Title
Reducing Trust When Trust Is Essential

Permalink
https://escholarship.org/uc/item/7301296m

Author
Seyalioglu, Hakan Ali-John

Publication Date
2012

Peer reviewed|Thesis/dissertation
Reducing Trust
When Trust Is Essential

A dissertation submitted in partial satisfaction
of the requirements for the degree
Doctor of Philosophy in Mathematics

by

Hakan Ali-John Seyalioglu

2012
Trust is unavoidable in many complex systems. As users demand more functionality and convenient access, there is often a point at which a systems designer has no choice but to ask clients to place some level of trust in one participant in the system, be it a key management server or even just the integrity of some external hardware. This is in contrast to many cryptographic functionalities that will either perform as expected, or at least remain secure, without a client having to trust anything but their own integrity. The main focus of this thesis is to conduct a thorough analysis of two settings in which trust is essential and find new dimensions in which to reduce the amount of trust the participants are asked to put in the system for it to function as expected. First, we analyze remote storage where a third party is trusted to store and do access control on a client’s data. Second, we study what additional assumptions are necessary in order to circumvent impossibility of fully secure MPC without an honest majority.
The dissertation of Hakan Ali-John Seyalioglu is approved.

Don Blasius

Sebastien Roch

Amit Sahai

Rafail Ostrovsky, Committee Chair

University of California, Los Angeles
2012
For my family and those I love,
# Table of Contents

1 Introduction .......................................................... 1  
   1.1 Related Topics. ................................................... 3  

2 Revoking Access to Stored Data ................................. 7  
   2.1 Preliminaries and Notation ...................................... 12  
   2.2 Revocable Attribute-Based Encryption .......................... 17  
   2.3 Revocable-Storage Attribute Based Encryption .................. 21  
   2.4 Ciphertext Delegation ............................................ 22  
   2.5 Revocable Storage KP-ABE ....................................... 35  
   2.6 Piecewise KP-ABE Construction ................................ 39  
   2.7 Revocable Storage CP-ABE ....................................... 53  
   2.8 Piecewise CP-ABE Construction ................................ 62  

3 Identifying Cheaters with No Honest Majority ............ 74  
   3.1 Preliminaries and Notation ..................................... 79  
   3.2 Locally Identifiable Secret Sharing .............................. 83  
   3.3 Relaxing Local Identifiability .................................. 93  
   3.4 Unanimously Identifiable Secret Sharing ........................ 97  
   3.5 Applications ...................................................... 102  
   3.6 Limitations of Pairwise Authentication .......................... 116  
   3.7 On the Limits of Identifiability. ................................. 121
ACKNOWLEDGMENTS

I would like to offer my gratitude to Professors Rafail Ostrovsky and Amit Sahai for their close guidance over the past four years. They have been both insightful and kind beyond what I could have expected. I also owe my love of the discipline to Moses Liskov who first taught me the material as an undergraduate, supervised my earliest research and has provided great guidance over the years.

I would also like to acknowledge the outstanding support from the National Science Foundation for the last three years.

The work in this thesis has formed the basis for two academic papers. The first paper is to appear at CRYPTO 2012 under the title ‘Dynamic Credentials and Ciphertext Delegation for Attribute-Based Encryption’ and is co-authored by Amit Sahai and Brent Waters. The second paper is titled ‘Identifying Cheaters Without an Honest Majority’ and is co-authored with Yuval Ishai and Rafail Ostrovsky and has appeared at the IACR Theory of Cryptography Conference 2012. Both papers are © IACR. The work presented here is an expanded version of the paper that appeared at the conference and is presented with the co-authors’ permission. I would like to thank Thomas King and Daniel Manchala for originally sparking the idea of researching revocation on stored data and for many helpful conversations during our study.

On a more personal note, while I can never fully acknowledge all they have given to me, I owe everything I have and who I am to my family. I also want to thank Kate for putting up with many more stressful deadlines than I had any right to expect of her and her constant support.
Vita

2007 B.S. (Mathematics) Summa Cum Laude with Highest Honors, The College of William & Mary

2007–2008 Fulbright Scholar, Budapest, Hungary

2008–2009 Teaching Assistant, Mathematics, UCLA. Led sections for Pre-calculus, Linear Algebra and Infinite Sequences and Series

2009–2012 National Science Foundation Graduate Fellow, UCLA

2012 M.A. (Mathematics), UCLA

Publications


CHAPTER 1

Introduction

Reducing the amount of trust necessary to fulfil a certain functionality is the fundamental problem of cryptography. However, trust is difficult to analyze concretely and can be implicitly present in myriad different ways. Identifying trust in a system is a difficult and sometimes holistic problem that is a necessary first step to building secure systems.

Even though cryptographic assumptions have been very successful in reducing necessary trust, this does not mean that they can be used to eliminate trust in every application. Many applications do not make semantic sense without a trusted authority (e.g. identity [71] or attribute-based [68] encryption). Similarly, there are many applications that on the surface seem to employ trust which can be removed through clever cryptography but which can be proved to be impossible to achieve without additional assumptions (e.g. multi-party computation [76] without an honest majority [18]). This thesis focuses on two problems which have some level of trust that is intractable, access control in remote storage and fully secure multi-party computation without an honest majority. Instead of accepting that some level of trust is essential for these systems and moving on, we identify new dimensions in which this trust can be reduced and provide achieve constructions that achieve this lower level of trust.

Revoking Access on Stored Data. A setting in which trust is clearly essential is remote (or cloud) storage where a third party is entrusted with storing sensitive data for an individual or organization. Despite being an area of massive importance in modern computing, the question of exactly what level of trust we need to place in the cloud is not
well understood. In this work we investigate the problem of cryptographic access control [68, 40, 6] and ask how much trust we need to place in the cloud to enforce dynamic access control on our data. We show that this functionality can be achieved without regular secure communication with the cloud while remaining secure even if data stored in the cloud is completely compromised (including the case where the attacker is a client or group of clients that previously had access). In the process we define and give a systematic study of ciphertext delegation the process in which a message encrypted under a given policy can be re-encrypted under a more restrictive policy without increasing ciphertext size or decryption complexity.

Small Primitives for MPC. Fully secure multi-party computation (with guaranteed output delivery and fairness) is impossible to achieve for arbitrary functions without an honest majority [18]. One line of work has tried to address this problem by assuming the players have access to a trusted primitive that computes some fixed function on their inputs and distributes the output [25, 33]. Here the trust is in the correct manufacture and application of this primitive. When trust is placed in hardware level, a natural way in which to reduce trust is by making this primitive as simple and reusable as possible. In our work, we give the first non-cryptographic primitive that is complete for fully secure (information theoretic) MPC that does not depend on the specific function being evaluated or its circuit complexity. In the process we formally define the new concept of locally identifiable secret sharing and give a secure construction.

Organization. As these two problems are reducing trust in two very different settings, they share little notation or preliminary information. Therefore, we present both problems and relevant notation discretely in their own chapters. Before beginning with our two main areas of contribution we will begin with giving a brief overview of several related research topics that also deal with reducing trust in applications where it is essential.
1.1 Related Topics.

In this section we give a brief overview of several other directions for reducing trust in systems where trust is essential that have achieved significant research interest. We defer a thorough treatment of the topic for revocable ABE and fully secure MPC to Chapters 2. and 3. respectively.

Reducing Trust in Identity-Based Encryption.

Identity-based encryption (IBE) was introduced by Shamir [71] in 1984 in order to address a simple problem: Looking up an individual’s public key can be time consuming and unreliable without strong infrastructure. He proposed a new system where encryption takes a master public key as input (shared amongst all users) and a single target user’s identity, allowing decryption only by the user that possesses the secret key corresponding to this identity. This allows encryption to any user as long as the encrypter holds the master public key without requiring any individual public key lookup. Identity-based encryption is a rich topic of interest in public key cryptography (e.g. [9, 8, 74]) having spawned many related topics such as fuzzy IBE [68], attribute-based encryption and hierarchical IBE [29, 30].

Identity-based encryption has the unfortunate drawback that unlike generic public-key encryption, trust must be placed in the central key generation authority to not misuse its ability to create secret keys corresponding to any identity. Several approaches have been proposed to deal with this problem, and we outline two of them in the following section.

Accountable-Authority Identity Based Encryption (A-IBE). Introduced by Goyal [38] in 2007, accountable-authority identity based encryption offers a fresh direction for reducing this essential trust placed in the key generation authority (called the public-key generator or PKG). Accepting that the authority would need this trust in order to fulfil the functionality, Goyal took a new direction and asked whether or not it might be possible to
prove the PKG had behaved maliciously in a court of law and hold it accountable if this trust is violated. This is an interesting direction for the problem of trust since it doesn’t focus on eliminating trust (since it’s essential for the functionality) and instead places outside deterrents against dishonest behavior.

The first scheme proposed by Goyal [38] achieved a notion called ‘white-box’ security where if a honest user ever recovers a secret key for his identity that was created by the PKG and disseminated without his consent, he can prove to a third party with overwhelming probability that the PKG behaved maliciously and sold a duplicate version of his secret key. A stronger notion of ‘black-box’ security requires that even if a user only recovers a decoder box with some non-negligible probability of decrypting messages encrypted to his identity, he can still prove malicious behavior on the part of the PKG. This notion was achieved in restricted settings [50, 39] before being achieved by Sahai and Seyalioglu [70] in 2011.

**Threshold Cryptography.** There are many settings where a group of players wish to designate a specific task (such as reconstructing a previously shared secret [72] or evaluating a joint function [76]). In all of these functions, there is again a measure of trust. For example, in any secret sharing scheme, if only a single player remains honest and all others refuse to return their shares there is no way for the remaining player to reconstruct the secret. Therefore, a common method in such schemes is to allow the secret to be reconstructed as long as a certain number of players remain honest. This common method is called *threshold cryptography* and is a way of splitting the trust in a system amongst many players so any small subset is unable to violate this trust. See [26] for a survey on classical approaches to threshold cryptography.

Threshold cryptography has applications in identity-based encryption as well by splitting trust in the PKG into multiple entities as shown by Boneh and Franklin [8]. In such a setting, a user collaborates with multiple PKG’s in order to receive his secret key and malicious key generation is only possible if a large number of PKG’s collude. Similar threshold methods
have also been used for signatures with great success (e.g. [20, 27, 73] and references therein).

**Reducing Trust in Attribute-Based Encryption.**

Attribute-based encryption (presented in full detail in Chapter 2.) is a generalization of IBE where an encryption can enforce very fine grained control over exactly who can successfully decrypt, unlike IBE where encryption can only be done to a single individual at a time. In this setting, a PKG assigns different users keys with certain attributes and the encryption algorithm takes a policy as input, allowing any user whose key satisfies the policy to decrypt (this in particular is called CP-ABE, a symmetric notions where the policies are stored in the key is called KP-ABE and is also presented in Chapter 2).

As in IBE, attribute-based encryption suffers from trust in a central authority. There have been several interesting directions for reducing the trust placed in the PKG in this setting as well.

**Multi-Authority ABE.** Similar to the threshold approach to splitting trust in ABE, it is also possible to entrust different capabilities to different PKG’s, limiting the impact of any one malicious entity. In a multi-authority scheme, different authorities may only be entrusted with the ability to generate secret keys for certain attributes in the scheme for a user who can then combine these components into his full secret key. Whether such a scheme could be achieved was originally asked by Sahai and Waters [68] in 2005. The first construction was due to Chase [15] in 2007 and required an online central authority while a followup by Chase and Chow [16] in 2009 improved this to trusted setup with strongly amplified privacy guarantees.

**Worry-Free Encryption.** Introduced by Sahai and Seyalioglu [69] in 2010, worry-free encryption gave a way to completely eliminate a great amount of trust in the PKG in a setting that maintained the access control guarantees of ABE (no user without an authorized key can decrypt) but at the cost of eliminating the shared public key property of general
ABE. In such a scheme, encryption takes as input a policy and identity, and the user with that identity can only decrypt if his key (produced with help from PKG) satisfies this access policy. The interesting observation here is that since the PKG and user collaborate to make the public key, it is possible to guarantee that if the PKG is honest, the user will not be able to decrypt any information he is not authorized to while keeping the user’s credentials completely secret to any outsider. And at the same time, if the user is honest, the PKG can never decrypt anything encrypted to his identity. This gives another way of reducing trust, by identifying certain applications where the full guarantee of the proposed functionality is not needed. Settings where a shared public key may not be necessary may be a small company or university where the cost to lookup a colleague’s public key is low but where users may need to do access control while keeping specific user’s credentials private (e.g. for internal auditing or tenure review).

Reducing Trust in Hardware.

A very active field in modern cryptography concerns the problem of leakage. Research in this area asks what security guarantees are possible if an attacker is allowed to gain some information on the inner state of the hardware running the application.

The work of Dziembowski and Pietrzak [22] in 2008, gave a stream cipher construction that remains secure in the presence of a bounded amount of leakage over any time period, but an unbounded amount of total leakage. Further relaxations in the model include splitting the state of the device and assuming that each side leaks information independently (for example due to being physically separated).

Many additional primitives have been constructed with strong leakage guarantees such as signatures [23, 10], pseudorandom functions [21] and public key encryption [41, 58]. We refer the reader to [2] for a survey of recent results.
CHAPTER 2

Revoking Access to Stored Data

As discussed previously, a prominent area in modern cryptography where trust can not be avoided is attribute-based encryption (ABE).

In this chapter we study the problem of reducing trust when using ABE where encryptions are stored by a third party. We ask how much trust we need to invest in this third party to ensure that we can dynamically control access on our data. We find surprisingly that we can let a third party do access control on stored data while remaining secure even if all data stored on the third party is compromised at some point without any regular secure communication between us and the third party.

We give our final construction through a black-box reduction from any scheme that satisfies certain easy to verify properties concerning key generation. We also introduce the notion of ciphertext delegation and provide a full analysis of the type of delegation possible in a number of existing schemes [40, 68, 75, 49].

The work in this section is based on joint work with Amit Sahai and Brent Waters and will appear under the title ‘Dynamic Credentials and Ciphertext Delegation for Attribute-Based Encryption’ (Copyright IACR) at the 32nd Annual International Cryptology Conference (CRYPTO 2012).
Introduction

The need to store information externally has never been higher: With users and organizations expecting to access and modify information across multiple platforms and geographic locations, there are numerous advantages to storing data *in the cloud*. However, there is a natural resistance to the idea of handing over sensitive information to external storage. Since these databases are often filled with valuable data, they are high value targets for attackers and security breaches in such systems are not uncommon, especially by insiders. In addition, organizations with access to extremely sensitive data might not want to give an outside server any access to their information at all. Similar problems can easily arise when dealing with centralized storage within a single organization, where different users in different departments have access to varying levels of sensitive data.

A first step in addressing this problem of trust is to only store information in encrypted form. However, data access is not static – as employees are hired, fired or promoted, it will be necessary to change who can access certain data. A natural solution to this problem is to have users authenticate their credentials before giving them access to data; but such an approach requires a great deal of trust in the server: a malicious party may be able to penetrate the server and bypass authentication by exploiting software vulnerabilities. A solution that avoids this problem is to use cryptographically enforced access control such as attribute-based encryption (ABE) [68]. However, this fails to address the problem that the credentials of a user may change with time. This problem motivated the study of revocation [7] where a nightly key update would only allow non-revoked users to update their keys to decrypt newly encrypted data. Dynamic credentials in the context of stored data, however, present novel challenges that have not been considered in previous study on revocation. Take the following example.

A Motivating Story. Consider an employee with access to sensitive documents neces-
sary for his work\textsuperscript{1}. One day, this employee is terminated and has his access revoked. Now, this employee with insider knowledge of the organization’s systems, and who has retained his old key, may attempt to penetrate the database server and decrypt all the files that he once had access to. How can we deal with this type of attack? At first glance, there appears to be an inherent intractability to this problem. Any encrypted file that the user could decrypt with his old key will still be decryptable, after all.

Despite these problems, we believe this situation presents a very real security threat to the organization and is important to address. One method to handle this problem is to decrypt and re-encrypt all stored data every time some employee’s credentials are revoked. However, the involvement of secret key information in this process both makes this process cumbersome and opens up the overall system to problematic new vulnerabilities. In general, we want to limit the use of secret key information to only key generation and not to database upkeep as the database is handling constant two way communication in our system and is therefore modeled as the most vulnerable party.

We propose a novel method to deal with this problem: We devise a revocable ABE system where the database, using only publicly available information, can periodically update the ciphertexts stored on the system, so that as soon as access is revoked for a user $U$, all stored files (no matter how old) immediately become inaccessible to $U$ after the update process. The database does not even need to know the identities of users whose access was revoked. We emphasize that this is a significant security improvement over decrypting and re-encrypting (which cannot be done with only public information) since in our solution, the database server never needs access to any secret keys. Furthermore, secret key holders do not have to interact with the database for the purpose of maintaining access control.

We also note in passing that while re-encrypting each ciphertext after every revocation

\textsuperscript{1}While gainfully employed, the worker may have incentives to exercise discretion by only accessing the files necessary for his work and not download all files he has access to. Such discretion may be enforced, for example, through access logs.
(in a repeated nesting fashion) can also be applied to solve the problem of access control, this solution is *inefficient* when a ciphertext needs to be updated many times. In such a solution, decryption time will increase linearly and the ciphertext may grow significantly upon each invocation (Even using hybrid encryption, this would add a potentially large ABE header each time).

**Our Results.**

In this work, we provide the first Revocable ABE scheme that deals with the problem of efficiently revoking stored data. This result is obtained through two main technical contributions:

**Revocable Storage Attribute-Based Encryption.** We provide ABE encryption schemes with a new property we call *revocable storage*. Revocable storage allows a third party storing ciphertexts to revoke access on previously encrypted data. Additionally, our scheme satisfies strong efficiency guarantees with regard to the lifetime of the database.

We realize revocable storage by introducing a notion of *ciphertext delegation*. In ABE, ciphertext delegation is the ability to restrict a ciphertext with access policy $P$ to a more restrictive policy $P'$ using only *publicly available information*, but without causing the ciphertext size to increase. We initiate the first systematic study of ciphertext delegation for both Key-Policy ABE (KP-ABE) and Ciphertext-Policy ABE (CP-ABE) by analyzing the type of delegation possible in a variety of existing schemes [40, 68, 75, 49].

**Protecting Newly Encrypted Data.** To utilize revocable storage we need a method for efficiently revoking users credentials such that newly encrypted data will not be decryptable by a user’s key if that user’s access has been revoked. This topic of revoking credential was considered by Boldyreva et al. [7] in the context of Identity-Based Encryption (and to restricted notions of security for ABE); however was not paired with the revocation of
ciphertexts. In addition, ours is the first fully (vs. selectively) secure system.

While the Boldyreva et al. system needed to be proven “from scratch”, we provide a methodology to obtain a simple construction and proof. We propose a natural modification to standard ABE which we call *Piecewise Key Generation*. Informally (KP-)ABE with piecewise key generation is similar semantically to a standard ABE scheme, except decryption takes as input two keys $K_{P_0,U}^{(0)}$ and $K_{P_1,U}^{(1)}$ from a user $U$ such that decryption only succeeds if both key policies $P_0$ and $P_1$ are authorized to decrypt the challenge ciphertext. The scheme is ‘piecewise’ in that a user may query two different key generation oracles $\text{KeyGen}_0$ or $\text{KeyGen}_1$ in order to get either side of the key adaptively. This allows the user to build his key in a piecewise fashion, after having already seen other pieces of his key. This requirement is close to standard ABE requirements and many existing proof methods extend to prove piecewise security, however, unlike standard ABE, it suffices to imply full revocability in a black-box fashion.

We then show that variants of the transformation method of Boldyreva et al. [7] succeed in converting *any* ABE scheme with piecewise key generation to a revocable ABE scheme. We give a modification of Lewko et al.’s fully secure ABE scheme [49] that satisfies our requirement and prove its security. Combined with our new techniques for dealing with revocable storage, this yields our Revocable Storage KP-ABE and CP-ABE schemes.

