $\mathbb{Z}_r$.

  For each row, $1 \le x < i$, choose random $\vec{\boldsymbol{z_x}} \in \mathbb{Z}_r^3$ and $a_x, b_x \in \mathbb{Z}_r$. The row cipher text components are,

$$\vec{\boldsymbol{R_x}} = g^{\vec{\boldsymbol{z_x}}} \qquad \tilde{\vec{\boldsymbol{R}}}_{\boldsymbol{x}} = g^{\eta \vec{\boldsymbol{z_x}}}$$

$$A_x = g^{a_x} \qquad B_x = G_x^{b_x}$$

$$T_x = (\prod_{k \in S_x} u_k)^{a_x}$$

  For row, $x = i$, pick random $\vec{\boldsymbol{v_i}}$ randomly $\in \mathbb{Z}_r^3$. Note that $\vec{\boldsymbol{v_i}} \cdot \vec{\boldsymbol{v'_c}} \neq \vec{\boldsymbol{v_i}} \cdot \vec{\boldsymbol{v_c}}$. This prevents parties (i, y), with $y < j$ from decrypting the message.

  The row cipher text component for $x = i$ is,

$$\vec{\boldsymbol{R_i}} = g^{r_i s_i \vec{\boldsymbol{v_i}}} \qquad \tilde{\vec{\boldsymbol{R}}}_{\boldsymbol{i}} = g^{\eta r_i s_i \vec{\boldsymbol{v_i}}}$$

$$A_i = g^{s_i t (\vec{\boldsymbol{v_i}} \cdot \vec{\boldsymbol{v_c}})} \qquad B_i = M \cdot G_i^{s_i t (\vec{\boldsymbol{v_i}} \cdot \vec{\boldsymbol{v_c}})}$$

$$T_i = (\prod_{k \in S_i} u_k)^{s_i t (\vec{\boldsymbol{v_i}} \cdot \vec{\boldsymbol{v_c}})}$$

  For rows, $x > i$, pick random $\vec{\boldsymbol{v_x}} = \tilde{q}_x \vec{\boldsymbol{v_1}} + \tilde{q}'_x \vec{\boldsymbol{v_2}}$ where $\tilde{q}_x, \tilde{q}'_x$ are random $\in \mathbb{Z}_r$. Note that $\vec{\boldsymbol{v_x}} \cdot \vec{\boldsymbol{v'_c}} = \vec{\boldsymbol{v_x}} \cdot \vec{\boldsymbol{v_c}}$. This allows all parties (x, y) to decrypt the message, if $x > i$.

The row cipher text components for all $x > i$ are,

$$\vec{\boldsymbol{R}}_{\boldsymbol{x}} = g^{r_x s_x \vec{\boldsymbol{v}}_{\boldsymbol{x}}} \qquad \tilde{\vec{\boldsymbol{R}}}_{\boldsymbol{x}} = g^{\eta r_x s_x \vec{\boldsymbol{v}}_{\boldsymbol{x}}}$$

$$A_x = g^{s_x t(\vec{\boldsymbol{v}}_{\boldsymbol{x}} \cdot \vec{\boldsymbol{v}}_{\boldsymbol{c}})} \qquad B_x = M \cdot G_x^{s_x t(\vec{\boldsymbol{v}}_{\boldsymbol{x}} \cdot \vec{\boldsymbol{v}}_{\boldsymbol{c}})}$$

$$T_x = (\prod_{k \in S_x} u_k)^{s_x t(\vec{\boldsymbol{v}}_{\boldsymbol{x}} \cdot \vec{\boldsymbol{v}}_{\boldsymbol{c}})}$$

And for every column $y < j$, the column ciphertext components are,

$$\vec{\boldsymbol{C}}_{\boldsymbol{y}} = g^{c_y t \vec{\boldsymbol{v}}'_{\boldsymbol{c}}} \cdot g^{\eta \vec{\boldsymbol{w}}_{\boldsymbol{y}}} \qquad \tilde{\vec{\boldsymbol{C}}}_{\boldsymbol{y}} = g^{\vec{\boldsymbol{w}}_{\boldsymbol{y}}}$$

And for every column $y \geq j$, the column ciphertext components are,

$$\vec{\boldsymbol{C}}_{\boldsymbol{y}} = g^{c_y t \vec{\boldsymbol{v}}_{\boldsymbol{c}}} \cdot g^{\eta \vec{\boldsymbol{w}}_{\boldsymbol{y}}} \qquad \tilde{\vec{\boldsymbol{C}}}_{\boldsymbol{y}} = g^{\vec{\boldsymbol{w}}_{\boldsymbol{y}}}$$

- $M \leftarrow Decrypt_{AugBE}(C, S, K_{(x,y)}, (x, y))$

  Let $K'_{(x,y)} = g^{\alpha_x} g^{r_x c_y} \prod_{k \in S_x} u_k^{\sigma_{x,y}}$ be the key used by recipient (x, y). Note that user (x, y) can always compute the product when $y \in S_x$ and cannot compute this product otherwise.

$$M = \frac{B_x}{\frac{e(K'_{(x,y)}, A_x)}{e(T_x, g^{\sigma_{x,y}})}} \cdot \frac{e(\vec{\boldsymbol{R}}_{\boldsymbol{x}}, \vec{\boldsymbol{C}}_{\boldsymbol{y}})}{e(\tilde{\vec{\boldsymbol{R}}}_{\boldsymbol{x}}, \tilde{\vec{\boldsymbol{C}}}_{\boldsymbol{y}})}$$

The broadcast encryption procedure is to obtained by encrypting using $Encrypt_{AugBE}(PK, 0, M)$. This illustrates the public traceability of our system.The correctness of decryption follows by inspection.

2.4.2 AugBE using Asymmetric Bilinear Groups

The AugBE scheme consists of the algorithms: $Setup_{AugBE}$, $Encrypt_{AugBE}$, $Decrypt_{AugBE}$.

- $(PK, K_{(1,1)}, \cdots K_{(1,m)}, K_{(2,1)} \cdots K_{(m,m)}) \leftarrow Setup_{AugBE}\ (1^\lambda, N = m^2)$

  The $Setup_{AugBE}$ algorithm takes as input the security parameter λ and the number of users N in the system. The algorithm generates two prime order groups $\mathbb{G}_1, \mathbb{G}_2$ with a pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_\mathbb{T}$. It outputs, g_1 and g_2, generators of $\mathbb{G}_1$ and $\mathbb{G}_2$ and let

r (depends on the security parameter) denote the size of $\mathbb{G}_1$. It then chooses random $r_1, r_2, r_3, \ldots r_m,\ c_1, c_2 \ldots c_m,\ \alpha_1, \alpha_2 \ldots \alpha_m \in \mathbb{Z}_r$. The public key PK of the AugBE system (along with the group description) is set to:

$$g_1, E_1 = g_1^{r_1}, E_2 = g_1^{r_2}, \ldots, E_m = g_1^{r_m},$$

$$G_1 = e(g_1, g_2)^{\alpha_1}, G_2 = e(g_1, g_2)^{\alpha_2}, \ldots, G_m = e(g_1, g_2)^{\alpha_m},$$

$$g_2, H_1 = g_2^{c_1}, H_2 = g_2^{c_2}, \ldots, H_m = g_2^{c_m}$$

$$u_1, u_2, \ldots, u_m \in \mathbb{G}_2.$$

The secret key of each user (x, y) is $K_{(x,y)} = \{g_2^{\alpha_x} \cdot g_2^{r_x c_y} \cdot u_y^{\sigma_{x,y}}, g_2^{\sigma_{x,y}}, \forall i, (i \neq y), u_y^{\sigma_{x,y}}\}$.

- $C \leftarrow Encrypt_{AugBE}(PK, S, (i, j), M)$

The algorithm chooses random $t, \eta, s_1, s_2 \ldots s_m \in \mathbb{Z}_r$ and $\vec{\boldsymbol{w_1}}, \vec{\boldsymbol{w_2}}, \ldots, \vec{\boldsymbol{w_m}}, \vec{\boldsymbol{v_1}}, \vec{\boldsymbol{v_c}} \in \mathbb{Z}_r^2$. Let $\vec{\boldsymbol{v_2}}$ be a random vector $\in \mathbb{Z}_r^2$ such that $\vec{\boldsymbol{v_1}} \cdot \vec{\boldsymbol{v_2}} = 0$. Let $\vec{\boldsymbol{v'_c}} = \vec{\boldsymbol{v_c}} + v_{cr} \cdot \vec{\boldsymbol{v_2}}$ be another vector, with v_{cr} randomly chosen from $\mathbb{Z}_r$. Note that, $\vec{\boldsymbol{v'_c}} \cdot \vec{\boldsymbol{v_1}} = \vec{\boldsymbol{v_c}} \cdot \vec{\boldsymbol{v_1}}$, a key property we will use in the correctness of our scheme.

All elements $g^{\vec{\boldsymbol{v}}}$ when $\vec{\boldsymbol{v}}$ is a along $\vec{\boldsymbol{v_1}}$ define the V_q space. These elements belong to the space V_p when the vector $\vec{\boldsymbol{v}}$ is parallel to $\vec{\boldsymbol{v_2}}$. Based on the XDH assumption the details about the V_p space are private.

For each row, $1 \leq x < i$, pick random $\vec{\boldsymbol{z_x}} \in \mathbb{Z}_r^2$ and $a_x, b_x \in \mathbb{Z}_r$. The row cipher text components are,

$$\vec{\boldsymbol{R_x}} = g_1^{\vec{\boldsymbol{z_x}}} \qquad \tilde{\vec{\boldsymbol{R}}}_{\boldsymbol{x}} = g_1^{\eta \vec{\boldsymbol{z_x}}}$$

$$A_x = g_1^{a_x} \qquad B_x = G_x^{b_x}$$

$$T_x = (\prod_{k \in S_x} u_k)^{a_x}$$

For row, $x = i$, pick random vector $\vec{\boldsymbol{v_i}} \in \mathbb{Z}_r^2$. Note that $\vec{\boldsymbol{v_i}} \cdot \vec{\boldsymbol{v'_c}} \neq \vec{\boldsymbol{v_i}} \cdot \vec{\boldsymbol{v_c}}$. This is prevent parties (i, y), with $y < j$ from decrypting the message.

The row cipher text component for $x = i$ is,

$$\vec{\boldsymbol{R}}_{\boldsymbol{i}} = g_1^{r_i s_i \vec{\boldsymbol{v}}_{\boldsymbol{i}}} \qquad \tilde{\vec{\boldsymbol{R}}}_{\boldsymbol{i}} = g_1^{\eta r_i s_i \vec{\boldsymbol{v}}_{\boldsymbol{i}}}$$

$$A_i = g_2^{s_i t(\vec{\boldsymbol{v}}_{\boldsymbol{i}} \cdot \vec{\boldsymbol{v}}_{\boldsymbol{c}})} \qquad B_i = M \cdot G_i^{s_i t(\vec{\boldsymbol{v}}_{\boldsymbol{i}} \cdot \vec{\boldsymbol{v}}_{\boldsymbol{c}})}$$

$$T_i = (\prod_{k \in S_i} u_k)^{s_i t(\vec{\boldsymbol{v}}_{\boldsymbol{i}} \cdot \vec{\boldsymbol{v}}_{\boldsymbol{c}})}$$

For rows, $x > i$, pick random $v'_x \in \mathbb{Z}_r$, let $\vec{\boldsymbol{v}}_{\boldsymbol{x}} = v'_x \cdot v_1$. Note that $\vec{\boldsymbol{v}}_{\boldsymbol{x}} \cdot \vec{\boldsymbol{v}}'_{\boldsymbol{c}} = \vec{\boldsymbol{v}}_{\boldsymbol{x}} \cdot \vec{\boldsymbol{v}}_{\boldsymbol{c}}$. This allows all parties (x, y), for all values of y to decrypt the message, if $x > i$.

The row cipher text components for all $x > i$ are,

$$\vec{\boldsymbol{R}}_{\boldsymbol{x}} = g_1^{r_x s_x \vec{\boldsymbol{v}}_{\boldsymbol{x}}} \qquad \tilde{\vec{\boldsymbol{R}}}_{\boldsymbol{x}} = g_1^{\eta r_x s_x \vec{\boldsymbol{v}}_{\boldsymbol{x}}}$$

$$A_x = g_2^{s_x t(\vec{\boldsymbol{v}}_{\boldsymbol{x}} \cdot \vec{\boldsymbol{v}}_{\boldsymbol{c}})} \qquad B_x = M \cdot G_x^{s_x t(\vec{\boldsymbol{v}}_{\boldsymbol{x}} \cdot \vec{\boldsymbol{v}}_{\boldsymbol{c}})}$$

$$T_x = (\prod_{k \in S_x} u_k)^{s_x t(\vec{\boldsymbol{v}}_{\boldsymbol{x}} \cdot \vec{\boldsymbol{v}}_{\boldsymbol{c}})}$$

And for every column $y < j$, the column ciphertext components are,

$$\vec{\boldsymbol{C}}_{\boldsymbol{y}} = g_2^{c_y t \vec{\boldsymbol{v}}'_{\boldsymbol{c}}} \cdot g_2^{\eta \vec{\boldsymbol{w}}_{\boldsymbol{y}}} \qquad \tilde{\vec{\boldsymbol{C}}}_{\boldsymbol{y}} = g_2^{\vec{\boldsymbol{w}}_{\boldsymbol{y}}}$$

And for every column $y \geq j$, the column ciphertext components are,

$$\vec{\boldsymbol{C}}_{\boldsymbol{y}} = g_2^{c_y t \vec{\boldsymbol{v}}_{\boldsymbol{c}}} \cdot g_2^{\eta \vec{\boldsymbol{w}}_{\boldsymbol{y}}} \qquad \tilde{\vec{\boldsymbol{C}}}_{\boldsymbol{y}} = g_2^{\vec{\boldsymbol{w}}_{\boldsymbol{y}}}$$

- $M \leftarrow Decrypt_{PLBE}(C, S, K_{(x,y)}, (x, y))$

  Let $K'_{(x,y)} = g_2^{\alpha_x} g_2^{r_x c_y} \prod_{k \in S_x} u_k^{\sigma_{x,y}}$ be the key used by recipient (x, y). Note that user (x, y) can always compute the product when $y \in S_x$ and cannot compute this product otherwise.

$$M = \frac{B_x}{\frac{e(A_x, K'_{(x,y)})}{e(T_x, g_2^{\sigma_{x,y}})}} \cdot \frac{e(\vec{\boldsymbol{R}}_{\boldsymbol{x}}, \vec{\boldsymbol{C}}_{\boldsymbol{y}})}{e(\tilde{\vec{\boldsymbol{R}}}_{\boldsymbol{x}}, \tilde{\vec{\boldsymbol{C}}}_{\boldsymbol{y}})}$$

The normal encryption procedure is to just encrypt to
$Encrypt_{AugBE}(PK, 0, M)$. This illustrates the public traceability of our system. The correctness of decryption follows by inspection.

2.4.3 PLBE

The two AugBE schemes based on symmetric and asymmetric prime order groups respectively can be converted to the corresponding PLBE schemes by removing the u terms from the public key. We will also need to get rid of the u and σ terms in the secret key. Row ciphertexts will not include T_x terms and decryption will not require a pairing corresponding to the term T_x. Rest of the parts of the scheme remain the same. Details can be found in an earlier version [GKS09] of this paper.

2.5 Security Proof

Here we only give the proof for the AugBE scheme using symmetric prime order bilinear groups. The proof for the AugBE scheme based on asymmetric prime order bilinear groups is also similar. The only difference is that security depends on the XDH assumption. The security of the PLBE schemes is implied by the security of the AugBE schemes.

2.5.1 Index Hiding

Theorem 2.5.1. *If the Decision 3-party Diffie Hellman assumption and the decisional linear assumption hold, then no probabilistic polynomial time adversary can distinguish between an encryption to two adjacent recipients in the index hiding game for any* (i, j) *where* $1 \leq i, j \leq m$ *with non-negligible probability.*

Proof. We consider two possible cases. First, when the adversary tries to distinguish between ciphertexts encrypted to (i, j) and $(i, j+1)$ when $1 \leq j < m$. Second, when the adversary tries to distinguish between ciphertexts encrypted to (i, m) and $(i+1, 1)$ when $1 \leq i < m$. The first case follows by Lemma 2.5.2 and the second case follows by Lemma 2.5.3. ■

Lemma 2.5.2. *If the Decision 3-party Diffie Hellman assumption holds, then no probabilistic polynomial time adversary can distinguish between an encryption to recipient* (i, j) *and* $(i, j+1)$ *in the index hiding game for any* (i, j) *where* $j < m$ *with non-negligible probability.*

Proof. This proof is similar to proof of Lemma 5.2 of [BSW06]. Consider an adversary $\mathcal{A}$ that succeeds in the index hiding game with a probability greater than ε. The adversary is considered successful if it can distinguish between encryptions made to positions (i, j) and $(i, j+1)$. We build a reduction $\mathcal{R}$ that uses $\mathcal{A}$ to solve the Decision 3-party Diffie Hellman problem. The reduction receives the Decision 3-party Diffie Hellman challenge as:

$$\mathbb{G}, g, A = g^a, B = g^b, C = g^c, T$$

and it is expected to guess if T is g^{abc} or if it is random.

In this system two cases arise. The simulator guesses the challenge value s' and generates the public parameters correctly. In case the value of the s' does not match the value later provided by the adversary then the simulator aborts. Since the simulator will successfully guess the right value of s' with probability at least 1/2 the simulation will work with probability at least 1/2.

Case 1: $s' = 0$**:** In the Setup phase the reduction based on the input (i, j) (the row and column the adversary will attack) sets up the public and the private parameters. The reduction chooses random $r_1, r_2, \ldots r_m, c_1, c_2 \ldots c_m, \alpha_1, \alpha_2 \ldots \alpha_m, \delta_1, \delta_2 \ldots \delta_m \in \mathbb{Z}_r$. It also chooses $\sigma_{x,y} \in \mathbb{Z}_r$ for every $x, y \in \{1 \ldots m\}$. It sets up the public parameters as:

$$g, E_1 = g^{r_1}, E_2 = g^{r_2}, \ldots E_i = B^{r_i} \ldots, E_m = g^{r_m},$$

$$G_1 = e(g,g)^{\alpha_1}, G_2 = e(g,g)^{\alpha_2}, \ldots, G_m = e(g,g)^{\alpha_m},$$

$$H_1 = g^{c_1}, H_2 = g^{c_2}, \ldots H_j = C^{c_j} \ldots, H_m = g^{c_m}$$

$$u_1 = g^{\delta_1}, u_2 = g^{\delta_2} \ldots u_m = g^{\delta_m}$$

And the private key $K_{(x,y)}$ of user (x,y) is:

$$K_{(x,y)} = \{g^{\alpha_x} \cdot B^{r_x \cdot c_y} \cdot u_y^{\sigma_{x,y}}\} : x = i, y \neq j$$

$$K_{(x,y)} = \{g^{\alpha_x} \cdot C^{r_x \cdot c_y} \cdot u_y^{\sigma_{x,y}}\} : x \neq i, y = j$$

$$K_{(x,y)} = \{g^{\alpha_x} \cdot g^{r_x \cdot c_y} \cdot u_y^{\sigma_{x,y}}\} : x \neq i, y \neq j$$

The private keys also contain $\{g^{\sigma_{x,y}}, \forall i, (i \neq y), u_y^{\sigma_{x,y}}\}$ for each user (x, y). Note that the distribution of the public and private parameters matches the distribution of parameters in the real scheme.

In the challenge phase the adversary sends the message $M \in \mathbb{G}_T$ to the reduction. The reduction then chooses random $t, w_{1,k}, w_{2,k}, \dots w_{m,k}, s_1, s_2 \dots s_m \in \mathbb{Z}_r$ where $k = \{1, 2, 3\}$. It also chooses random $a, b, c \in \mathbb{Z}_r$ and sets $\vec{\boldsymbol{v_1}} = (a, 0, c)$, $\vec{\boldsymbol{v_2}} = (0, b, c)$ and $\vec{\boldsymbol{v_3}} = (-bc, -ac, ab)$.

Set $g^\eta = B$.

It then sets $\vec{\boldsymbol{v_c}} = (v_{c,1}, v_{c,2}, v_{c,3})$ where $v_{c,1}, v_{c,2}, v_{c,3}$ are chosen randomly in $\mathbb{Z}_r$. Let $\vec{\boldsymbol{v^q}}$ denote the projection of $\vec{\boldsymbol{v}}$ along the plane formed by $\vec{\boldsymbol{v_1}}$ and $\vec{\boldsymbol{v_2}}$. And let $\vec{\boldsymbol{v^p}}$ be the component along $\vec{\boldsymbol{v_3}}$.

It also chooses random $z_{1,x,k}, z_{2,x}, z_{3,x} \in \mathbb{Z}_r$ where $1 \le x < i$ and $k \in \{1, 2, 3\}$ and sets up the ciphertext as follows.

$$\begin{aligned} x < i: \quad & R_{x,k} = g^{z_{1,x,k}} : k = \{1,2,3\} \\ & \widetilde{R}_{x,k} = g^{z_{1,x,k}\eta} : k = \{1,2,3\} \\ & A_x = g^{z_{2,x}} \\ & B_x = e(g,g)^{z_{3,x}} \\ & T_x = A_x{}^{\sum_{k \in S_x} \delta_k} \end{aligned} \tag{2.5.1}$$

It sets $\vec{\boldsymbol{v_i}} = \tilde{q}_i \cdot \vec{\boldsymbol{v_3}} + \tilde{q}'_i \cdot \vec{\boldsymbol{v_3}} + \tilde{p}_i \cdot \vec{\boldsymbol{v_3}}$ where $\tilde{q}_i, \tilde{q}'_i, \tilde{p}_i$ are random in $\mathbb{Z}_r$.

$$\begin{aligned} x = i: \quad & R_{i,k} = g^{r_i v_{i,k} s_i} : k = \{1,2,3\} \\ & \widetilde{R}_{i,k} = B^{r_i v_{i,k} s_i} : k = \{1,2,3\} \\ & A_i = A^{s_i t(\vec{\boldsymbol{v_i^p}} \cdot \vec{\boldsymbol{v_c^p}})} \cdot g^{s_i t(\vec{\boldsymbol{v_i^q}} \cdot \vec{\boldsymbol{v_c^q}})} \\ & B_i = M \cdot e(g, A_i)^{\alpha_i} \\ & T_i = A_i{}^{\sum_{k \in S_i} \delta_k} \end{aligned}$$

For each $x \in \{i+1 \cdots m\}$, it picks $\vec{\boldsymbol{v_x}} = \vec{\boldsymbol{v_x^q}} = \tilde{q}_x \cdot \vec{\boldsymbol{v_1}} + \tilde{q}'_x \cdot \vec{\boldsymbol{v_2}}$ where $\tilde{q}_x, \tilde{q}'_x$ are random in $\mathbb{Z}_r$.

$$\begin{aligned} x > i: \quad & R_{x,k} = g^{r_x v^q_{x,k} s_x} : k = \{1,2,3\} \\ & \widetilde{R}_{x,k} = B^{r_x v^q_{x,k} s_x} : k = \{1,2,3\} \\ & A_x = B^{s_x t(\vec{\boldsymbol{v_x^q}} \cdot \vec{\boldsymbol{v_c}})} \\ & B_x = M \cdot e(g,B)^{\alpha_x s_x t(\vec{\boldsymbol{v_x^q}} \cdot \vec{\boldsymbol{v_c}})} \\ & \\ & T_x = A_x{}^{\sum_{k \in S_x} \delta_k} \end{aligned}$$

Choose a random $z \in \mathbb{Z}_r$ and for $k = \{1,2,3\}$.

$$\begin{aligned} y < j: \quad & C_{y,k} = g^{z v^p_{c,k}} \cdot B^{w_{y,k}} \\ & \widetilde{C}_{y,k} = g^{-c_y t v^q_{c,k}} \cdot g^{w_{y,k}} \\ y = j: \quad & C_{y,k} = T^{c_y t v^p_{c,k}} \cdot B^{w_{y,k}} \\ & \widetilde{C}_{y,k} = C^{-c_y v^q_{c,k} t} \cdot g^{w_{y,k}} \\ y > j: \quad & C_{y,k} = B^{w_{y,k}} \\ & \widetilde{C}_{y,k} = A^{-c_y v^p_{c,k} t} \cdot g^{-c_y t v^q_{c,k}} \cdot g^{w_{y,k}} \end{aligned}$$

If T corresponds to g^{abc}, then the ciphertext corresponding to (i,j) is well formed; and if T is randomly chosen, then the encryption corresponds to $(i, j+1)$. The reduction will receive the guess γ from $\mathcal{A}$ and it passes on the same value to the Decision 3-party Diffie Hellman challenger. The advantage of the reduction is exactly equal to the advantage of the adversary $\mathcal{A}$.

Case 2: $s' = 1$: In the Setup phase the reduction based on the input (i,j) (the row and column the adversary will attack) sets up the public and the private parameters. The reduction chooses random $r_1, r_2, \ldots r_m, c_1, c_2 \ldots c_m, \alpha_1, \alpha_2 \ldots \alpha_m, \delta_1, \delta_2 \ldots \delta'_j \ldots \delta_m \in \mathbb{Z}_r$. It also chooses $\sigma_{x,y} \in \mathbb{Z}_r$ for every $x, y \in \{1 \ldots m\} \& y \neq j$. It also chooses $\sigma'_{(x,j)} \in \mathbb{Z}_r$ for every

$x \in \{1 \dots m\}$. It sets up the public parameters as:

$$\begin{aligned} & g, E_1 = g^{r_1}, E_2 = g^{r_2}, \dots E_i = B^{r_i} \dots, E_m = g^{r_m}, \\ & G_1 = e(g,g)^{\alpha_1}, G_2 = e(g,g)^{\alpha_2}, \dots, G_m = e(g,g)^{\alpha_m}, \\ & H_1 = g^{c_1}, H_2 = g^{c_2}, \dots H_j = C^{c_j} \dots, H_m = g^{c_m} \\ & u_1 = g^{\delta_1}, u_2 = g^{\delta_2} \dots u_j = C^{\delta'_j} \dots u_m = g^{\delta_m} \end{aligned} \tag{2.5.2}$$

In our system $\delta_j = c \cdot \delta'_j$. Set $\sigma_{(i,j)} = -\frac{br_x c_y}{\delta'_j} + \frac{\sigma'_{(i,j)}}{\delta'_j}$. And the private key $K_{(x,y)}$ of user (x,y) is:

$$\begin{aligned} K_{(x,y)} &= \{g^{\alpha_x} \cdot B^{r_x \cdot c_y} \cdot u_y^{\sigma_{x,y}}, g^{\sigma_{x,y}}\} : x = i, y \neq j \\ K_{(x,y)} &= \{g^{\alpha_x} \cdot C^{r_x \cdot c_y} \cdot u_y^{\sigma_{x,y}}, B^{\sigma_{x,y}}\} : x \neq i, y = j \\ K_{(i,j)} &= \{g^{\alpha_x} \cdot C^{\sigma'_{i,j}}, g^{\sigma_{i,j}}\} : x = i, y = j \\ K_{(i,j)} &= \{g^{\alpha_x} \cdot g^{r_x \cdot c_y} \cdot u_y^{\sigma_{x,y}}, B^{\sigma_{x,y}}\} : x \neq i, y \neq j \end{aligned}$$

The secret key also contain $\forall i, (i \neq y), u_y^{\sigma_{x,y}}$ for each user (x, y). Note that the distribution of the public and private parameters matches the distribution of parameters in the real scheme.

In the challenge phase the adversary sends the message $M \in \mathbb{G}_T$ to the reduction. The reduction then chooses random $t, w_{1,k}, w_{2,k}, \dots w_{m,k}, s_1, s_2 \dots s_m \in \mathbb{Z}_r$ where $k = \{1, 2, 3\}$. It also chooses random $a, b, c \in \mathbb{Z}_r$ and sets $\vec{v_1} = (a, 0, c)$, $\vec{v_2} = (0, b, c)$ and $\vec{v_3} = (-bc, -ac, ab)$.

Set $g^\eta = B$.

It then sets $\vec{v_c} = (v_{c,1}, v_{c,2}, v_{c,3})$ where $v_{c,1}, v_{c,2}, v_{c,3}$ are chosen randomly in $\mathbb{Z}_r$. Let $\vec{v^q}$ denote the projection of $\vec{v}$ along the plane formed by $\vec{v_1}$ and $\vec{v_2}$. And let $\vec{v^p}$ be the component along $\vec{v_3}$.

It also chooses random $z_{1,x,k}, z_{2,x}, z_{3,x} \in \mathbb{Z}_r$ where $1 \leq x < i$ and $k \in \{1, 2, 3\}$ and sets up the ciphertext as follows.

$$\begin{aligned} x < i : \quad & R_{x,k} = g^{z_{1,x,k}} : k = \{1,2,3\} \\ & \widetilde{R}_{x,k} = g^{z_{1,x,k}\eta} : k = \{1,2,3\} \\ & A_x = g^{z_{2,x}} \\ & B_x = e(g,g)^{z_{3,x}} \\ & T_x = {A_x}^{\sum_{k \in S_x} \delta_k} \end{aligned} \tag{2.5.3}$$

It sets $\vec{v_i} = \tilde{q}_i \cdot \vec{v_3} + \tilde{q}'_i \cdot \vec{v_3} + \tilde{p}_i \cdot \vec{v_3}$ where $\tilde{q}_i, \tilde{q}'_i, \tilde{p}_i$ are random in $\mathbb{Z}_r$. The ciphertext for row i can be easily generated because we do not need to use δ_j.

$$\begin{aligned}
x = i: \quad & R_{i,k} = g^{r_i v_{i,k} s_i} : k = \{1,2,3\} \\
& \widetilde{R}_{i,k} = B^{r_i v_{i,k} s_i} : k = \{1,2,3\} \\
& A_i = A^{s_i t(\vec{v_i^p} \cdot \vec{v_c^p})} \cdot g^{s_i t(\vec{v_i^q} \cdot \vec{v_c^q})} \\
& B_i = M \cdot e(g, A_i)^{\alpha_i} \\
& T_i = A_i^{\sum_{k \in S_i} \delta_k}
\end{aligned}$$

For each $x \in \{i+1 \cdots m\}$, it picks $\vec{v_x} = \vec{v_x^q} = \tilde{q}_x \cdot \vec{v_1} + \tilde{q}'_x \cdot \vec{v_2}$ where $\tilde{q}_x, \tilde{q}'_x$ are random in $\mathbb{Z}_r$. The terms with δ_j can be separated and evaluated as all values $\sigma_{(x,y)}$ for $x > i$ are known.

$$\begin{aligned}
x > i: \quad & R_{x,k} = g^{r_x v^q_{x,k} s_x} : k = \{1,2,3\} \\
& \widetilde{R}_{x,k} = B^{r_x v^q_{x,k} s_x} : k = \{1,2,3\} \\
& A_x = B^{s_x t(\vec{v_x^q} \cdot \vec{v_c})} \\
& B_x = M \cdot e(g, B)^{\alpha_x s_x t(\vec{v_x^q} \cdot \vec{v_c})} \\
& T_x = (\prod_{k \in S_x} u_k)^{s_x t(\vec{v_x^q} \cdot \vec{v_c})}
\end{aligned}$$

Choose a random $z \in \mathbb{Z}_r$.

$$\begin{aligned}
y < j: \quad & C_{y,k} = g^{z v^p_{c,k}} \cdot B^{w_{y,k}} : k = \{1,2,3\} \\
& \widetilde{C}_{y,k} = g^{-c_y t v^q_{c,k}} \cdot g^{w_{y,k}} : k = \{1,2,3\} \\
y = j: \quad & C_{y,k} = T^{c_y t v^p_{c,k}} \cdot B^{w_{y,k}} : k = \{1,2,3\} \\
& \widetilde{C}_{y,k} = C^{-c_y v^q_{c,k} t} \cdot g^{w_{y,k}} : k = \{1,2,3\} \\
y > j: \quad & C_{y,k} = B^{w_{y,k}} : k = \{1,2,3\} \\
& \widetilde{C}_{y,k} = A^{-c_y v^p_{c,k} t} \cdot g^{-c_y t v^q_{c,k}} \cdot g^{w_{y,k}} : k = \{1,2,3\}
\end{aligned}$$

If T corresponds to g^{abc}, then the ciphertext corresponding to (i, j) is well formed; and if T is randomly chosen, then the encryption corresponds to $(i, j+1)$. The reduction will receive the guess γ from $\mathcal{A}$ and it passes on the same value to the Decision 3-party Diffie Hellman challenger. The advantage of the reduction is exactly equal to the advantage of the adversary $\mathcal{A}$. ■

Lemma 2.5.3. *If the Decision 3-party Diffie Hellman assumption and the decisional linear assumption hold, then no probabilistic polynomial time adversary can distinguish between an encryption to recipient (i, m) and $(i+1, 1)$ in the index hiding game for any $1 \leq i < m$ with non-negligible probability.*

Proof. The proof of this lemma follows from a series of lemmas that establish the indistinguishability of the following games.

- H_1 Encrypt to column[2] m, row i is the target row,[3] row $i+1$ is the greater-than row.[4]
- H_2 Encrypt to column $m+1$, row i is the target row, row $i+1$ is the greater-than row.
- H_3 Encrypt to column $m+1$, row i is the less-than row, row $i+1$ is the greater-than row (no target row).
- H_4 Encrypt to column 1, row i is the less-than row, row $i+1$ is the greater-than row (no target row).
- H_5 Encrypt to column 1, row i is the less-than row, row $i+1$ is the target row.

It can be observed that game H_1 corresponds to the encryption being done to (i, m) and game H_5 corresponds to encryption to $(i+1, 1)$. The indistinguishability of the games H_1 and H_5, which follows from Lemmas 2.5.4, 2.5.5, 2.5.6, and 2.5.7, implies the lemma. ■

Lemma 2.5.4. *If the Decision 3-party Diffie Hellman assumption holds, then no probabilistic polynomial time adversary can distinguish between games H_1 and H_2 with non-negligible probability.*

Proof. This lemma can be proved by applying the result of Lemma 2.5.2. ■

Lemma 2.5.5. *If the Decision 3-party Diffie Hellman assumption holds, then no probabilistic polynomial time adversary can distinguish between games H_2 and H_3 with non-negligible probability.*

[2]Columns greater than or equal to m are well formed, both in V_p and V_q.
[3]The row for which the row component of the ciphertext has well formed components, both in V_p and V_q.
[4]The first row with the row component of ciphertexts only in V_q.

Proof. The basic intuition behind the proof is to embed the problem in the $\vec{v_c^p}$ part of $\vec{v_c}$. Since all columns have a random component in V_p, we don't need to actually generate this part. Consider an adversary $\mathcal{A}$ that can distinguish between H_2 and H_3 with a probability greater than ε. We build a reduction $\mathcal{R}$ that uses $\mathcal{A}$ to solve the Decision 3-party Diffie Hellman problem. The reduction receives the Decision 3-party Diffie Hellman challenge as:

$$\mathbb{G}, g, A = g^a, B = g^b, C = g^c, T$$

and it is expected to guess if T is g^{abc} or if it is random.

Next, in the setup phase the reduction based on the input i (the row the adversary wants to attack) sets up the public and the private parameters. The reduction chooses random $r_1, r_2, \ldots r_{i-1}, r_{i+1} \ldots r_m,\ c_1, c_2 \ldots c_m,\ \alpha_1, \alpha_2 \ldots \alpha_{i-1}, \alpha_{i+1} \ldots\ \alpha_m, \delta_1, \delta_2 \ldots \delta_m \in \mathbb{Z}_r$. It also chooses $\sigma_{x,y} \in \mathbb{Z}_r$ for every $x, y \in \{1 \ldots m\}$. It sets $g^{\alpha_x} = g^{a \cdot b}$ and $g^{r_x} = B$. It doesn't know g^{ab} but can generate $G_i = e(A, B)$ and $K_{x,y} = g^{ab} g^{((c_y - a)b)} = B^{c_y}$. It sets up the public parameters as:

$$\begin{aligned}
&g, E_1 = g^{r_1} \ldots E_i = B \ldots E_m = g^{r_m}, \\
&G_1 = e(g,g)^{\alpha_1}, \ldots G_i = e(A,B), \ldots, G_m = e(g,g)^{\alpha_m}, \\
&H_1 = g^{c_1} \cdot A^{-1}, H_2 = g^{c_2} \cdot A^{-1} \ldots H_m = g^{c_m} \cdot A^{-1} \\
&u_1 = g^{\delta_1}, u_2 = g^{\delta_2} \ldots u_m = g^{\delta_m}
\end{aligned} \tag{2.5.4}$$

And the private key $K_{(x,y)}$ of user (x,y) is:

$$\begin{aligned}
&K_{(x,y)} = \{g^{\alpha_x} \cdot (g^{c_y} \cdot A^{-1})^{r_x} \cdot u_y^{\sigma_{x,y}}, g^{\sigma_{x,y}}, \forall i, (i \neq y), u_y^{\sigma_{x,y}}\} \\
&: x \neq i \\
&K_{(x,y)} = \{B^{c_y} \cdot u_y^{\sigma_{x,y}}, g^{\sigma_{x,y}}, \forall i, (i \neq y), u_y^{\sigma_{x,y}}\} : x = i
\end{aligned}$$

Note that the distribution of the public and private parameters matches the distribution of parameters in the real scheme.

In the challenge phase the adversary sends the message $M \in \mathbb{G}_T$ to the reduction. The reduction then chooses random t, η, $w_{1,k}$, $w_{2,k}$, $\ldots w_{m,k}$, s_1, s_2 $\ldots s_m \in \mathbb{Z}_q$ where $k = \{1, 2, 3\}$. It also chooses random $a, b, c \in \mathbb{Z}_r$ and sets $\vec{v_1} = (a, 0, c)$, $\vec{v_2} = (0, b, c)$

and $\vec{\boldsymbol{v_3}} = (-bc, -ac, ab)$. It then sets $\vec{\boldsymbol{u_c}} = (u_{c,1}, u_{c,2}, u_{c,3})$ where $u_{c,1}, u_{c,2}, u_{c,3}$ are chosen randomly in $\mathbb{Z}_r$. Let $\vec{\boldsymbol{u^q}}$ denote the projection of $\vec{\boldsymbol{u}}$ along the plane formed by $\vec{\boldsymbol{v_1}}$ and $\vec{\boldsymbol{v_2}}$ and $\vec{\boldsymbol{u^p}}$ denote the projection of $\vec{\boldsymbol{u}}$ along $\vec{\boldsymbol{v_3}}$. Let $g^{v^p_{c,k}} = C^{u^p_{c,k}}$. Note that by using this value of $v^p_{c,k}$, we will not be able to generate a column ciphertext that has the right component in V_p; but since all columns are random in V_p, we do not need to generate this term. Let $g^{{v'}^p_{c,k}} = g^{z \cdot u^p_{c,k}}$, where z is random in $\mathbb{Z}_r$. It also sets , $\vec{\boldsymbol{v_i}} = \tilde{q}_i \cdot \vec{\boldsymbol{v_1}} + \tilde{q}'_i \cdot \vec{\boldsymbol{v_2}} + \tilde{p}_i \cdot \vec{\boldsymbol{v_3}}$ where $\tilde{q}_i, \tilde{q}'_i, \tilde{p}_i$ are random in $\mathbb{Z}_r$. It also chooses random $z_{1,x,k}, z_{2,x}, z_{3,x} \in \mathbb{Z}_q$ where $1 \leq x < i$ and $k \in \{1, 2, 3\}$.

Then it creates the ciphertext as:

$$
\begin{aligned}
x < i: \quad & R_{x,k} = g^{z_{1,x,k}} : k = \{1,2,3\} \\
& \widetilde{R}_{x,k} = g^{z_{1,x,k}\eta} : k = \{1,2,3\} \\
& A_x = g^{z_{2,x}} \\
& B_x = e(g,g)^{3_{3,x}} \\
& T_x = {A_x}^{\sum_{k \in S_x} \delta_k} \\
x = i: \quad & R_{i,k} = B^{v_{i,k} s_i} : k = \{1,2,3\} \\
& \widetilde{R}_{i,k} = B^{v_{i,k} s_i \eta} : k = \{1,2,3\} \\
& A_i = g^{s_i t (\vec{\boldsymbol{v_i^q}} \cdot \vec{\boldsymbol{v_c^q}})} \cdot C^{s_i t (\vec{\boldsymbol{v_i^p}} \cdot \vec{\boldsymbol{u_c^p}})} \\
& B_i = M \cdot e(A,B)^{s_i t (\vec{\boldsymbol{v_i^q}} \cdot \vec{\boldsymbol{v_c^q}})} e(g,T)^{t s_i (\vec{\boldsymbol{v_i^p}} \cdot \vec{\boldsymbol{u_c^p}})} \\
& T_i = {A_i}^{\sum_{k \in S_i} \delta_k}
\end{aligned}
$$

For each $x \in \{i+1 \cdots m\}$, it picks $\vec{\boldsymbol{v_x}} = \vec{\boldsymbol{v_x^q}} = \tilde{q}_x \cdot \vec{\boldsymbol{v_1}} + \tilde{q}'_x \cdot \vec{\boldsymbol{v_2}}$ where $\tilde{q}_x, \tilde{q}'_x$ are random in $\mathbb{Z}_r$.

$$
\begin{aligned}
x > i: \quad & R_{x,k} = g^{r_x v^q_{x,k} s_x} : k = \{1,2,3\} \\
& \widetilde{R}_{x,k} = g^{r_x v^q_{x,k} s_x \eta} : k = \{1,2,3\} \\
& A_x = g^{s_x t (\vec{\boldsymbol{v_x^q}} \cdot \vec{\boldsymbol{v_c^q}})} \\
& B_x = M \cdot e(g,g)^{\alpha_x s_x t (\vec{\boldsymbol{v_x^q}} \cdot \vec{\boldsymbol{v_c^q}})} \\
& T_x = {A_x}^{\sum_{k \in S_x} \delta_k}
\end{aligned}
$$

$$C_{y,k} = \left(g^{c_y} \cdot A^{-1}\right)^{t(v^q_{c,k} + v'^p_{c,k})} \cdot g^{w_{y,k}\eta} : k = \{1, 2, 3\}$$
$$\widetilde{C}_{y,k} = g^{w_{y,k}} : k = \{1, 2, 3\}$$

If T corresponds to g^{abc}, then the ciphertext corresponding to row i corresponds to the target row; and if T is randomly chosen, then the encryption corresponds to game H_3. The reduction will receive the guess γ from $\mathcal{A}$, and it passes on the same value to the Decision 3-party Diffie Hellman challenger. The advantage of the reduction is exactly equal to the advantage of the adversary $\mathcal{A}$. ■

Lemma 2.5.6. *If the Decision 3-party Diffie Hellman assumption holds, then no probabilistic polynomial time adversary can distinguish between games H_3 and H_4 with non-negligible probability.*

Proof. H_3 to H_4 can be expressed as a series of games $H_{3,m+1}$, $H_{3,m}$ $\cdots$ $H_{3,1}$. In the game $H_{3,j}$, all column ciphertexts $(C_y, \widetilde{C}_y)$ are well formed for all y such that $j \leq y \leq m$. It can be seen that $H_{3,1}$ is the same as H_4, and $H_{3,m}$ is the same as H_3. We prove the indistinguishability of games $H_{3,j}$ and $H_{3,j+1}$ for all j where $1 \leq j \leq m$. The proof for this is similar to that of Lemma 2.5.2. Consider an adversary $\mathcal{A}$ that solves the index hiding game with a probability greater than ε. The adversary is considered successful if it can distinguish between games $H_{3,j}$ and $H_{3,j+1}$. We build a reduction $\mathcal{R}$ that uses $\mathcal{A}$ to solve the Decision 3-party Diffie Hellman problem. The reduction receives the Decision 3-party Diffie Hellman challenge as:

$$\mathbb{G}, g, A = g^a, B = g^b, C = g^c, T$$

and it is expected to guess if T is g^{abc} or if it is random.