**Related Work.**

Originally proposed by Sahai and Waters [68], *attribute-based encryption* [40, 61, 75, 49, 60] has been an active research area in cryptography in part since it is a primitive with interesting functional applications [39, 7] and can be implemented efficiently [6]. In a key-

---

2Note that a related work [65] proposes a “fully” secure Revocable ABE scheme under a significantly different model, where the revocation list is given as an input to the encryption algorithm. This requirement makes such a scheme unsuitable for the main application of secure storage, since we cannot expect users who create documents to know the credentials of all other users in the system. Indeed, the exact credentials of any user may be a secret meant to be protected.
policy attribute-based encryption (KP-ABE) scheme every secret key is generated with a policy \( P \) and ciphertexts are generated with a set of attributes \( U \) and decryption is only possible if \( P(U) = \text{True} \). The parallel notion where ciphertexts are associated with policies and keys with sets of attributes is called ciphertext-policy attribute-based encryption (CP-ABE). While the problem of delegating a key to a more restrictive key has been considered [40], it is analyzed only in the context of the scheme proposed in the paper. The problem of revocation is also a well studied problem, both for general PKI [53, 57, 1, 54, 56, 28, 37], identity based encryption [7, 51] and attribute-based encryption [65]. At a high level, our revocable storage results can be seen as taking methods from forward secure encryption [13] which were introduced for key management and applying them to ciphertext management by noticing that the key delegation infrastructure can be replicated for the ciphertext through the delegation mechanism we introduce.

2.1 Preliminaries and Notation

We will assume \( e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T \) is a non-degenerate bilinear pairing whenever it is used. We will use \( \vec{v} \) notation for vectors and \([i,j]\) to denote the set of all integers from \( i \) to \( j \) inclusive or \([i]\) as shorthand for \([1,i]\). We will use \( l(\vec{v}) \) to denote the dimension of the vector. Throughout this chapter, \( \log(x) \) will denote the logarithm of \( x \) to the base 2. The notation \( V(T) \) for a tree \( T \) will denote the set of nodes of this tree. The notation \( x \leftarrow X \) for \( X \) a randomized algorithm may denote by context either that \( x \) is a possible output of that algorithm with positive probability or that \( x \) is drawn from the distribution \( X \). We now briefly give the syntax for the two central notions for this chapter.
Attribute-Based Encryption

Attribute-based encryption schemes are generally divided into two types depending on if the access policy is embedded in the keys or ciphertexts. We now define KP-ABE, where keys have a access policies incorporated within while ciphertexts are associated with sets of attributes:

**Definition.** *(Key-Policy ABE)* A KP-ABE scheme with attribute set $\Omega$ that supports policies $\mathcal{P}$ with message space $\mathbb{M}$ is defined by the following polynomial time algorithms:

- **Setup**$(1^\lambda) \rightarrow (PK, MSK)$: Setup takes as input the security parameter and outputs the public key and master secret key.

- **KeyGen**$(MSK, P) \rightarrow SK_P$: Key generation outputs a secret key with policy $P \in \mathcal{P}$.

- **Encrypt**$(PK, M, S) \rightarrow C_S$: Encrypts a message $M \in \mathbb{M}$ under the attribute set $S \subseteq \Omega$.

- **Decrypt**$(SK_P, C_S) \rightarrow M$: Decryption successfully recovers the encrypted message if and only if $P(S) = 1$ (in other words, the attribute set satisfies the policy).

The security game requires that an adversary queries keys corresponding to policies, and in the challenge phase requests the encryption of one of two adaptively chosen messages with an attribute set of its choice (generated during the challenge phase). The adversary succeeds if it correctly outputs the encrypted message without ever querying a key for a policy that is satisfied by the attribute set of the challenge ciphertext (we defer a formal security guarantee until we also introduce revocability). A more restrictive notion of selective security is also present in the literature in which the adversary must commit to the challenge policy before the security game begins. In a Ciphertext-Policy ABE scheme the placement of the policy is reversed, the key generation algorithm takes a set of attributes as input while encryption takes a policy.
Revocable Attribute-Based Encryption

A revocable attribute-based encryption scheme [7] has the added functionality that a user may be placed on a revocation list that will make him unable to decrypt any message encrypted after he was revoked. Such a scheme has a periodic broadcast by a trusted key generation authority that allows the un-revoked users to update their keys and continue decryption. In such a scheme we assume that the total number of users and the number of key updates the scheme can survive are both very large and therefore the scheme’s parameters should only depend polylogarithmically on the total number of non-revoked users and time bound. As in previous work, we will assume the key generation procedure is *stateful* for these constructions (in other words, it can store information used to create other keys in internal state, which we will denote with \( \sigma \)).

From this point on we will use the convention that a set of credentials \( S \) satisfies a policy \( P \) if and only if \( P(S) = 1 \).

**Definition.** *(Revocable KP-ABE)* A Revocable KP-ABE scheme with attribute set \( \Omega \) that supports policies \( \mathcal{P} \) with message space \( \mathbb{M} \), time bound \( T \) and identity length \( I \) consists of the following algorithms:

- **Setup**(1^\lambda) → (PK, MSK, \( \sigma \)): Setup takes as input the security parameter and outputs the public key, master secret key and initializes \( \sigma = \emptyset \).

- **KeyGen**(MSK, P, ID, \( \sigma \)) → (SK_{P,ID}, \( \sigma \)): Key generation outputs a secret key with policy \( P \in \mathcal{P} \) for the user \( ID \in \{0,1\}^{I} \).

- **Encrypt**(PK, M, S, t) → \( C_{S,t} \): Encrypts \( M \in \mathbb{M} \) with attribute set \( S \subseteq \Omega \) at time \( t \leq T \).

- **KeyUpdate**(MSK, rl, t, \( \sigma \)) → (\( K_t \), \( \sigma \)): The update phase takes as input a revocation list \( rl \) (a set of strings in \( \{0,1\}^{I} \)) and the master secret key and outputs the key update.
information for time $t$.

- **Decrypt**($SK_{P,ID}, K_{t'}, C_{S,t}) \rightarrow M$: Decryption successfully recovers the encrypted message if and only if $P(S) = 1$, $t' \geq t$ and $ID$ was not revoked at time $t$ (it was not present on $rl$ when $K_{t'}$ was generated).

Before we define the security game, we first define the oracles that will be used in its definition.

**Security Game Oracles.** Define the following oracles to use in the security game. These oracles are given access to $(PK, MSK, \sigma)$ that are generated at the beginning of the security game, and may have been modified since, at the time of the oracle’s invocation.

1. The Secret Key Generation oracle $SK(\cdot, \cdot)$ takes as input $(P, ID)$ and return $SK_{P,ID}$ generated from:

   $$(SK_{P,ID}, \sigma) \leftarrow \text{KeyGen}(MSK, P, ID, \sigma).$$

2. The Key Update Generation oracle $K(\cdot, \cdot)$ takes as input $t$ and a revocation list $rl$ and returns $K_t$ generated from: $(K_t, \sigma) \leftarrow \text{KeyUpdate}(MSK, t, rl, \sigma)$.

Note that for both oracles $\sigma$ is not sent to the adversary but is used to update the current $\sigma$ value of the scheme. For a p.p.t. adversary $A$ define the following experiment (some additional constraints on the adversary’s actions will be enumerated after the experiment’s definition). We use the term *challenger* for an internal agent in the security game who participates in the experiment. The challenger’s behavior is described in the security game.
RKP-Security$_A(1^λ)$:

1. The challenger runs Setup$(1^λ) \rightarrow (PK, MSK, σ)$ and returns $PK$ to $A$;

2. The adversary is given oracle access to $SK(\cdot, \cdot)$, $K(\cdot, \cdot)$ until it signals the query phase is over;

3. After the query phase, $A$ returns $(M_0, M_1, S^*, t^*)$ to the challenger;

4. The challenger picks $b$ a random bit and returns to the adversary:

   $$C_{S^*, t^*} \leftarrow \text{Encrypt}(PK, M_b, S^*, t^*)$$

5. The adversary is once again given oracle access to the two oracles above;

6. The adversary returns a bit $b'$. The Experiment returns 1 if and only if $b' = b$ and the conditions below concerning the adversary’s query history are satisfied.

The conditions placed on the adversary’s queries is as follows: For any query, $SK(P, ID)$ such that $P(S^*) = 1$, $ID \in rl$ for every query $K(t, rl)$ with $t \geq t^*$. Informally this corresponds to the fact that every user with sufficient credentials to decrypt the challenge ciphertext should be revoked by time $t^*$ for the message to remain hidden.

**Definition.** A Revocable KP-ABE scheme is secure if for any polynomial time adversary $A$ the advantage of this adversary in the RKP-Security game:

$$2 \Pr \left[ \text{RKP-Security}_A(1^λ) = 1 \right] - 1$$

is negligible in $λ$.

**Remark.** Note that in previous works on revocation [7] the security definition is pre-
sented a little differently. Their definition is much more in line with an actual implementation in which an attacker is not given the authority to query key-updates in a non-sequential order in time (in other words, after seeing a key-update for time \( t \), it can not ‘go back in time’ and query one for time \( t - 1 \)). Their definition corresponds more closely to attacking the scheme in practice, while ours is stronger. We choose our definition both because it is stronger and because it’s more straightforward to prove using our techniques. However, for most practical applications (unless there is threat of an attacker being able to generate key updates out of order, or with a time component of his choice), showing the weaker version in [7] would suffice.

2.2 Revocable Attribute-Based Encryption

The Revocable ABE and IBE constructions given by Boldyreva et al. [7] are built from an underlying ABE scheme through a new transformation they introduce. However, security of the underlying ABE scheme does not imply security of the transformation, and their resulting scheme was proven secure (in the ABE case, in the restricted selective security model) from scratch. In this work we aim to both extend and simplify previous work by investigating the additional properties an ABE scheme needs to satisfy to imply a Revocable ABE scheme following their transformation. Using our result, we modify the fully secure scheme due to Lewko et al. [49] to satisfy our requirement in both the KP-ABE and CP-ABE setting. This yields the first fully secure Revocable KP-ABE and CP-ABE schemes.

**The Requirement: Piecewise Key Generation.** We find that the necessary condition an ABE scheme should satisfy in order to imply a revocable ABE scheme is that key generation can be done in a dual componentwise fashion. In the KP-ABE setting keys will have two separate policies \( P_0 \) and \( P_1 \) such that decryption succeeds for an encryption with attribute set \( S \) if and only if \( P_0(S) = 1 \) and \( P_1(S) = 1 \). The adversary in the se-
security game is allowed to query these components separately with access to two oracles 
\( \text{KeyGen}_0 \) and \( \text{KeyGen}_1 \) which take as input a policy and an identifier \( U \) such that a key 
\( \text{KeyGen}_0(MSK, P_0, U) \) and \( \text{KeyGen}_1(MSK, P_1, U') \) can only be combined if \( U = U' \) (in our 
applications this \( U \) value will be related to, but will not exactly be a user’s identity. For this 
reason, we switch notation from identities \( ID \) to identifiers \( U \) at this point).

This security definition is stronger than the standard ABE definition because these com-
ponents may be queried in an \textit{adaptive} manner, allowing the adversary to build his key \textit{piece}
by \textit{piece}. Note the actual notation we use in our scheme is slightly different than the way
it is presented above. For the rest of this chapter we will assume key generation is allowed
to be stateful (as captured with \( \sigma \) in the Revocable ABE definition) but will omit the state
being updated as part of the syntax of the key generation algorithm from this point on for
notational simplicity.

**Definition.** A KP-ABE scheme is said to have \textit{piecewise key generation} with attribute
set \( \Omega \) supporting policies in \( \mathcal{P} \), message space \( \mathcal{M} \) and identifier length \( I \) if key generation
and encryption are modified to the following syntax:

\( \text{KeyGen} \) takes as input the master key \( MSK \), a bit \( b \) a policy \( P \in \mathcal{P} \) and an identifier
\( U \in \{0, 1\}^I \):

- \( \text{KeyGen}(MSK, b, P, U) \rightarrow K_{(b)P,U} \).

\( \text{Decrypt} \) takes two outputs of \( \text{KeyGen} \) as the key input instead of one:

- \( \text{Decrypt}(C_S, K_0, K_1) \rightarrow M \).

We now define correctness of the scheme:

**Definition.** (Correctness) A KP-ABE scheme with piecewise key generation is \textit{correct}
if for any \( S \subseteq \Omega \) and:

- \( (PK, MSK) \leftarrow \text{Setup}(1^\lambda) \)
\begin{itemize}
  \item $C_S \leftarrow \text{Encrypt}(PK, M, S)$
  \item $P_0, P_1 \in \mathcal{P}$ such that $P_0(S) = P_1(S) = 1$:
\end{itemize}

If $\text{KeyGen}(MSK, 0, P_0, U) \rightarrow K_{P_0,U}^{(0)}$ and $\text{KeyGen}(MSK, 1, P_1, U) \rightarrow K_{P_1,U}^{(1)}$, then,

$$\text{Decrypt}(C_S, K_{P_0,U}^{(0)}, K_{P_1,U}^{(1)}) = M.$$ 

The definition for security for a scheme with \textit{piecewise key generation} is now defined as one would expect: Unless the adversary has queried a single identifier $U$ to $\text{KeyGen}(MSK, 0, P_0, U)$ and $\text{KeyGen}(MSK, 1, P_1, U)$ such that $P_0(S) = P_1(S) = 1$, he should not be able to distinguish which message has been encrypted. We formalize this through the following game:

<table>
<thead>
<tr>
<th>Piecewise KPABE Security (_A(1^\lambda)):</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. The challenger runs $\text{Setup}(1^\lambda) \rightarrow (PK, MSK)$ and sends $PK$ to $A$;</td>
</tr>
<tr>
<td>2. $A$ makes queries of the type $(b, P, U)$ for $b \in {0, 1}, P \in \mathcal{P}$ and $U \in {0, 1}^I$.</td>
</tr>
<tr>
<td>The challenger runs $\text{KeyGen}(MSK, b, P, U)$ and returns the key to $A$;</td>
</tr>
<tr>
<td>3. $A$ signals the query phase is over and returns $(M_0, M_1, S)$;</td>
</tr>
<tr>
<td>4. The challenger picks $b' \xleftarrow{$} {0, 1}$ and returns $\text{Encrypt}(PK, M_b, S)$ to $A$;</td>
</tr>
<tr>
<td>5. $A$ has another query phase as previously;</td>
</tr>
<tr>
<td>6. $A$ sends a bit $b'$ to the challenger;</td>
</tr>
<tr>
<td>7. If for any $U$, $A$ has queried $\text{KeyGen}(MSK, 0, P_0, U)$ and $\text{KeyGen}(MSK, 1, P_1, U)$ such that $P_0(S) = P_1(S) = 1$ return 0;</td>
</tr>
<tr>
<td>8. If $b' = b$ return 1.</td>
</tr>
</tbody>
</table>
**Definition.** A KP-ABE scheme with piecewise key generation is secure if for any polynomial time adversary $A$ the advantage of this adversary in the Piecewise KPABE Security game:

$$2 \Pr[\text{Piecewise KPABE Security}_A(1^\lambda) = 1] - 1$$

is negligible in $\lambda$.

### 2.2.1 Revocability from Piecewise KP-ABE

To apply our results to revocability we will need to make a requirement on exactly what types of policies we are assuming our piecewise KP-ABE scheme can support. Since our ultimate goal is to apply our result to the construction of [49], we will state our result if the KP-ABE scheme supports access policy formatted as LSSS matrices (as in [49]).

**LSSS Matrices.** We begin with a brief overview of LSSS matrices. A LSSS (linear secret sharing scheme) policy is of the type $(A, \rho)$ where $A$ is an $n \times l$ matrix over the base field $\mathbb{F}$ (whose dimensions may be chosen at the time of encryption) and $\rho$ is a map from $[n]$ to $\Omega$, the universe of attributes in the scheme. A policy $(A, \rho)$ satisfies an attribute set $S \subseteq \Omega$ if and only if $1 = (1, 0, 0, \ldots, 0) \in \mathbb{F}^l$ is contained in $\text{Span}_\mathbb{F}(A_i : \rho(i) \in S)$ where $A_i$ is the $i^{th}$ row of $A$. An LSSS policy $(A, \rho)$ is called injective if $\rho$ is injective. We now state our result on a black-box reduction between KP-ABE schemes with piecewise key generation supporting LSSS policies to Revocable KP-ABE supporting LSSS policies.

**Theorem.** Let $E$ be a KP-ABE scheme with piecewise key generation supporting injective LSSS matrices with attribute set of size $\omega$. Then there exists a Revocable KP-ABE scheme $F$ supporting injective LSSS matrices with time bound $T$ under the same complexity assumptions as $E$ with an attribute set of size $\omega - 2\log(T)$.

In Section 2.5 we prove a stronger statement: In language that will be introduced shortly, we will prove that a KP-ABE scheme with piecewise key generation and ciphertext delegation
implies a Revocable-Storage KP-ABE scheme. The construction presented in Section 2.5 when the underlying scheme does not have ciphertext delegation can easily be observed to fulfill the requirement of being a Revocable KP-ABE scheme and therefore we defer the construction to that section. In Section 2.6 we give a modification of the fully secure ABE scheme due to Lewko et al. [49] that satisfies the above requirements.

2.3 Revocable-Storage Attribute Based Encryption

Motivated by settings in which a third party is managing access on encrypted data, we now present a new direction for revocability. We call the property we achieve revocable storage -the ability for a ciphertext to be updated using only the public key so that revoked users can no longer decrypt the refreshed ciphertexts. This is an additional functionality that is added onto standard revocability, allowing an untrusted third party storing ciphertexts to update them in a way to make revoked users unable to decrypt messages they were once authorized to access. This can be thought of as an adaptation of forward security [13] which is used to manage keys, to managing ciphertexts.

Definition (Revocable Storage) A Revocable KP-ABE scheme has Revocable Storage if it has the following additional algorithm:

- **CTUpdate**(CS,t,PK) → CS,t+1: The nightly update that transforms a ciphertext encrypted at time t to an independent encryption of the same message under the same set S at time t + 1.

Formally, the requirement is:

For any attribute set S ⊂ Ω, time t ∈ [T − 1] and message M ∈ M, if PK and C are such that KeyGen(1λ) → (MSK,PK) and Encrypt(PK,M,S,t) → C then,

Encrypt(PK,M,S,t + 1) ≡ CTUpdate(C,PK).
Where \( \equiv \) denotes equality in distribution. Note that this is a very strong distributional requirement, as we are insisting that starting from any ciphertext allows complete resampling from the output of the encryption algorithm. This is in contrast to merely requiring that the updated ciphertext be a valid ciphertext under the new time component. Having this strong distributional guarantee will be vital in applications where we may be delegating multiple times from the same base ciphertext.

We will call a Revocable KP-ABE scheme with Revocable Storage, a Revocable Storage KP-ABE scheme for brevity. Notice that the above procedure allows us to accomplish our motivating applications; it allows a third party storing ciphertexts to update the ciphertexts after revocation has been done at time \( t \) so that only the non-revoked users can continue decrypting. We will impose the restriction that all parameters should only depend polylogarithmically on \( T \), the upper bound for the time component and \( n \), the total number of users in the scheme. It is worth observing that there are trivial inefficient ways to satisfy this requirement assuming a vanilla KP-ABE scheme (i.e. by having \( C_{S,t} = \{ \text{Encrypt}(PK,M,S,t') : t' \leq t \} \) and having the update procedure simply delete the lowest indexed ciphertext) that depend polynomially on \( T \).

### 2.4 Ciphertext Delegation

Revocable storage centers around allowing an untrusted third party to manage access on ciphertexts by incrementing the time component. To accomplish this, we need a process by which a ciphertext can be made harder to decrypt using only public operations in a more efficient way than simply re-encrypting under the more restrictive policy.

We call this problem *ciphertext delegation* - where a user who has access to only the ciphertext and public key may process this information into a completely new encryption under a more restrictive access policy. We consider this problem for attribute based en-
cryption and show a simple method to classify delegation made possible in existing ABE schemes. We say that a ciphertext with a given access policy can be delegated to a more restrictive policy if there is a procedure that given any valid encryption of a message under the first policy produces a independent and uniformly chosen encryption of the same message under the new access policy. We again stress that delegation is required to produce a new encryption of the same message that is independent of the randomness and access policy of the the original ciphertext being delegated from. This requirement is crucial if multiple delegations from the same base ciphertext are ever used in a scheme. Without this guarantee, multiple delegations may have correlated randomness and the security of the underlying scheme would not imply any security in these applications.

Remark. Note that we only consider CPA security in this paper as the re-randomizability guarantee makes CCA2 security unattainable (as the challenge ciphertext can simply be re-randomized and sent to the decryption oracle). An analysis of possible relaxations of CCA2 security that are possible in our model is an interesting question for future work.

**KP-ABE Ciphertext Delegation**

For monotone access policies (as are generally considered in the literature [36, 49, 68]), the natural way to restrict access is by removing attributes from the ciphertext’s attribute set. Note that for non-monotone access policies, delegation is not achievable without severely limiting the key policies permitted as any change in attribute set will make the ciphertext decryptable to certain simple policies not previously authorized, implying that delegation would violate security if these policies are supported.

**Definition.** *(KP Delegation)* A KP-ABE scheme $\mathcal{E}$ with message space $\mathbb{M}$ and attribute space $\Omega$ is said to have ciphertext delegation if there is an algorithm Delegate with the following guarantee: For any $S' \subseteq S \subseteq \Omega$ and any $(PK, MSK) \leftarrow \mathcal{E}.Setup(1^\lambda)$, $M \in \mathbb{M}$
and \( C_S \leftarrow \mathcal{E}.\text{Encrypt}(PK, M, S) : \)

\[
\mathcal{E}.\text{Delegate}(PK, C_S, S') \equiv \mathcal{E}.\text{Encrypt}(PK, M, S').
\]

We show briefly how ciphertext delegation is possible in the KP-ABE scheme due to Goyal et al. [36] as the delegation procedures in the other listed schemes follow similarly. In this scheme, a ciphertext \( C_S \) with attribute set \( S \) is encrypted as:

\[
C_S = (S, MY^s, \{ T_i^s : i \in S \})
\]

where \( s \) is chosen uniformly in \( \mathbb{Z}_p \) and \( \{ T_i : i \in S \} \), \( Y \) are part of the public key. To delegate this to an encryption under attribute set \( S' \subseteq S \), we first modify the ciphertext to be \( (S', MY^s, \{ T_i^s : i \in S' \}) \) by replacing \( S \) with \( S' \) and deleting elements in the third component. While this is a valid ciphertext under attribute set \( S' \) notice that it is not a valid delegation since we require the delegated ciphertext to be a completely independent encryption. By generating \( Y^{s'}, \{ T_i^{s'} : i \in S' \} \) with \( s' \) uniformly chosen from \( \mathbb{Z}_p \), the ciphertext can be modified to \( (S', MY^{s+s'}, \{ T_i^{s+s'} : i \in S' \}) \) which is a fresh uniform encryption of \( M \) with attribute set \( S' \). A similar analysis also holds for [49, 68] which allows us to conclude:

**Remark.** The KP-ABE schemes defined in [36, 49, 68] have ciphertext delegation.

### 2.4.1 Ciphertext Policy Delegation

The more involved analysis for delegation comes when considering CP-ABE schemes as the ciphertexts may be associated with complex access policies. The most prominent CP-ABE schemes in the literature are built upon an underlying secret sharing scheme corresponding to their access policy: [68, 6] take their policies input as threshold trees while [75, 49] take their policies as LSSS matrices. The first step in the encryption procedure in these schemes is to share a uniformly chosen secret according to the implied secret sharing scheme (as either Shamir secret sharing applied to threshold trees, defined shortly, or according to a LSSS matrix) with the shares of the secret embedded into certain components of the ciphertext.
The encryption scheme is said to be based on a given secret sharing scheme if it falls into the above paradigm for this secret sharing scheme. Before we continue, we remind the reader of the syntax for LSSS matrices and threshold trees.

**Secret Sharing Notation**

**Linear Secret Sharing Schemes.** A linear secret sharing scheme over \( \mathbb{F} \) for a set of players \( \mathbb{P} \) is defined through a pair \((A, \rho)\) with \( A \) the share generating matrix of dimension \( n \times l \) and \( \rho \) the assignment function from \([n] \rightarrow \mathbb{P}\). To evaluate the shares of a secret, the vector \( w = (s, r_1, \ldots, r_l) \) is constructed where \( s \) is the secret to be shared and \( r_1, \ldots, r_l \in \mathbb{F} \) are uniformly chosen at random. The share vector is then defined as \( \vec{v} = Mw \) with party \( i \) receiving all \( \vec{v}[j] \) such that \( \rho(j) = i \).

**Threshold Trees.** An access tree is defined as a tree \( T \) where every non-leaf node \( x \) has assigned a number \( n_x \) (corresponding to the threshold level of that node) and every leaf \( x \) is assigned a (not necessarily unique) player \( i \in [n] \) (call this value \( \beta(x) \)). Whether or not an access tree is satisfied by a set of players \( A \subseteq [n] \) is determined by first assigning each leaf \( x \), the boolean \( \text{true} \) if \( \beta(x) \in A \) and setting all other leaves assigned to players not in \( A \) to \( \text{false} \). A non-leaf node \( x \) on the second level of the tree is now assigned the value \( \text{true} \) if at least \( n_x \) of its children are \( \text{true} \) and set to \( \text{false} \) otherwise. This process is recursed through all levels of the tree and \( A \) is said to satisfy \( T \) if the root of \( T \) is finally assigned \( \text{true} \). Implementing a secret sharing scheme with the access structure defined by a threshold tree is straightforward using a simple generalization of Shamir secret sharing. The secret to be shared is first assigned to the root node of the tree - every descendent of a fixed node \( x \) is then recursively assigned the evaluation of a random polynomial \( p \) of degree \( n_x - 1 \) at the node’s index - a non-zero number assigned to the node so that no two children of the same node share the same index and is part of the description of the tree such that the polynomial such that \( p(0) \) is the value assigned to \( x \). After this process has recursed through
the tree, the player $i$ is returned all leaf values labeled $i$ (note that the labeling and indexing are two different operations). Call this secret sharing scheme, *Shamir secret sharing applied to threshold trees*.

For our applications we will use slightly non-standard notation for secret sharing schemes for notational efficiency. We say that a secret sharing scheme $S$ is made of two probabilistic polynomial time algorithms $\text{Share}$ and $\text{Rec}$. The reconstruction algorithm $\text{Rec}$ behaves as expected by reconstructing the shares returned by the parties. The sharing function $\text{Share}$ takes as input a secret in $S$ as well as a policy $P$ such that a group of users $\mathcal{P}$ are authorized to reconstruct the secret if and only if $P(\mathcal{P}) = 1$. Instead of the $\text{Share}$ functionality returning one share to each party, it will output $(\vec{v}, \alpha)$ where the components of $\vec{v}$ are elements in the share space and the second output $\alpha$ is an assignment of indices of $\vec{v}$ to $[n]$ (where $n$ is the number of players) that determines which components of $\vec{v}$ should be sent to which player (if $\alpha(i) = j$, then $\vec{v}[i]$ is sent to player $P_j$; note there may be multiple elements sent to some or all players).

**Delegation Procedures**

In this section we analyze policy delegation in CP-ABE, the process by which a ciphertext $C_P$ (encrypted with policy $P$) can be delegated into a new $C_{P'}$. To this end, we observe that most known CP-ABE schemes actually have the access structure ‘built in’ to the ciphertext in the form of a secret sharing scheme. Informally, these schemes work as follows: The encryption algorithm will first generate some secret $s \in S$ at random and internally share it according to the policy it is being encrypted under. In the decryption phase, the decryptor is then able to ‘use’ all the shares that correspond to attributes he possesses to reconstruct the underlying secret, succeeding in recovering the message if he does. Unfortunately formally defining this intuition is somewhat difficult. It is with this motivation that we define $F$, the share-extractor that recovers the shares embedded in the ciphertext. We will call a
ciphertext valid if it is a possible encryption of some message in the message space $M$ under some supported policy.

We say a CP-ABE scheme is based on a specific secret sharing scheme $S$ with share space $S$ if there is a (possibly inefficient) share-extractor $F$ that on input any valid ciphertext $C_P$ outputs $(s, (\vec{v}, \alpha))$ where $s \in S$, $(\vec{v}, \alpha) \leftarrow S. \text{Share}(s, P)$. The guarantee is that the shares the extractor recovers correspond correctly to a secret shared under the ciphertext access structure: If $F(C) = (s, (\vec{v}, \alpha))$ with $(\vec{v}, \alpha) \leftarrow S. \text{Share}(s, P)$ then $C$ is a valid ciphertext with policy $P$.

**Elementary Ciphertext Manipulations**

We will define a CP-ABE scheme as having ciphertext delegation if it allows for certain operations to be performed on its ciphertext that manipulate the shares of the underlying secret sharing scheme in a fixed way. A CP-ABE scheme allows elementary ciphertext manipulations if there exist the following efficient public operations on valid ciphertexts. As the first requirement does not change the underlying structure of the ciphertext but merely refreshes the encryption, we call this **Property 0**.

**Property 0.** Any well-formed ciphertext under a given access policy can be re-randomized to an independent encryption of the same message under the same policy.

In addition, there are the following four efficient operations on valid ciphertexts (for all examples below let $C$ be a valid ciphertext with $F(C) = (s, (\vec{v}, \alpha))$ and $[n]$ the set of attributes):

**Property 1.** *Linearly combining shares with the same label:* There is a p.p.t. algorithm $\text{Combine}$ such that for any $i, j \leq l(\vec{v})$ with $\alpha(i) = \alpha(j)$ if $\text{Combine}(C, i, j, a_i, b_j, d) = C'$ where
\( F(C') = (s, (\vec{v}', \alpha')) \), then:

\[
\vec{v}'[k] = \begin{cases} 
\vec{v}[k], & \text{for all } k \neq i; \\
a_i \vec{v}[i] + b_j \vec{v}[j] + d, & \text{if } k = i.
\end{cases}
\]

**Property 2.** *Deleting components of \( \vec{v} \):* There is a p.p.t. algorithm \textsf{Delete} such that for any \( i \leq l(\vec{v}) \) if \( \textsf{Delete}(C, i) = C' \) where \( F(C') = (s, (\vec{v}', \alpha')) \). Then, \( l(\vec{v}') = l(\vec{v}) - 1 \) and:

\[
(\vec{v}'[k], \alpha'(k)) = \begin{cases} 
(\vec{v}[k], \alpha(k)), & \text{for } k < i; \\
(\vec{v}[k+1], \alpha(k+1)), & \text{if } k \geq i.
\end{cases}
\]

**Property 3.** *Adding new entries to \( \vec{v} \):* There is a p.p.t. algorithm \textsf{Add} such that for any \( i \in [n] \) if \( \textsf{Add}(C, i) = C' \) where \( F(C') = (s, (\vec{v}', \alpha')) \). Then \( l(\vec{v}') = l(\vec{v}) + 1 \) and:

\[
(\vec{v}'[k], \alpha'(k)) = \begin{cases} 
(\vec{v}[k], \alpha(k)), & \text{for } i \leq l(\vec{v}); \\
(0, i), & \text{if } k = l(\vec{v}) + 1.
\end{cases}
\]

**Property 4.** *Swapping entries in \( \vec{v} \):* There is a p.p.t. algorithm \textsf{Swap} such that for any \( i, j \leq l(\vec{v}) \) if \( \textsf{Swap}(C, i, j) = C' \) where \( F(C') = (s, (\vec{v}', \alpha')) \), then:

\[
(\vec{v}'[k], \alpha'(k)) = \begin{cases} 
(\vec{v}[k], \alpha(k)), & \text{for } k \notin \{i, j\}; \\
(\vec{v}[i], \alpha(i)), & \text{if } k = j; \\
(\vec{v}[j], \alpha(j)), & \text{if } k = i.
\end{cases}
\]

We now state our observation on known CP-ABE schemes that fall under the framework we have just described. Consulting the constructions for each scheme, it is immediate to observe where the underlying secret sharing scheme is embedded during encryption and how the operations above can be performed.

**Observation.** The CP-ABE schemes given in SW05 [68], BSW07 [6], Waters11 [75] and LOSTW10 [49] allow for elementary ciphertext manipulations. These schemes are based on Shamir secret sharing, Shamir secret sharing generalized to threshold trees and linear secret sharing schemes respectively.
These elementary ciphertext manipulations are our basic tools for ciphertext delegation.

Notice that if a CP-ABE scheme allows delegation of any well-formed ciphertext under policy $P$ to be processed into a well-formed ciphertext under policy $P'$ using the elementary delegation operations, this implies that any ciphertext encrypted with policy $P$ may be delegated to a uniformly random encryption of the same message under policy $P'$ by the re-randomization guarantee. Depending on what secret sharing scheme the CP-ABE scheme is based on, the elementary delegation operations may have significantly different capabilities which we outline below.

**Theorem.** Let $\mathcal{E}$ be a CP-ABE scheme based on Shamir secret sharing generalized to threshold trees that allows for elementary ciphertext manipulations. A ciphertext $C_T$ encrypted under threshold tree $T$ can be delegated to a ciphertext $C_T'$ if $T'$ can be derived from $T$ by any sequence of the following operations:

1. Inserting a node $x$ along an edge with $n_x = 1$ (In other words, splitting an edge into two, connected by a node $x$). Note that this does not affect the access structure.

2. Increasing $n_x \rightarrow n_x + 1$ while optionally adding another leaf $y$ labeled arbitrarily with index not equal to that of any sibling.

3. Deleting a subtree.

Properties 1. and 3. are trivial as a valid secret sharing under one tree will also be valid under the second after 1. is performed and 3. only requires repeated invocations of Delete. We now give the full proof of 2. A similar method was used for key delegation in [40].

**Proof of 2.** Take $C$ with $F(C) = (s, (\vec{v}, \alpha))$ where $(\vec{v}, \alpha) \leftarrow \text{Share}(s, T)$ for the threshold tree $T$. We will show how to increase the threshold of a node $x$ while inserting the leaf node labeled $i \in [n]$ with index $\mu$, if this leaf node is not desired it can simply be deleted by the Delete functionality after the delegation. We will insert this new share to be the last entry of the share vector, which can be changed with the Swap functionality if desired.
Let \( q(X) \) be the \( n_x - 1 \) degree polynomial associated with the node \( x \) (recall that in threshold trees, each node is associated with a polynomial). We will show how to change the implied polynomials in the tree so that the polynomial at this node is actually changed to \( q'(X) = (1 - X/\mu)q(X) \). This will add 1 to the degree of the polynomial at \( x \), corresponding to incrementing \( n_x \) while still interpolating to the same point as \( q \) at 0. To change the polynomial to the above value with an additional leaf \( y \) assigned to \( i \) added as a child to \( x \):

1. Use \( \text{Add}(C, i) \) to set the share at this new vector component to 0.

2. The remaining net effect of changing the polynomial as above will be to multiply the shares of leaves that are descendants of a child \( z \) of \( x \) by \( (1 - \text{index}(z)/\mu) \) where the multiplication is done using \( \text{Combine} \) to scale entries of \( \vec{v} \). This is implicitly changing the value assigned to the node \( z \) and all descendants of \( z \) by multiplying the previous value with \( (1 - \text{index}(z)/\mu) \).

Note that the delegation allowed through the elementary ciphertext manipulations on Shamir threshold tree based schemes can be elegantly quantified, the effect on the access structure of each of the operations we discuss above can be immediately seen. Similar delegation properties are possible for general LSSS schemes in a straightforward application of the manipulations.

**Theorem** Let \( \mathcal{E} \) be a CP-ABE scheme based on linear secret sharing matrices that allows for elementary ciphertext manipulations. A ciphertext \( C_A \) encrypted under policy \((A, \rho)\) can be delegated to a ciphertext \( C'_{A'} \) with policy \((A', \rho')\) if \((A', \rho')\) can be derived from \((A, \rho)\) through any sequence of the following operations:

1. Swapping rows: The output of the function \( \rho \) remains unchanged except for at these rows, where it is also swapped. Note that this does not affect the access structure.

2. Deleting rows: The output of \( \rho' \) on a row of \( A' \) corresponds to the output of \( \rho \) on the corresponding row of \( A \) (before it was shifted by deletion).
3. Adding a new row: Let $A$ be $m \times n$, then $A'$ is $m + 1 \times n + 1$ with $A$ the upper left $m \times n$ submatrix of $A'$ equal to $A$ and $m + 1 \times n + 1$ entry equal to 1. The remaining entries and $\rho(n + 1)$ may be assigned arbitrarily.

4. Linearly combining rows: If $\rho(i) = \rho(j)$, $A'$ may be achieved from $A$ by multiplying row $i$ by a constant and adding it to row $j$ to get a new value for row $j$.

**Key Delegation.** Note that the methods we propose for ciphertext delegation can also be applied to delegation of key policies in some existing KP-ABE schemes [36, 68]. Some key delegation techniques possible for a KP-ABE scheme based on threshold trees are given in [36].

**Linear Ciphertext Delegation**

Since KP-ABE delegation only allows for very restricted manipulation of the ciphertext, it is somewhat surprising that it can be used to handle ciphertext updates while only introducing a logarithmic dependence on $T$ since these ciphertexts should be resilient to being updated up to $T$ times. For a revocable-storage ABE scheme we have two problems to address simultaneously, the first one is the revocability infrastructure, and the second is the process by which ciphertexts are managed so that the time component can be incremented through $\text{CTUpdate}$. We have already introduced the main tool we will use to address the first problem, piecewise key generation. In this section we introduce the methods we use to address the second problem. In Section 2.5 we combine our previous results with the techniques we develop here to achieve full revocable-storage ABE.

In this section we define PKE with Linear Delegation as a toy example to introduce the reader to our methods to address the second problem listed above. This is a public key encryption scheme where keys are additionally associated with a time $t \in [T]$ and encryption takes an additional input $t'$. Decryption should only succeed if $t' \leq t$, corresponding to the
requirement that a key issued at time $t$ should be valid only to decrypt ciphertexts generated before this time. Furthermore, we will require it to allow a $\text{CTUpdate}$ procedure as we defined previously where an encryption at time $t$ can be delegated using only the public key to an independent encryption at time $t+1$. This gives us a chance to study the updating procedure before additionally imposing the ABE access control infrastructure.

**Managing the Time Structure Efficiently**

In this section we will show how to build an efficient PKE scheme with linear delegation from a KP-ABE scheme with ciphertext delegation. To solve this problem we use a binary tree $Q$ of depth $\log(T) = r$ (from now on we will assume $T$ is a perfect power of 2 for notational convenience, if not then the $r$ will just be taken to $\log(T)$ rounded up to the next integer), in a method introduced by Canetti et al. [13] in the context of forward secure encryption. Nodes of this tree will correspond to different attribute sets, while a single encryption of the delegatable scheme, interestingly, will be comprised of multiple encryptions from the underlying KP-ABE scheme, each one corresponding to an attribute set from a different node of the tree. While only one of these ciphertext components may be necessary for a secret key holder to decrypt, the ciphertexts include multiple components for delegation purposes.

**Labeling Nodes of the Tree.** Associate with each leaf of $Q$ a string corresponding to its path from the root with 0's denoting that the path traverses the left child of the previous node and 1's indicating traversing through the right child. As an example, the string $0^r$ corresponds to the leftmost leaf of the tree while $0^{r-1} \circ 1$ corresponds to its right sibling. Associate non-leaf nodes to strings by the path from the root by using $*$ values to pad the string to $r$ bits. For example, the root node will be referred to by the string $*^r$ while $0 \circ *^{r-1}$ refers to its left child. The string associated with a node $v$ will be labeled $b(v)$. We refer to the node associated with a string $x$ as $v_x$; notice this gives a bijection between the
time components \( t \in \{0, 1\}^r \), and the leaves of the tree \( Q \).

Managing the access structure will require associating each time \( t \in [T] \) with a set of nodes in the tree through a process we describe below. The following theorem will be the main method through which we will handle the time component of our final revocable storage construction. This theorem is also implicit in the work of [13] but we provide a proof here for containment. For the theorem statement we will replace the time bound \( T \) with \( q \) to avoid confusing it with the trees, that will be called \( T \). The value \( r \) is now \( \log(q) \). Below the term ‘efficiently computable’ means in time linear in \( r \).