Next, in the Setup phase the reduction based on the input (i, j) (the row and column the adversary will attack) sets up the public and the private parameters. The reduction chooses random $r_1, r_2, \ldots r_m$, $c_1, c_2 \ldots c_m$, $\alpha_1, \alpha_2 \ldots \alpha_m \delta_1, \delta_2 \ldots \delta_m \in \mathbb{Z}_r$. It also chooses $\sigma_{x,y} \in \mathbb{Z}_r$ for every $x, y \in \{1 \ldots m\}$. It sets up the public parameters as:

$$\begin{aligned}
&g, E_1 = g^{r_1}, E_2 = g^{r_2}, \ldots E_m = g^{r_m}, \\
&G_1 = e(g,g)^{\alpha_1}, G_2 = e(g,g)^{\alpha_2}, \ldots, G_m = e(g,g)^{\alpha_m}, \\
&H_1 = g^{c_1}, H_2 = g^{c_2}, \ldots H_j = C^{c_j} \ldots, H_m = g^{c_m} \\
&u_1 = g^{\delta_1}, u_2 = g^{\delta_2} \ldots u_m = g^{\delta_m}
\end{aligned} \tag{2.5.5}$$

And the private key $K_{(x,y)}$ of user (x,y) is:

$$K_{(x,y)} = \{g^{\alpha_x} \cdot g^{r_x \cdot c_y} \cdot u_y^{\sigma_{x,y}}, g^{\sigma_{x,y}}, \forall i, (i \neq y), u_y^{\sigma_{x,y}}\} : y \neq j$$
$$K_{(x,y)} = \{g^{\alpha_x} \cdot C^{r_x \cdot c_y} \cdot u_y^{\sigma_{x,y}}, g^{\sigma_{x,y}}, \forall i, (i \neq y), u_y^{\sigma_{x,y}}\} : y = j$$

Note that the distribution of the public and private parameters matches the distribution of parameters in the real scheme.

In the challenge phase the adversary sends the message $M \in \mathbb{G}_T$ to the reduction. The reduction then chooses random $t, w_{1,k}, w_{2,k}, \ldots w_{m,k}, s_1, s_2 \ldots s_m \in \mathbb{Z}_r$ where $k = \{1, 2, 3\}$. It also chooses random $a, b, c \in \mathbb{Z}_r$ and sets $\vec{\boldsymbol{v_1}} = (a, 0, c)$, $\vec{\boldsymbol{v_2}} = (0, b, c)$ and $\vec{\boldsymbol{v_3}} = (-bc, -ac, ab)$.

Set $g^\eta = B$.

It then sets $\vec{\boldsymbol{v_c}} = (v_{c,1}, v_{c,2}, v_{c,3})$ where $v_{c,1}, v_{c,2}, v_{c,3}$ are chosen randomly in $\mathbb{Z}_r$. Let $\vec{\boldsymbol{v}}^{\boldsymbol{q}}$ denote the projection of $\vec{\boldsymbol{v}}$ along the plane formed by $\vec{\boldsymbol{v_1}}$ and $\vec{\boldsymbol{v_2}}$. And $\vec{\boldsymbol{v}}^{\boldsymbol{p}}$ be the component along $\vec{\boldsymbol{v_3}}$.

It chooses random $z_{1,x,k}, z_{2,x}, z_{3,x} \in \mathbb{Z}_r$ where $1 \leq x < i$ and $k \in \{1, 2, 3\}$ and sets up the ciphertext as follows.

$$\begin{aligned}
x \leq i: \quad & R_{x,k} = g^{z_{1,x,k}} : k = \{1, 2, 3\} \\
& \widetilde{R}_{x,k} = g^{z_{1,x,k}\eta} : k = \{1, 2, 3\} \\
& A_x = g^{z_{2,x}} \\
& B_x = e(g,g)^{z_{3,x}} \\
& T_x = {A_x}^{\sum_{k \in S_x} \delta_k}
\end{aligned} \tag{2.5.6}$$

For each $x \in \{i+1 \cdots m\}$, it picks $\vec{\boldsymbol{v_x}} = \vec{\boldsymbol{v^q_x}} = \tilde{q}_x \cdot \vec{\boldsymbol{v_1}} + \tilde{q}'_x \cdot \vec{\boldsymbol{v_2}}$ where $\tilde{q}_x, \tilde{q}'_x$ are random in $\mathbb{Z}_r$.

$$\begin{aligned} x > i: \quad & R_{x,k} = g^{r_x v^q_{x,k} s_x} : k = \{1,2,3\} \\ & \widetilde{R}_{x,k} = B^{r_x v^q_{x,k} s_x} : k = \{1,2,3\} \\ & A_x = B^{s_x t(\vec{\boldsymbol{v^q_x}} \cdot \vec{\boldsymbol{v_c}})} \\ & B_x = M \cdot e(g,B)^{\alpha_x s_x t(\vec{\boldsymbol{v^q_x}} \cdot \vec{\boldsymbol{v_c}})} \\ & T_x = {A_x}^{\sum_{k \in S_x} \delta_k} \end{aligned}$$

Choose a random $z \in \mathbb{Z}_r$.

$$\begin{aligned} y < j: \quad & C_{y,k} = g^{z v^p_{c,k}} \cdot B^{w_{y,k}} : k = \{1,2,3\} \\ & \widetilde{C}_{y,k} = g^{-c_y t v^q_{c,k}} \cdot g^{w_{y,k}} : k = \{1,2,3\} \\ y = j: \quad & C_{y,k} = T^{c_y t v^p_{c,k}} \cdot B^{w_{y,k}} : k = \{1,2,3\} \\ & \widetilde{C}_{y,k} = C^{-c_y v^q_{c,k} t} \cdot g^{w_{y,k}} : k = \{1,2,3\} \\ y > j: \quad & C_{y,k} = B^{w_{y,k}} : k = \{1,2,3\} \\ & \widetilde{C}_{y,k} = A^{-c_y v^p_{c,k} t} \cdot g^{-c_y t v^q_{c,k}} \cdot g^{w_{y,k}} : k = \{1,2,3\} \end{aligned}$$

If T corresponds to g^{abc}, then we are in game $H_{3,j}$; and if T is randomly chosen, then the encryption corresponds to the game $H_{3,j+1}$. The reduction will receive the guess γ from $\mathcal{A}$, and it passes on the same value to the Decision 3-party Diffie Hellman challenger. The advantage of the reduction is exactly equal to the advantage of the adversary $\mathcal{A}$. ■

Lemma 2.5.7. *If the decisional linear assumption holds, then no probabilistic polynomial time adversary can distinguish between games H_4 and H_5 with non-negligible probability.*

Proof. Consider an adversary $\mathcal{A}$ that can distinguish between games H_4 and H_5 with a probability greater than ε. We build a reduction $\mathcal{R}$ that uses $\mathcal{A}$ to solve the decisional linear problem. The reduction receives the decisional linear challenge as:

$$\mathbb{G}, g, g^a, g^b, g^c, g^{ax}, g^{by}, T$$

and it is expected to guess if T is $g^{c(x+y)}$ or if it is random.

Next, in the Setup phase the reduction based on the input i (the row the adversary will attack) sets up the public and the private parameters. The reduction chooses random $r_1, r_2, \ldots r_m, c_1, c_2 \ldots c_m, \alpha_1, \alpha_2 \ldots \alpha_m \delta_1, \delta_2 \ldots \delta_m \in \mathbb{Z}_r$. It also chooses $\sigma_{x,y} \in \mathbb{Z}_r$ for every $x, y \in \{1 \ldots m\}$. It sets up the public parameters as:

$$\begin{aligned} & g, E_1 = g^{r_1}, E_2 = g^{r_2}, \ldots E_m = g^{r_m}, \\ & G_1 = e(g,g)^{\alpha_1}, G_2 = e(g,g)^{\alpha_2}, \ldots G_m = e(g,g)^{\alpha_m}, \\ & H_1 = g^{c_1}, H_2 = g^{c_2}, \ldots H_m = g^{c_m} \\ & u_1 = g^{\delta_1}, u_2 = g^{\delta_2} \ldots u_m = g^{\delta_m} \end{aligned} \tag{2.5.7}$$

And the private key $K_{(x,y)}$ of user (x,y) is:

$$K_{(x,y)} = \{ g^{\alpha_x} \cdot g^{r_x \cdot c_y} \cdot u_y^{\sigma_{x,y}}, g_2^{\sigma_{x,y}}, \forall i, (i \neq y), u_y^{\sigma_{x,y}} \} : \forall x, y$$

Note that the distribution of the public and private parameters matches the distribution of parameters in the real scheme.

It sets $g^{v_{1,1}} = g^a$, $g^{v_{1,2}} = g^0$, $g^{v_{1,3}} = g^c$, $g^{v_{2,1}} = g^0$, $g^{v_{2,2}} = g^b$ and $g^{v_{2,3}} = g^c$. A valid decisional linear tuple will lie in the subspace formed by vectors $\vec{\boldsymbol{v_1}}$ and $\vec{\boldsymbol{v_2}}$. A decisional linear problem tuple will be used for setting row ciphertext for row $i+1$. A valid tuple leads to encryption as in game H_4, and a random tuple will cause the encryption to be as in game H_5.

In the challenge phase the adversary sends the message $M \in \mathbb{G}_T$ to the reduction. The reduction then chooses random $t, \eta, w_{1,k}, w_{2,k}, \ldots w_{m,k}, s_1, s_2 \ldots s_m \in \mathbb{Z}_r$ where $k = \{1, 2, 3\}$. It then sets $\vec{\boldsymbol{v_c}} = (v_{c,1}, v_{c,2}, v_{c,3})$ where $v_{c,1}, v_{c,2}, v_{c,3}$ are chosen randomly in $\mathbb{Z}_r$.

$$g^{(\vec{\boldsymbol{v_x}} \cdot \vec{\boldsymbol{v_c}})} = \prod_{k=1}^{3} [g^{v_{x,k}}]^{v_{c,k}}$$

It also chooses random $z_{1,x,k}, z_{2,x}, z_{3,x} \in \mathbb{Z}_r$ where $1 \leq x \leq i$ and $k \in \{1, 2, 3\}$. Then it creates the ciphertext as follows.

$$\begin{aligned} x \le i: \quad & R_{x,k} = g^{z_{q,x,k}} : k = \{1,2,3\} \\ & \widetilde{R}_{x,k} = g^{z_{1,x,k}\eta} : k = \{1,2,3\} \\ & A_x = g^{z_{2,x}} \\ & B_x = e(g,g)^{z_{3,x}} \\ & T_x = {A_x}^{\sum_{k \in S_x} \delta_k} \end{aligned}$$

It sets $g^{v_{i+1,1}} = g^{ax}$, $g^{v_{i+1,2}} = g^{by}$ and $g^{v_{i+1,3}} = T$. For each $x \in \{i+2 \cdots m\}$, it picks $g^{v_{x,1}} = g^{a\tilde{q}_x}$, $g^{v_{x,2}} = g^{b\tilde{q}'_x}$ and $g^{v_{x,3}} = g^{c(\tilde{q}_x + \tilde{q}'_x)}$ where $\tilde{q}_x, \tilde{q}'_x$ are random in $\mathbb{Z}_r$.

$$\begin{aligned} x > i: \quad & R_{x,k} = g^{r_x v_{x,k} s_x} : k = \{1,2,3\} \\ & \widetilde{R}_{x,k} = g^{r_x v_{x,k} s_x \eta} : k = \{1,2,3\} \\ & A_x = g^{s_x t(\vec{v_x} \cdot \vec{v_c})} \\ & B_x = M \cdot e(g,g)^{\alpha_x s_x t(\vec{v_x} \cdot \vec{v_c})} \\ & T_x = {A_x}^{\sum_{k \in S_x} \delta_k} \end{aligned}$$

$$C_{y,k} = g^{c_y t v_{c,k}} \cdot g^{w_{y,k}\eta} \quad \widetilde{C}_{y,k} = g^{w_{y,k}} : k = \{1,2,3\}$$

If T corresponds to $g^{c(x+y)}$, then the ciphertext corresponds to game H_4; and if T is randomly chosen, then it corresponds to game H_5. The reduction will receive the guess γ from $\mathcal{A}$, and it passes on the same value to the decisional linear challenger. The advantage of the reduction is exactly equal to the advantage of the adversary $\mathcal{A}$. ■

2.5.2 Message Hiding

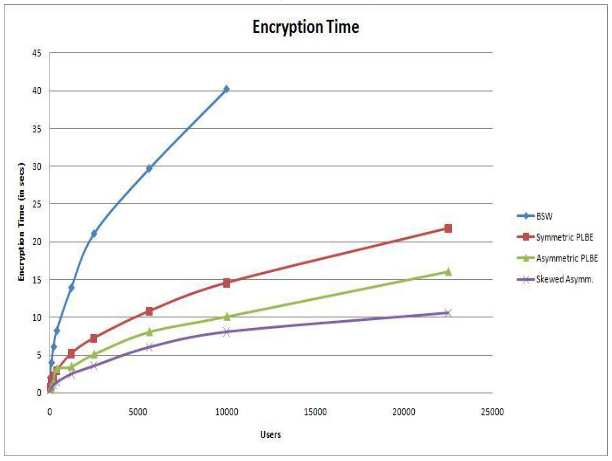
Theorem 2.5.8. *No adversary can distinguish between two ciphertexts when the encryption is done to the* $(m+1,1)$.

Proof. This means that all rows will be completely random and independent of the message. Hence, information theoretically the adversary has no way of identifying which message has been encrypted. ■

2.6 Implementation

We provide the first implementation of fully collusion resistant traitor tracing and trace & revoke schemes. We use only *prime* order bilinear groups in this implementation. We implement all of our schemes using the Pairing Based Crypto (PBC) library [Lyn]. For schemes that use asymmetric bilinear groups, we generate them using MNT curves [MNT00]. The group size is 170 bits long, the group representations are 512 bits long, and the security is equivalent to 1024 bits of discrete log. It is also believed that the XDH assumption holds on these curves (Section 8.1 [BLS01]). For symmetric bilinear groups, we use super singular curves (with fastest pairing times but bad group element size). We use 512 bit group representations and have 1024 bits of discrete log security. One can choose other alternative symmetric groups that have smaller group size with faster exponentiation but slower pairing operations. This kind of tradeoff was not possible in previous systems [BSW06, BW06].

Figure 2.2: Encryption Time (in secs) of traitor tracing schemes



We contrast our schemes' efficiency with an implementation of [BSW06]. [BSW06] only provides traitor tracing functionality. We compare our traitor tracing scheme with [BSW06] in Tables 2.1,2.2 and Figure 2.4. We also provide additional data on our trace & revoke implementation (Table 2.3)). Currently, the only known way to generate composite order groups is by using symmetric bilinear groups. Also, their subgroup decision assumption mandates that the order of the composite group be at least 1024 bits (to avoid sub-exponential factoring based attacks). We compare the encryption time, decryption time and ciphertext

Table 2.1: Encryption Time of traitor tracing schemes

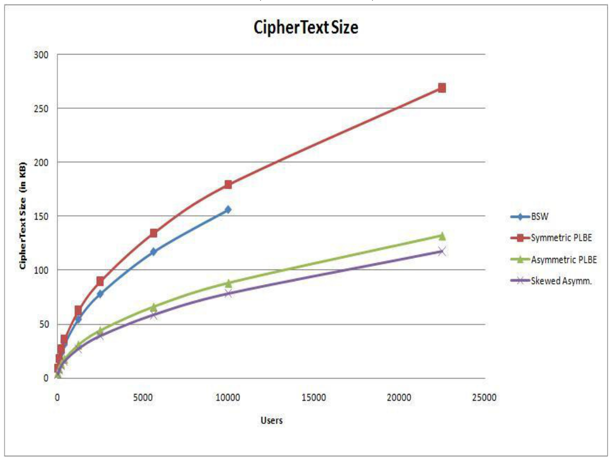
Users	Boneh et al.	Symmetric PLBE	Asymmetric PLBE	Skewed Asymm.
25	1.977s	0.749s	0.494s	0.3611s
100	3.971s	1.503s	1s	0.694s
225	6.069s	2.183s	1.512s	1.081s
400	8.2s	2.922s	3.187s	1.424s
1225	13.898s	5.1885s	3.495s	2.523s
2500	21.046s	7.227s	5.104s	3.583s
5625	29.681s	10.797s	8.069s	6.056s
10000	40.189s	14.552s	10.099s	8.074s
22500	-	21.769s	16.028s	10.577s

sizes as the number of users grow for all these schemes.

A real implementation of broadcast encryption will use a symmetric key cipher under some key K [BSW06]. But this key K still needs to be distributed and one can use our schemes for key distribution. By converting our encryption system to a Key Encapsulation Mechanism we can save on computation. Under this optimization, we do not need to evaluate B_x or include it in the ciphertext. A user (x, y) can extract the key $K_x = e(K_{(x,y)}, A_x)\frac{\prod_{i=1}^{3} e(\widetilde{R}_x, \widetilde{C}_y)}{\prod_{i=1}^{3} e(R_x, C_y)}$. The ciphertext would now have to contain an encryption of K under each of the K_x. The user can then derive K from an encryption of it under K_x.

In Table 2.1 and 2.2 we provide a comparison of our PLBE scheme for the case of symmetric and asymmetric prime order groups with that of [BSW06] (which uses composite order groups). The implementation was done on an Intel i3 2.9GHz quad core desktop PC with 2GB RAM. The groups were chosen to guarantee 1024 bits of discrete log security for encryption time and ciphertext size.

Figure 2.3: Ciphertext Size (in bytes) of traitor tracing schemes



2.6.1 Encryption Time

The encryption time (Table 2.1, Figure 2.2) is heavily dependent on a large number of exponentiation operations, one for each row of ciphertext. It depends on the number of users as $O(\sqrt{N})$, explaining the parabolic nature of the graph(s). The cost of exponentiation operations in elliptic curves depend both on group representation size and the actual order of the group. The order of symmetric groups that we have chosen for this implementation are constructed to be of the form $2^a \pm 2^b \pm 1$, for some integers a, b. This makes exponentiation in them very efficient. The asymmetric order groups are efficient because of their smaller group size. The composite order groups perform significantly worse by a factor of 6.

Figure 2.4: Decryption time of traitor tracing schemes

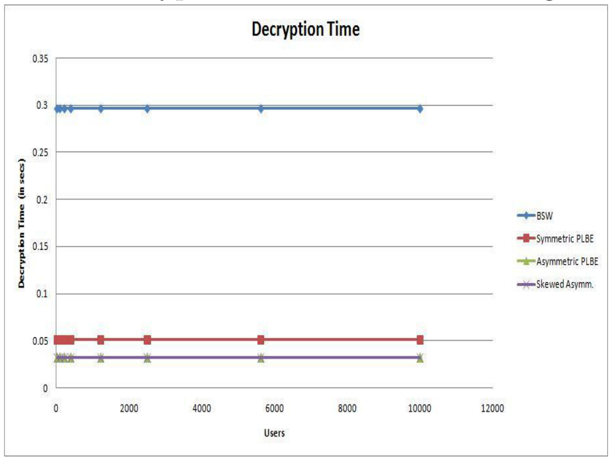


Table 2.2: CipherText Size (in bytes) of traitor tracing schemes

Users	Boneh et.al	Symmetric PLBE	Asymmetric PLBE	Skewed Asymm.
25	7800	8960	4400	3840
100	15600	17920	8800	7680
225	23400	26880	13200	11840
400	31200	35840	17600	15680
1225	54600	62720	30800	27360
2500	78000	89600	44000	39200
5625	117000	134400	66000	58720
10000	156000	179200	88000	78400
22500	-	268800	132000	117440

2.6.2 Ciphertext Size

The ciphertext size (Table 2.2, Figure 2.3) is dependent on the representation size of the elliptic curve and the number of group elements used. Our construction, although using a larger number of group elements, has smaller total ciphertext size (in the asymmetric case) because the group sizes are significantly smaller. The asymmetric groups, by their nature allows us to optimize ciphertext size by increasing the number of rows and decreasing the columns in (Fig. 2.1). We call this the Skewed Asymmetric group version. Note that by design, most of the group elements in the ciphertext are placed in the smaller group $\mathbb{G}_1$. Skewing has no effect on security proofs and allows us to optimize on ciphertext size.

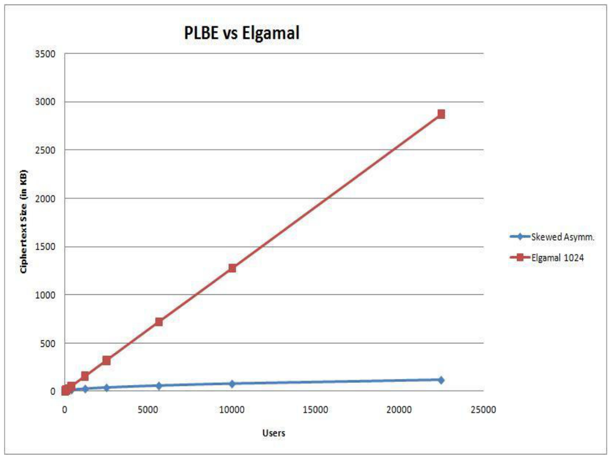
Calculations show that using $25 \times 16 (= 400)$ rectangle for generating ciphertexts produces only 15680 bytes which gives us a 50% improvement compared to the scheme of [BSW06]

We provide and implement efficient broadcast, trace & revoke system. Table 2.3 provides encryption times and ciphertext size (in bytes) for up to 5625 users. The security guaranteed on the elliptic curves used are 1024 bit discrete log security.

Table 2.3: Encryption Time and CipherText size (in bytes)

Users	Symm. Enc. Time	Asymm. Enc. Time	Symm. Ciphertext Size	Asymm. Ciphertext Size
25	0.611s	0.540s	9600B	4600B
100	1.179s	1.027s	19200B	9200B
225	1.695s	1.550s	28800B	13800B
400	2.213s	2.059s	38400B	18400B
1225	3.765s	3.594s	67200B	32200B
2500	5.272s	5.248s	96000B	46000B
5625	8.104s	7.759s	144000B	69000B

Figure 2.5: Ciphertext Size



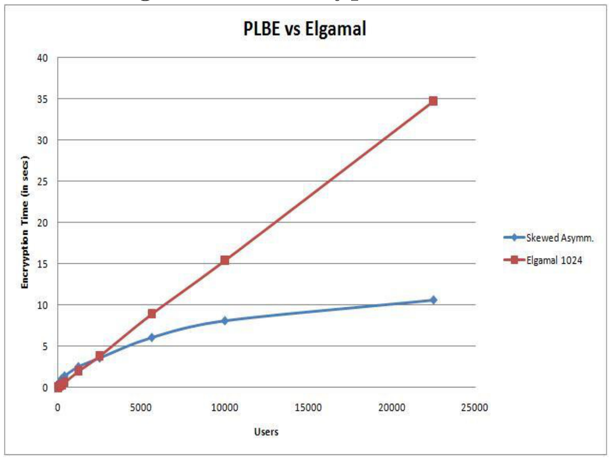
2.6.3 Decryption Time

The decryption time for the various scenarios (Figure 2.4) above are relatively constant and independent of the number of users for each scheme. This is because decryption time is dominated by the cost of pairing operations on the elliptic curves. The composite order schemes decrypt at $0.296s$ per ciphertext and the primer order symmetric and asymmetric groups decrypt at $0.051s$ and $0.032s$ respectively. Thus we see the prime order groups are relative similar w.r.t decryption times and are *10 times* faster due to faster pairing operations in these groups.

2.6.4 Comparison with the ElGamal Encryption

We compare the efficiency of our scheme with an implementation of a naïve (but optimized) ElGamal based traitor tracing scheme. The advantage of using an ElGamal based scheme is that the group that it works on could support very efficient arithmentic operations (we choose the multiplicative group Z_p^* for a 1024 bit prime p) making encryption very fast. The disadvantage is that for N users ElGamal based systems use $O(N)$ steps whereas our scheme uses $O(\sqrt{N})$ steps. We observe that the ElGamal implementation has a huge cipher-text size overload compared to our scheme (Figure 2.5). We also observe that asymptotic improvements in the encryption time begin to show up for as few as 2500 users (Figure 2.6).

Figure 2.6: Encryption Time



2.7 Conclusion

Boneh et al. [BSW06,BW06] provide traitor tracing and trace & revoke systems using composite order bilinear groups. These groups have large exponentiation and pairing times making them impractical. We provide the first implementation of a traitor tracing and trace & revoke systems, using symmetric and asymmetric prime order bilinear groups. Our implementation and comparisons with [BSW06] show that we achieve about 10 times faster decryption, 6 times faster encryption and 50% reduction in ciphertext size. The ideas presented in this work are general and can be applied to convert other composite order cryptosystems to efficient prime order based cryptosystems.

Part 2

CHAPTER 3

Cryptography using Captcha

3.1 Preliminaries

In this work, to model "access to a human", we will provide some parties (modeled as interactive Turing machines–ITM) *oracle* access to a function H. An ITM M with oracle access to H is an ordinary ITM except that it has two special tapes: a write-only *query tape* and a read-only *answer tape*. When M writes a string q on its query tape, the value $H(q)$ is written on its answer tape. If q is not a valid query (i.e., not in the domain of H), a special symbol $\perp$ is written on the output tape. Such a query and answer step is counted as one step in the running time of M. We use the notation M^H to mean that M has oracle access to H. The reader is referred to [Gol01, AB09] for a detailed treatment of this notion.

Notation. The output of an oracle ITM M^H is denoted by a triplet $(\texttt{out}, \bar{q}, \bar{a})$ where $\texttt{out}, \bar{q}$, and $\bar{a}$ denote the contents of M's output tape, a vector of strings written to the query tape in the current execution, and the answer to the queries present in $\bar{q}$ respectively.

Let $k \in \mathbb{N}$ denote the security parameter, where $\mathbb{N}$ is the set of natural numbers. All parties are assumed to receive 1^k as an implicit input (even if not mentioned explicitly). When we say that an (I)TM M (perhaps with access to an oracle H) runs in polynomial time, we mean that there exists a polynomial $T(\cdot)$ such that for every input, the total number of steps taken by M are at most $T(k)$. For two strings a and b, their concatenation is denoted by $a \circ b$. The statistical distance between two distributions X, Y is denoted $\Delta(X, Y)$.

In all places, we only use standard notations (with their usual meaning) for describing algorithms, random variables, experiments, protocol transcripts and so on. We assume familiarity with standard concepts such as computational indistinguishability, negligible functions, and so on (see [Gol01]).

Statistically Secure Oblivious Transfer We now recall the notion of a statistically secure, two message oblivious transfer (OT) protocol, as defined by Halevi and Kalai [HK10].

Definition 3.1.1. ***(Statistically Secure Oblivious Transfer), [HK10]*** *Let $\ell(\cdot)$ be a polynomial and $k \in \mathbb{N}$ the security parameter. A two-message, two-party protocol $\langle S_{\mathrm{OT}}, R_{\mathrm{OT}} \rangle$ is said to be a* statistically secure oblivious transfer *protocol for bit-strings of length $\ell(k)$ such that both the sender S_{OT} and the receiver R_{OT} are* PPT *ITMs receiving 1^k as common input; in addition, S_{OT} gets as input two strings $(m_0, m_1) \in \{0,1\}^{\ell(k)} \times \{0,1\}^{\ell(k)}$ and R_{OT} gets as input a choice bit $b \in \{0,1\}$. We require that the following conditions are satisfied:*

- Functionality: *If the sender and the receiver follow the protocol then for every $k \in \mathbb{N}$, every $(m_0, m_1) \in \{0,1\}^{\ell(k)} \times \{0,1\}^{\ell(k)}$, and every $b \in \{0,1\}$, the receiver outputs m_b.*

- Receiver security: *The ensembles $\{R_{\mathrm{OT}}(1^k, 0)\}_{k \in \mathbb{N}}$ and $\{R_{\mathrm{OT}}(1^k, 1)\}_{k \in \mathbb{N}}$ are computationally indistinguishable, where $\{R_{\mathrm{OT}}(1^k, b)\}_{k \in \mathbb{N}}$ denotes the (first and only) message sent by R_{OT} on input $(1^k, b)$. That is,*

$$\{R_{\mathrm{OT}}(1^k, 0)\}_{k \in \mathbb{N}} \overset{c}{\equiv} \{R_{\mathrm{OT}}(1^k, 1)\}_{k \in \mathbb{N}}$$

- Sender security: *There exists a negligible function* $\mathrm{negl}(\cdot)$ *such that for every $(m_0, m_1) \in \{0,1\}^{\ell(k)} \times \{0,1\}^{\ell(k)}$, every first message $\alpha \in \{0,1\}^*$ (from an arbitrary and possibly unbounded malicious receiver), and every sufficiently large $k \in \mathbb{N}$, it holds that either*

$$\Delta_0(k) := \Delta(S_{\mathrm{OT}}(1^k, m_0, m_1, \alpha), S_{\mathrm{OT}}(1^k, m_0, 0^{\ell(k)}, \alpha)) \text{ or,}$$

$$\Delta_1(k) := \Delta(S_{\mathrm{OT}}(1^k, m_0, m_1, \alpha), S_{\mathrm{OT}}(1^k, 0^{\ell(k)}, m_1, \alpha))$$

is negligible, where $S_{\mathrm{OT}}(1^k, m_0, m_1, \alpha)$ denotes the (only) response of the honest sender S_{OT} with input $(1^k, m_0, m_1)$ when the receiver's first message is α.

Statistically secure OT can be constructed from a vareity of cryptographic assumptions. In [HK10], Halevi and Kalai construct protocols satisfying the above definition under the assumption that *verifiable smooth projective hash families with hard subset membership problem* exist (which in turn, can be constructed from a variety of standard assumptions such as the quadratic-residue problem). [HO09] show the equivalence of 2-message statistically secure oblivious transfer and lossy encryption.

3.2 Modeling Captcha Puzzles

As said earlier, CAPTCHA puzzles are problem instances that are easy for "humans" but hard for computers to solve. Let us first consider the "hardness" of such puzzles for computers. To model "hardness," one approach is to consider an asymptotic formulation. That is, we envision a randomized generation algorithm G which on input a security parameter 1^k, outputs a puzzle from a (discrete and finite) set $\mathcal{P}_k$ called the *puzzle-space*. Indeed, this is the formulation that previous works [ABH03, Dzi10, CHS06] as well as our work here follow. assume that there is a fixed polynomial $\ell(\cdot)$ such that every puzzle instance $z \in \mathcal{P}_k$ is a bit string of length at most $\ell(k)$.

Of course, not all CAPTCHA puzzle systems satisfy such an asymptotic formulation. It is possible to have a (natural) non-asymptotic formulation to define CAPTCHA puzzles which takes into consideration this issue and defines hardness in terms of a "human population" [ABH03]. However, a non-asymptotic formulation will be insufficient for cryptographic purposes. For many puzzles, typically hardness can be amplified by sequential or parallel repetition [CHS04].

Usually, CAPTCHA puzzles have a unique and well defined solution associated with every puzzle instance. We capture this by introducing a discrete and finite set $\mathcal{S}_k$, called the *solution-space*, and a corresponding *solution function* $H_k : \mathcal{P}_k \rightarrow \mathcal{S}_k$ which maps a puzzle instance $z \in \mathcal{P}_k$ to its corresponding solution. Without loss of generality we assume that every element of $\mathcal{S}_k$ is a bit string of length k. We will require that G generates puzzles together with their solutions. This restriction is also required in previous works [ABH03, Dzi10]. To facilitate the idea that the puzzle-generation is a completely *automated* process, G will not be given "access to a human."

With this formulation, we can view "humans" as computational devices which can "efficiently" compute the solution function H_k. Therefore, to capture "access to a human", the algorithms can simply be provided with *oracle* access to the family of solution functions $H := \{H_k\}_{k \in \mathbb{N}}$. Recall that by definition, oracle-access to H means that algorithms can only provide an input z to some function $H_{k'}$ in the family H, and then read its output $H_{k'}(z)$; if z is not in the domain $\mathcal{P}_{k'}$, the response to the query is set to a special symbol, denoted $\perp$. Every query to $H_{k'}$ will be assumed to contribute one step to the running time of the querying algorithm. The discussion so far leads to the following definition for Captcha puzzles.

Definition 3.2.1. ***(*Captcha *Puzzles)*** *Let $\ell(\cdot)$ be a polynomial, and $\mathcal{S} := \{\mathcal{S}_k\}_{k \in \mathbb{N}}$ and $\mathcal{P} := \{\mathcal{P}_k\}_{k \in \mathbb{N}}$ be such that $\mathcal{P}_k \subseteq \{0,1\}^{\ell(k)}$ and $\mathcal{S}_k \subseteq \{0,1\}^k$. A* Captcha *puzzle system $\mathcal{C} := (G, H)$ over $(\mathcal{P}, \mathcal{S})$ is a pair such that G is a randomized polynomial time turing machine, called the* generation algorithm*, and $H := \{H_k\}_{k \in \mathbb{N}}$ is a collection of* solution functions *such that $H_k : \mathcal{P}_k \rightarrow \mathcal{S}_k$. Algorithm G, on input a security parameter $k \in \mathbb{N}$, outputs a tuple $(z, a) \in \mathcal{P}_k \times \mathcal{S}_k$ such that $H_k(z) = a$. We require that there exists a negligible function* negl$(\cdot)$ *such that for every* PPT *algorithm $\mathcal{A}$, and every sufficiently large $k \in \mathbb{N}$, we have that:*

$$p_{inv}(k) := \Pr\left[(z, a) \leftarrow G(1^k); \mathcal{A}(1^k, z) = a\right] \leq \text{negl}(k)$$

where the probability is taken over the randomness of both G and $\mathcal{A}$.

Turing Machines vs Oracle Turing Machines. We emphasize that the Captcha puzzle generation algorithm G is an ordinary turing machine with no access to any oracles. Furthermore, the security of a Captcha system holds *only* against PPT adversaries $\mathcal{A}$ who are turing machines. It *does not* hold against oracle turing machines with oracle access to H. However, we use Captcha systems defined as above in protocols which guarantee security against adversaries who may even have access to the oracle H. This distinction between machines which have access to an (human) oracle and machines which don't occurs throughout the text.

The Issue of Malleability. As noted earlier, Captcha puzzles are usually easily *malleable* [DDN00]. That is, given a challenge puzzle z, it might be possible for an algorithm $\mathcal{A}$ to efficiently generate a new puzzle $z' \neq z$ such that given the solution of z', $\mathcal{A}$ can efficiently solve z. It turns out that in all previous works this creates several difficulties in the security proofs. In particular, in reducing the "security" of a cryptographic scheme to the "hardness" of the Captcha puzzle, it becomes unclear how to handle such an adversary.

Due to this, previous works [Dzi10, CHS06] only prove security against a very restricted class of adversaries called the *conservative* adversaries. Such adversaries are essentially those who do not query H_k on any Captcha instances other than the ones that are provided to them by the system. To facilitate a proof against all PPT adversaries, we develop the notion of human-extractable Captcha puzzles below.

Human-Extractable Captcha Puzzles. The notion of human-extractable Captcha puzzles stems from the intuition that if a PPT algorithm $\mathcal{A}$ can solve a random instance z produced by G, then it must make queries $\bar{q} = (q_1, q_2, \ldots)$ to (functions in) H that contain sufficient information about z. More formally, suppose that z_1 and z_2 are generated by two random and independent executions of G. If $\mathcal{A}$ is given z_1 as input and it produces the correct solution, then the queries $\bar{q}$ will contain sufficient information about z_1 and no information about z_2 (since z_2 is independent of z_1 and never seen by $\mathcal{A}$). Therefore, by looking at the queries $\bar{q}$, it should be possible with the help of the human to deduce which of the two instances is solved by $\mathcal{A}$. We say that a Captcha puzzle system is human-extractable if there exists a PPT algorithm Extr which, by looking at the queries $\bar{q}$, can tell with the help of the human which of the two instances was solved by $\mathcal{A}$. The formal definition follows; recall the convention that output of oracle Turing machines includes the queries $\bar{q}$ they make to H and corresponding answers $\bar{a}$ received.

Definition 3.2.2. ***(Human-extractable* Captcha*)*** *A* Captcha *puzzle system* $\mathcal{C} := (G, H)$ *is said to be* human-extractable *if there exists an oracle* PPT *algorithm* Extr^H*, called the* extractor*, and a negligible function* $\text{negl}(\cdot)$*, such that for every oracle* PPT *algorithm* $\mathcal{A}^H$*,* *and every sufficiently large $k \in \mathbb{N}$, we have that:*

$$p_{\mathsf{fail}}(k) := \Pr\left[\begin{array}{l}(z_0, s_0) \leftarrow G(1^k); (z_1, s_1) \leftarrow G(1^k); b \xleftarrow{\$} \{0,1\}\,; \\ (s, \bar{q}, \bar{a}) \leftarrow \mathcal{A}^H(1^k, z_b); b' \leftarrow \mathsf{Extr}^H(1^k, (z_0, z_1), \bar{q}); \\ s = s_b \wedge b' \neq b\end{array}\right] \leq \mathrm{negl}(k)$$

where the probability is taken over the randomness of $G, \mathcal{A}$, and Extr*.*

Observe that except with negligible probability, $s_0 \neq s_1$, since otherwise one can break the hardness of $\mathcal{C}$(definition 3.2.1).

We believe that the notion of human-extractable CAPTCHA puzzles is a very natural notion; it may be of independent interest and find applications elsewhere. We note that while assuming the existence of human-extractable CAPTCHA puzzles may be a strong assumption, it is very different from the usual extractability assumptions in the literature such as the Knowledge-Of-Exponent (KOE) assumption [HT98, BP04]. In particular, often it might be possible to empirically test whether a given CAPTCHA system is human-extractable. For example, one approach for such a test is to just ask sufficiently many humans to correlate the queries $\bar{q}$ to one of the puzzles z_0 or z_1. If sufficiently many humans can correctly correlate $\bar{q}$ to z_b with probability noticeably better than 1/2, one can already conclude some form of weak extraction. Such weak extractability can then be amplified by using techniques from parallel repetition. In contrast, there is no such hope for KOE assumption (and other problems with similar "non-black-box" flavor) since they are not falsifiable [Nao03].

In this work, we only concern ourselves with human-extractable CAPTCHA puzzles. Thus we drop the adjective human-extractable as convenient.

Drawbacks of Our Approach and Other Considerations. While our framework significantly improves upon previous works [Dzi10, CHS06], it still has certain drawbacks which are impossible to eliminate in an asymptotic formulation such as ours.

The first drawback is that as the value of k increases, the solution becomes larger. It is not clear if the humans can consistently answer such a long solution. Therefore, such a formulation can become unsuitable for even very small values of k. The second drawback is that the current formulation enforces strict "rules" on how a human and a Turing machine communicate via oracle access to H. This does not capture "malicious" humans who can communicate with their computers in arbitrary ways. It is not even clear how to formally define such "malicious" humans for our purpose.

Finally, definition 3.2.1 enforces the condition that $|\mathcal{S}_k|$ is super-polynomial in k. For many CAPTCHA puzzle systems in use today, $|\mathcal{S}_k|$ may be small (e.g., polynomial in k or even a constant). Such CAPTCHA puzzles are not directly usable in our setting. Observe that if $|\mathcal{S}_k|$ is small, clearly $\mathcal{A}$ can solve a given challenge puzzle with noticeable probability. Therefore, it makes sense to consider the following weaker variant in definition 3.2.1: instead of requiring $\mathsf{p}_{\mathsf{inv}}$ to be negligible, we can consider it to be a small constant ϵ. Likewise, we can also consider weakening the extractability condition by in definition 3.2.2 by requiring $\mathsf{p}_{\mathsf{fail}}$ to be only noticeably better than 1/2.

A subtle point to observe here is that while it might be possible to *individually* amplify $\mathsf{p}_{\mathsf{inv}}$ and $\mathsf{p}_{\mathsf{fail}}$ by using parallel or sequential repetitions, it may not be possible to amplify *both at the same time*. Indeed, when $|\mathcal{S}_k|$ is small, the adversary $\mathcal{A}$ can simply ask one CAPTCHA puzzle for every solution $a \in \mathcal{S}_k$ multiple times and "hide" the challenge puzzle z_b (in some mauled form z'_b) somewhere in this large list of queries. Such a list of queries might have sufficient correlation with *both* z_0 and z_1 simply because the solutions of these both are in $\mathcal{S}_k$ and $\mathcal{A}$ has asked at least one puzzle for each solution in the whole space. In this case, even though parallel repetition may amplify $\mathsf{p}_{\mathsf{inv}}$, extraction might completely fail because the correlation corresponding to the challenge puzzle is not easy to observe in $\mathcal{A}$'s queries and answers.

As a consequence of this, our formulation essentially rules out the possibility of using such "weak" CAPTCHA puzzles for which both $\mathsf{p}_{\mathsf{inv}}$ and $\mathsf{p}_{\mathsf{fail}}$ are not suitable. This is admittedly a strong limitation, which seems to come at the cost of proving security beyond the class of conservative adversaries.

3.3 A Straight-line Extractable Commitment Scheme

In this section we present a straight-line extractable commitment scheme which uses human-extractable CAPTCHA puzzles. The hiding and binding properties of this commitment scheme rely on standard cryptographic assumptions, and the straight-line extraction property relies on the extraction property of CAPTCHA puzzles.

We briefly recall the notion of secure commitment schemes, with emphasis on the changes from the standard definition and then define the notion of straight-line extractable commitments.

Commitment Schemes. First, we present a definition of commitment schemes augmented with CAPTCHA puzzles. Let $\mathcal{C} := (G, H)$ be a CAPTCHA puzzle system, and let $\text{Com}_{\mathcal{C}} := \langle \mathsf{C}^H, \mathsf{R} \rangle$ be a two-party interactive protocol where (only) C has oracle access to the solution function family H[1]. We say that $\text{Com}_{\mathcal{C}}$ is a commitment scheme if: both C and R are PPT (interactive) TM receiving 1^k as the common input; in addition, C receives a string $m \in \{0,1\}^k$. Further, we require C to *privately* output a *decommitment* string d, and R to *privately* output an auxiliary string aux. The transcript of the interaction is called the *commitment* string, denoted by c. During the course of the interaction, let $\bar{q}$ and $\bar{a}$ be the queries and answers obtained by C via queries to the CAPTCHA oracle H. To denote the sampling of an honest execution of $\text{Com}_{\mathcal{C}}$, we use the following notation: $(c, (d, \bar{q}, \bar{a}), \mathsf{aux}) \leftarrow \langle \mathsf{C}^H(1^k, m), \mathsf{R}(1^k) \rangle$.

Notice that $(d, \bar{q}, \bar{a})$ is the output of oracle ITM C^H as defined in section 3.1. For convenience, we associate a polynomial time algorithm DCom which on input (c, d, aux) either outputs a message m, or $\perp$. It is required that for all honest executions where C commits to m, DCom always outputs m. We say that $\text{Com}_{\mathcal{C}}$ is an *ordinary* commitment scheme if $\bar{q}$ (and hence $\bar{a}$) is an empty string.

Furthermore, our definition of a commitment scheme allows for stateful commitments. In

[1]The reason we do not provide R with access to H, is because our construction does not need it, and therefore we would like to avoid cluttering the notation. In general, however, both parties can have access to H. Also, in our adversarial model, we consider all malicious receivers to have access to the oracle H

particular the output aux might be necessary for a successful decommitment of the committed message.

We assume that the reader is familiar with perfect/statistical binding and computational hiding properties of a commitment scheme. Informally, straight-line extraction property means that there exists an extractor $\mathsf{ComExtr}^H$ which on input the commitment string c (possibly from an interaction with a malicious committer), aux (from an honest receiver), and $\bar{q}$, outputs the committed message m (if one exists), except with negligible probability. If m is not well defined, there is no guarantee about the output of $\mathsf{ComExtr}$.

For any commitment, we use $\mathcal{M} = \mathcal{M}(c, \mathsf{aux})$ to denote a possible decommitment message defined by the commitment string c and the receiver state aux. If such a message is not well defined (say there could be multiple such messages or none at all) for a particular (c, aux), then define $\mathcal{M}(c, \mathsf{aux}) = \perp$.