**Theorem.** *(Tree Theorem)* There are (efficiently computable) subsets of \( V(Q) \) (the node set of \( Q \) where \( Q \) is a tree of depth \( r \)), \( \{T_i : i \in [q]\} \) such that for all \( t \in \{0, 1\}^r \):

- **Property 1.** \( T_t \) contains an ancestor of \( v_{t'} \) if and only if \( t \leq t' \);
- **Property 2.** If \( u \in T_{t+1} \) then there is an ancestor of \( u \) in \( T_t \);
- **Property 3.** \( |T_t| \leq r \).

We first give an informal intuition of how this sequence of trees will be used in the scheme. A secret key for time \( t \) will be associated with the leaf \( v_t \) of \( Q \) while a ciphertext at time \( t' \) will be associated with the set of nodes \( T_{t'} \). A key for time \( t \) will succeed in decryption (by using the underlying KP-ABE scheme) if and only if \( v_t \) is a descendant of the node of the ciphertext *(Property 1. above will then imply that a key at time \( t \) will only succeed in decrypting ciphertexts from earlier times).*

Additionally, in our implementation delegation will be possible by traversing down the tree - a ciphertext associated with a set of nodes will be delegatable to a ciphertext associated with another set if and only if for every node in the target set (for the ciphertext being delegated to), one of its ancestors is in the first set (the set associated with the ciphertext being delegated from). **Property 2.** allows us to conduct linear delegation. **Property 3.** guarantees that this process can be done efficiently.
Proof. We now describe the process to construct $T_t$. Let $v_t$ be as before and $\text{Path}_t$ be the set of nodes from $v_t$ to the root of $Q$ (including $v_t$ and the root). Let $T_t$ be the set of right children of $\text{Path}_t$ that are not included in $\text{Path}_t$. Notice that since the number of elements in $T_t$ is at most the number of elements in $\text{Path}_t$ (which is of size $r$), we have Property 3 immediately.

We now prove Property 1. We begin by showing that if $t \leq t'$ then $T_t$ contains an ancestor of $v_{t'}$. Let $w$ be the first common ancestor of $v_t$ and $v_{t'}$. Since $t \leq t'$, $v_t$ is an element of the left subtree (the subtree of $Q$ with the left child of $w$ as its root) of $w$ and $v_{t'}$ is an element of its right subtree, meaning it is a descendant of the right child of $w$. Since $w$ is the lowest common ancestor of $v_{t'}$ and $v_t$ and the right child of $w$ is an ancestor of $v_{t'}$, it is not an ancestor of $v_t$ and therefore is not on $\text{Path}_t$. Since the right child of $w$ is not on $\text{Path}_t$ and $w$ is on $\text{Path}_t$, this implies it is in $T_t$. As we’ve established the right child of $w$ is an ancestor of $v_t$ this implies there is a node in $T_t$ that is an ancestor of $v_t$.

We now show the other direction of Property 1. Let $t > t'$ and we will show $T_t$ does not contain an ancestor of $v_{t'}$. Considering $\text{Path}_{t'}$, by a similar argument to the above case, we see that $\text{Path}_{t'}$ joins $\text{Path}_t$ through the left child of some node in $w$. As $\text{Path}_{t'}$ and $\text{Path}_t$ are the same after they join, it’s easy to see $\text{Path}_{t'}$ never overlaps with the right child of a member of $\text{Path}_t$ unless this child is also in $\text{Path}_t$. This implies that $\text{Path}_{t'} \cap T_t = \emptyset$ as desired.

We now show Property 2. Take any $u \in T_{t+1}$ and assume that $u \not\in T_t$ (otherwise the statement is trivially satisfied) and we will show that $u$ has an ancestor in $T_t$. By construction $u$ is the right child of some node $p(u)$ in $\text{Path}_{t+1}$. Let $w$ be the lowest common ancestor of $v_t$ and $v_{t+1}$. Then, similarly to the proof of the previous case, $v_{t+1}$ is a descendent of the right child of $w$ and therefore, either $p(u) \in \text{Path}_t$ (in which case $u \in T_t$ and the statement is true), or $p(u)$ is a descendent of the right child of $w$ (recall the right child of $w$ is in $T_t$), implying $p(u)$ and therefore $u$ has an ancestor in $T_t$. 

34
2.5 Revocable Storage KP-ABE

By combining the method above for achieving linear delegation with our fully secure KP-ABE scheme with piecewise key generation and ciphertext delegation (given in Section 2.6), we will now show the following theorem. We defer the construction of our KP-ABE scheme with the required guarantees until after this section as the specific construction and security proof is involved and unnecessary for understanding the connection between piecewise key generation, delegation and revocable storage.

**Theorem** Let $E$ be KP-ABE scheme with ciphertext delegation and piecewise key generation that supports injective LSSS matrices with attribute set size $\omega$. Then there exists a Revocable Storage KP-ABE scheme $F$ that supports injective LSSS matrices with time bound $T$ under the same complexity assumptions as $E$ with attribute set size $\omega - 2 \log(T)$.

To prove the above theorem, we will use a second tree $U$ for revocation management (as in [7]) with the identities $\{0, 1\}^T$ labeling the leaves. For a set of leaves $V$, the function $U(V)$ returns a (deterministically chosen) collection of nodes of $U$ such that some ancestor of a leaf $v$ is included in $U(V)$ if and only if $v \notin V$. That such a function exists and can be computed in polynomial time in the size of $V$ and $I$ is shown in [7]. Define $\text{Path}(ID)$ for a leaf $v_{ID}$ (where the name of the node identifies the path from the root to this node, as in Section 2.4.1) as the set of nodes from the root of $U$ to the leaf $ID$ (including the root and leaf).

We separate the attribute set of $E$, reserving some attributes to only be used to manage the time component. Write the attribute set of $E$ as $\Omega' \cup \Omega$ through some strict ordering (for example lexicographically by their name) and label the components of $\Omega'$ as:

$$\Omega' = \{\omega_{i,b} : i \in [\log(T)], b \in \{0, 1\}\}$$

and for each node $y$ define (where the string $b(y)$ is comprised of 0, 1 and $*$’s defined previously corresponding to the path from the root to this node with $*$s padding the string
to length $\log(T)$:

$$s_y = \text{for all } i \in [\log(T)] \begin{cases} \text{if } b(y)[i] \in \{0, 1\} & \omega_{i,y[i]} \in s_y, \\ \text{otherwise } & \omega_{i,0} \text{ and } \omega_{i,1} \in s_y \end{cases}$$

Next let $P_t$ be the policy defined by:

$$P_t(S) = 1 \text{ if and only if } \omega_{i,t[i]} \in S \ \forall \ i \in [\log(T)]$$

The important observation about $P_t$ is that $P_t(s_y) = 1$ if and only if $y$ is an ancestor of the leaf corresponding to $t$ and that injective LSSS matrices suffice to express the policies $P_t$. The encryption and key generation procedures of $\mathcal{F}$ will operate solely over $\Omega$ (this is why we lose $2\log(T)$ elements from our attribute set). We now describe how to construct our Revocable Storage KP-ABE scheme $\mathcal{F}$ from $\mathcal{E}$ as defined above. We will use properties about $T_t$ defined in the previous section heavily during this construction.

- **Setup($1^\lambda$) :** Return $\mathcal{E}.\text{Setup}(1^\lambda) = (PK, MSK)$

- **KeyGen($MSK, P, ID$):** For all $x \in \text{Path}(ID)$ set $SK_{p,x}^{(0)} = \mathcal{E}.\text{KeyGen}(MSK, 0, P, x)$. Return:

$$SK_{p,ID}^{(0)} = \{SK_{p,x}^{(0)} : x \in \text{Path}(ID)\}.$$

- **Encrypt($PK, M, S, t$) where $S \subseteq \Omega$:** For all $x \in T_t$ set:

$$C_{S,x} = \mathcal{E}.\text{Encrypt}(PK, M, S \cup s_x).$$

Return:

$$C_{S,t} = \{C_{S,x} : x \in T_t\}.$$

- **KeyUpdate($MSK, rl, t$) :** For all $x \in \mathcal{U}(rl)$ set:

$$SK_{p,x}^{(1)} = \mathcal{E}.\text{KeyGen}(MSK, 1, P, x).$$
Return:

\[ K_t = \{ SK_{P_t,x}^{(1)} : x \in U(rl) \}. \]

- **Decrypt** \((SK_{P,ID}, K_{t'}, C_{S,t})\): If \( ID \notin rl \) when \( K_{t'} \) was generated, there is some \( x \in U(rl) \cap \text{Path}(ID) \) (by the definition of \( U(V) \)). For this \( x \) there is:

\[ SK_{P,x}^{(0)} \in SK_{P,ID} \text{ and } SK_{P_{t'},x}^{(1)} \in K_{t'}. \]

Additionally, if \( t' \geq t \) there is some \( y \in T_t \) such that \( y \) is an ancestor of the leaf \( v_{t'} \), which implies \( P_{t'}(s_y) = 1 \). For this \( y \), take \( C_{S,y} \in C_{S,t} \) and return:

\[ \mathcal{E}. \text{Decrypt}(SK_{P,x}^{(0)}, SK_{P_{t'},x}^{(1)}, C_{S,y}). \]

If \( P(S) = 1 \) then \( P(S \cup s_y) = P_{t'}(S \cup s_y) = 1 \) which implies that decryption succeeds.

- **CTUpdate** \((PK, C_{S,t})\): For all \( x \in T_{t+1} \) find \( y \in T_t \) such that \( y \) is an ancestor of \( x \) such that there is a \( C_{S,y} \) component in \( C_{S,t} \). For all such \( x \) set:

\[ C_{S,x} = \mathcal{E}. \text{Delegate}(PK, C_{S,y}, S \cup s_x) \]

Which is possible since \( y \) being an ancestor of \( x \) implies \( s_x \subset s_y \). Return:

\[ C_{S,t+1} = \{ C_{S,x} : x \in T_{t+1} \} \]

We now describe how security of the underlying \( \mathcal{E} \) implies security of \( \mathcal{F} \) in the Revocable KP-ABE security game. The fact that the **CTUpdate** procedure is correct can be observed similarly to the construction from Section 2.4.1 and it therefore only remains to argue that the above is a secure Revocable KP-ABE scheme.

**Proof of RKP-ABE Security.** Let \( A \) be an adversary such that \( \text{RKP-Security}_A(1^\lambda) \) is non-negligible, we will construct an \( A' \) such that \( \text{PIECEWISE KP-ABE}_{A'} \) is non-negligible. We will modify the **PIECEWISE** security game slightly and give an \( A' \) with non-negligible
advantage when instead of a single challenge query, the adversary gives a pair of messages 
\((M_0, M_1)\) as well as a tuple of sets \((S_1^*, S_2^*, \ldots, S_{\rho}^*)\) and is returned the tuple:

\[
\{\text{Encrypt}(PK, M_b, S_i^*) : i \in [\rho]\}
\]

by the challenger with the restriction that all \(S_i^*\) obey the restriction placed on \(S^*\) in the
standard game. This implies security in the standard Piecewise KP-ABE security game
by a standard hybrid argument.

\(A'\) begins by initializing the Piecewise KP-ABE security game and forwarding \(PK\)
to \(A\). To respond to an \(SK(P, ID)\) query \(A'\) sends a query \((0, P, x)\) to its key generation
oracle for all \(x \in \text{Path}(ID)\) which drawn from the same distribution as the construction
above. Similarly, for all queries \(K(t, rl)\), \(A'\) sends a query \((1, P_t, x)\) for all \(x \in \mathcal{U}(rl)\) to its
key generation oracle to simulate the key update information.

When \(A\) makes a challenge query \((M_0, M_1, S^*, t^*)\) in order the simulate this, \(A'\) in the
modified game we described above will send as its challenge query \((M_0, M_1)\) and the tuple
\(S^* \cup s_x\) for all \(x \in T_{t^*}\). Notice that by responding to the queries in this fashion we have
perfectly simulated the expected distribution for \(A\). It remains only to show that as long as
\(A\) does not submit an invalid query that causes the experiment to automatically output 0 our
\(A'\) has not submitted an invalid query to the Piecewise KP-ABE oracle in the modified
game.

Take any \(S^* \cup s_x\) in the challenge query that \(A'\) makes as described above. Take any
\(y \in \mathcal{U}\) we now claim that either for either \(b = 0\) or \(b = 1\) all queries of the type \((b, P, y)\) that
\(A'\) makes while simulating the queries of \(A\), \(P(S^* \cup s_x) = 0\).

First consider the case where for some descendent leaf of \(y\) (which is a \(I\) bit string) that
\(A\) makes a query to \(SK(P, ID)\) with \(P(S^*) = 1\). In this case by the guarantee on the queries
of \(A\) for all \(K(t, rl)\) queries with \(t \geq t^*\), \(ID \in rl\). This implies that for all queries of the
type \((1, P_t, z)\) that \(A'\) makes, either \(t < t^*\) (in which case \(P_t(S^* \cup s_x) = 0\) since \(P_t\) does not

38
depend on $S^*$ and $P_t(s_x) = 0$ since $x$ is not an ancestor of $t$ because $x \in T_r$ or $ID \in rl$ which implies no ancestor of $ID$ is contained in $U(rl)$ and therefore, $z$ is not an ancestor of $ID$ and therefore $z \neq y$. So we have established that in this case, all $(1, P, y)$ queries have the property that $P(S^* \cup s_x) = 0$ as desired.

Next, consider the case where $A$ does not make a query to a descendent leaf $ID$ of $y$ with $SK(P, ID) = 1$. Then, in simulating $A'$ only makes $(0, P, y)$ queries where $P(S^*) = 0$ which implies $P(S^* \cup s_x) = 0$ (since the policies for the Revocable scheme are only over $\Omega$). This shows the desired statement in both cases, proving the theorem. Using the construction given in Section 2.6 we conclude:

**Theorem** Under Assumptions 1, 2, and 3. (defined below), when $E$ is set to be the scheme given in Section 2.6 the above $F$ is a fully secure KP-ABE scheme with Revocable Storage supporting injective LSSS Matrices.

### 2.6 Piecewise KP-ABE Construction

We now introduce the assumptions we will be using to build our scheme. These are the same assumptions from the fully secure ABE construction due to Lewko et al. [49]. The notation of giving $G$ as input will imply that this includes a description of the group, the bilinear pairing operation and the order of the group.

**Assumption 1.** Let $G$ be a cyclic group of size $N = p_1 p_2 p_3$ with bilinear map $e$ selected according to the group generator $G(1^\lambda)$. Consider the event that we generate $g \leftarrow G_{p_1}, X_3 \leftarrow G_{p_3}, T_1 \leftarrow G_{p_1, p_2}, T_2 \leftarrow G_{p_1}$ uniformly at random. **Assumption 1.** states that for any probabilistic polynomial time $A$:

$$\left| \Pr[A(G, g, X_3, T_1) = 0] - \Pr[A(G, g, X_3, T_2) = 0] \right|$$

is negligible in $\lambda$.

**Assumption 2.** Let $G$ be a cyclic group of size $N = p_1 p_2 p_3$ with bilinear map $e$ selected
according to the group generator $G(1^\lambda)$. Consider the event that we generate $g, X_1 \leftarrow \mathbb{G}_{p_1}, X_2, Y_2 \leftarrow \mathbb{G}_{p_2}, X_3, Y_3 \leftarrow \mathbb{G}_{p_3}, T_1 \leftarrow \mathbb{G}$ and $T_2 \leftarrow \mathbb{G}_{p_1p_3}$ uniformly at random. Assumption 2 states that for any probabilistic polynomial time $A$ (if we let $D = (\mathbb{G}, g, X_1X_2, X_3, Y_2Y_3)$):

$$\left| \Pr[A(D, T_1) = 0] - \Pr[A(D, T_2) = 0] \right|$$

is negligible in $\lambda$.

Assumption 3. Let $\mathbb{G}$ be a cyclic group of size $N = p_1p_2p_3$ with bilinear map $e$ selected according to the group generator $G(1^\lambda)$ with target group $\mathbb{G}_T$. Consider the event that we generate $g \leftarrow \mathbb{G}, \alpha, s \leftarrow \mathbb{Z}_N, X_2, Y_2, Z_2 \leftarrow \mathbb{G}_{p_2}, X_3 \leftarrow \mathbb{G}_{p_3}$ uniformly at random. Finally select $T_1 = e(g, g)^{\alpha s}$ and $T_2 \leftarrow \mathbb{G}_T$. Assumption 3. states that for any probabilistic polynomial time $A$, if we let $D$ denote $D = (\mathbb{G}, \mathbb{G}_T, g, g^\alpha X_2, X_3, g^s Y_2, Z_2)$, then:

$$\left| \Pr[A(D, T_1) = 0] - \Pr[A(D, T_2) = 0] \right|$$

is negligible in $\lambda$.

**Theorem.** The KP-ABE scheme given below has piecewise key generation, supports injective LSSS policies and has ciphertext delegation if Assumptions 1, 2, 3. hold.

Since $\mathbb{G}$ is cyclic, it has unique subgroups of size $p_1, p_2$ and $p_3$ which we call $\mathbb{G}_{p_1}, \mathbb{G}_{p_2}$ and $\mathbb{G}_{p_3}$ respectively.

Note that the proof of security requires significant divergence from [49] but the construction is a fairly straightforward adaptation. For our construction and security proof we will assume that the access policy $(A, \rho)$ has $A$ a $n \times l$ matrix where $n$ and $l$ are fixed for convenience and the obvious adaptation of the construction and proof suffices to cover the more general case where the dimensions are allowed to vary. We let the vector 1 stand as shorthand for $1 \circ 0 \circ 0 \ldots 0$ when the dimension is specified by context. Note that below we will assume the decryption algorithm has the public key as input, which was not part of our original definition. Our construction below can be modified to fit our formal definition by
including the public key as part of all secret keys, however we produce it where they are separate to give the leanest instantiation.

**Setup**($1^\lambda$) → ($PK, MSK$): Choose a bilinear group of order $N = p_1p_2p_3$ (three distinct primes) according to $G(1^\lambda)$. Then choose $\alpha \leftarrow \mathbb{Z}_N$ and $g \leftarrow \mathbb{G}_{p_1}$ uniformly. For each $i \in \Omega$, $s_i \leftarrow \mathbb{Z}_N$ uniformly at random. Pick $X_3 \in \mathbb{G}_{p_3}$ uniformly with $X_3 \neq 1$ and set:

$$PK = (N, g, e(g, g)^\alpha, X_3, T_i = g^{s_i} \text{ for all } i \in \Omega), \quad MSK = (\alpha, PK).$$

**KeyGen**($MSK, b, (A, \rho), U, PK$). If $\alpha_U$ has not been generated yet, generate it and store it. For each row of $A_i$ of $A$ choose a random $r_i \leftarrow \mathbb{Z}_N$ and $W_i, V_i \in \mathbb{G}_{p_3}$. If $b = 0$ let $u$ be a random $l$ dimensional vector over $\mathbb{Z}_N$ such that $1 \cdot u = \alpha_U$ otherwise, sample it subject to the restriction that $1 \cdot u = \alpha - \alpha_U$. For all $i \in [n]$ set:

$$K_i^{(1)} = g^{A_i \cdot u}T^{r_i}_{\rho(i)}W_i, \quad K_i^{(2)} = g^{r_i}V_i$$

and return $SK^{(b)}_{U,(A,\rho)} = \{K_i^{(1)}, K_i^{(2)} : i \in [n]\}$.

**Encrypt**($PK, M, S$). Choose $s \leftarrow \mathbb{Z}_N$ at random. Return:

$$CT_S = (C = Me(g, g)^{\alpha s}, C_0 = g^s, (C_i = T_i^s : i \in S)).$$

**Decrypt**($CT_S, SK^{(0)}_{U,(A,\rho)}, SK^{(1)}_{U,(B,\beta)}, PK$). Let $\omega_i \in \mathbb{Z}_N$ be such that $\sum_i \omega_i A_i = 1$.

Label the components of $SK^{(0)}_{U,(A,\rho)}$ as $K_i^{(1)}, K_i^{(2)}$ for $i \in [n]$ and set:

$$\prod_{\rho(i) \in S} \frac{e(C_0, K_i^{(1)})^{\omega_i}}{e(C_{\rho(i)}, K_i^{(2)})^{\omega_i}} = \prod_{\rho(i) \in S} \frac{e(g, g)^{s\omega_i A_i \cdot u}e(g, T_{\rho(i)}^{s r_i \omega_i})}{e(g, T_{\rho(i)}^{s r_i \omega_i})} = e(g, g)^{s\alpha U}$$

And using the identical procedure with $SK^{(1)}_{U,(B,\beta)}$ recovers $e(g, g)^{s(\alpha - \alpha_U)}$ allowing recovery of $e(g, g)^{s\alpha}$ and $M$ as $C/e(g, g)^{s\alpha}$.
2.6.1 Proof of Security

We now define semi-functional ciphertexts and keys for our purposes. While the ciphertext modification is unchanged from the original scheme, the semi-functional key description is significantly changed. We will use the notation $X := f(X)$ to mean the value of a variable $X$ is changed to be $f(X)$ (where the input to $X$ is the old value of $X$). Throughout this section the notation $x \leftarrow X$ for a set $X$ will mean $x$ is uniformly sampled from $X$ unless there is further qualification.

**Semi-Functional Ciphertext.** Let $g_2$ be a generator of $\mathbb{G}_{p_2}$ (the generation procedure is the same no matter which generator is chosen). Generation for a semi-functional ciphertext generates $\text{Encrypt}(PK, M, S) = (C, C_0, (C_i : i \in S))$ and then generates $c \leftarrow \mathbb{Z}_N$ and $z_i \leftarrow \mathbb{Z}_N$ for all $i \in S$ and sets:

$$C := C_0 g_2^c, \quad C_i := C_i g_2^z_i \quad \forall \ i \in S.$$ 

and returns $(C, C_0, (C_i : i \in S))$. This is equivalent to generating $s, c, z_i \leftarrow \mathbb{Z}_N$ uniformly for all $i \in S$ and setting:

$$C = \text{Me}(g, g)^{as}, \quad C_0 = g^{as} g_2^c, \quad C_i = T_i^{as} g_2^z_i \quad \text{for all } i \in S.$$ 

In general we will use the ‘updating the values’ notation used in the first definition above as we believe it gives the clearest intuition as to how the generation process is modified.

**Semi-Functional Key.** An identifier $U$ has keys generated sem-functionally if the key generation process is modified as follows. Generate $\beta_{U,0}, \beta_{U,1} \leftarrow \mathbb{Z}_N$ uniformly at random once for this $U$ (if this $U$ has had a semi-functional key generated earlier, re-use those $\beta$ values). A key $SK^{(b)}_{U,(A,\rho)}$ is generated semi-functionally by first calling the real generation procedure

$$\text{KeyGen}(MSK, b, (A, \rho), U, PK)$$
then sampling a random vector \( v \in \mathbb{Z}_N^l \) subject to the restriction that \( v \cdot 1 = \beta_{U,b} \) and setting:

\[
K^{(1)}_i := K^{(1)}_i \times g^{A_i v}
\]

for all \( i \in [n] \).

**Definition.** \((\text{GAME}_{\text{REAL}})\) An adversary in \(\text{GAME}_{\text{REAL}}\) is interacting with the actual functionality as described in Piecewise KPABE Security\(_A\).

**Definition.** \((\text{GAME}_0)\) The response to the adversary’s queries in \(\text{GAME}_0\) will differ from \(\text{GAME}_{\text{REAL}}\) only in the challenge ciphertext phase. In \(\text{GAME}_0\), the challenge ciphertext will be generated as a semi-functional ciphertext.

**Lemma.** Let \(\epsilon_{\text{REAL}},\epsilon_0\) be the advantage of an adversary \(A\) in \(\text{GAME}_{\text{REAL}}\) and \(\text{GAME}_0\) respectively. If Assumption 1. holds then \(|\epsilon_{\text{REAL}} - \epsilon_0|\) is negligible in \(\lambda\).

**Proof.** Let \((\mathbb{G}, g, X_3, T)\) be a uniform instance from Assumption 1. where \(T\) is either from \(\mathbb{G}_{p_1p_2}\) or \(\mathbb{G}_{p_1}\) with equal probability, we will show how to reply to the adversary’s queries given only this information such that if it comes from the first distribution we are responding as a uniform instance of \(\text{GAME}_{\text{REAL}}\) and otherwise, we are responding as a uniform instance of \(\text{GAME}_0\).

Having a generator of \(\mathbb{G}_{p_3}, X_3\), and \(g\) suffices to reproduce the honest \text{Setup} and \text{KeyGen} queries completely by choosing \(\alpha, s_i \leftarrow \mathbb{Z}_N\) uniformly at random for all \(i \in \Omega\). The challenge ciphertext will be generated with the following modification:

On input \(M_0, M_1, S\) the algorithm will choose \(b \leftarrow \{0, 1\}\) at random. It will then set:

\[
C = M_b \cdot (g^\alpha, T), \quad C_0 = T, \quad C_i = T^{s_i}, \quad \forall i \in S
\]

Then, if \(T \in \mathbb{G}_{p_1}\) this is a properly distributed normal ciphertext but if \(T \in \mathbb{G}_{p_1p_2}\) the above is a property distributed semi-functional ciphertext (the \(\mathbb{G}_{p_1}\) and \(\mathbb{G}_{p_2}\) subgroups are orthogonal to each other in \(e\)), as desired since the value of any \(s_i\) value modulo \(p_2\) is
completely undetermined from the point of view of a distinguisher since it’s only used as an exponent for elements in $G_{p_1}$ in the public key.

**Definition.** (Game $k$) For this game, the keys corresponding to the first $k$ identifiers that are queried by the adversary have keys generated semi-functionally and the challenge ciphertext is semi-functional.

Notice that above we are not concerned with the strict order in which the adversary makes his queries, but instead with the first $k$ unique identifiers he queries. This means that if an identifier was among the first $k$ that $A$ has made a key generation query to, all subsequent keys for this identifier are generated semi-functionally. We let $q$ be an upper bound on the number of queries the adversary makes.

**Lemma.** Let $\epsilon_k$ be the advantages of an adversary $A$ in game GAME$_k$. For any $k \in [q]$, if Assumption 2. holds then $|\epsilon_k - \epsilon_{k+1}|$ is negligible in $\lambda$.

This statement will require a sequence of hybrids. We define an adversary $A$ to be **Type 1.** if for the $k^{th}$ identifier that it queries a key for (call this identifier $U$) and any $\text{KeyGen}(MSK, b, (A, \rho), U)$ query it makes in the security game with $b = 0$, $(A, \rho)(S^*) = 0$ where $S^*$ is the attribute set of the challenge ciphertext. Similarly, we define $A$ to be **Type 2.** if $(A, \rho)(S^*) = 0$ whenever $b = 1$. Note that while the adversary’s queries must fall in one of the above categories in order for the security game to not automatically output 0, the actual adversary does not fall in either of these cases. However, we will use this analysis to conclude our result for general adversaries.

**Definition.** We define the game $\mathcal{H}_i$ as follows. The challenge ciphertext and keys for the first $k - 1$ identifiers queried are generated semi-functionally while the keys for the identifiers after and including the $k + 1^{st}$ identifier are generated normally. The keys to the $k^{th}$ identifier $U$ are modified as follows:

For the first $i - 1$ queries to $\text{KeyGen}(MSK, 0, (A, \rho), U)$, after generating the key from
the normal distribution and then generate \( u_2 \leftarrow \mathbb{Z}_N \) and set for all \( j \in [n] \):

\[
K_j^{(1)} := K_j^{(1)} \times g_2^{A_j \cdot u_2}
\]

For the \( i \)th query to \( \text{KeyGen}(\text{MSK}, 0, (A, \rho), U) \), generate the key from the normal distribution and then generate \( u_2 \leftarrow \mathbb{Z}_N^l \), \( \gamma_j \leftarrow \mathbb{Z}_N \) and set for all \( j \in [n] \):

\[
K_j^{(1)} := K_j^{(1)} \times g_2^{A_j \cdot u_2 + \gamma_j / \rho(j)} \quad K_j^{(2)} := K_j^{(2)} \cdot g_2^\gamma_j
\]

Following the \( i \)th query, or to \( \text{KeyGen}(\text{MSK}, 1, (A, \rho), U) \) for the \( k \)th identifier \( U \), no modification is made to the normal key generation algorithm.

**Definition.** Define the game \( \mathcal{I}_i \) as \( \mathcal{H}_i \) where the \( i \)th query to \( \text{KeyGen}(\text{MSK}, 0, (A, \rho), U) \) is modified by setting \( \gamma_i = 0 \) rather than generating it uniformly over \( \mathbb{Z}_N \).

For the following analysis let \( \nu_i \) denote the advantage of the adversary in the security game \( \mathcal{H}_i \) and \( \mu_i \) the advantage of the adversary in \( \mathcal{I}_i \):

**Lemma.** If \( A \) is of Type 1. then \( |\mu_{i-1} - \nu_i| \) is negligible in \( \lambda \) if Assumption 2. holds.

**Proof.** We will describe how to embed an instance of the security game from Assumption 2. into either \( \mathcal{I}_{i-1} \) or \( \mathcal{H}_i \) depending on the distribution chosen in the Assumption 2. experiment. In the security game for Assumption 2. we are given \( G, g, X_1, X_2, X_3, Y_2, Y_3, T \) where \( T \) is either in \( G \) or \( G_{p_1 p_2} \). Generate \( \alpha \leftarrow \mathbb{Z}_N \) and \( s_i \leftarrow \mathbb{Z}_N \) for each attribute \( i \in \Omega \).

As the public key, generate and send to the adversary:

\[
PK = \{N, g, e(g, g)^\alpha, T_i = g^{s_i} \text{ for all } i \in \Omega\}
\]

The challenge ciphertext with attribute set \( S \) will be generated as:

\[
C = M_\beta e(g, g^sX_2)^\alpha, \quad C_0 = g^sX_2, \quad C_i = (g^sX_2)^{s_i}, \forall i \in S.
\]

- **The first \( k-1 \) identifiers queried to** \( \text{KeyGen}(\text{MSK}, b, U, (A, \rho)) \). If \( \alpha_U, \beta_U, 0, \beta_U, 1 \) has not yet been generated, choose them uniformly in \( \mathbb{Z}_N \). For all such queries choose
a random value \( r_j \in \mathbb{Z}_N \), and uniformly generate \( W_j, V_j \in G_{p_3} \) (using \( X_3 \)) for each row, and choose \( u \in \mathbb{Z}_N^l \) uniformly subject to the restriction that if \( b = 0 \), \( u \cdot 1 = \alpha_U \) and otherwise \( u \cdot 1 = \alpha - \alpha_U \). Generate additionally \( u_2 \leftarrow \mathbb{Z}_N^l \) so that \( u_2 \cdot 1 = \beta_{u,b} \) and set:

\[
K_j^{(1)} = g^{A_j u T_{\rho(j)}^r} W_i(Y_2 Y_3)^{A_j u_2}, \quad K_j^{(2)} = g^{r_j V_j}
\]

- **The \( k^{th} \) identifier where \( b = 0 \).** If \( \alpha_U \) has not yet been generated, choose it uniformly in \( \mathbb{Z}_N \). The first \( i - 1 \) times \( \text{KeyGen}(MSK, 0, (A, \rho), U) \) is called, choose \( u \) a random vector over \( \mathbb{Z}_N^l \) subject to the restriction that \( u \cdot 1 = \alpha_U \) and \( u_2 \leftarrow \mathbb{Z}_N^l \), \( V_j, W_j \leftarrow G_{p_3}, r_j \leftarrow \mathbb{Z}_N \) for all \( j \in [n] \) and set:

\[
K_j^{(1)} = g^{A_j u^{T\rho_j} W_j(Y_2 Y_3)^{A_j u_2}}, \quad K_j^{(2)} = g^{r_j V_j}
\]

For the \( i \)th query, choose \( u \) a random vector over \( \mathbb{Z}_N^l \) subject to the restriction that \( u \cdot 1 = \alpha_U \) and generate \( v \) randomly over \( \mathbb{Z}_N^l \) so that \( v \cdot 1 = 0 \) and \( V_j, W_j \leftarrow G_{p_3}, r_j \leftarrow \mathbb{Z}_N \) for all \( j \in [n] \) and set:

\[
K_j^{(1)} = g^{A_j u^{T\rho_j} W_j T_{\rho_j}^{s_{\rho(j)}}}, \quad K_j^{(2)} = T_{\rho_j}^{s_{\rho(j)}} V_j
\]

For queries after the \( i \)th query choose \( u \) a random vector over \( \mathbb{Z}_N^l \) subject to the restriction that \( u \cdot 1 = \alpha_U \) and \( V_j, W_j \leftarrow G_{p_3}, r_j \leftarrow \mathbb{Z}_N \) for all \( j \in [n] \) set:

\[
K_j^{(1)} = g^{A_j u^{T\rho_j} W_j}, \quad K_j^{(2)} = g^{r_j V_j}
\]

- **After the \( k^{th} \) identifier or the \( k^{th} \) identifier with \( b = 1 \).** If \( \alpha_U \) has not yet been generated, choose it uniformly in \( \mathbb{Z}_N \). Choose \( u \) a random vectors over \( \mathbb{Z}_N^l \) subject to the restriction that \( u \cdot 1 = \alpha_U \) if \( b = 0 \) or \( u \cdot 1 = \alpha - \alpha_U \) if \( b = 1 \) and \( V_j, W_j \leftarrow G_{p_3}, r_i \leftarrow \mathbb{Z}_N \) for all \( j \in [n] \) set:

\[
K_j^{(1)} = g^{A_j u^{T\rho_j} W_j}, \quad K_j^{(2)} = g^{r_j V_j}
\]
Note that when \( T \in \mathbb{G}_{p_1} \) the above method is distributed identically to an instance of \( \mathcal{I}_{i-1} \). However, when \( T \in \mathbb{G} \) it is modified in that for the \( i^{th} \) query rather than each \( K_j^{(1)} \) component being multiplied by \( g_2^{A_j \cdot u_2 + r_j s_{\rho(j)}} \) for a randomly generated \( u_2 \) it is multiplied by \( T_{G_{p_2}} \) (the \( \mathbb{G}_{p_2} \) part of \( T \)) raised to \( A_j \cdot u' + r_j s_{\rho(j)} \) where \( u' \) is generated uniformly at random subject to the restriction that \( 1 \cdot u = 0 \) (notice that the fact that \( u' \) is used as an exponent in \( \mathbb{G}_{p_1} \) does not affect this analysis as its value as an exponent in \( \mathbb{G}_{p_2} \) is independent of its value as an exponent in \( \mathbb{G}_{p_1} \)). This is different from the actual generation procedure in \( \mathcal{I}_i \) where \( u' \) is generated as a truly uniform vector, however we will show that these distributions are actually identical from the point of view of \( A \) because of the queries the adversary is allowed to make.

Notice that since \( s_{\rho(i)} \) is not used as an exponent in \( \mathbb{G}_{p_2} \) for attributes not in the challenge ciphertext, this is equivalent as when all \( j \) such that \( \rho(j) \) is not in the challenge ciphertext is simply multiplied by a uniform element in \( \mathbb{G}_{p_2} \) and all rows that do have \( \rho(j) \) in the challenge ciphertext are multiplied by \( T_{G_{p_2}} \) raised to \( A_i \cdot u' + r_i s_{\rho(j)} \), since \( 1 \) is not in the span of these \( A_i \) values, generating the challenge ciphertext by this procedure is equivalent to generating it by generating \( u' \) to be uniformly chosen vector as we demonstrate below.

We now argue that for a set of rows \( J \), if \( 1 \notin \text{Span}(A_j : j \in J) \), then the distributions on \((A_j \cdot u' : j \in J)\) is equivalent when either \( u' \) is chosen as a uniformly random \( l \) dimensional vector over \( \mathbb{Z}_N \) or as one subject to the restriction that \( u' \cdot 1 = 0 \). This will suffice to prove that in the case that \( T \in \mathbb{G} \) that the above generation process is equivalent to the process from \( \mathcal{I}_i \).

Notice that for any \(|J| \) dimensional vector \( \bar{z} \), if we let \( S \) be the set of vectors \( u'(\beta) \) such that \( \bar{z}[j] = A_j \cdot u' \) for all \( j \in J \) and \( u' \cdot 1 = \beta \), then by using \( f \) an \( l \) dimensional vector orthogonal to all \( A_j \) such that \( 1 \cdot A_j \neq 0 \), there is a one-to-one correspondence between \( u'(\beta) \) and any other \( u'(\beta') \). Then we have that the probability that \( A_j \cdot u' \) takes a fixed set of values for all \( j \in J \) restricted to the condition that \( u' \cdot 1 = 0 \) is \( u'(0)/p_2^{l-1} \). Similarly, the
probability it takes this set of values with no restrictions is \( \sum_{\beta \in \mathbb{Z}^l} u'(\beta)/p^2 \). By our previous observation on the equivalence of \( u'(\beta) \) and \( u'(\beta') \) for any \( \beta \neq \beta' \) these quantities are equal, as desired. We note that a similar argument is presented in [49].

**Lemma.** If \( A \) is of Type 1. then \( |\mu_i - v_i| \) is negligible in \( \lambda \) if Assumption 2 holds.

**Proof.** The only step that changes from the previous reduction is for the \( i^{th} \) query with \( b = 0 \) to the \( k^{th} \) identifier which is now generated as follows:

- **The \( k^{th} \) identifier.** For the \( i^{th} \) query where \( b = 0 \), choose \( u \) a random vector over \( \mathbb{Z}_N^l \) subject to the restriction that \( u \cdot 1 = \alpha_U \) and generate \( v \leftarrow \mathbb{Z}_N^l \) and \( V_j, W_j \leftarrow \mathbb{G}_{p_3} \), \( r_i \leftarrow \mathbb{Z}_N \) for all \( j \in [n] \) set:

\[
K_j^{(1)} = g^{A_j \cdot u(Y_2 Y_3)} A_j \cdot v W_j T^{r_j s_{\rho(j)}}, \quad K_j^{(2)} = T^{r_j} V_j
\]

The above is a uniform instance of \( H_i \) if \( T \in \mathbb{G} \) and \( I_i \) if \( T \in \mathbb{G}_{p_1 p_3} \) as desired.

**Definition.** Define the game \( J \) as a modification of \( I_q \) as follows. For the \( k^{th} \) identifier \( U \), rather than generating \( u_2 \) as a random vector for every KeyGen\((MSK, PK, 0, (A, \rho), U)\) query, generate a fixed value \( \beta_{U,0} \in \mathbb{Z}_N \) randomly for \( U \) (to be re-used across all such queries) and generate \( u_2 \) uniformly over \( \mathbb{Z}_N^l \) subject to the restriction that \( u_2 \cdot 1 = \beta_{U,0} \) rather than generating it completely uniformly.

Let \( \eta \) be the advantage of the advantage of the adversary \( A \) in \( J \).

**Lemma.** If \( A \) is of Type 1. then \( |\mu_q - \eta| \) is negligible in \( \lambda \) if Assumption 2 holds.

We remind the reader of the generation procedure in \( I_q \) and \( J \). In \( I_q \) and \( J \) the ciphertext and keys for the first \( k - 1 \) identifiers are all semi-functional and the keys corresponding to the \( k + 1^{st} \) and following identifiers are generated according to the true distribution. In \( I_q \) and \( J \) the \( k^{th} \) identifier has keys generated as below:

- **The \( k^{th} \) identifier in \( I_q \).** For \( b = 1 \) the standard key generation algorithm is used.

  For \( b = 0 \), after the standard key generation algorithm, \( u_2 \leftarrow \mathbb{Z}_N^l \) (new for every query
to this identifier) and after the normal key generation, the $K_j$ components are modified as:

$$K_j^{(1)} := K_j^{(1)} \times g_{A_j}^u$$

- **The $k^{th}$ identifier in $\mathcal{J}$.** For $b = 1$ the standard key generation algorithm is used.