Definition 3.3.1. ***(Straight-line Extractable Commitment)*** *A statistically-binding computationally-hiding commitment scheme* $\mathrm{Com}_{\mathcal{C}} := \langle \mathsf{C}^H, \mathsf{R} \rangle$ *defined over a human-extractable* CAPTCHA *puzzle system* $\mathcal{C} := (G, H)$ *is said to admit* straight-line extraction *if there exists a* PPT *algorithm* $\mathsf{ComExtr}^H$ *(the* extractor*) and a negligible function* $\mathrm{negl}(\cdot)$*, such that for every* PPT *algorithm* $\widehat{\mathsf{C}}$ *(a* malicious *committer whose input could be arbitrary), and every sufficiently large* $k \in \mathbb{N}$*, we have that:*

$$\Pr\left[\begin{array}{l} (c^*, (d^*, \bar{q}, \bar{a}), \mathsf{aux}) \leftarrow \langle \widehat{\mathsf{C}}^H(1^k, \cdot), \mathsf{R}(1^k) \rangle; \mathcal{M} = \mathcal{M}(c^*, aux); \\ m \leftarrow \mathsf{ComExtr}^H(1^k, \bar{q}, (c^*, \mathsf{aux})) : (\mathcal{M} \neq \perp) \wedge (m \neq \mathcal{M}) \end{array}\right] \leq \mathrm{negl}(k)$$

where the probability is taken over the randomness of $\widehat{\mathsf{C}}, \mathsf{R}$*, and* $\mathsf{ComExtr}$*.*

The Commitment Protocol. At a high level, the receiver R of our protocol will choose two CAPTCHA puzzles (z^0, z^1) (along with their solutions s^0, s^1). To commit to bit b, the sender C will select z^b using the OT protocol and commit to its solution s^b using an ordinary (perfectly-binding) commitment scheme $\langle C_{\mathrm{PB}}, R_{\mathrm{PB}} \rangle$. The solution to the puzzle is obtained by querying H on z^b. To decommit, first decommit to s^b which the receiver verifies; and then the receiver accepts b as the decommitted bit if the solution it received is equal to s^b. To facilitate this task, the receiver outputs an auxiliary string aux which contains (z^0, z^1, s^0, s^1). To commit to a k-bit string $m \in \{0,1\}^k$, this atomic protocol is repeated in *parallel* k-times (with some minor modifications as in Figure 3.1)

For convenience we assume that $\langle C_{\mathrm{PB}}, R_{\mathrm{PB}} \rangle$ is *non-interactive* (i.e., C sends only one message to R) for committing strings of length k^2. The decommitment string then consists of the committed messages and the randomness of C_{PB}. The formal description of our protocol appears in figure 3.1.

Theorem 3.3.2. *Assume that $\langle C_{\mathrm{PB}}, R_{\mathrm{PB}} \rangle$ is an ordinary, non-interactive, perfectly-binding and computationally-hiding commitment scheme, $\mathcal{C} = (G, H)$ is a human-extractable* CAPTCHA *puzzle system, and $\langle S_{\mathrm{OT}}, R_{\mathrm{OT}} \rangle$ is a two-round statistically-secure oblivious transfer protocol. Then, protocol $\mathrm{Com}_{\mathcal{C}} = \langle \mathsf{C}^H, \mathsf{R} \rangle$ described in figure 3.1 is a 3-round perfectly-binding and computationally-hiding commitment scheme which admits straight-line extraction.*

Proof. [Sketch] Statistical binding of our scheme follows from perfect binding of $\langle C_{\mathrm{PB}}, R_{\mathrm{PB}} \rangle$ and the fact that except with negligible probability over the randomness of G, $s_i^0 \neq s_i^1$ for every $i \in [k]$ (since otherwise $\mathsf{p}_{\mathsf{inv}}(k)$ will not be negligible).

In addition, the computational-hiding of this scheme follows by a standard hybrid argument which uses the following two conditions: the *receiver security* property of $\langle S_{\mathrm{OT}}, R_{\mathrm{OT}} \rangle$, and computational-hiding of $\langle C_{\mathrm{PB}}, R_{\mathrm{PB}} \rangle$. The proof of this part is standard, and omitted.

We now show that the scheme admits straight-line extraction. Suppose that string $(c^*, (d^*, \bar{q}, \bar{a}), \mathsf{aux})$ represents the output of an execution of our commitment protocol; then by statistical binding of our commitment scheme, except with negligible probability there is a unique message defined by this string. In fact, this unique message is completely defined by only the strings $(c^*, \{z_i^0, z_i^1\}_{i=1}^k)$, where $\{z_i^0, z_i^1\}_{i=1}^k$ are CAPTCHA puzzles of the honest receiver (included in aux). Recall that to refer to this message, we use the variable $\mathcal{M}$, and write $\mathcal{M}(c^*, \mathsf{aux})$ to explicitly mention a transcript.

Now suppose that for a commitment scheme, it holds that $\Pr[\mathcal{M} \neq \perp]$ is negligible, then straight-line extraction property as in Definition 3.3.1 holds trivially. Therefore, in our analysis, it suffices to analyze malicious committers who satisfy the condition that

Let k be the security parameter, $\mathcal{C} := (G, H)$ a human-extractable CAPTCHA puzzle system, $\langle C_{\text{PB}}, R_{\text{PB}} \rangle$ a non-interactive perfectly-binding commitment scheme for strings of length k^2, and $\langle S_{\text{OT}}, R_{\text{OT}} \rangle$ a two-message two-party OT protocol.

Commitment. Let $m = (m_1, \dots, m_k) \in \{0,1\}^k$ be the message to be committed.

1. CAPTCHA GENERATION: For every $i \in [k]$, R generates a pair of independent CAPTCHA puzzles: $(z_i^0, s_i^0) \leftarrow G(1^k)$ and $(z_i^1, s_i^1) \leftarrow G(1^k)$.

2. PARALLEL OT: C and R perform k parallel executions of OT, where the i^{th} execution proceeds as follows. Party R acts as the OT-sender S_{OT} on input (z_i^0, z_i^1) and party C acts the OT-receiver R_{OT} on input the bit m_i. At the end of the execution, let the puzzle instances obtained by C be $z_1^{m_1}, \dots, z_k^{m_k}$.

3. COMMIT TO CAPTCHA SOLUTIONS: For every $i \in [k]$, C queries H_k on $z_i^{m_i}$ to obtain the solution $s_i^{m_i}$. Let $\bar{s} := s_1^{m_1} \circ \dots \circ s_k^{m_k}$, which is of length k^2. C commits to $\bar{s}$ using protocol $\langle C_{\text{PB}}, R_{\text{PB}} \rangle$. Let r be the randomness used and c be the message sent by C in this step.

4. OUTPUTS: R sets $\mathsf{aux} = \{(z_i^0, z_i^1, s_i^0, s_i^1)\}_{i=1}^k$, and C sets $d = (\bar{s}, r)$.

Decommitment. On input the commitment transcript, and strings $d = (\bar{s}, r)$ and $\mathsf{aux} = \{(z_i^0, z_i^1, s_i^0, s_i^1)\}_{i=1}^k$ do the following: parse the transcript to obtain string c from the last step, and verify that $(\bar{s}, r)$ is a valid decommitment for c. If yes, parse $\bar{s} = a_1 \circ \dots \circ a_k$ and test that for every $i \in [k]$, there exists a *unique* bit b_i such that $a_i = s_i^{b_i}$. If any test fails, output $\perp$; otherwise output $m = (b_1, \dots, b_k)$.

Figure 3.1: Straightline Extractable Commitment Protocol $\langle \mathsf{C}^H, \mathsf{R} \rangle$

the event $\mathcal{M} \neq \perp$ happens with non-negligible probability. The formal description of our commitment-extractor, $\mathsf{ComExtr}$, follows. It uses the (extractor-)machine Extr guaranteed for the CAPTCHA system $\mathcal{C}$ (by definition 3.2.2), to extract m bit-by-bit.

```
ALGORITHM $\mathsf{ComExtr}^H(1^k, \bar{q}, (c^*, \mathsf{aux}))$:
    1. Parse $\mathsf{aux}$ to obtain the puzzles $\{z_i^0, z_i^1\}_{i=1}^k$.
    2. For every $i \in [k]$, set $b_i \leftarrow \mathsf{Extr}^H(1^k, (z_i^0, z_i^1), \bar{q})$. If $\mathsf{Extr}^H$ outputs $\perp$, then set $b_i = \perp$.
    3. Output the string $b_1 \circ \ldots \circ b_k$.
```

Observe that the extraction algorithm *does not* use the solutions $\{s_i^0, s_i^1\}_{i=1}^k$ included in the string aux, and this information is redundant. Also, the output of the algorithm above, may include the $\perp$ symbols in some places.

We say that $\mathsf{ComExtr}^H(1^k, \bar{q}, (c^*, \mathsf{aux}))$ *fails at step* i if $b_i = \perp$ or if $b_i \neq \mathcal{M}_i$, where $\mathcal{M}_i$ denotes the i^{th} bit of the unique message $\mathcal{M}$ if it exists. Further, let $p_i(k)$ denote the probability that this happens. Define the following probability $p_i(k)$ over the randomness of $\widehat{\mathsf{C}}$ and R and Extr:

$$p_i(k) := \Pr\left[\begin{array}{l}(c^*, (d^*, \bar{q}, \bar{a}), \mathsf{aux}) \leftarrow \langle \widehat{\mathsf{C}}^H(1^k, \cdot), \mathsf{R}(1^k)\rangle; \mathcal{M} := \mathcal{M}(c^*, \mathsf{aux}); \\ b_i = \mathsf{Extr}^H(1^k, (z_i^0, z_i^1), \bar{q}) : \mathcal{M}_i \neq \perp \wedge b_i \neq \mathcal{M}_i\end{array}\right] \tag{3.3.1}$$

We remark again, that $p_i(k)$ is not affected by the actual solutions $\{s_i^0, s_i^1\}_{i=1}^k$ included in aux in the equation above. This is because $\mathcal{M}$ is completely defined by c^* and the puzzles $\{z_i^0, z_i^1\}_{i=1}^k$, and every other expression in the equation does not depend on aux.

If we prove that $p_i(k)$ is negligible for every $i \in [k]$, then by union bound, it follows that, except with negligible probability, the output of $\mathsf{ComExtr}$ is always equal to $\mathcal{M}$ (or $\mathcal{M}$ equals $\perp$). This will complete the proof of the theorem. To show $p_i(k)$ is negligible, we construct an adversary for the extraction game in Definition 3.2.2(we will call such an adversary an CAPTCHA *extraction adversary*), and then show that $p_i(k)$ and $\mathrm{p}_{\mathsf{fail}}$ are negligibly close.

Informally, the proof follows from the fact that when the commitment adversary solves a particular CAPTCHA puzzle, say z_i^b, and commits to its solution s_i^b, the "other" CAPTCHA puzzle z_i^{1-b} is statistically hidden from his view due to the statistical security of the OT protocol. Thus, this commitment adversary can be converted to a CAPTCHA extraction adversary by setting $z_b^i = \tilde{z}$, where $\tilde{z}$ is the CAPTCHA puzzle that the CAPTCHA extraction adversary gets to see in the extraction game (Definition 3.2.2). Due to space limitations, we defer the formal proof to Appendix 4.1.

■

3.4 Constructing UC-Puzzles using Captcha

We provided a basic background in the section 1.2 to our results on protocol composition, and mentioned that there are two ways in which we can incorporate CAPTCHA puzzles in the UC-framework: the indirect access model, and the direct access model. This section is about constructing *UC puzzles* [LPV09] in the indirect access model. Recall that in the indirect access model, the environment $\mathcal{Z}$ is not given direct access to a human (or the solution function family of the CAPTCHA system) H; instead, $\mathcal{Z}$ must access H exclusively through the adversary $\mathcal{A}$. This allows the simulator to look at the queries of $\mathcal{Z}$, which in turn allows for a positive result. Due to space constraints, we shall assume basic familiarity with the UC-framework [Can01], and directly work with the notion of UC-puzzles. A more detailed review of the UC framework, and concurrent composition, is given in appendix 4.3 .

Lin, Pass and Venkitasubramaniam [LPV09] defined the notion of a *UC puzzle*, and demonstrated that to obtain universal-composition in a particular model (e.g., the CRS model), it suffices to construct a UC puzzle in that model. We will adopt this approach, and construct a UC puzzle using CAPTCHA. We recall the notion of a UC-puzzle with necessary details, and refer the reader to [LPV09] for an extensive exposition. Our formulation directly incorporates CAPTCHA puzzles in the definition and hence does not refer to any setup $\mathcal{T}$; other than this semantic change, the description here is essentially identical to that of [LPV09].

The UC-puzzle is a protocol which consists of two parties—a sender S, and a receiver R, and a PPT-relation $\mathcal{R}$. Let $\mathcal{C} := (G, H)$ be a CAPTCHA puzzle system. Only the sender will be given oracle access to H, and the resulting protocol will be denoted by $\langle S^H, R\rangle$. Informally, we want that the protocol be *sound*: no efficient receiver R^* can successfully complete an interaction with S and also obtain a "trapdoor" y such that $\mathcal{R}(\mathsf{TRANS}, y) = 1$, where

TRANS is the transcript of that execution. We also require *statistical UC-simulation*: for every efficient adversary $\mathcal{A}$ participating as a sender in many executions of the protocol with multiple receivers $R_1, \ldots, R_m$, and communicating with an environment $\mathcal{Z}$ simultaneously, there exists a simulator Sim which can *statistically* simulate the view of $\mathcal{A}$ for $\mathcal{Z}$ and output trapdoors to all successfully completed puzzles at the same time.

Formally, we consider a concurrent execution of the protocol $\langle S^H, R\rangle$ for an adversary $\mathcal{A}$. In the concurrent execution, $\mathcal{A}$ exchanges messages with a puzzle-environment $\mathcal{Z}$ and participates as a sender concurrently in $m = \mathrm{poly}(k)$ (puzzle)-protocols with honest receivers $R_1, \ldots, R_m$. At the onset of a execution, $\mathcal{Z}$ outputs a session identifier *sid* that all receivers receive as input. Thereafter, $\mathcal{Z}$ is allowed to exchange messages only with the adversary $\mathcal{A}$. In particular, for any queries to the CAPTCHA solving oracle, $\mathcal{Z}$ cannot query H; instead, it can send its queries to $\mathcal{A}$, who in turn, can query H for $\mathcal{Z}$, and report the answer back to $\mathcal{Z}$. We compare a real and an ideal execution.

REAL EXECUTION. The real execution consists of the adversary $\mathcal{A}$, which interacts with $\mathcal{Z}$, and participates as a sender in m concurrent interactions of $\langle S^H, R\rangle$. Further, the adversary and the honest receivers have access to H which they can query and receive the solutions over secure channels. The environment $\mathcal{Z}$ does not have access to H; it can query H, by sending its queries to $\mathcal{A}$, who queries H with the query and reports the answers back to $\mathcal{Z}$. Without loss of generality, we assume that after every interaction, $\mathcal{A}$ honestly sends TRANS to $\mathcal{Z}$, where TRANS is the transcript of execution. Let $\textsc{real}^H_{\mathcal{A},\mathcal{Z}}(k)$ be the random variable that describes the output of $\mathcal{Z}$ in the real execution.

IDEAL EXECUTION. The ideal execution consists of a PPT machine (the simulator) with oracle access to H, denoted Sim^H. On input 1^k, Sim^H interacts with the environment $\mathcal{Z}$. At the end of the execution, the environment produces an output. We denote the output of $\mathcal{Z}$ in the ideal execution by the random variable $\textsc{ideal}_{\mathsf{Sim}^H,\mathcal{Z}}(k)$.

Definition 3.4.1. ***(*UC-Puzzle*, adapted from [LPV09])*** *Let* $\mathcal{C} := (G, H)$ *be a* CAPTCHA *puzzle system. A pair $(\langle S^H, R\rangle, \mathcal{R})$ is called UC-puzzle for a polynomial time computable relation $\mathcal{R}$ and the* Captcha *puzzle system $\mathcal{C}$, if the following conditions hold:*

- Soundness. *There exists a negligible function* $\text{negl}(\cdot)$ *such that for every* PPT *receiver $\mathcal{A}$, and every sufficiently large k, the probability that $\mathcal{A}$, after an execution with the sender S^H on common input 1^k, outputs y such that $y \in \mathcal{R}(\mathsf{TRANS})$ where TRANS is the transcript of the message exchanged in the interaction, is at most* $\text{negl}(k)$.

- Statistical Simulation. *For every* PPT *adversary $\mathcal{A}$ participating in a* concurrent puzzle execution, *there exists an oracle* PPT *machine called the simulator, Sim^H, such that for every* PPT *environment $\mathcal{Z}$ and every sufficiently large k, the random variables* $\text{REAL}^H_{\mathcal{A},\mathcal{Z}}(k)$ *and* $\text{IDEAL}_{\mathsf{Sim}^H,\mathcal{Z}}(k)$ *are statistically close over $k \in \mathbb{N}$, and whenever Sim sends a message of the form TRANS to $\mathcal{Z}$, it outputs y in its special output tape such that $y \in \mathcal{R}(\mathsf{TRANS})$.*

Some Tools and Notation. To construct the UC-puzzle, we will use our straight-line extractable commitment scheme $\text{Com}_{\mathcal{C}} := \langle \mathsf{C}^H, \mathsf{R}\rangle$ from figure 3.1. However, this commitment scheme is too weak for our purposes. In particular, it has the following issues: it is possible for a cheating committer to commit to an invalid string $\perp$ (simply by committing incorrect solutions to Captcha puzzles), and this event cannot be detected by the receiver. We would like to ensure that if the receiver accepts the transcript, then except with negligible probability, there be a unique and valid string fixed by the transcript of the communication. However, since the Captcha solutions cannot be verified by a PPT Turing machine, we cannot use zero-knowledge proofs right-away to guarantee this.

Therefore, we resort to using our extractable commitment scheme along with an ordinary commitment scheme, and then use simple "cut-and-choose" techniques to ensure that the two commitment schemes commit to same values. Once this is ensured, the ordinary commitment scheme will provide us with **NP**-relations which we can work with.

Formally, let $\langle C_{\text{PB}}, R_{\text{PB}}\rangle$ be a non-interactive perfectly-binding and computationally-hiding commitment scheme; and let $\text{Com}_{\mathcal{C}} := \langle \mathsf{C}^H, \mathsf{R}\rangle$ be our extractable-commitment scheme in figure 3.1 defined over an human-extractable CAPTCHA puzzle system $\mathcal{C} := (G, H)$. Then, define the following commitment scheme:

SCHEME $\widehat{\mathsf{Com}}(v)$:

> To commit to a value v, the sender commits to v twice: first commit to v using the protocol $\langle C_{\mathrm{PB}}, R_{\mathrm{PB}} \rangle$, then commit to v using the protocol $\langle \mathsf{C}^H, \mathsf{R} \rangle$. The Receiver accepts the commitment if both, R_{PB} and R, accept their respective commitments.

To open to a value v, the sender executes the opening phases of both C_{PB} and $\mathrm{Com}_{\mathcal{C}}$; and if both opening phases accept v as the decommitted value, the receiver of $\widehat{\mathsf{Com}}$ also accepts v as the decommitted value. That fact that $\widehat{\mathsf{Com}}$ is statistically-binding and computationally-hiding follows directly from the corresponding properties of $\langle C_{\mathrm{PB}}, R_{\mathrm{PB}} \rangle$ and $\langle \mathsf{C}^H, \mathsf{R} \rangle$. Now define the following 3-round "cut-and-choose" protocol for committing to a string s, which is essentially taken from [Ros04, PRS02], except that it uses the scheme $\widehat{\mathsf{Com}}$:

SCHEME $\mathsf{PRSCom}(s)$:

1. Sender selects $2k$ strings $\{ s_0^i \}_{i=1}^k, \{ s_1^i \}_{i=1}^k$ such that $s = s_0^i \oplus s_1^i$. Now the sender commits to string s, as well as all $2k$ strings $\{ s_0^i \}_{i=1}^k, \{ s_1^i \}_{i=1}^k$ using the special commitment scheme $\widehat{\mathsf{Com}}$.
2. Receiver sends a uniformly selected challenge $\sigma = \sigma_1 \circ \ldots \circ \sigma_k \in \{ 0, 1 \}^k$.
3. Sender opens to $s_{\sigma_i}^i$ for each $1 \leq i \leq k$ by sending the decommitment information for $\widehat{\mathsf{Com}}$. Receiver accepts the commitment phase, if these are valid openings.

To open the committed string s according to the scheme PRSCom, the sender simply opens the rest of commitment values as well, i.e., decommitment to s as well as to the remaining k unopened strings. The fact that PRSCom is actually a statistically-binding and computationally-hiding commitment scheme follows (only) from the fact that $\widehat{\mathsf{Com}}$ is a statistically-binding and computationally-hiding commitment scheme. This is a standard proof, and is omitted (see, e.g., [Ros04, PRS02]).

For convenience, we define the notion of *well-formedness* of PRSCom. Informally, we say that a PRS commitment is well formed if each pair of commitments in the first step is indeed to valid shares of s.

Definition 3.4.2. ***(Partial Transcripts)*** *Let $\langle \rho, (\rho_0^1, \rho_1^1), \ldots, (\rho_0^k, \rho_1^k) \rangle$ be the transcripts of the commit phases of the $2k+1$ executions of $\widehat{\mathsf{Com}}$ in Step 1 of a successfully completed execution of PRSCom. For each $(i, b) \in \{1, \ldots, k\} \times \{0, 1\}$, parse ρ_b^i as a pair of transcripts (τ_b^i, θ_b^i), where τ_b^i is the transcript of the commit phase of $\langle C_{\mathrm{PB}}, R_{\mathrm{PB}} \rangle$, and θ_b^i is the transcript of the commit phase of $\langle \mathsf{C}^H, \mathsf{R} \rangle$, in the (i, b)-execution of $\widehat{\mathsf{Com}}$. Similarly, let $\rho = (\tau, \theta)$. Define the* partial transcript *to be the tuple $\langle \tau, (\tau_0^1, \tau_1^1), \ldots, (\tau_0^k, \tau_1^k) \rangle$.*

Definition 3.4.3. ***(Well-formed PRS)*** *Let $\Psi = \langle \tau, (\tau_0^1, \tau_1^1), \ldots, (\tau_0^k, \tau_1^k) \rangle$ be the partial transcript of the commitment phase of PRSCom. Then, we say Ψ is* well-formed *if for every $1 \leq i \leq k$: $s_0^i \oplus s_1^i = s$ where s_b^i and s denote the strings committed in τ_b^i and τ respectively and $b \in \{0, 1\}$.*

For a partial transcript $\Psi = \langle \tau, (\tau_0^1, \tau_1^1), \ldots, (\tau_0^k, \tau_1^k) \rangle$ we define the string committed in Ψ, s_Ψ, as follows: if Ψ is well-formed, then $s_\Psi := s$, where s is the string committed in τ. Else, $s_\Psi := \perp$.