For $b = 0$, if $\beta_{U,0}$ has not yet been generated, it is generated uniformly from $\mathbb{Z}_N$. After the standard key generation algorithm $u_2$ is generated uniformly over $\mathbb{Z}_N^l$ subject to the restriction that $u_2 \cdot 1 = \beta_{U,0}$ and the $K_j$ components are modified as:

$$K_j^{(1)} := K_j^{(1)} \times g_{A_j}^{u_2}$$

**Proof.** For this we will define modified versions of $\mathcal{I}_i$ and $\mathcal{H}_i$. In both modifications, the net change is only to the $K_j^{(1)}$ components of the keys for the $k^{th}$ identifier where $b = 0$. In $\mathcal{T}'_i$ and $\mathcal{H}'_i$ the values are all drawn as in $\mathcal{I}_i$ and $\mathcal{H}_i$ except that an additional value $\beta_{U,0}$ is chosen uniformly from $\mathbb{Z}_N$ once when the $k^{th}$ identifier $U$ is queried to the key generation algorithm and the below modification is made:

- **Queries to the $k^{th}$ identifier with $b = 0$ for $\mathcal{T}'_i$ or $\mathcal{H}'_i$.** Generate the key as in the $\mathcal{I}_i$ or $\mathcal{H}_i$ respectively. Generate $v$ uniformly over $\mathbb{Z}_N^l$ subject to the restriction that $v \cdot 1 = \beta_{U,0}$ and set for all $j \in [n]$:  

$$K_j^{(1)} := K_j^{(1)} \times g_{A_j}^{u_2 v}$$

Notice that $\mathcal{T}'_0 = \mathcal{H}'_0 = \mathcal{J}$ and $\mathcal{T}'_q = \mathcal{T}'_q'$ since the completely random $u_2$ factor subsumes the additional $v$ factor introduced in $\mathcal{T}'_q'$. Define similarly to before, $\nu'_i$ to be the advantage of $A$ in $\mathcal{H}'_i$ and $\mu_i$ to be its advantage in $\mathcal{T}'_i$.

**Lemma.** If $A$ is of Type 1, then $|\mu_{i-1}' - \nu'_i|$ is negligible in $\lambda$ if Assumption 2. holds.

This lemma proceeds identically to the proof for $\mu_{i-1}$ and $\nu_i$ with the following modification: At the beginning of the security game generate $\beta$ uniformly from $\mathbb{Z}_N$. For all queries to
the $k^{th}$ identifier where $b = 0$, generate a uniform $v$ such that $v \cdot 1 = \beta$ and after generating the $K_j^{(1)}$ component as in the previous proof, modify it as:

$$K_j^{(1)} := K_j^{(1)}(Y_2Y_3)^{A_j \cdot v}$$

It’s simple to check using the argument from the previous lemma that this provides the same indistinguishability guarantee between $I_{i-1}'$ and $H_i'$. Similarly, by reproducing the argument from the next lemma we get:

**Lemma.** If $A$ is of Type 1. then $|\mu_i' - \nu_i'|$ is negligible in $\lambda$ if Assumption 2. holds.

Combining these lemmas with the previous two allows us to conclude that $I_0'$ is indistinguishable from $I_0' = J$. Let $\eta$ be the advantage of $A$ in $J$. We can now conclude:

**Corollary.** If $A$ is of Type 1. then $|\epsilon_{k-1} - \eta|$ is negligible in $\lambda$ if Assumption 2. holds.

We now are ready to connect our hybrids to $\text{GAME}_k$:

**Lemma.** If $A$ is of Type 1. then $|\eta - \epsilon_k|$ is negligible in $\lambda$ if Assumption 2 holds.

**Proof.** At a high level, we will be using the $\mathbb{G}_{p_1}$ part of $T$ as $g^{\alpha U}$. We generate the public key and challenge ciphertext as in the previous two reductions.

- **The first $k-1$ identifiers queried to $\text{KeyGen}(MSK, b, U, (A, \rho))$.** If $\alpha_U, \beta_U, 0, \beta_U, 1$ have not yet been generated, choose them uniformly in $\mathbb{Z}_N$. For all such queries pick $r_j \leftarrow \mathbb{Z}_N$, $W_j, V_j \leftarrow \mathbb{G}_{p_3}$ for each row, and choose $u \in \mathbb{Z}_N^l$ uniformly subject to the restriction that if $b = 0$ $u \cdot 1 = \alpha_U$ and otherwise $u \cdot 1 = \alpha - \alpha_U$. Generate additionally $u_2$ uniformly over $\mathbb{Z}_N^l$ so that $u_2 \cdot 1 = \beta_{U,b}$, and set for all $j \in [n]$:

$$K_j^{(1)} = g^{A_j \cdot u}T_{\rho(j)}^{r_j}W_j(Y_2Y_3)^{A_j \cdot u_2}, \quad K_j^{(2)} = g^{r_j}V_j$$

- **The $k^{th}$ identifier.** If $\beta_{U,0}$ has not yet been generated, choose it uniformly in $\mathbb{Z}_N$. For queries to $\text{KeyGen}(MSK, 0, (A, \rho), U)$. Generate $u$ at random over $\mathbb{Z}_N^l$ so that $u \cdot 1 = 1$
and generate $u_2$ uniformly so that $u_2 \cdot 1 = \beta_{U,0}$ generate $V_j, W_j \leftarrow \mathbb{G}_{p_3}$ for all $j \in [n]$ and set:

$$K_j^{(1)} = T^{A_j \cdot u_1^T} W_j (Y_2 Y_3)^{A_j \cdot u_2}, \quad K_j^{(2)} = g^{r_j} V_j$$

- For queries to $\text{KeyGen}(MSK, 1, (A, \rho), U)$, generate $u, u_2$ at random over $\mathbb{Z}_N^l$ so that $u \cdot 1 = \alpha$ and $u_2 \cdot 1 = -1$ generate $V_j, W_j \leftarrow \mathbb{G}_{p_3}$ for all $j \in [n]$ and set:

$$K_j^{(1)} = g^{A_j \cdot u_1^T} W_j (Y_2 Y_3)^{A_j \cdot u_2}, \quad K_j^{(2)} = g^{r_j} V_j$$

- **After the $k^{th}$ identifier.** If $\alpha_U$ has not yet been generate, choose it uniformly in $\mathbb{Z}_N$. Generate $u$ at random over $\mathbb{Z}_N^l$ so that $u \cdot 1 = \alpha_u$ if $b = 0$ and $u \cdot 1 = \alpha - \alpha_U$ if $b = 1$. Generate $r_j \leftarrow \mathbb{Z}_N, V_j, W_j \leftarrow \mathbb{G}_{p_3}$ for all $j \in [n]$ and set:

$$K_j^{(1)} = g^{A_j \cdot u_1^T} W_j (Y_2 Y_3)^{A_j \cdot u_2}, \quad K_j^{(2)} = g^{r_j} V_j$$

When $T \in \mathbb{G}_{p_1 p_3}$ the above is a uniform instance of $\mathcal{J}$ where the $\mathbb{G}_{p_1}$ part of $T$ is implicitly set to be $\alpha_U$ for the $k^{th}$ identifier $U$. Similarly, if $T \in \mathbb{G}$ it is a uniform instance of $\text{GAME}_k$ where the $\mathbb{G}_{p_1}$ part of $T$ is set to be $\alpha_U$, the $\beta_{U,0}$ value for the $k^{th}$ identifier $U$ is actually set to be the $Y_2$ logarithm of the $\mathbb{G}_{p_2}$ part of $T$ plus the $\beta_{U,0}$ value in the construction and $-\beta_{U,1}$ is implicitly set to be the $Y_2$ logarithm of the $\mathbb{G}_{p_2}$ part of $T$.

Therefore we have that $\text{GAME}_{k-1}$ is indistinguishable from $\text{GAME}_k$ to **Type 1**. adversaries - and through a symmetric repetition of the above argument that it is also indistinguishable to **Type 2**. adversaries (where the $\mathcal{H}_i, \mathcal{I}_i, \mathcal{H}_i', \mathcal{I}_i'$ hybrids now modify the generation procedure when $b = 1$). If there was a general adversary $A$ that could distinguish between $\text{GAME}_k$ and $\text{GAME}_{k+1}$ it would be possible to create either a **Type 1**. or **Type 2**. adversary that could also distinguish by creating the two adversaries $A_1$ and $A_2$ where $A_1$ aborts is $A$ ever makes a query that violates the condition of being **Type 1**. and guesses randomly and similarly for $A_2$. Therefore the above implies that $\text{GAME}_k$ and $\text{GAME}_{k+1}$ is indistinguishable for general adversaries. By a hybrid argument we have:
Corollary. \(|\epsilon_0 - \epsilon_q|\) is negligible in \(\lambda\) if Assumption 2. holds.

We briefly remind the reader of the modified key and ciphertext generation in Game\(_q\).

- **Setup and the Public Key.** Choose a bilinear group of order \(N = p_1p_2p_3\) and \(\alpha \leftarrow \mathbb{Z}_N\) and \(g \leftarrow \mathbb{G}_{p_1}, g_2 \leftarrow \mathbb{G}_{p_2}\) uniformly. For each \(i \in \Omega\) pick \(s_i \leftarrow \mathbb{Z}_N\), \(X_3 \leftarrow \mathbb{G}_{p_3}\) uniformly and set:

\[
PK = (N, e(g, g)^\alpha, X_3, T_i = g^{s_i})
\]

- **Key Generation.** On a query to KeyGen(MSK, b, U, (A, \(\rho\)), PK) proceed as follows:
  
If \(\alpha_U\) or \(\beta_{U,b}\) has not been generated yet, generate it and store it. Pick \(u\) and \(v\) random vectors over \(\mathbb{Z}_N\) so that \(u \cdot 1 = \alpha_U\) if \(b = 0\) and \(u \cdot 1 = \alpha - \alpha_U\) if \(b = 1\) and \(v\) so that \(v \cdot 1 = \beta_{U,b}\). Pick \(W_i, V_i\) randomly for all \(i \in [n]\) from \(\mathbb{G}_{p_3}\) and set, for all \(i \in [n]\):

\[
K_i^{(1)} = g^{A_i \cdot u} g_2^{A_i \cdot v} T_{\rho(i)} W_i, \quad K_i^{(2)} = g^{r_i} V_i
\]

- **The Challenge Ciphertext.** With challenge message \(M\) and attribute set \(S\). Pick \(s, c, z_i \leftarrow \mathbb{Z}_N\) at random for all \(i \in S\) and return:

\[
C = Me(g, g)^{as}, \quad C_0 = g^s g_2^c, \quad C_i = T_i s_i z_i \forall i \in S
\]

**Definition.** Let Game\(_{\text{Final}}\) be a modification of Game\(_q\) where a random message is encrypted in the challenge phase rather than the challenge message.

**Lemma.** For any \(A\), \(|\epsilon_q - \epsilon_{\text{Final}}|\) is negligible if Assumption 3. holds.

**Proof.** Recall that in the Assumption 3. security game the distinguisher is given \(g, g^\alpha X_2, X_3, g^s Y_2, Z_2, T\) where \(\alpha, s \leftarrow \mathbb{Z}_N\), \(X_2, Y_2, Z_2 \leftarrow \mathbb{G}_{p_2}\), \(X_3 \leftarrow \mathbb{G}_{p_3}\) and \(T\) is either \(e(g, g)^{as}\) or randomly chosen from \(\mathbb{G}_T\). Our reduction proceeds as follows:

- **Setup and the Public Key.** Generate the public key as:

\[
PK = (N, g, X_3, e(g, g)^\alpha = e(g, g^\alpha X_2))
\]
- **Private Key Queries.** On a query to generate $\text{KeyGen}(MSK, b, (A, \rho), U)$ proceed as follows: If $\beta_{U,b}$ or $\alpha_U$ hasn’t been chosen yet, generate it uniformly from $\mathbb{Z}_N$ and pick $u, u_2$ uniformly over $\mathbb{Z}_N^l$ subject to the restriction that $u \cdot 1 = 1$ and $u_2 \cdot 1 = \beta_{U,b}$, and pick $V_j, W_j$ randomly in $G_{p_3}$ for all $j \in [n]$. If $b = 0$ set for all $j \in [n]$:

$$K_j^{(1)} = g^{\sum_{j'=2}^{l} A_j[j']u[j']}(g^{\alpha_U})^{A_j[1]} Z_2^{A_ju_2 \rho_{(j)}^p} W_j , \quad K_j^{(2)} = g^{r_j}V_j$$

If $b = 1$ follow the same procedure but set:

$$K_j^{(1)} = g^{\sum_{j'=2}^{l} A_j[j']u[j']}(g^{-\alpha_U}(g^X_{2}))^{A_j[1]} Z_2^{A_ju_2 \rho_{(j)}^p} W_j , \quad K_j^{(2)} = g^{r_j}V_j$$

- **The Challenge Ciphertext.** To generate the challenge ciphertext with messages $(M_0, M_1)$ and attribute set $S$ pick a random bit $b$ and set:

$$C = M_b T , \quad C_0 = g^s X_2 , \quad C_i = (g^s X_2)^{s_i} \forall i \in S$$

In the case where $T = e(g, g)^{as}$ let us analyze the distribution of the exponent in $K_j^{(1)}, K_j^{(2)}$ when $b = 0$. In this case (if we call $X_2 = g^c_{2}$ and $Z_2 = g^d_{2}$) the above generation process is equivalent to generating a key from $\text{GAME}_q$ where $v$ is chosen so that the $\beta_{U,b}$ value is set to be $\beta_{U,b} + c$ in the new generation procedure. Similarly, if $T$ is a random target group element this is an encryption of a randomly chosen message.

As the advantage of any adversary in $\text{GAME}_{\text{FINAL}}$ is clearly negligible (it has no dependence on $b$) we have $\epsilon_{\text{FINAL}}$ is negligible which proves the original claim.

### 2.7 Revocable Storage CP-ABE

We now describe the requirement needed for a CP-ABE scheme to imply a revocable CP-ABE scheme and provide a construction. We first formally define security for a revocable CP-ABE scheme. Once again we begin by defining some oracles for use in the security game:
**Security Game Oracles.** Define the following oracles to use in the security game. These oracles are given access to \((PK, MSK, \sigma)\) that are generated at the beginning of the security game, and may have been modified since, at the time of the oracle’s invocation.

1. The Secret Key Generation oracle \(SK(\cdot, \cdot)\) takes as input \((S, ID)\) and return \(SK_{S, ID}\) generated from:
   \[
   (SK_{S, ID}, \sigma) \leftarrow \text{KeyGen}(MSK, S, ID, \sigma).
   \]

2. The Key Update Generation oracle \(K(\cdot, \cdot)\) takes as input \(t\) and a revocation list \(rl\) and returns \(K_t\):
   \[
   (K_t, \sigma) \leftarrow \text{KeyUpdate}(MSK, t, rl, \sigma)
   \]
   Only \(K_t\) is sent to the adversary while \(\sigma\) is used to update the current \(\sigma\) value.

The security definition now is similar to the key-policy case.
RCP-Security$_A(1^λ)$:

1. The challenger runs $\text{Setup}(1^λ) → (PK, MSK, σ)$ and returns $PK$ to $A$;

2. $A$ is given oracle access to $SK(·, ·)$, $K(·, ·)$ until it signals the query phase is over;

3. After the query phase, $A$ returns $(M_0, M_1, P^*, t^*)$ to the challenger;

4. The challenger picks $b$ a random bit and returns to $A$:

   $$C_{P^*,t^*} ← \text{Encrypt}(PK, M_b, P^*, t^*)$$

5. $A$ is once again given oracle access to the three oracles above;

6. $A$ returns a bit $b'$. The Experiment returns 1 if and only if $b' = b$ and the conditions below concerning the adversary’s query history are satisfied.

The conditions placed on the adversary’s queries is as follows: For any query, $SK(S, ID)$ such that $P^*(S) = 1$, $ID ∈ rl$ for every query $K(t, rl)$ with $t ≥ t^*$.

**Definition.** A Revocable CP-ABE scheme is secure if for any polynomial time adversary $A$ the advantage of this adversary in the RCP-Security game:

$$2 \Pr [\text{RCP-Security}_A(1^λ) = 1] − 1$$

is negligible in $λ$.

**Definition.** A CP-ABE scheme is said to have piecewise key generation with attribute set $Ω$ supporting policies in $P$, message space $M$ and identifier length $I$ if key generation and encryption are modified to the following syntax:
**KeyGen** takes as input the master key $MSK$, a bit $b$, a set $S \subseteq \Omega$ and an identifier $U \in \{0, 1\}^z$ and returns:

- $\textbf{KeyGen}(MSK, b, S, U) \rightarrow K^{(b)}_{S,U}$.

**Decrypt** takes two outputs of **KeyGen** as input instead of one:

- $\textbf{Decrypt}(C_S, K_0, K_1) \rightarrow M$

**Definition. (Correctness)** A CP-ABE scheme with piecewise key generation is correct if for any $P \in \mathcal{P}$:

- $(PK, MSK) \leftarrow \textbf{Setup}(1^\lambda)$
- $C_P \leftarrow \textbf{Encrypt}(PK, M, P)$
- $S, T \subset \Omega$ such that $P(S) = 1$ and $P(T) = 1$:

If $\textbf{KeyGen}(MSK, 0, S, U) \rightarrow K^{(0)}_{S,U}$ and $\textbf{KeyGen}(MSK, 1, T, U) \rightarrow K^{(1)}_{T,U}$, then,

$$\textbf{Decrypt}(K^{(0)}_{S,U}, K^{(1)}_{T,U}, C_{(A,\rho)}) = M.$$  

Security for a scheme with piecewise key generation now follows similarly to the KP-ABE definition:
**Piecewise CPABE Security**$_A(1^\lambda)$:

1. The challenger runs **Setup**$(1^\lambda) \rightarrow (PK, MSK)$ and sends $PK$ to $A$;
2. $A$ makes queries of the type $(b, S, U)$ for $b \in \{0, 1\}, S \subset \Omega$ and $U \in \{0, 1\}^T$;
   The challenger runs KeyGen$(MSK, b, S, U)$ and returns the key to $A$;
3. $A$ signals the query phase is over and returns $(M_0, M_1, P)$;
4. The challenger picks $b \leftarrow \{0, 1\}$ and returns Enc$(PK, M_b, P)$ to the adversary;
5. $A$ has another query phase as previously;
6. $A$ sends a bit $b'$ to the challenger;
7. If for any $U$, $P(S) = P(T) = 1$ for some $S, T \subset \Omega$ s.t. $A$ has queried KeyGen$(MSK, 0, S, U)$ and KeyGen$(MSK, 1, T, U)$ return 0.
8. If $b' = b$ return 1, otherwise return 0.

**Definition.** A CP-ABE scheme with piecewise key generation is secure if for any polynomial time adversary $A$ the advantage of this adversary in the **Piecewise CPABE Security** game:

$$2 \Pr[\text{Piecewise CPABE Security}_A(1^\lambda) = 1] - 1$$

is negligible in $\lambda$.

**Revocable Storage CP-ABE from Piecewise CP-ABE with Delegation**

Using a CP-ABE scheme with piecewise key generation $\mathcal{E}$ that supports elementary ciphertext manipulations and policies that include injective LSSS matrices, we will build a Revocable Storage CP-ABE scheme $\mathcal{F}$. First, we will split the attribute set of $\mathcal{E}$ as $\Omega \cup \Omega'$ where:

$$\Omega' = \{\omega_{i,b} : i \in [\log(T)], b \in \{0, 1\}\}$$

The construction of $\mathcal{F}$ is as follows. Let the tree $\mathcal{U}$ be defined as before of depth $I$ with all identifiers corresponding to leaf nodes and $\mathcal{U}(rl)$ defined as before, and $\mathcal{T}$ the tree
with leaves numbered $[T]$ with the subsets $T_t$ also defined as before. We now show how to
handle the policy and delegation by constructing the ‘time-policy’ in a delegatable fashion
from injective LSSS matrices. Once again we let $r = \log(T)$ and assume for notational
convenience that $T$ is a power of two.

Let $B$ be the $2r \times r$ matrix such with $2 \times 2$ blocks of the all ones matrix cascading along
the diagonal with all zeros elsewhere. Below is an example of this construction when $r = 3$.

\begin{equation*}
B = \begin{pmatrix}
1 & 1 & 0 \\
1 & 1 & 0 \\
0 & 1 & 1 \\
0 & 1 & 1 \\
0 & 0 & 1 \\
0 & 0 & 1
\end{pmatrix}
\end{equation*}

Label the rows of the matrix $B$ as $b_{1,0}, b_{1,1}, b_{2,0}, b_{2,1}, \ldots, b_{r,0}, b_{r,1}$ in descending order. The
relevant property of the construction of $B$ and this ordering that we will need is that the
vector 1 will be in the span of a set of the rows $S$ if and only if for each $i \in [r]$ either $b_{i,0} \in S$
or $b_{i,1} \in S$. We now define the labeling $\beta$ as taking $\beta(b_{i,d}) \rightarrow \omega_{i,d}$ for all rows of $B$. Note
that $(B, \beta)$ now describe an LSSS policy that is satisfied by a set of attributes $S$ if and only
if for each $i \in [r]$, $\omega_{i,b} \in S$ for some $b \in \{0, 1\}$. Notice that $(B, \beta)$ is injective.

We now describe the time policies corresponding to nodes of the tree. Recall each $y \in T$ admits a natural string representation of length $r$ where * values are used to pad the
description of the path from the root to this node to $r$ bits. We describe the policy $(B_y, \beta_y)$
now. For each index $i \in [r]$ if $i$ is not * we will eliminate the row $b_{i,\overline{y[i]}}$ from the rows of $B$
(keeping the output of $\beta_y$ the same as $\beta$ on all non-deleted rows). As an example if $r = 3$
and $y = 01*$, the matrix $B$ and labeling $\beta_y$ (represented by the arrows below) are set to be:
Notice that \((B_y, \beta_y)\) is satisfied by a set of attributes \(S\) if and only if for all \(i\) such that \(y[i] \neq \ast\), \(\omega_{i,y[i]} \in S\) and for each \(i\) such that \(y[i] = \ast\) either \(\omega_{i,0}\) or \(\omega_{i,1}\) is in \(S\). Additionally, any scheme that allows elementary ciphertext manipulations can delegate any \((B_y, \beta_y) \rightarrow (B_{y'}, \beta_{y'})\) for any descendent \(y'\) of \(y\) (as shown in Section 2.4 by use of the Delete operation). Additionally, notice that \((B_y, \beta_y)(s_z) = 1\) (as defined in Section 2.5) if and only if \(y\) is an ancestor of \(z\).

We now begin the description of our scheme. We use the notation \((A, \rho) \lor (B, \beta)\) for two LSSS policies to mean the LSSS policy that is made by sequentially adding the rows of \(B\) with assignment \(\beta\) to the matrix \(A\) (and padding both to the same length). Furthermore, if \((A, \rho)\) and \((B, \beta)\) are injective with disjoint ranges, \((A, \rho) \lor (B, \beta)\) is injective as well and if \(y'\) is a descendent of \(y\), \((A, \rho) \lor (B_{y'}, \beta_{y'})\) can be delegated from \((A, \rho) \lor (B_y, \beta_y)\). Notice that in contrast to our use of \(\lor\) in the definition, this policy may be satisfied by a set of attributes even if this set doesn’t satisfy either of the policies individually, but in our application below we will assign attributes relevant to each half disjointly and therefore, a set will only satisfy the entire policy if it satisfies one of the two halves.

- **Setup**\((1^\lambda)\): Return \(E\). \(\text{Setup}(1^\lambda) \rightarrow (PK, MSK)\).

- **KeyGen**\((MSK, S, ID)\): For all \(x \in \text{Path}(ID)\) run \(E\). \(\text{KeyGen}(MSK, 0, S, x) \rightarrow SK_{S,x}^{(0)}\).

  Return:

  \[SK_{S,1D}^{(0)} = \{SK_{S,x} : x \in \text{Path}(ID)\}\].

59
- **Encrypt**(\(PK, M, (A, \rho), t\)): For each \(y \in T_t\), set:

\[
C_{(A, \rho), y} = \mathcal{E}.\text{Encrypt}(PK, M, (A, \rho) \lor (B_y, \beta_y)).
\]

Return:

\[
C_{(A, \rho), t} = \{C_{(A, \rho), y} : y \in T_t\}.
\]

- **KeyUpdate**(\(MSK, rl, t\)): For all \(x \in \mathcal{U}(rl)\) set:

\[
SK_{s_t, x}^{(1)} = \mathcal{E}.\text{KeyGen}(MSK, 1, s_t, x) \text{ where } s_t = \{\omega_{i, t[i]} : i \in [r]\}
\]

Return \(K_t = \{SK_{t, x}^{(1)} : x \in \mathcal{U}(rl)\}\).

- **Decrypt**(\(C_{(A, \rho), t}, SK_{S, ID}, K_{t'}\)): If user \(ID \not\in rl\) when \(K_{t'}\) was created there is some node of \(x \in \mathcal{U}(rl) \cap \text{Path}(ID)\). For this \(x\) there is:

\[
SK_{S, x}^{(0)} \in SK_{S, ID}^{(0)} \text{ and } SK_{s_{t'}, x} \in K_{t'}
\]

Additionally if \(t' \geq t\) this implies there is some \(y \in T_t\) such that \(y\) is an ancestor of \(t'\) and therefore \((B_y, \beta_y)(s_{t'}) = 1\). For this \(y\), take \(C_{(A, \rho), y} \in C_{(A, \rho), t}\) and return:

\[
\mathcal{E}.\text{Decrypt}(SK_{S, x}^{(0)}, SK_{s_{t'}, x}, C_{(A, \rho), y})
\]

If \((A, \rho)(S) = 1\) then \((A, \rho) \lor (B_y, \beta_y)(S) = (A, \rho) \lor (B_y, \beta_y)(s_{t'}) = 1\) which implies that decryption succeeds.

- **CTUpdate**(\(PK, C_{(A, \rho), t}\)): For all \(x \in T_{t+1}\) find \(y \in T_t\) such that \(y\) is an ancestor of \(x\) and such that there is a \(C_{(A, \rho), y}\) component in \(C_{(A, \rho), t}\). For all such \(x\) set:

\[
C_{(A, \rho), x} = \mathcal{E}.\text{Delegate}(PK, C_{(A, \rho), y}, (A, \rho) \lor (B_x, \beta_x))
\]

And finally return:

\[
C_{(A, \rho), t+1} = \{C_{(A, \rho), x} : x \in T_{t+1}\}
\]
Theorem. If $E$ is a secure CP-ABE with piecewise key generation that allows elementary ciphertext manipulations, $F$ described above is a secure CP-ABE scheme with revocable storage supporting injective LSSS matrices.

Proof of RCP-ABE Security. Let $A$ be an adversary such that $RCP$-security$_A(1^\lambda)$ is non-negligible and we will construct an $A'$ such that $Piecewise CP$-ABE$_A(1^\lambda)$ is non-negligible. Once again, similar to the KP-ABE case we consider a modified security game where the adversary returns a tuple of policies $(P_1^*, P_2^*, \ldots, P_\rho^*)$ along with two messages $(M_0, M_1)$ and a random bit $b$ is chosen and the adversary receives an encryption of $M_b$ under each of these policies. Security should hold as long as the identities for any key with sufficient credentials to decrypt any of the challenge policies is on every revocation list equal to or exceeding the challenge time.

$A'$ initializes the $Piecewise CP$-ABE$_A$ security game and forwards $PK$ to $A$ (note that the attribute set of the $Reversible CP$-ABE scheme that $A$ is interacting with is $\Omega$ and therefore all keys and policies sent by $A$ do not involve $\Omega'$). To respond to a $SK(S, ID)$ query, $A'$ queries $(0, S, x)$ to its key generation oracle for all $x \in Path(ID)$, simulating the oracle of $A$. Similarly for all queries $K(t, rl)$, $A'$ sends a query $(1, s_t, x)$ for all $x \in U(rl)$ and combined them to answer the key update query of $A$.

To respond to the challenge query of $A$, which we call $(M_0, M_1, P^*, t^*)$, $A'$ sends as its challenge query $(M_0, M_1)$ and the tuple of policies $P^* \lor (B_y, \beta_y)$ for all $y \in T_{t^*}$ in the modified security game we described above. It now remains to only show that for any $x \in U$ and any $P^* \lor (B_y, \beta_y)$ queried in the challenge phase, $A'$ does not query its oracle on two values $(0, S, x)$ and $(1, S', x)$ such that $P^* \lor (B_y, \beta_y)(S) = P^* \lor (B_y, \beta_y)(S') = 1$.

Fix any $y \in T_t$, we will analyze each $P^* \lor (B_y, \beta_y)$ separately. Begin by considering an $x$ such that no descendent leaf $ID$ of this node is queried on a set $SK(S, ID)$ such that $P^*(S) = 1$. In this case, $(0, S, x)$ is only queried on sets by $A'$ such that $P^*(S) = 0$ and therefore, since $(B_y, \beta_y)$ does not depend on the attributes in $\Omega$, $P^* \lor (B_y, \beta_y)(S) = 0$ as
desired.

Next, consider an \( x \) such that a descendent leaf \( ID \) of \( x \) is queried on a set \( SK(S, ID) \) such that \( P^*(S) = 1 \). Then, \( ID \) must be included on \( rl \) for all queries by \( A \) to \( K(t, rl) \) where \( t \geq t^* \). Therefore, for all the \( (1, s_t, z) \) queries that \( A' \) makes, either \( t < t^* \) (in which case \( P^* \lor (B_y, \beta_y)(s_t) = 0 \) because \( y \) is from \( T_t^* \) which does not contain an ancestor of \( t \)), or \( z \neq x \) since no ancestor of \( ID \) is included on these queries when \( ID \in rl \). Therefore, for all queries \( (1, S, x) \) we have \( P^* \lor (B_y, \beta_y)(S) = 0 \), completing the proof.

2.8 Piecewise CP-ABE Construction

In this section we give our construction of a CP-ABE scheme with piecewise key generation from Assumptions 1, 2 and 3. Throughout this construction we assume the size of all queried policies are \( n \times l \) for notational convenience but this can be extended in a straightforward manner.

- **Setup(1^\lambda) \rightarrow (PK, MSK)**: Choose a bilinear group of order \( N = p_1p_2p_3 \) (from the definition of Assumptions 1, 2 and 3) according to \( G(1^\lambda) \). Then choose \( \alpha, a \leftarrow \mathbb{Z}_N \) and \( g \leftarrow \mathbb{G}_{p_1} \) uniformly. For each \( i \in \Omega \), pick \( s_i \leftarrow \mathbb{Z}_N \) and set \( MSK = (\alpha, X_3) \) with \( X_3 \leftarrow \mathbb{G}_{p_3} \) such that \( X_3 \neq 1 \):

  \[
  PK = (N, g, g^\alpha, e(g, g)^\alpha, \{T_i = g^{s_i} \text{ for all } i \in \Omega\})
  \]

- **KeyGen(MSK, b, S, U)**. If \( \alpha_U \) has not been generated yet, select it uniformly from \( \mathbb{Z}_N \); if it has been for a previous query, re-use this value. Generate \( t_U \leftarrow \mathbb{Z}_N \) and \( V, Z \leftarrow \mathbb{G}_{p_3} \) and \( W_i \leftarrow \mathbb{G}_{p_3} \) for each \( i \in S \). Let \( \gamma = \alpha_U \) if \( b = 0 \) and \( \alpha - \alpha_U \) if \( b = 1 \). Return \( SK_{S,U}^{(b)} = (K_U, K_S,U, K_U^*) \) generated as:

  \[
  K_U = g^{t_U}V \quad , \quad K_{S,U} = \{K_{U,i} = T_i^{t_U}W_i : i \in S\} \quad , \quad K_U^* = g^\gamma g^{at_U}Z.
  \]
**Encrypt**\((PK, M, (A, \rho))\). Choose \(v \leftarrow \mathbb{Z}_N^l\) and \(r_i \leftarrow \mathbb{Z}_N\) for all \(i \in [n]\), label the components as \(v = (s, v_2, \ldots, v_n)\) and set:

\[
C = Me(g, g)^{as}, \quad C' = g^s
\]

\[
C_i = g^{a_A \cdot v_T - r_i}, \quad D_i = g^{r_i} \text{ for all } i \in [n]
\]

Return:

\[
C(A, \rho) = (C, C', (C_i, D_i : i \in [n]))
\]

**Decrypt**\((C(A, \rho), SK_{S, U}^{(0)}, SK_T^{(1)})\). The decryption algorithm first computes \(\omega_i \in \mathbb{Z}_N\) such that \(\sum_{\rho(i) \in S} \omega_i A_i = 1\). Then, writing \(SK_{S, U}^{(0)} = (K_U, K_{S, U} = (K_{i, U} : i \in S), K_U^*)\) it generates:

\[
\prod_{\rho(i) \in S} (e(C_i, K_U)e(D_i, K_{U, \rho(i)}))^{\omega_i} = e(g, g)^{sat_U}
\]

\[
e(K_U^*, C') = e(g^{ou}, g^*)e(g^{at_U}, g^*)
\]

Which allows us to recover \(e(g^{ou}, g^*)\). Similarly, with \(SK_T, U\) we can recover \(e(g^{a - ou}, g^*)\) which allows us to recover \(e(g, g)^{sa}\) and \(M\) as \(C/e(g, g)^{sa}\).

We now begin the proof of security. For this we once again define notions of semi-functionality for ciphertexts and keys.

**Semi-Functional Ciphertext**. For a semi-functional ciphertext, we add a multiplicative factor to some of the components according to the below algorithm. After computing the ciphertext honestly through \(Encrypt(PK, M, (A, \rho)) = (C, C', (C_i, D_i : i \in [n]))\), make the following modifications. First, generate \(c \leftarrow \mathbb{Z}_N\) at random and set:

\[
C' := C' g_2^c
\]

Next generate a random vector \(u \leftarrow \mathbb{Z}_N^l\) and for all \(i \in [n]\) generate \(\gamma_i \leftarrow \mathbb{Z}_N\) and set:

\[
C_i := C_i \times g_2^{A_i \cdot u - \gamma_i \rho(i)}, \quad D_i := D_i \times g_2^\gamma_i
\]
The semi-functional ciphertext is now set as $C_{(A,\rho)} = (C, C', (C_i, D_i : i \in [n]))$.

**Semi-Functional Key.** An identifier $U$ has keys generated *semi-functionally* if the following change is made to all queries to KeyGen. Generate $d_{U,0}, d_{U,1} \leftarrow \mathbb{Z}_N$ once for this $U$.

A key $SK_{S,U}^{(b)}$ is generated semi-functionally by first calling the real generation procedure:

$$\text{KeyGen}(MSK, b, S, U, pk) = (K_U, K_{S,U}, K^*_U)$$

and making the modification:

$$K^*_U := K^*_U \times g^{d_{U,b}}_2$$

Notice that for all queries to the same $U$ for the same $b$ value, the $K^*_U$ component is offset by the same multiplicative factor in $G_{p_2}$.

**Definition. (GameReal)** An adversary in GameReal is interacting with the actual functionality as described in Piecewise CPABE Security. A.

**Definition. (Game0)** The response to the adversary’s queries in Game0 will differ from GameReal only in the challenge ciphertext phase. In Game0, the challenge ciphertext will be generated as a semi-functional ciphertext.

Throughout the rest of the proof we will let $\epsilon_i$ denote the advantage of $A$ in Game_i.

**Lemma.** If Assumption 1. holds then $|\epsilon_{\text{Real}} - \epsilon_0|$ is negligible in $\lambda$.

**Proof.** Recall in Assumption 1. the challenger is given $g \leftarrow \mathbb{G}_{p_1}$ and $X_3 \leftarrow \mathbb{G}_{p_3}$ and $T$ that either comes from $\mathbb{G}_{p_1p_2}$ or from $\mathbb{G}_{p_1}$. We simulate either GameReal or Game0 depending on the distribution $T$ is drawn from as follows. Choose $a, \alpha \leftarrow \mathbb{Z}_N$ and $s_i \in \mathbb{Z}_N$ for all $i \in \Omega$. Then, sends $A$ the public parameters:

$$PK = \{N, g, g^a, e(g, g)^\alpha, \{T_i = g^{s_i} : i \in \Omega\}\}$$

which is drawn from the correct distribution and internally set $MSK = \{\alpha, X_3\}$. This suffices to simulate honest key generation keys perfectly. The only difficulty lies in generating the
challenge ciphertext. On input $(M_0, M_1, P)$ for the challenge ciphertext, generate $b \leftarrow \{0, 1\}$ and sets:

$$C = M_b e(g^\alpha, T), \quad C' = T$$

We are implicitly setting the $G_{p_1}$ part of $T$ to be $g^s$. Then, pick a random vector $v \in \mathbb{Z}_N^n$ such that $v \cdot 1 = 1$ and generate \( r_i \leftarrow \mathbb{Z}_N \) for all $i \in [n]$ and sets:

$$C_i = T^{a_i v} T^{-r_i s_{\rho(i)}}, \quad D_i = T^{r_i}$$

Notice that if $T = g^s$ then the above is a properly generated normal ciphertext where the $v$ value in the real encryption algorithm is set to be $s \times v$ and the $r_i$ values in the actual scheme are set to be $s \times r_i$ which are both drawing from the correct distribution.

**Definition.** (Game \( k \)) For this game, the keys corresponding to the first $k$ identifiers that are queried by the adversary are generated semi-functionally and the challenge ciphertext is semi-functional.

**Lemma.** For any $k \in [q]$ if Assumption 2. holds, $|\epsilon_k - \epsilon_{k+1}|$ is negligible in $\lambda$.

We will analyze two restricted types of adversaries separately and later show that we can use this analysis to conclude the final result. $A$ is **Type 1** if for the $k^{th}$ identifier $U$ it only queries a attribute sets $S$ to $\text{KeyGen}(MSK, 0, S, U)$ such that $(A^*, \rho)(S) = 0$ where $(A^*, \rho)$ is the policy for the challenge ciphertext.

Similarly, $A$ is **Type 2** if for the $k^{th}$ identifier $U$ it only queries attribute sets $S$ to $\text{KeyGen}(MSK, 1, S, U)$ such that $(A^*, \rho)(S) = 0$.

We will show that for either type of $A$ has advantage $|\epsilon_{k-1} - \epsilon_k|$ negligible. Note that this does not immediately imply the theorem statement as the actual $A$ does not fall in either class, but we will use it to prove our lemma.

**Definition.** We define the game $\mathcal{H}_k$ as follows. The challenge ciphertext and keys for the first $k - 1$ identifiers queried are generated semi-functionally while the keys for the identifiers after and including the $k + 1$\textsuperscript{st} are generated normally. The keys for the $k^{th}$
identifier are modified as follows:

For the first \( i - 1 \) queries to \( \text{KeyGen}(MSK, 0, S, U) \) generate \((K_U, K_{SU}, K^*_U)\) from the normal distribution, generate \( d_U \leftarrow \mathbb{Z}_N \) (new for each query) and set:

\[
K^*_U := K^*_U \times g^{d_U}.
\]

For the \( i \)th query to \( \text{KeyGen}(MSK, 0, S, U) \) generate \((K_U, K_{SU}, K^*_U)\) from the normal distribution and then generate \( e_U, f_U \leftarrow \mathbb{Z}_N \) and set:

\[
K_U := K_U \times g^{e_U}, \quad K_{U,i} := K_{U,i} \times g^{e_{U,i}}, \quad K^*_U := K^*_U \times g^{f_U}
\]

Following the \( i \)th query, or to \( \text{KeyGen}(MSK, 1, (A, \rho), U) \) for the \( k \)th identifier \( U \), no modification is made to the normal key generation algorithm.

**Definition.** Define \( \mathcal{I}_i \) as a modification to \( \mathcal{H}_i \) where instead of \( e_U \) being generated uniformly for the \( i \)th query with \( b = 0 \) to the \( k \)th identifier, set \( e_U = 0 \).

Let \( \nu_i \) be the advantage of \( A \) in the security game \( \mathcal{H}_i \) and \( \mu_i \) its advantage in \( \mathcal{I}_i \).