Note that as $\langle C_{\mathrm{PB}}, R_{\mathrm{PB}} \rangle$ is a perfectly-binding commitment scheme, given Ψ, the statement that "Ψ is well-formed" is an **NP**-statement where the witness consists of all the committed strings along with the correct openings (which, in turn, is just the randomness used by algorithm C_{PB}).

The UC-puzzle System. The construction of our UC-puzzle over an human-extractable Captcha puzzle system $\mathcal{C} := (G, H)$, denoted $\left(\langle S^H, R \rangle, \mathcal{R}\right)$ appears in figure 3.2.

$\mathcal{R}(\mathsf{TRANS}, s) = 1$ if and only if:

1. $\mathsf{TRANS} := ((z, f_k), \tau_2, \tau_3) \in \{0,1\}^{\ell_1(k)}$, and $s \in \{0,1\}^k$.
2. $z = f_k(s)$, where (z, f_k) represents sender's Phase-1 message
3. τ_3, representing the transcript of ZK-proof in Phase-3, is accepting.

All parties are given k as the security parameter, and $\{ f_k \}_{k \in \mathbb{N}}$ is a family of one-way permutations such that $f_k : \{0,1\}^k \to \{0,1\}^{\ell'(k)}$.

Phase-1. Sender chooses a string $s \in \{0,1\}^k$ uniformly, and sends $z = f_k(s)$ and the description of f_k.

Phase-2. Sender commits to the string s using the scheme $\mathsf{PRSCom}(s)$. Let $\Psi := \langle \tau, (\tau_0^1, \tau_1^1), \ldots, (\tau_0^k, \tau_1^k) \rangle$ be the *partial transcript*. If the zero-knowledge proof is accepting, the Receiver accepts and halts.

Figure 3.2: UC Puzzle $\langle S^H, R \rangle$ using a CAPTCHA puzzle system $\mathcal{C} := (G, H)$

The string s is defined to be the *trapdoor* of the accepting execution TRANS.

Theorem 3.4.4. *Assume that $\{f_k\}_{k \in \mathbb{N}}$ is a family of one-way permutations, and that $\mathcal{C} := (G, H)$ is an human-extractable* CAPTCHA *puzzle system. Then, the protocol $\langle S^H, R \rangle$ in figure 3.2 is a UC puzzle.*

Due to space constraints, we refer the reader to a complete proof of this theorem provided in appendix 4.2 . Very briefly and informally, the high-level ideas of the proof proceed as follows. First, the cut-and-choose method in PRSCom ensures that the corresponding components of two commitment schemes (included in $\widehat{\mathsf{Com}}$), are indeed equal with high probability. Then, the ZK-proof guarantees that for one of them that the XOR of the majority of pairs committed to yield a unique and well defined value, say s_Ψ and that it is equal to the desired trapdoor s, with high probability. Then, s_Ψ can be recovered from the extractable-commitment part to prove the statistical simulation. Soundness is proven using a standard hybrid argument.

3.5 Conclusion

Open Questions and Future Work. Our work presents a basic technique using human-extractable CAPTCHA puzzles to enable straight-line extraction and shows how to incorporate it into the framework of protocol composition to obtain new and interesting feasibility results. However, many other important questions remain to be answered. For examples, can we obtain zero-knowledge proofs for **NP** in 3 or less rounds?[2] Can we obtain plain-text aware encryption-schemes? What about *non-interactive* non-malleable commitments *without* setup [DDN00, CIO98, CKO01, PPV08]?

One interesting direction is to consider improving upon the recent work of Goyal, Jain, and Ostrovsky on generating a password-based session-keys in the concurrent setting [GJO10]. One of the main difficulties in [GJO10] is to get a control on the number of times the simulator rewinds any given session. They accomplish this by using the technique of precise-simulation [MP06, PPS08]. However, since we obtain straight-line simulation, it seems likely that our techniques could be used to improve the results in [GJO10]. The reason we are not able to do this is that our techniques are limited to only simulation—they do *not* yield both straight-line simulation and extraction, whereas [GJO10] needs a control over both.

Another interesting direction is to explore the design of extractable CAPTCHA puzzles. In general, investigating the feasibility and drawbacks of the asymptotic formulation for CAPTCHA puzzles presented here and in [ABH03, CHS06, Dzi10] is an interesting question in its own right. We presented a discussion of these details in section 3.2, however they still present numerous questions for future work.

[2]By using standard techniques, e.g., coin-tossing using our commitment scheme along with Blum's protocol [Blu87], we can obtain a 5-round (concurrent) zero-knowledge protocol. But we do not know how to reduce it to 3 rounds.

CHAPTER 4

Concurrent Non-Malleable Zero Knowledge

4.1 Proof of Theorem 3.3.2

In this section we present the complete proof of security of our extractable commitment scheme.

Theorem 4.1.1 (Restated). *Assume that $\langle C_{\mathrm{PB}}, R_{\mathrm{PB}} \rangle$ is an ordinary, non-interactive, perfectly-binding and computationally-hiding commitment scheme, $\mathcal{C} = (G, H)$ is a human-extractable* Captcha *puzzle system, and $\langle S_{\mathrm{OT}}, R_{\mathrm{OT}} \rangle$ is a two-round statistically-secure oblivious transfer protocol. Then, protocol* $\mathrm{Com}_{\mathcal{C}} = \langle \mathsf{C}^H, \mathsf{R} \rangle$ *described in figure 3.1 is a 3-round perfectly-binding and computationally-hiding commitment scheme which admits straight-line extraction.*

Proof. Statistical binding of our scheme follows from perfect binding of $\langle C_{\mathrm{PB}}, R_{\mathrm{PB}} \rangle$ and the fact that except with negligible probability over the randomness of G, $s_i^0 \neq s_i^1$ for every $i \in [k]$ (since otherwise $\mathrm{p}_{\mathsf{inv}}(k)$ will not be negligible).

In addition, the computational-hiding of this scheme follows by a standard hybrid argument which uses the following two conditions: the *receiver security* property of $\langle S_{\mathrm{OT}}, R_{\mathrm{OT}} \rangle$, and computational-hiding of $\langle C_{\mathrm{PB}}, R_{\mathrm{PB}} \rangle$. The proof of this part is standard, and omitted.

We now show that the scheme admits straight-line extraction. Suppose that string $(c^*, (d^*, \bar{q}, \bar{a}), \mathsf{aux})$ represents the output of an execution of our commitment protocol; then by statistical binding of our commitment scheme, except with negligible probability there is a unique message defined by this string. In fact, this unique message is completely defined by only the strings $(c^*, \{z_i^0, z_i^1\}_{i=1}^k)$, where $\{z_i^0, z_i^1\}_{i=1}^k$ are Captcha puzzles of the honest receiver (included in aux). Recall that to refer to this message, we use the variable $\mathcal{M}$, and write $\mathcal{M}(c^*, \mathsf{aux})$ to explicitly mention a transcript.

Now suppose that for a commitment scheme, it holds that $\Pr[\mathcal{M} \neq \perp]$ is negligible, then straight-line extraction property as in Definition 3.3.1 holds trivially. Therefore, in our analysis, it suffices to analyze malicious committers who satisfy the condition that the event $\mathcal{M} \neq \perp$ happens with non-negligible probability. The formal description of our commitment-extractor, $\mathsf{ComExtr}$, follows. It uses the (extractor-)machine Extr guaranteed for the CAPTCHA system $\mathcal{C}$ (by definition 3.2.2), to extract m bit-by-bit.

```
ALGORITHM $\mathsf{ComExtr}^H(1^k, \bar{q}, (c^*, \mathsf{aux}))$:
    1. Parse $\mathsf{aux}$ to obtain the puzzles $\{z_i^0, z_i^1\}_{i=1}^k$.
    2. For every $i \in [k]$, set $b_i \leftarrow \mathsf{Extr}^H(1^k, (z_i^0, z_i^1), \bar{q})$. If $\mathsf{Extr}^H$ outputs $\perp$, then set $b_i = \perp$.
    3. Output the string $b_1 \circ \ldots \circ b_k$.
```

Observe that the extraction algorithm *does not* use the solutions $\{s_i^0, s_i^1\}_{i=1}^k$ included in the string aux, and this information is redundant. Also, the output of the algorithm above, may include the $\perp$ symbols in some places.

We say that $\mathsf{ComExtr}^H(1^k, \bar{q}, (c^*, \mathsf{aux}))$ *fails at step* i if $b_i = \perp$ or if $b_i \neq \mathcal{M}_i$, where $\mathcal{M}_i$ denotes the ith bit of the unique message $\mathcal{M}$ if it exists. Further, let $p_i(k)$ denote the probability that this happens. Define the following probability $p_i(k)$ over the randomness of $\widehat{\mathsf{C}}$ and R and Extr:

$$p_i(k) := \Pr\left[\begin{array}{l}(c^*, (d^*, \bar{q}, \bar{a}), \mathsf{aux}) \leftarrow \langle \widehat{\mathsf{C}}^H(1^k, \cdot), \mathsf{R}(1^k)\rangle; \mathcal{M} := \mathcal{M}(c^*, \mathsf{aux}); \\ b_i = \mathsf{Extr}^H(1^k, (z_i^0, z_i^1), \bar{q}) : \mathcal{M}_i \neq \perp \wedge b_i \neq \mathcal{M}_i\end{array}\right] \tag{4.1.1}$$

We remark again, that $p_i(k)$ is not affected by the actual solutions $\{s_i^0, s_i^1\}_{i=1}^k$ included in aux in the equation above. This is because $\mathcal{M}$ is completely defined by c^* and the puzzles $\{z_i^0, z_i^1\}_{i=1}^k$, and every other expression in the equation does not depend on aux.

If we prove that $p_i(k)$ is negligible for every $i \in [k]$, then by union bound, it follows that, except with negligible probability, the output of $\mathsf{ComExtr}$ is always equal to $\mathcal{M}$ (or $\mathcal{M}$ equals

$\perp$). This will complete the proof of the theorem. To show $p_i(k)$ is negligible, we construct an adversary for the extraction game in Definition 3.2.2(we will call such an adversary an CAPTCHA *extraction adversary*), and then show that $p_i(k)$ and $\mathrm{p}_{\mathsf{fail}}$ are negligibly close.

Informally, the proof follows from the fact that when the commitment adversary solves a particular CAPTCHA puzzle, say z_i^b, and commits to its solution s_i^b, the "other" CAPTCHA puzzle z_i^{1-b} is statistically hidden from his view due to the statistical security of the OT protocol. Thus, this commitment adversary can be converted to a CAPTCHA extraction adversary by setting $z_b^i = \tilde{z}$, where $\tilde{z}$ is the CAPTCHA puzzle that the CAPTCHA extraction adversary gets to see in the extraction game (Definition 3.2.2).

To prove the above statement formally, we analyze the following hybrid games.

Hybrid $\mathcal{H}_1$. In this hybrid experiment, we consider the interaction of $\widehat{\mathsf{C}}$ with the following simulator S. Simulator S interacts with $\widehat{\mathsf{C}}$ on common input 1^k exactly like an honest receiver R of our commitment protocol except that in session i, the messages corresponding to the OT protocol are forwarded to an external (honest) OT-sender S_{OT}. The OT-sender S_{OT} is given inputs (x_0, x_1) where $(x_b, s_b) \leftarrow G(1^k)$ for $b \in \{0,1\}$ by S. At the end of the experiment, denote by let $(d', \bar{q}', \bar{a}')$ denote the contents of output tape of $\widehat{\mathsf{C}}$, and c' denote the commitment string. Define the puzzles and solutions corresponding to session i as follows: $z_i^0 = x_0, z_i^1 = x_1, s_i^0 = \perp, s_i^1 = \perp$. By defining these values, we have completely defined aux' (which includes puzzles and solutions for all sessions). Define $(c', (d', \bar{q}', \bar{a}'), \mathsf{aux}')$ to be the output of $\mathcal{H}_1$.

Let $q_1(k)$ denote the probability that the event in equation (4.1.1) occurs, when we replace the string $(c^*, (d^*, \bar{q}, \bar{a}), \mathsf{aux})$ by the output of $\mathcal{H}_1$—i.e., $(c', (d', \bar{q}', \bar{a}'), \mathsf{aux}')$.

Observe that except for the solutions (s_i^0, s_i^1), all components of the string $(c', (d', \bar{q}', \bar{a}'), \mathsf{aux}')$, are distributed identically to the corresponding components of string $(c^*, (d^*, \bar{q}, \bar{a}), \mathsf{aux})$. Further, as noted earlier, the event in equation (4.1.1) is independent of (s_i^0, s_i^1). Therefore, $\mathsf{ComExtr}$ *fails at step i* with probability $p_i(k)$ even if we use the strings $(c', (d', \bar{q}', \bar{a}'), \mathsf{aux}')$ sampled according to Hybrid 1, in equation 4.1.1. Therefore, we have that $q_1(k) = p_i(k)$. Note that experiment $\mathcal{H}_1$ is a polynomial time process.

Before going to the next set of hybrid experiments, we make some observations. Recall that our protocol consists of k parallel OT-executions; let us denote them by $\mathsf{OT}_1, \ldots, \mathsf{OT}_k$. Each OT-execution has exactly 2-rounds, and the message corresponding to execution OT_i, say α_i is forwarded to S_{OT}(in $\mathcal{H}_1$). To compute α_i, our simulator S, selects a random-tape r which is independent from the randomness used to sample x_0 and x_1 of experiment $\mathcal{H}_1$. Let $\mathcal{R}$ be the domain from which r is sampled.

Consider an exponential number of experiments $\mathcal{H}_r$, defined below, one for each $r \in \mathcal{R}$. We show that in each one of them, $q_r(k)$, defined analogous to $q_1(k)$, is negligible. We note that unlike "standard" hybrid arguments which use indistinguishability of consecutive hybrids, we have that $q_1(k)$ is related to $q_r(k)$ via equation 4.1.2.

Hybrid $\mathcal{H}_r$. This experiment is identical to $\mathcal{H}_1$ except that the randomness sampled by S apart from that used to sample (x_0, x_1) of is fixed to the string r. The output of this experiment is defined exactly as in $\mathcal{H}_r$. Let $q_r(k)$ denote the probability that event in equation (4.1.1) occurs when we replace the string $(c^*, (d^*, \bar{q}, \bar{a}), \mathsf{aux})$ by the output of $\mathcal{H}_1$. By definition, we have that for all $k \in \mathbb{N}$:

$$q_1(k) = \sum_{r \in \mathcal{R}} q_r(k) \cdot \Pr[r \text{ is the randomness sampled by } S] \leq \max_r q_r(k) \tag{4.1.2}$$

We consider the hybrids $\mathcal{H}_r$ to enable us to use the statistical security property of the OT protocol. Consider any r (one that makes the value $q_r(k)$ maximum is most convenient). To measure q_r, we will make use of definition 3.1.1 and get rid of one of the puzzles (x_0, x_1). Note that, every string $r \in \mathcal{R}$ results in our simulator S sending a string α_i to S_{OT}. Since r generates α_i, and r is hardwired in $\mathcal{H}_r$, this experiment always sends the same string α_i to S_{OT}, who responds with some reply, say $\beta \leftarrow S_{\mathrm{OT}}((x_0, x_1), \alpha_i)$. By definition 3.1.1, it holds that random-variable β is statistically-close to one of the following: either $S_{\mathrm{OT}}((x_0, 0^{\ell(k)}), \alpha_i)$ or $S_{\mathrm{OT}}((0^{\ell(k)}, x_1), \alpha_i)$. Let $\gamma(r)$ be the bit that indicates which of these two cases is true (with ties broken arbitrarily). Note that the value of $\gamma(r)$ for every $r \in \mathcal{R}$, is fixed and can be given to a non-uniform machine as an advice.

Hybrid $\mathcal{H}_r^1$. This hybrid is identical to $\mathcal{H}_r$, except that it non-uniformly receives the value of the bit $\gamma(r)$. The output of this experiment is therefore distributed identically to that of $\mathcal{H}_r$.

Hybrid $\mathcal{H}_r^2$. This hybrid is identical to $\mathcal{H}_r$ except for the following difference: instead of computing $\beta \leftarrow S_{\mathrm{OT}}((x_0, x_1), \alpha_i)$ it computes the output of S_{OT} on either $(x_0, 0^{\ell(k)})$ or $(0^{\ell(k)}, x_1)$ depending upon the value of $\gamma(r)$. If $\gamma(r) = 0$, S_{OT} is given $((x_0, 0^{\ell(k)}), \alpha_i)$ as input; else, if $\gamma(r) = 1$, it is given $((0^{\ell(k)}, x_1), \alpha_i)$.

By construction, and by definition 3.1.1, it holds that the output of this distribution is statistically-close to that of $\mathcal{H}_r^1$, with distance $\Delta_{\gamma(r)}(k)$. Therefore, if $q_r'(k)$ denotes the probability that the event in equation (4.1.1) occurs, when we replace the string $(c^*, (d^*, \bar{q}, \bar{a})$ by the output of $\mathcal{H}_r^2$, then

$$q_r'(k) \leq q_r(k) + \Delta_{\gamma(r)}(k) \tag{4.1.3}$$

Next, we show that q_r' is at most $\mathsf{p}_{\mathsf{fail}}(k)$ as defined in definition 3.2.2. To see this, consider the following adversary for breaking the extractability property of the Captcha system $\mathcal{C}$.

Adversary $A_{\mathcal{C}}^H$. The adversary incorporates the entire hybrid experiment $\mathcal{H}_r^2$, including the values of r and $\gamma(r)$. The adversary receives a challenge puzzle z as input. The execution of the adversary, on input z, internally simulates $\mathcal{H}_r^2$ (including the honest sender S_{OT}), except for the following difference: if $\gamma(r) = 0$, it sets $x_0 = z$, and otherwise it sets $x_1 = z$. Note that $A_{\mathcal{C}}$ must use its oracle H to correctly simulate $\mathcal{H}_r^2$ since the hybrid includes a cheating committer $\widehat{\mathsf{C}}$ who requires access to H.

When the execution of H_2^r halts, let the contents of the tapes of the cheating committer $\widehat{\mathsf{C}}$, corresponding to the i^{th}-execution be $(d_i^*, \bar{q}, \bar{a})$. The decommitment information d_i^* includes a value s_i^*, supposedly a solution for $x_{\gamma(r)} = z$. The adversary outputs s_i^* as its solution to the challenge z.

First off, note that if $\mathcal{M} \neq \perp$, it holds that except with negligible probability s_i^* is a unique and well-defined value, which is indeed the solution of z. Next, observe that the internal execution of $A_{\mathcal{C}}$, on input z sampled honestly using G, is in fact identical to that of $\mathcal{H}_r^2$. Finally, observe that the queries made by A, are independent of the "other" CAPTCHA puzzle. That is, if $\gamma(r) = 0$, the queries made by A are independent of x_1 or any other puzzle. The case for $\gamma(r) = 1$ is symmetric.

Therefore, if the event in equation (4.1.1) occurs on outputs of $\mathcal{H}_2^r$ with probability $q_r'(k)$, then it must happen with same probability in internal execution of A on a random challenge puzzle. But occurrence of this event is equivalent to the failure of the CAPTCHA extractor as defined in definition 3.2.2. Therefore, it holds that $q_r'(k) \leq \mathrm{p}_{\mathsf{fail}}(k)$. By combining the results from all the hybrids in the sequence, we conclude that $p_i(k)$ is at most negligible. This completes the proof straight-line extraction. ■

4.2 Proof of UC Puzzle Construction

We now prove that $(\langle S^H, R\rangle, \mathcal{R})$, as defined in Section 3.4, is a UC puzzle, and begin with the soundness property. Recall that the soundness property requires that no malicious receiver can output the trapdoor s. In our case, the trapdoor is the pre-image (under the one-way permutation family $\{ f_k \}_{k\in\mathbb{N}}$) of the first message of the protocol, z. We will show that if there exists a malicious receiver that succeeds in outputting the trapdoor with non-negligible probability, then there exists an adversary that inverts the one-way permutation with some non-negligible probability. Informally, it will follow from the hiding property of PRSCom, and the zero-knowledge property of the proof in Phase 3, that the receiver does not learn s from Phases 2 and 3 of the protocol. Thus, if it succesfully outputs the trapdoor, it must do so by inverting the one-way permutation. As a technical point, note that when we are constructing the adversary for $f_k(\cdot)$, we are no longer inside the UC framework, and in particular, we are allowed to rewind the adversary. Details follow.

Lemma 4.2.1. *For any malicious receiver R^{*H}, the probability that it outputs s such that $\mathcal{R}(\mathsf{TRANS}, s) = 1$, where TRANS is an accepting transcript, is negligible in n.*

Proof. Let R^{*H} be a malicious receiver that outputs the correct pre-image for the first message of an accepting execution with probability ϵ. We construct an adversary $\mathcal{A}$ that inverts $f_k(\cdot)$ with probability negligibly close to ϵ. We construct $\mathcal{A}$ through a series of hybrid adversaries, wherein we maintain the invariant that each intermediate adversary outputs the pre-image with probability negligibly close to ϵ. As observed before, we are no longer in the UC framwork and are thus allowed to rewind the adversary.

Hybrid $\mathcal{A}_0$: Adversary $\mathcal{A}_0$ starts an internal execution of $\langle S^H, R^{*H}\rangle$ along with the environment. In particular, $\mathcal{A}_0$ starts by setting the random tapes of the environment and R^{*H}, and starts simulating the execution of $\langle S^H, R^{*H}\rangle$, conveying messages between the various parties. The adversary $\mathcal{A}_0$ simulates an honest S^H itself. In the end, $\mathcal{A}_0$ outputs whatever R^{*H} outputs. It is clear that this internal simulation is identical to the real execution, and $\mathcal{A}_0$ outputs the correct pre-image of the first message with probability ϵ.

Hybrid $\mathcal{A}_1$: Adversary $\mathcal{A}_1$ is the same as $\mathcal{A}_0$ except that in Phase 3, instead of giving the zero-knowledge proof from S^H to R^{*H}, it uses the simulator. More precisely, let $\mathsf{Sim}_{\mathsf{ZK}}$ be the simulator for the zero-knowledge proof in Phaes 3. Adversary $\mathcal{A}_1$ runs the execution of the puzzle similar to $\mathcal{A}_0$, except that it conveys all Phase 3 messages from R^{*H} to $\mathsf{Sim}_{\mathsf{ZK}}$, and all $\mathsf{Sim}_{\mathsf{ZK}}$ messages back to R^{*H}. If $\mathsf{Sim}_{\mathsf{ZK}}$ needs to rewind the verifier, then $\mathcal{A}_1$ rewinds the entire execution, including the environment and the honest sender. This is simply done be restarting the entire execution with same random tapes for the environment and R^{*H}, and using same messages from the sender till the point of rewind.

We claim that $\mathcal{A}_1$ outputs the trapdoor with probability negligibly close to ϵ: consider the following stand-alone malicious verifier $\bar{V}^H$ for the zero-knowledge proof used in Phase 3 - verifier $\bar{V}^H$ has access to S^H, R^{*H} and the environment, and starts an execution as in hybrid $\mathcal{A}_0$. The verifier $\bar{V}^H$ relays the the Phase 3 messages from the R^{*H} to the external prover, and sends the responses of the external prover back to R^{*H}. Finally, $\bar{V}^H$ outputs whatever R^{*H} outputs.

Note that the output of $\mathcal{A}_0$ is identical to the output of an interaction of $\bar{V}^H$ with a real ZK prover, while the output of $\mathcal{A}_1$ is identical to the output of a simulation. If the probabilities of outputting the trapdoor by $\mathcal{A}_0$ and $\mathcal{A}_1$ differ noticeably, then we can distinguish between the real interaction and a simulation of the ZK protocol, which contradicts the zero-knowledge property.

Hybrid $\mathcal{A}_2$: Adversary $\mathcal{A}_2$ is the same as $\mathcal{A}_1$ except in Phase 2, instead of committing to s, it commits to all-zeros string. The rest of the execution is the same.

we claim that the probabilities that $\mathcal{A}_1$ and $\mathcal{A}_2$ output the trapdoor are negligibly close. Consider the following malicious receiver $\bar{R}^H$ for PRSCom, that interacts with the external challenger. The challenger either commits to the all-zeros string, or to a random string, and $\bar{R}^H$ tries to distinguish between the two cases. Receiver $\bar{R}^H$ has access to S^H, R^{*H} and the environment, and starts an execution as in $\mathcal{A}_1$. It runs the PRSCom protocol with the challenger on one side, and conveys the messages to R^{*H} on the other side. In the end, it outputs whatever R^{*H} outuputs.

Observe that the output of $\mathcal{A}_1$ is identical to the output of the above game when the challenger commits to a random string, while the output of $\mathcal{A}_2$ is identical to the output of the above game when the challenger commits to the all-zeros string. Thus, if the probability of outputting the trapdoor differ noticeably, then we contradict the hiding property of PRSCom.

Our final adversary $\mathcal{A}$ receives $f_k(s)$ from the external challenger, for a randomly chosen s. It runs the same execution as $\mathcal{A}_2$, except that in Step 1, it sends $f_k(s)$ to R^{*H}. This execution is identical to the execution of $\mathcal{A}_2$, and thus $\mathcal{A}$ outputs the correct pre-image with probability negligibly close to ϵ. It follows from the hardness of $f_k(\cdot)$ that ϵ is negligible in n. ■

Now we consider the simulation and extraction property. It is easy to statistically simulate a malicious sender's view, as the simulator only has to play the honest receivers' parts. For extraction, we can extract from the ExtCom part of $\widehat{\mathsf{Com}}$.

Lemma 4.2.2. *Let S^{*H} be a malicious sender. Then there exists a simulator Sim^H that statistically simulates the view of S^{*H}. Further, the probability that S^{*H} sends an accepting* *transcript* TRANS *to the environment and* Sim^H *does not output the trapdoor* s *such that* $\mathcal{R}(\mathsf{TRANS}, s) = 1$*, is negligible in* n*.*

Proof. The simulator Sim^H, on input 1^k, starts an internal execution of the adversary S^{*H} on input 1^k. All messages that S^{*H} sends to receivers are handled by Sim^H by emulating honest receiver strategies. All communicatin between S^{*H} and the environment is relayed back and forth by Sim^H. As the simulator plays the honest receiver strategy, $\text{IDEAL}_{\mathsf{Sim}^H,\mathcal{Z}}(k)$ and $\text{REAL}^H_{\mathcal{A},\mathcal{Z}}(k)$ are identical.

We now show how Sim^H extracts the trapdoor. For each puzzle execution j, for each $(i, b) \in [k] \times \{ 0, 1 \}$, let $t^j_{i,b} := (\bar{q}, c, \mathsf{aux})^j_{i,b}$ be the queries, transcript and auxilliary information corresponding to the $(i, b)^{th}$ execution of of ExtCom in the j^{th} puzzle execution. Once a puzzle execution terminates, the simulator runs procedure $\mathsf{TrapExtr}$ (described below) on input $\{ t^j_{i,b} \}_{(i,b)\in[k]\times\{0,1\}}$. This procedure returns a string s^j, which Sim^H outputs as the trapdoor for the j^{th} puzzle execution. We call $\{ t^j_{i,b} \}_{(i,b)\in[k]\times\{0,1\}}$ the $\mathsf{extraction\ transcript}$ of the puzzle execution.

ALGORITHM $\mathsf{TrapExtr}(\{ t_{i,b} \}_{(i,b)\in[k]\times\{0,1\}})$

1. For each $(i, b) \in [k] \times \{ 0, 1 \}$, obtain $s^i_b \leftarrow \mathsf{ComExtr}(1^k, t_{i,b})$.
2. For each $i \in [k]$, set $s^i = s^i_0 \oplus s^i_1$. Let s be the most frequent string in the sequence $(s^1, \ldots, s^k)$. Output s.

We prove that $\mathsf{TrapExtr}$ returns the correct trapdoor with overwhelming probability. First, observe that soundness of the zero-knowledge proof in Phase 3 of the UC Puzzle immediately implies the following lemma,

Proposition 4.2.3. *The probability (over that random coins of the receiver) that the zero-knowledge proof in Phase 3 of puzzle execution is accepting and the partial transcript of that execution,* Ψ*, is not well-formed, is negligible in* k*.*

Next, we show that whenever that partial transcript is well-formed, the string returned by $\mathsf{TrapExtr}$ is the same as the string commited in partial transcript Ψ, with overwhelming probability.

Proposition 4.2.4. *Let Ψ be the partial transcript of a puzzle execution, and let s_Ψ be the string committed in Ψ. Further, let s be the string returned by $\mathsf{TrapExtr}$ when run on the extraction transcript of the puzzle execution. Then conditioned on the event that Ψ is well formed, the probability (over receiver's random coins) that $s_\Psi \neq s$ is negligible in k.*

Proof. For a particular execution of $\widehat{\mathsf{Com}}$, we say that the commitment is *invalid* if the strings committed in both the commit phases (that is, C_{PB} and ExtCom) are not the same. Note that if the number of invalid commitments is $\omega(\mathsf{log}(k))$, then with probability negligibly close to 1, the reciever will reject. Thus, given that Ψ is well-formed, more than half of $\widehat{\mathsf{Com}}$ exeuctions are valid. Therefore, the probability that $s_\Psi \neq s$ is negligible. ■

Finally we observe that it follows from the soundness of the zero-knowledge proof in Phase 3 of the puzzle that with all but negligible probability, $f_k(s_\Psi) = z$. The lemma follows from combining this with Propositions 4.2.3 and 4.2.4.

■

4.3 Brief Review of Protocol Composition

Protocol composition is a general term to describe how the security of various cryptographic protocols behave when they execute in a complex environment in which many other types of protocols are running at the same time. Roughly speaking, there are mainly three types of protocol compositions considered in the literature: self-composition, general-composition, and universal-composition. Barring some technical conditions, a sequence of results in the literature shows that for most "interesting" functionalities (except Zero Knowledge), all three notions are essentially (equivalent and) impossible to achieve [CF01, Lin03a, Lin03b, Lin04].

In section 3.4, we constructed UC puzzle. Below, we provide a brief review of UC framework, and how to incorporate the Captcha systems in this framework. This is followed by a brief discussion about the modeling, and concurrent self-composition. For a detailed exposition of these topics we refer the reader to the works of Canetti [Can01] and Lindell [Lin03b].

4.3.1 Universal Composition

The framework for universal composition considers the execution of a protocol π in a complex environment by introducing a special entity, called the *environment* $\mathcal{Z}$.

The environment drives the whole execution. The execution of π with the environment $\mathcal{Z}$, an adversary A, and a trusted party $\mathcal{G}$ proceeds as follows. To start an execution of π, $\mathcal{Z}$ initiates a *protocol execution session*, identified by session identifier *sid*, and activates all parties and assigns a unique identifier to each of them at invocation. An honest party, upon activation, starts executing π on inputs provided by $\mathcal{Z}$; adversarially controlled parties may deviate from the protocol arbitrarily. During the execution, $\mathcal{Z}$ is allowed to interact with $\mathcal{A}$ arbitrarily; in addition, it can see all the outputs of honest parties. We assume asynchronous authenticated communication over point-to-point channels; the scheduling of all messages is controlled by the adversary/environment. Some protocol executions may involve calls to "trusted parties" $\mathcal{G}$, who compute a specific functionality for the parties. Let k be the security parameter. We consider two types of executions.

Ideal Execution. Let $\mathcal{F}$ be a functionality (i.e., a trusted party); and let π_{ideal} be the "ideal protocol" which instructs its parties to call $\mathcal{F}$ with their private inputs. At the end of the computation, the parties then receive the output of the computation from $\mathcal{F}$. The *ideal model execution* of functionality $\mathcal{F}$ is then execution of protocol π_{ideal} with environment $\mathcal{Z}$, adversary $\mathcal{A}$, and trusted party $\mathcal{F}$. At the end of the execution, $\mathcal{Z}$ outputs a bit, denoted by the random variable $\textsc{ideal}^{\mathcal{F}}_{\pi_{\mathsf{ideal}},\mathcal{A},\mathcal{Z}}(k)$.

Real Execution. Let π be a multiparty protocol implementing $\mathcal{F}$. The *real model execution* of π, is the execution of π with $\mathcal{Z}$, and $\mathcal{A}$. Note that there are no calls to $\mathcal{F}$ in this execution. At the end of the execution, $\mathcal{Z}$ outputs a bit, denoted by the random variable $\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}}(k)$.

Informally speaking, we say that π UC-realizes $\mathcal{F}$, if π is a *secure emulation* of the protocol π_{ideal}. This is formulated by saying that for every adversary $\mathcal{A}$ participating in the real model execution of π, there exists an adversary $\mathcal{A}$', called the simulator, which participates in the ideal model execution of $\mathcal{F}$ such that no environment $\mathcal{Z}$ can tell apart whether it is interacting with $\mathcal{A}$ or $\mathcal{A}$'. That is, variables $\text{IDEAL}^{\mathcal{F}}_{\pi_{\mathsf{ideal}},\mathcal{A}',\mathcal{Z}}(k)$ and $\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}}(k)$ are computationally indistinguishable.

Modeling Access to H. Let $\mathcal{C} := (G, H)$ be an extractable CAPTCHA puzzle system. To incorporate the use of $\mathcal{C}$ in the UC framework, we provide all entities—i.e., the honest parties, the adversary $\mathcal{A}$, and the environment $\mathcal{Z}$—access to the solution function H.[1] Note that providing access to H in the UC framework is not a new formulation; it has been considered before by Canetti, Halevi, and Steiner [CHS06], who construct a UC-secure password-based key-generation protocol in this model. We follow the same approach and allow the honest parties and the adversary $\mathcal{A}$ to directly access the solution function H.

However, we observe that there are two *different* ways in which we the environment $\mathcal{Z}$ can access H. This is a crucial point and the difference between what is possible and what is not.

- INDIRECT ACCESS: The work of Canetti et. al. [CHS06], does *not* provide $\mathcal{Z}$ direct access to H. Instead, in their framework, all queries of $\mathcal{Z}$ to H are first sent to $\mathcal{A}$, who queries H and obtains the answers. These answers are then forwarded to $\mathcal{Z}$. We call this, the *indirect access model.*

[1]In analogy with the "trusted set up" models such as the CRS model, we assume that all parties are using the same CAPTCHA puzzle system $\mathcal{C}$. However, we insist that this is *not* essential to obtain our results. If there are multiple types of CAPTCHA puzzles $\mathcal{C}_1, \ldots, \mathcal{C}_{\text{poly}(k)}$ in use, then all we really need is that the simulator can access the queries made by cheating parties to the corresponding solution functions, say $H_1, \ldots, H_{\text{poly}(k)}$.

- Direct Access: In the *direct access model*, $\mathcal{Z}$ is given direct access to H. In particular, $\mathcal{A}$ cannot see the queries sent by $\mathcal{Z}$ to H.

In section 3.4, we prove that in the indirect access model, under the assumption that extractable Captcha puzzles and one-way permutations exist, for *every* PPT functionality $\mathcal{F}$, there exists a protocol π that UC-realizes $\mathcal{F}$.

On the other hand, in the direct access model, clearly there is no advantage that a simulator (acting in the ideal world for adversary $\mathcal{A}$) will have compared to the classical UC-framework. This is because since $\mathcal{A}$ cannot see the queries of $\mathcal{Z}$, the simulator will also not be able to do so. Therefore, existing impossibility results for the UC-framework should also hold in the direct-access model. This is indeed quite trivial to show—for example by reproducing the proof of Canetti-Fischlin [CF01] for commitment schemes (see Appendix 4.5.1).

Discussion. An interesting question to consider is which of these two models is the "right" model. Let us first compare the indirect access model to the other "trusted setup" models such as the CRS-model. In the CRS-model, the simulator S is in control of generating the CRS in the ideal world—this enables S to have a "trapdoor" to continue its actions without having to "rewind" the environment. All parties, *including* the environment $\mathcal{Z}$ use the (same) CRS generated S. Viewed this way, the indirect access model can be seen as some sort of a setup (i.e., the oracle H) where all parties, including $\mathcal{Z}$, use the same setup. However, the indirect access model is better than the CRS model (or other "trusted setup" models) in the sense that there is *no trust involved*. That is, in the indirect access model, there is no party who is trusted to generate the setup according to some specific settings, e.g., a random string in case of the CRS-model.

However, since the environment *must* access H through the adversary (enforcing, in some sense, the same setup condition), the indirect access model does not retain the true spirit of the *plain* or the vanilla model (where there is no setup to begin with). Intuitively, the Captcha model does not have any setup since every party is going to have its own "human" helping it to solve the Captcha puzzles: e,.g., our commitment scheme in section 3.3, is a scheme in the *plain* model. However, intuitively, in the plain model (irrespective of access no H), UC-security should ideally imply self-composition. The fact that the positive results in the indirect access model, do *not* carry over to the setting of self-composition, show that the direct access model is more natural and retains the true spirit of the plain model.

4.3.2 Concurrent Self Composition

Concurrent self composition, refers to the situation where many instances of a *single* protocol π are executed *concurrently* many times on the network. The concurrent attack model has a specific meaning in which the adversary is allowed to control the schedule and delivery of various protocol messages. The adversary can corrupt parties participating in various execution of π, either adaptively (i.e., in the middle of the execution) or statically (before any of the protocol executions begin).

In a series of results, Lindell [Lin03a, Lin03b, Lin04] proves a general theorem which, informally speaking, shows that for the so called "bi-directional bit-transmitting functionalities", security in the concurrent self-composition model implies security in the universal-composition model. It is not hard to see that his proof in fact holds in our setting (where access to H is granted to all parties) as well with respect to the *direct access model* (i.e. where environment accesses H directly and not through the adversary). Therefore, we obtain similar impossibility results for concurrent self-composition.

While the class of bi-directional bit-transmitting functionalities includes almost all interesting functionalities, zero-knowledge functionality does not fall in this class. Indeed, it is possible to have concurrent self-composition for zero-knowledge. By modifying Blum-Hamiltonicity protocol so that verifier's challenge is decided by using a coin-tossing phase (in which verifier first commits to its challenge using our extractable commitment scheme from figure 3.1), we can obtain constant-round and straight-line concurrent zero-knowledge for **NP**. Likewise, by replacing the initial PRS-phase by our extractable commitment scheme, the protocol of Barak, Prabhakaran, and Sahai [BPS06] yields a concurrent non-malleable zero-knowledge protocol which is constant-round with straight-line simulation. For completeness, the resulting protocol is given in appendix 4.4.

One might wonder, if we can get straight-line simulation in concurrent NMZK, why are we not able to obtain UC-Zero-knowledge in the direct access model. The reason is that our protocol (in appendix 4.4) can only guarantee straight-line simulation, *but not straight-line extraction* of the witness from man-in-the-middle. Also, there cannot be any method to convert this protocol so that one gets *both* simulation and extraction in straight-line since such a construction will imply UC-ZK which in turn will imply UC-security for computing all PPT functionalities in the direct access model [CLO02], which is impossible.

4.4 Concurrent NMZK: Constant Round, Straight-line

We assume basic familiarity with zero-knowledge protocols and their execution in a concurrent non-malleable experiment. Briefly, in such an experiment a man-in-the-middle A interacts with many provers $P_1, \ldots, P_m$ on "left" side, and with many verifiers $V_1, \ldots, V_m$ or "right" side. Interaction of A with P_i is called the i^{th}-left-session; likewise, A's interaction with V_i is called i^{th}-right-session. The statements being proven to A by the provers are chosen before the execution; the statements on right that A proves to the verifiers are chosen adaptively by A as the interaction proceeds. A controls the scheduling and delivery of all messages in this experiment. Without loss of generality, we can assume that A is a deterministic polynomial time machine with $z \in \{0,1\}^*$ as its auxiliary input. Concurrent Non-Malleable Zero-Knowledge is defined as follows ([BPS06, PR05]):

Definition 4.4.1 (Concurrent Non-Malleable Zero-Knowledge)**.** *A protocol is a Concurrent Non-Malleable Zero Knowledge (CNMZK) argument of knowledge for membership in an NP language L with witness relation R (that is, $y \in L$ iff there exists w such that $R(y, w) = 1$), if it is an interactive proof system between a prover and a verifier such that*

Completeness: *if both the prover and the verifier are honest, then for every (y, w) such that $R(y, w) = 1$, the verifier will accept the proof, and*

Soundness, Zero-Knowledge and Non-Malleability: *for every (non-uniform PPT)* *adversary A interacting with provers $P_1, \ldots, P_{m_L}$ in m_L "left sessions" and verifiers $V_1, \ldots, V_{m_R}$ in m_R "right sessions" of the protocol (with A controlling the scheduling of all the sessions), there exists a simulator S such that for every set of "left inputs" $y_1, \ldots, y_{m_L}$, we have $S(y_1, \ldots, y_{m_L}) = (\nu, z_1, \ldots, z_{m_R})$, such that*

1. *ν is a simulated view of A: i.e., ν is distributed indistinguishably from the view of A (for any set of witnesses $(w_1, \ldots, w_{m_L})$ that $P_1, \ldots, P_{m_L}$ are provided with).*
2. *For all $i \in \{1, \ldots, m_R\}$, if in the i^{th} right hand side session in ν the common input is x_i and the verifier V_i accepts the proof, then z_i is a valid witness to the membership of x_i in the language, except with negligible probability ($z_i =\perp$ if V_i does not accept.)*

Further, we call the protocol a black-box CNMZK if there exists a universal simulator S_{BB} such that for any adversary A, it is the case that $S = S_{BB}^A$ satisfies the above requirements.

To obtain concurrent non-malleable zero-knowledge (CNMZK), we use the protocol of Barak-Prabhakaran-Sahai [BPS06]. Briefly, the protocol has five phases, of which the first phase is the the PRS-preamble [PRS02]. We obtain our protocol by using our extractable commitment scheme from section 3.3 in phase I, instead of the PRS preamble. The protocol uses following ingredients, all of which can be constructed from standard number theoretic assumptions (or even general assumptions such as Claw Free permutations [GK96]): a two-round statistically-hiding commitment scheme denoted $\mathsf{Com}_{\mathsf{SH}}$ [GK96], a (constant round) statistical zero-knowledge argument-of-knowledge for **NP**, denoted SZKAOK [Blu87, GK96], and a (constant round) non-malleable commitment scheme denoted $\mathsf{Com}_{\mathsf{NM}}$ [Goy11, LP11, PR05, PPV08]. The resulting protocol is depicted in figure 4.1.

To prove the security of their protocol BPS only require the following two properties form the PRS-preamble: computational-hiding of the PRS-challenge, and extraction of each session-wise PRS-challenge (by means of rewinding) as soon as the PRS-phase ends. Since both of these properties are also satisfied by our extractable commitment scheme, we do not need to change the proof of BPS. In addition, since there are no rewindings involved

Common Input: $x \in L$.

Prover's Auxiliary Input: $y \in R_L(x)$.

Phase I: $(V \leftrightarrow P)$ The verifier chooses a random string ρ and commits to it using the commitment scheme $\mathrm{Com}_{\mathcal{C}} := \langle \mathsf{C}^H, \mathsf{R} \rangle$ from section 3.3.

Phase II: $(P \leftrightarrow V)$ P commits to the all-zero string using $\mathsf{Com_{SH}}$. Then it uses SZKAOK to prove knowledge of the randomness and inputs to this execution of $\mathsf{Com_{SH}}$.

Phase III: $(V \leftrightarrow P)$ Execute the Opening Phase of $\mathrm{Com}_{\mathcal{C}}$. Let the committed string (as revealed by the verifier) be σ.

Phase IV: $(P \leftrightarrow V)$ P commits to the witness y using $\mathsf{Com_{NM}}$.

Phase V: $(P \leftrightarrow V)$ P proves the following statement using SZKAOK: either the value committed to in Phase IV is y such that $y \in R_L(x)$, or the value committed to in Phase II is σ. P uses the witness corresponding to the first part of the statement.

Figure 4.1: Straight-Line Concurrent Non-Malleable Zero-Knowledge $(\mathbf{P}, \mathbf{V})$.

during simulation (to extract the PRS-challenge), the proof in fact gets simpler. Note that the extraction of witness is performed from Phase IV which still uses rewindings. The details are omitted.

4.5 Negative Results

4.5.1 Universal Composition in the Direct Access Model

Canetti and Fischlin [CF01] prove that universally composable commitments are impossible to construct in the plain model. We reproduce here the details of their result in our framework where we assume that there exists CAPTCHA puzzles (as in Definition 3.2.1) and the environment has direct access to a CAPTCHA solving oracle H. This implies that the environment's queries cannot be seen by the ideal world adversary. For this, we just focus on the the one-time commitment functionality, and prove that it is impossible to UC-realize it in direct access model. This functionality is shown in figure 4.2.

Functionality $\mathcal{F}_{\mathbf{com}}$

$\mathcal{F}_{\text{COM}}$ acts in the presence of parties $P_1^H \cdots P_n^H$ and an ideal model adversary $\mathcal{S}^H$

1. Upon receiving the message $(\texttt{Commit}, sid, P_i, P_j, b)$, from the party P_i, $b \in \{0,1\}$ record the value b and send the message $(\texttt{Receipt}, sid, P_i, P_j)$ to $\mathcal{S}$ and P_j, who writes it to his output tape. Ignore any subsequent $(\texttt{Commit}, \cdots)$ messages

2. Upon receiving the message $(\texttt{Open}, sid, P_i, P_j)$ from P_i, if some bit b was previously recorded, send $(\texttt{Open}, sid, P_i, P_j, b)$ to $\mathcal{S}$ and P_j (who writes it to his output tape) and halt. Else do nothing. Halt.

Figure 4.2: A one-time commitment functionality [CF01] $\mathcal{F}_{\text{COM}}$

In any commitment protocol which attempts to implement the functionality Figure 4.2, denote by C the committer P_i and by R the receiver P_j.

Definition 4.5.1. ***(Terminating Commitment Protocol)*** *A commitment protocol* $\pi = \langle \mathsf{C}^H, \mathsf{R}^H \rangle$ *is called* terminating *if there is a non-negligible probability that* R^H *outputs* $(\texttt{Receipt}, \cdots)$ *and moreover if the receiver, upon getting a valid decommitment for a session id sid and a committed bit b from the sender, outputs* $(\texttt{Open}, sid, C, R, b)$ *with non-negligible probability*

We say that a protocol is *bilateral* if only two parties participate in the protocol and all other parties are idle and do not send or receive messages.

Theorem 4.5.2. *There exists no bilateral terminating commitment protocol π that UC securely realizes the functionality $\mathcal{F}_{\text{COM}}$ in the direct access model of* CAPTCHA *puzzles. This holds even if the ideal-model adversary $\mathcal{S}^H$ is allowed to depend on the environment $\mathcal{Z}^H$.*

Our impossibility result is a restatement of Canetti-Fischlin impossibility [CF01] in the plain model. We present it here for completeness, but the details are precisely as is from [CF01]. It proceeds as follows.

Recall that all turing machines described below are oracle turing machines. Assuming that a Captcha puzzle system (Definition 3.2.1) exists, and modelling it in the UC framework via the direct access model, all turing machines described below have access to the solution oracle H. Informally, let π be any protocol attempting to realize the functionality $\mathcal{F}_{\text{COM}}$ securely. We would like to construct an adversary $\mathcal{A}_2$ and environment $\mathcal{Z}_2$ such that no ideal world adversary exists satisfying the definition of UC security. To this goal, let $\mathcal{Z}_1$ and $\mathcal{A}_1$ be the environment and adversary who do the following. The adversary $\mathcal{A}_1$ corrupts the committer C and forwards all messages that he receives from the environment on behalf of the committer. The environment chooses a bit b at random and executes the honest protocol with b as input. Then $\mathcal{Z}_1$ advises the corrupt committer to start the decommitment phase, and once again lets the adversary $\mathcal{A}_1$ forward messages generated by $\mathcal{Z}_1$ on behalf of the committer. When the receiver R outputs a bit b', the environment outputs 1, iff b equals b'. Observe the following about the adversary $\mathcal{A}_1$ and π,

1. $\mathcal{A}_1$ sees nothing but the messages of the protocol. In particular, $\mathcal{A}_1$ does not make any oracle queries itself.

2. By definition of UC security, there exists an ideal model adversary $\mathcal{S}_1$ such that $\mathcal{S}_1$ sends a $(\texttt{Commit}, sid, C, R, b')$ message to the functionality $\mathcal{F}_{\text{COM}}$. Moreover, by the indistinguishability property of UC security, $\Pr[b = b']$ differs only negligibly between the real and ideal executions.

3. By the definition of a terminating commitment protocol, $\Pr[b = b']$ is non-negligible.

Formally, Let $\pi = (\mathsf{C}^H, \mathsf{R}^H)$ be any terminating protocol that UC-securely implements the functionality $\mathcal{F}_{\text{COM}}$. Let $\mathcal{A}_1^H$ and $\mathcal{Z}_1^H$ be the adversary and environment as above. Let $\mathcal{S}_1^H$ be the ideal model advesary (possibly depending on $\mathcal{Z}_1^H$) whose existence is guaranteed by the definition of UC-security. Note that due to us assuming the direct access model for Captcha,

- the environment can execute a honest committer's protocol (as it can access the human oracle to solve the CAPTCHA puzzles in $\mathsf{ComExtr}$).
- the view of the ideal model adversary $\mathcal{S}_1^H$ and that of the real adversary $\mathcal{A}_1^H$ both consist of only the messages of the protocol, and in particular, neither of these machines make any oracle queries.

To prove a contradiction, consider the following environment and adversary $\mathcal{Z}_2^H$ and $\mathcal{A}_2^H$. The environment $\mathcal{Z}_2^H$ instructs the committer to pick a (secret) bit b randomly and commit to it using an honest execution of the protocol π. The adversary $\mathcal{A}_2^H$, corrupts the receiver and then obtains all the messages sent to the now corrupted receiver and forwards it to an internal copy of the simulator $\mathcal{S}_1^H$ from the previous experiment. When $\mathcal{S}_1^H$ sends a $(\texttt{Commit}, \cdots, b')$ message to $\mathcal{F}_{\text{COM}}$, the adversary forwards the bit b' to the environment. The environment outputs 1 if $b = b'$ and 0 otherwise.

The contradictions follow from the following observation. In the real world experiment of the above protocol, using $\mathcal{S}_1^H$ internally, adversary $\mathcal{A}_2$ obtains an advantage in guess the bit b. However in the ideal world execution of the protocol, since a decommitment is not being done, the ideal world adversary, $\mathcal{S}_2$ (even if it depends on $\mathcal{Z}_2$) has no advantage over $\frac{1}{2}$ in guessing the bit b. Thus for every ideal world adversary $\mathcal{S}_2$ the environment is capable of distinguishing the real world from the ideal world, contradicting the definition of UC-security.

4.5.2 Concurrent Self-composition of general functionalities

As noted earlier, much like the Canetti-Fischlin impossibility result, it is possible to repeat the proof of Lindell [Lin08] step-by-step with intuitive changes to obtain a negative result for concurrent self-composition of general functionalities when modelling CAPTCHA puzzles via the direct access model. Here we briefly discuss the steps involved in Lindell's result. We recommend that the reader familiarizes himself with the proof of [Lin08].

At a very high level, the first step in Lindell's result is to show that concurrent self-composition implies concurrent general-composition. The proof outline for this step is as follows. For any bi-directional bit-transmitting functionality $\mathcal{F}$, if there exists a secure self-composing protocol π which implements $\mathcal{F}$, it can be used concurrently many times to simulate the execution of π concurrently composed with any general protocol $\mathcal{G}$ by transmitting the messages of $\mathcal{G}$ bit-by-bit. This step remains completely unaltered and goes through in our setting as well; this is because in our model, the only difference is that machines are equipped with with oracle access to H. This is only a "cosmetic" change which does not affect simulation of $\mathcal{G}$ by using π concurrently many times.

The next step (to complete negative result of Lindell), shows that general composition implies universal composition for the case of so called *specialized simulator UC* [Lin03b]. Once again, since the only modification in our model is to equip machines with access to H, this step also goes through without any change. Finally, the result of Canetti-Fischlin (reproduced in theorem 4.5.2 for our model) proves that specialized-simulator UC is not possible for the commitment functionality. This is essentially the entire outline of our impossibility claim for concurrent self-composition of bi-directional bit-transmitting functionalities. A complete proof can be obtained by a step-by-step reproduction of Lindell's results.

References

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach.* Cambridge University Press, 2009.

[ABH03] Luis Von Ahn, Manuel Blum, Nicholas J. Hopper, and John Langford. "Captcha: Using Hard AI Problems for Security." In *EUROCRYPT*, pp. 294–311, 2003.

[BBS04a] Dan Boneh, Xavier Boyen, and Hovav Shacham. "Short group signatures." In *Proceedings of CRYPTO .04, LNCS series*, pp. 41–55. Springer-Verlag, 2004.

[BBS04b] Dan Boneh, Xavier Boyen, and Hovav Shacham. "Short Group Signatures." In Matthew K. Franklin, editor, *CRYPTO*, volume 3152 of *Lecture Notes in Computer Science*, pp. 41–55. Springer, 2004.

[BF99] Dan Boneh and Matthew K. Franklin. "An Efficient Public Key Traitor Tracing Scheme." In *CRYPTO '99: Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology*, pp. 338–353, London, UK, 1999. Springer-Verlag.

[BGN05] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. "Evaluating 2-DNF Formulas on Ciphertexts." In *Second Theory of Cryptography Conference, TCC*, volume 3378 of *LNCS*, pp. 325–341, 2005.

[BGW05] Dan Boneh, Craig Gentry, and Brent Waters. "Collusion Resistant Broadcast Encryption with Short Ciphertexts and Private Keys." In *CRYPTO*, pp. 258–275, 2005.

[BLS01] Dan Boneh, Ben Lynn, and Hovav Shacham. "Short Signatures from the Weil Pairing." In *ASIACRYPT '01: Proceedings of the 7th International Conference on the Theory and Application of Cryptology and Information Security*, pp. 514–532, London, UK, 2001. Springer-Verlag.

[Blu87] Manual Blum. "How to prove a theorem so no one else can claim it." In *International Congress of Mathematicians*, pp. 1444–1451, 1987.

[BN08] Dan Boneh and Moni Naor. "Traitor tracing with constant size ciphertext." In *ACM Conference on Computer and Communications Security*, pp. 501–510, 2008.

[BP04] Mihir Bellare and Adriana Palacio. "The Knowledge-of-Exponent Assumptions and 3-Round Zero-Knowledge Protocols." In *CRYPTO*, pp. 273–289, 2004.

[BPS06] Boaz Barak, Manoj Prabhakaran, and Amit Sahai. "Concurrent Non-Malleable Zero Knowledge." In *FOCS*, pp. 345–354. IEEE Computer Society, 2006.

[BR94] Mihir Bellare and Phillip Rogaway. "Optimal Asymmetric Encryption." In *EUROCRYPT*, pp. 92–111, 1994.

[BSM91] Manuel Blum, Alfredo De Santis, Silvio Micali, and Giuseppe Persiano. "Noninteractive Zero-Knowledge." *SIAM J. Comput.*, **20**(6):1084–1118, 1991.

[BSW06] Dan Boneh, Amit Sahai, and Brent Waters. "Fully collusion resistant traitor tracing with short ciphertexts and private keys." In *EUROCRYPT 2006, volume 4004 of LNCS*, pp. 573–592. Springer-Verlag, 2006.

[BW06] Dan Boneh and Brent Waters. "A fully collusion resistant broadcast, trace, and revoke system." In *CCS '06: Proceedings of the 13th ACM conference on Computer and communications security*, pp. 211–220, New York, NY, USA, 2006. ACM.

[Can01] Ran Canetti. "Universally Composable Security: A New Paradigm for Cryptographic Protocols." In *FOCS*, pp. 136–145, 2001.

[CAP] The Official CAPTCHA Site. "www.captcha.net.".

[CF01] Ran Canetti and Marc Fischlin. "Universally Composable Commitments." In *CRYPTO*, pp. 19–40, 2001.

[CFN94] Benny Chor, Amos Fiat, and Moni Naor. "Tracing Traitors." In *CRYPTO '94: Proceedings of the 14th Annual International Cryptology Conference on Advances in Cryptology*, pp. 257–270, London, UK, 1994. Springer-Verlag.

[CHS04] Ran Canetti, Shai Halevi, and Michael Steiner. "Hardness Amplification of Weakly Verifiable Puzzles." In *TCC*, pp. 17–33. Springer-Verlag, 2004.

[CHS06] Ran Canetti, Shai Halevi, and Michael Steiner. "Mitigating Dictionary Attacks on Password-Protected Local Storage." In *ADVANCES IN CRYPTOLOGY, CRYPTO*. Springer-Verlag, 2006.

[CIO98] Giovanni Di Crescenzo, Yuval Ishai, and Rafail Ostrovsky. "Non-Interactive and Non-Malleable Commitment." In *STOC*, pp. 141–150, 1998.

[CKO01] Giovanni Di Crescenzo, Jonathan Katz, Rafail Ostrovsky, and Adam Smith. "Efficient and Non-interactive Non-malleable Commitment." In *EUROCRYPT*, pp. 40–59, 2001.

[CLO02] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. "Universally composable two-party and multi-party secure computation." In *STOC*, pp. 494–503, 2002.

[CPP05] Hervé Chabanne, Duong Hieu Phan, and David Pointcheval. "Public Traceability in Traitor Tracing Schemes." In *EUROCRYPT*, pp. 542–558, 2005.

[DC12] Sandra Diaz-Santiago and Debrup Chakraborty. "On Securing Communication from Profilers." In *SECRYPT*, pp. 154–162, 2012.

[DDN00] Danny Dolev, Cynthia Dwork, and Moni Naor. "Non-Malleable Cryptography." *SIAM J. on Computing*, **30**(2):391–437, 2000.

[DF03] Yevgeniy Dodis and Nelly Fazio. "Public Key Trace and Revoke Scheme Secure against Adaptive Chosen Ciphertext Attack." In *Public Key Cryptography*, pp. 100–115, 2003.

[DH76] Whitfield Diffie and Martin E. Hellman. "New Directions in Cryptography." *IEEE Transactions on Information Theory*, November 1976.

[DNS98] Cynthia Dwork, Moni Naor, and Amit Sahai. "Concurrent Zero-Knowledge." In *STOC*, pp. 409–418, 1998.

[Dzi10] Stefan Dziembowski. "How to Pair with a Human." In *SCN*, pp. 200–218, 2010.

[Fre09] David Mandell Freeman. "Converting pairing-based cryptosystems from composite-order groups to prime-order groups." In *Preprint*, 2009.

[GJO10] Vipul Goyal, Abhishek Jain, and Rafail Ostrovsky. "Password-Authenticated Session-Key Generation on the Internet in the Plain Model." In *CRYPTO*, pp. 277–294, 2010.

[GK96] Oded Goldreich and Ariel Kahan. "How to Construct Constant-Round Zero-Knowledge Proof Systems for NP." *J. Cryptology*, **9**(3):167–190, 1996.

[GKS09] Sanjam Garg, Abishek Kumarasubramanian, Amit Sahai, and Brent Waters. "Building Efficient Fully Collusion-Resilient Traitor Tracing and Revocation Schemes." Cryptology ePrint Archive, Report 2009/532, 2009. http://eprint.iacr.org/.

[GMW86] Oded Goldreich, Silvio Micali, and Avi Wigderson. "Proofs that Yield Nothing But their Validity and a Methodology of Cryptographic Protocol Design (Extended Abstract)." In *FOCS*, pp. 174–187, 1986.

[Gol01] Oded Goldreich. *Foundations of Cryptography: Basic Tools*. Cambridge University Press, 2001.

[Goy11] Vipul Goyal. "Constant round non-malleable protocols using one way functions." In *STOC*, pp. 695–704, 2011.

[GW09] Craig Gentry and Brent Waters. "Adaptive Security in Broadcast Encryption Systems (with Short Ciphertexts)." In *EUROCRYPT*, pp. 171–188, 2009.

[HK10] Shai Halevi and Yael Kalai. "Smooth Projective Hashing and Two-Message Oblivious Transfer." *Journal of Cryptology*, pp. 1–36, 2010. 10.1007/s00145-010-9092-8.

[HO09] Brett Hemenway and Rafail Ostrovsky. "Lossy Trapdoor Functions from Smooth Homomorphic Hash Proof Systems." *Electronic Colloquium on Computational Complexity (ECCC)*, **16**:127, 2009.

[HT98] Satoshi Hada and Toshiaki Tanaka. "On the Existence of 3-Round Zero-Knowledge Protocols." In *CRYPTO*, pp. 408–423, 1998.

[KD98] Kaoru Kurosawa and Yvo Desmedt. "Optimum Traitor Tracing and Asymmetric Schemes." In *EUROCRYPT*, pp. 145–157, 1998.

[KY02a] Aggelos Kiayias and Moti Yung. "Breaking and Repairing Asymmetric Public-Key Traitor Tracing." In *Digital Rights Management Workshop*, pp. 32–50, 2002.

[KY02b] Aggelos Kiayias and Moti Yung. "Traitor Tracing with Constant Transmission Rate." In *EUROCRYPT*, pp. 450–465, 2002.

[Lin03a] Yehuda Lindell. "Bounded-concurrent secure two-party computation without setup assumptions." In *STOC*, pp. 683–692, 2003.

[Lin03b] Yehuda Lindell. "General Composition and Universal Composability in Secure Multi-Party Computation." In *In 44th FOCS*, pp. 394–403, 2003.

[Lin04] Yehuda Lindell. "Lower Bounds for Concurrent Self Composition." In Moni Naor, editor, *Theory of Cryptography*, volume 2951 of *Lecture Notes in Computer Science*, pp. 203–222. Springer Berlin / Heidelberg, 2004.

[Lin08] Yehuda Lindell. "Lower Bounds and Impossibility Results for Concurrent Self Composition." *Journal of Cryptology*, **21**:200–249, 2008. 10.1007/s00145-007-9015-5.

[LP11] Huijia Lin and Rafael Pass. "Constant-round non-malleable commitments from any one-way function." In *STOC*, pp. 705–714, 2011.

[LPV09] Huijia Lin, Rafael Pass, and Muthuramakrishnan Venkitasubramaniam. "A unified framework for concurrent security: universal composability from stand-alone non-malleability." In *STOC*, pp. 179–188, 2009.

[Lyn] Ben Lynn. "The Pairing-Based Cryptography Library.".

[MNT00] Atsuko Miyaji, Masaki Nakabayashi, and Shunzo Takano. "Characterization of Elliptic Curve Traces under FR-Reduction." In *ICISC*, pp. 90–108, 2000.

[MP06] Silvio Micali and Rafael Pass. "Local zero knowledge." In *STOC*, pp. 306–315, 2006.

[MSK02] Shigeo Mitsunari, Ryuichi Sakai, and Masao Kasahara. "A new traitor tracing." In *IEICE Trans. Fundamentals*, pp. E85–A(2):481484, 2002.

[Nao03] Moni Naor. "On Cryptographic Assumptions and Challenges." In *CRYPTO*, pp. 96–109, 2003.

[NI] N.I.S.T. "N.I.S.T Cryptography Toolkit.".

[NP00] Moni Naor and Benny Pinkas. "Efficient Trace and Revoke Schemes." In *Financial Cryptography*, pp. 1–20, 2000.

[Pas04] Rafael Pass. "Bounded-concurrent secure multi-party computation with a dishonest majority." In *STOC*, pp. 232–241, 2004.

[Pfi96] Birgit Pfitzmann. "Trials of Traced Traitors." In *Information Hiding*, pp. 49–64, 1996.

[PPS08] Omkant Pandey, Rafael Pass, Amit Sahai, Wei-Lung Dustin Tseng, and Muthuramakrishnan Venkitasubramaniam. "Precise Concurrent Zero Knowledge." In *EUROCRYPT*, pp. 397–414, 2008.

[PPV08] Omkant Pandey, Rafael Pass, and Vinod Vaikuntanathan. "Adaptive One-Way Functions and Applications." In *CRYPTO*, pp. 57–74, 2008.

[PR03] Rafael Pass and Alon Rosen. "Bounded-Concurrent Secure Two-Party Computation in a Constant Number of Rounds." In *FOCS*, 2003.

[PR05] Rafael Pass and Alon Rosen. "New and improved constructions of non-malleable cryptographic protocols." In *STOC*, pp. 533–542, 2005.

[PRS02] Manoj Prabhakaran, Alon Rosen, and Amit Sahai. "Concurrent Zero Knowledge with Logarithmic Round-Complexity." In *FOCS*, pp. 366–375, 2002.

[PW97] Birgit Pfitzmann and Michael Waidner. "Asymmetric Fingerprinting for Larger Collusions." In *ACM Conference on Computer and Communications Security*, pp. 151–160, 1997.

[Ros04] Alon Rosen. "A Note on Constant-Round Zero-Knowledge Proofs for NP." In *TCC*, pp. 191–202, 2004.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems." *Commun. ACM*, **21**(2):120–126, 1978.

[Sha48] Claude Shannon. "A Mathematical Theory of Communication." *Bell System Technical Journal*, **27**:379–423, 623–656, July, October 1948.

[TSZ03] V. D. Tô, R. Safavi-Naini, and F. Zhang. "New traitor tracing schemes using bilinear map." In *DRM '03: Proceedings of the 3rd ACM workshop on Digital rights management*, pp. 67–76, New York, NY, USA, 2003. ACM.

[Wat09] Brent Waters. "Dual System Encryption: Realizing Fully Secure IBE and HIBE under Simple Assumptions." In *CRYPTO*, pp. 619–636, 2009.

[WHI01] Yuji Watanabe, Goichiro Hanaoka, and Hideki Imai. "Efficient Asymmetric Public-Key Traitor Tracing without Trusted Agents." In *CT-RSA*, pp. 392–407, 2001.

[Wik] Wikipedia. "History of Cryptography.".

[Yao82] Andrew Chi-Chih Yao. "Protocols for Secure Computations (Extended Abstract)." In *FOCS*, pp. 160–164. IEEE Computer Society, 1982.