**Lemma.** If \( A \) is of **Type 1** then \( |\mu_{i-1} - \nu_i| \) is negligible in \( \lambda \) for any \( i \in [q] \) if Assumption 2. holds.

**Proof.** Let \( g, X_1, X_2, X_3, Y_2, Y_3, N, T \) be generated from the indistinguishability instance of Assumption 2. - Recall then that either \( T \leftarrow \mathbb{G} \) or \( T \leftarrow \mathbb{G}_{\text{pair}} \). We generate the public key by generating \( a, \alpha \leftarrow \mathbb{Z}_N \) and \( s_i \leftarrow \mathbb{Z}_N \) for all \( i \in \Omega \) and setting:

\[
PK = (N, g^a, e(g, g)^\alpha, \{T_i = g^{s_i} \forall i \in \Omega\})
\]

We first describe the generation of the semi-functional challenge ciphertext with policy \((A, \rho)\):

- **Challenge Ciphertext Generation.** Generate the vectors \( u \) uniformly over \( \mathbb{Z}_N^l \) subject to the restriction that \( u \cdot 1 = a \) and generate \( r_i \leftarrow \mathbb{Z}_N \) for each \( i \in [n] \) set:

\[
C = Me(g^a, X_1, X_2), \quad C' = X_1, X_2
\]
We next describe how to generate semi-functional keys for the first $k - 1$ identifiers.

- **The first $k - 1$ identifiers queried to KeyGen($MSK, b, S, U'$):** If $\alpha_U$ or $d_{U, b}$ has not been generated yet, generate it randomly from $\mathbb{Z}_N$. Generate $t_U, d_U \leftarrow \mathbb{Z}_N$ and $V, Z \leftarrow \mathbb{G}_{p_3}$ and $W_i \leftarrow \mathbb{G}_{p_3}$ for all $i \in S$ and set where $\gamma = \alpha_U$ if $b = 0$ and $\alpha - \alpha_U$ if $b = 1$:

$$K_{U'} = g^{t_{U'} V}, \quad K_{U', S} = \{T_i^{t_{U'} W_i} : i \in S\} \quad K_{U'}^* = g^{\gamma g^{at_{U'}} Z(Y_2 Y_3)^{d_{U', b}}}$$

- **The $k^{th}$ identifier where $b = 0$:** Let $U$ be the $k^{th}$ identifier. For the first $i - 1$ queries to KeyGen($MSK, 0, S, U$) If $\alpha_U$ has not been generated yet, generate it randomly from $\mathbb{Z}_N$. Generate $t_U, d_U \leftarrow \mathbb{Z}_N$ and $V, Z \leftarrow \mathbb{G}_{p_3}$ and $W_i \leftarrow \mathbb{G}_{p_3}$ for all $i \in S$ and set:

$$K_U = g^{t_U V}, \quad K_{U, S} = \{T_i^{t_U W_i} : i \in S\} \quad K_V = g^{\alpha_U^{-1} Z(Y_2 Y_3)^{d_U}}$$

For the first $i^{th}$ queries to KeyGen($MSK, 0, S, U$): If $\alpha_U$ has not been generated yet, generate it randomly from $\mathbb{Z}_N$. Generate $V, Z \leftarrow \mathbb{G}_{p_3}$ and $W_i \leftarrow \mathbb{G}_{p_3}$ for all $i \in S$ and set:

$$K_U = TV, \quad K_{U, S} = \{T_i^{s_i W_i} : i \in S\} \quad K_V = g^{\alpha U T^a Z}$$

For the queries after the $i^{th}$ or where $b = 1$ to KeyGen($MSK, b, S, U$): If $\alpha_U$ has not been generated yet, generate it randomly from $\mathbb{Z}_N$. Generate $V, Z \leftarrow \mathbb{G}_{p_3}$ and $W_i \leftarrow \mathbb{G}_{p_3}$ for all $i \in S$ and set where $\gamma = \alpha_U$ if $b = 0$ and $\alpha - \alpha_U$ if $b = 1$:

$$K_U = g^{t_U V}, \quad K_{U, S} = \{T_i^{t_{U'} W_i} : i \in S\} \quad K_V = g^{\gamma g^{at_{U'}} Z}$$

Generating the fully functional keys for the identifiers numbered $k' > k$ from the correct distribution is simple since we have the master secret key $(\alpha, X_3)$ as they are generated identically to the first $k - 1$ identifiers where $d_{U, b}$ is set to be 0.
If \( T \in \mathbb{G}_{p_1 p_3} \), the above is a uniform instance of \( \mathcal{I}_{i-1} \). On the other hand, if \( T \in \mathbb{G} \) then the above procedure is almost equivalent to the procedure from \( \mathcal{H}_i \) except for the fact that the \( C_i \) components of the challenge ciphertext are multiplied by a new \( X_2^A \cdot u \) factor where instead of \( u \) being generated uniformly at random, it is uniform subject to the restriction that \( u \cdot 1 = a \). However, for all \( i \) such that \( \rho(i) \not\in T \) (where \( T \) is the attribute set for the \( i^{th} \) query to the \( k^{th} \) identifier with \( b = 0 \)) notice the \( \mathbb{G}_{p_2} \) components of \( C_i \) are independent from all other parts of the scheme (as for these values \( s_{\rho(i)} \) is nowhere else used as an exponent in \( \mathbb{G}_{p_2} \), note this statement requires the injectivity of \( \rho \)).

Therefore, the additional restriction that \( u \cdot 1 = a \) is only present in the ciphertext components for \( i \) with \( \rho(i) \not\in S \) where \( S \) is the attribute set for the \( i^{th} \) query to the \( k^{th} \) identifier when \( b = 0 \). Since \( 1 \not\in \text{Span}(A_i : \rho(i) \in T) \) this generation procedure is identical to when \( u \) is generated completely uniformly (this argument is identical to that presented in the KP-ABE proof for indistinguishability between \( \mathcal{I}_{i-1} \) and \( \mathcal{H}_i \)) which is the correct distribution.

**Lemma.** If \( A \) is Type 1. then \( |\mu_i - \nu_i| \) is negligible in \( \lambda \) for any \( i \in [q] \) if Assumption 2. holds.

**Proof.** The only divergence from the above proof is for the \( i^{th} \) query with \( b = 0 \) to the \( k^{th} \) identifier which we give below:

- **The \( i^{th} \) query to the \( k^{th} \) identifier with \( b = 0 \):** \( \text{KeyGen}(MSK, 0, S, U) \). If \( \alpha_U \) has not been generated, sample it uniformly from \( \mathbb{Z}_N \). Sample \( d_U \) uniformly at random from \( \mathbb{Z}_N \) and \( V, Z \leftarrow \mathbb{G}_{p_3} \) and \( W_i \leftarrow \mathbb{G}_{p_3} \) for all \( i \in S \) and set:

\[
K_U := TV , \quad K_{U,S} := \{ T^s W_i : i \in S \} \quad K_{U}^* := T^a g^{\alpha_U} Z(Y_2 Y_3)^{d_U}
\]

Notice that the key generation above is actually independent of the \( \mathbb{G}_{p_2} \) part of \( T^a \) (if \( T \in \mathbb{G} \)) to the additional re-randomization by \( Y_2^{d_U} \) and therefore the additional analysis in the previous case is not needed since sampling \( u \) subject to the restriction \( u \cdot 1 = a \) is
equivalent to generating it randomly if $a$ is not used as an exponent in $\mathbb{G}_{p_2}$ anywhere else in the scheme. It’s therefore a simple observation that if $T \in \mathbb{G}_{p_1 p_3}$ the above is a uniform instance of $\mathcal{I}_i$ and if $T \in \mathbb{G}$ the above is a uniform instance in $\mathcal{H}_i$, as desired.

Now, by a hybrid argument and the observation that $\mathcal{I}_0 = \text{GAME}_{k-1}$:

**Corollary.** If $A$ is of Type 1, then $|\mu_q - \epsilon_{k-1}|$ is negligible in $\lambda$ if Assumption 2. holds.

We still have not proven our desired result however as $\mathcal{I}_q$ differs from $\text{GAME}_k$ in two ways. First, each $\text{KeyGen}$ with $b = 0$ query to the $k^{th}$ identifier has the $\mathbb{G}_{p_2}$ component of $K^*_U$ re-randomized separately (whereas in a semi-functional identifier, the $\mathbb{G}_{p_2}$ component is constant for all such queries). Second, $\text{KeyGen}$ queries with $b = 1$ to $U$ should have the $K^*_U$ component offset by a constant $g_2^{d_{U,0}}$ amount. For this we need another sequence of hybrids.

**Definition.** Define $\mathcal{J}$ as a modification of $\mathcal{I}_q$ as follows. For the $k^{th}$ identifier $U$, rather than generating $f_U \leftarrow \mathbb{Z}_N$ for each query $\text{KeyGen}(MSK, 0, S, U)$, the value $f_U$ is generated uniformly once for $U$ and re-used across all such queries.

Let $\eta$ be the advantage of the adversary in $\mathcal{J}$.

**Lemma.** If $A$ is of Type 1, then $|\mu_q - \nu|$ is negligible in $\lambda$ if Assumption 2. holds.

To show this we use another series of games:

**Definition.** We define $\mathcal{I}'_i$ and $\mathcal{H}'_i$ as modifications of $\mathcal{I}_i$ and $\mathcal{H}_i$ respectively where for the $k^{th}$ identifier $U$ an additional value $d_{U,0}$ is generated uniformly at random and all queries to $U$ of the form $\text{KeyGen}(MSK, 0, S, U)$ after being generated from $\mathcal{I}_i$ or $\mathcal{H}_i$ respectively are modified by:

$$K^*_U := K^*_U \times g_2^{d_{U,0}}$$

Note that in the above definition $d_{U,0}$ is only generated once for $U$. We call the advantage of the adversary in $\mathcal{I}'_i$ and $\mathcal{H}'_i$, $\mu'_i$ and $\nu'_i$ respectively.

Notice first that $\mathcal{I}'_q = \mathcal{I}_q$ as the independent $f_U$ factor in each of these queries in $\mathcal{I}_q$ completely subsumes the additional $g_2^{d_{U,0}}$ factor. Additionally, through repeating the proofs
of the previous two lemmas with the only modification being that for the $k^{th}$ identifier $U$, $d_{U,0}$ is generated and all responses to $\text{KeyGen}(MSK,0,S,U)$ are modified as $K^*_U := K^*_U \times g_{d_{U,0}}^2$, we can conclude the following two lemmas:

**Lemma.** If $A$ is of Type 1. then $|\mu'_{i-1} - \nu'_i|$ is negligible in $\lambda$ for any $i \in [q]$ if Assumption 2. holds.

and,

**Lemma.** If $A$ is of Type 1. then $|\mu'_i - \nu_i'|$ is negligible in $\lambda$ for any $i \in [q]$ if Assumption 2. holds.

Since $T'_0 = J$, this allows us to combine these two lemmas and indistinguishability between $T'_q$ and $\text{Game}_{k-1}$ to conclude:

**Corollary.** If $A$ is of Type 1. then $|\eta - \epsilon_{k-1}|$ is negligible in $\lambda$ if Assumption 2. holds.

Now we have indistinguishability of $\text{GAME}_{k-1}$ form $J$ which is much closer to $\text{GAME}_{k}$. Like $\text{GAME}_{k}$, $J$ has all $\text{KeyGen}(MSK,0,S,U)$ queries for the $k^{th}$ identifier $U$ have the $K^*_U$ value offset by a constant amount in $G_{p_2}$. However, the $\text{KeyGen}(MSK,1,S,U)$ values are not yet offset. For this, we will need one final lemma.

**Lemma.** If $A$ is of Type 1. then $|\epsilon_k - \eta|$ is negligible in $\lambda$ if Assumption 2. holds.

**Proof.** Let $g, X_1, X_2, X_3, Y_2, Y_3, N, T$ be generated from the indistinguishability instance of Assumption 2. - Recall then that either $T \leftarrow G$ or $T \leftarrow G_{p_1,p_3}$. We generate the public key by generating $a, \alpha \leftarrow Z_N$ and setting:

$$PK = (N, g, g^a, e(g,g)^\alpha, \{T_i = g^{z_i} \forall i \in \Omega\})$$

First we demonstrate how to simulate semi-functional challenge ciphertext generation $(A, \rho)$:

- **Challenge Ciphertext Generation.** On a query $(M_0, M_1, (A, \rho))$, generate the vector $u \in Z_{N}^l$ uniformly subject to the restriction that $u \cdot 1 = a$. Then choose
\[ b \leftarrow \{0, 1\} \text{ uniformly and } r_i \leftarrow \mathbb{Z}_N \text{ for all } i \in [n] \text{ and set:} \]

\[ C = M_b e(g^\alpha, X_1 X_2), \quad C' = X_1 X_2 \]

\[ C_i = (X_1 X_2)^{A_i u} (X_1 X_2)^{-r_i s_{\rho(i)}}, \quad D_i = (X_1 X_2)^{r_i} \]

Next we show how to generate semi-functional keys for the first \( k - 1 \) identifiers.

- **The first \( k - 1 \) identifiers:** On a query \textbf{KeyGen}(MSK, b, S, U): If \( \alpha_U \) or \( d_{U,b} \) have not yet been generate them uniformly over \( \mathbb{Z}_N \). Generate \( t_U \leftarrow \mathbb{Z}_N, V, Z \leftarrow \mathbb{G}_{p_3} \) and \( W_i \leftarrow \mathbb{G}_{p_3} \) for all \( i \in S \). Let \( \gamma = \alpha_U \) if \( b = 0 \) and \( \alpha - \alpha_U \) if \( b = 1 \) and set:

\[ K_U = g^{t_U} V, \quad K_{U,S} = \{ T_i^{r_i} W_i : i \in S \} \]

\[ K_U^* = g^{\alpha U} g^{at_U} Z (Y_2 Y_3)^{d_{U,b}} \]

The key for the identifiers following \( k^{th} \) are also generated according to the above distribution where \( d_{U,b} \) is set to 0. We now detail the generation for the \( k^{th} \) identifier.

- **The \( k^{th} \) identifier:** On a query \textbf{KeyGen}(MSK, 0, S, U): If \( d_{U,0} \) has not yet been generate it uniformly over \( \mathbb{Z}_N \). Generate \( t_U \leftarrow \mathbb{Z}_N, V, Z \leftarrow \mathbb{G}_{p_3} \) and \( W_i \leftarrow \mathbb{G}_{p_3} \) for all \( i \in S \). Set:

\[ K_U = g^{t_U} V, \quad K_{U,S} = \{ g^{r_i} W_i : i \in S \} \]

\[ K_U^* = T g^{at_U} Z (Y_2 Y_3)^{d_{U,0}} \]

On a query \textbf{KeyGen}(MSK, 1, S, U): Generate \( t_U \leftarrow \mathbb{Z}_N, V, Z \leftarrow \mathbb{G}_{p_3} \) and \( W_i \leftarrow \mathbb{G}_{p_3} \) for all \( i \in S \). Set:

\[ K_U = g^{t_U} V, \quad K_{U,S} = \{ g^{r_i} W_i : i \in S \} \]

\[ K_U^* = g^\alpha T^{-1} g^{at_U} Z \]

Here if we write \( T_{\mathbb{G}_{p_1}} \) (the \( \mathbb{G}_{p_1} \) part of \( T \)) as \( g^t \) we are implicitly setting \( \alpha_U = t \). If \( T \in \mathbb{G}_{p_{1} p_3} \) then the keys for the \( k^{th} \) identifier have \( d_{U,0} \) as in \( T'_0 \). Similarly, if \( T \in \mathbb{G}_{p_{1} p_{2} p_3} \), \( g^{d_{U,0}} \) is set to be the \( g^{d_{U,0}} \) value generated in the above simulation plus the \( \mathbb{G}_{p_2} \) part of \( T \) and

71
is set to be the $G_{p_2}$ part of $T^{-1}$. As these are both uniform, independent and reused across all queries to this identifier with the same value for $b$, this draws from the correct distribution for GAME$_k$ as desired. Now combining all the lemmas we have shown we can conclude:

**Lemma.** If $A$ is of *Type 1*. then $|\epsilon_k - \epsilon_{k-1}|$ is negligible in $\lambda$ for any $k \in [q]$ if Assumption 2. holds.

Through a symmetric argument, the above also holds for *Type 2*. adversaries (where the $\mathcal{H}_i, \mathcal{I}_i, \mathcal{H}_i', \mathcal{I}_i'$ hybrids now modify the generation procedure when $b = 1$). If there was a general adversary $A$ (not *Type 1*. or *Type 2*. that could distinguish between GAME$_k$ and GAME$_{k+1}$ notice that it would be possible to make two adversaries $A_1$ and $A_2$ such that $A_1$ is *Type 1*. and *Type 2*. using $A$ (such that $A_1$ aborts and guesses randomly is $A$ ever makes a query to violate being *Type i*. that either $A_1$ or $A_2$ would have a non-negligible advantage if $A$ did, violating the above lemma. Therefore, GAME$_k$ and GAME$_{k+1}$ are indistinguishable for general adversaries and through a hybrid argument we have:

**Corollary.** $|\epsilon_0 - \epsilon_q|$ is negligible in $\lambda$ if Assumption 2. holds.

Let GAME$_{Final}$ denote the game identical to GAME$_q$ with the only exception that rather than choosing the challenge message to be $M_c$ where $c \leftarrow \{0,1\}$, the message is chosen uniformly over $G_T$.

**Lemma.** If Assumption 3. holds, $|\epsilon_{Final} - \epsilon_q|$ is negligible.

**Proof.** Recall that in Assumption 3. we are given $N, g, g^aX_2, X_3, g^sY_2, Z_2$ and $T$ which either is $e(g, g)^{\alpha s}$ or a uniform element over $G_T$. We now describe how to simulate either an instance of GAME$_q$ or GAME$_{Final}$ depending on which distribution $T$ comes from.

- To generate the public key, choose $a \leftarrow \mathbb{Z}_N$ and $z_i \leftarrow \mathbb{Z}_N$ for all $i \in \Omega$ and set:

$$PK = (N, g, g^a, e(g, g^aX_2) = e(g, g)^a, (T_i = g^{z_i} : i \in \Omega))$$
We next describe how to generate keys for semi-functional identifiers:

- **Semi-Functional Identifiers.** To queries to KeyGen(\textit{MSK}, \textit{b}, \textit{S}, \textit{U}) if \(dU,b\) or \(\alpha_U\) has not been generated yet, sample it uniformly from \(Z_N\) and store it. Generate \(t_U \leftarrow Z_N\), \(V, Z \leftarrow G_{p3}\) and \(W_i \leftarrow G_{p3}\) for all \(i \in S\). Set \(G = g^{\alpha_U}\) if \(b = 0\) and \(G = g^{\alpha_X}g_{2}^{-\alpha_U}\) if \(b = 1\) and set:

\[
K_U = g^{t_U}V, \quad K_{U,S} = \{T_i^{d_U}W_i \forall i \in S\}, \quad K^*_U = Gg^{at_U}Z_{2}^{d_{U,b}}Z
\]

- **The Ciphertext.** Generate \(v \in Z_{N}^l\) uniformly subject to the restriction that \(v \cdot 1 = a\) and \(r_i \leftarrow Z_N\) for all \(i \in [n]\) and \(s \leftarrow Z_N\) and set:

\[
C = M_cT, \quad C' = g^sY_2
\]

\[
C_i = (g^{s}Y_2)^{A_i-v(g^sY_2)^{-r_is_{\rho(i)}}}, \quad D_i = (g^sY_2)^{r_i}
\]

When \(T = e(g, g)^{a}\) the above is a correctly distributed semi-functional ciphertext, whereas when \(T\) is a random element in \(G_T\) it is an encryption of a random message. Since \(\epsilon_{\text{final}}\) must be negligible, as the challenge bit \(c\) is completely independent of anything in the challenge ciphertext, this proves security of our scheme.
CHAPTER 3

Identifying Cheaters with No Honest Majority

One of the most natural areas in which trust is essential in cryptography is in the context of multi-party computation without an honest majority. As proven by Cleve [18] fully secure MPC is not possible without additional assumptions, even in the presence of cryptography. In this work we circumvent this problem by having a trusted primitive that is guaranteed to always perform some small operation correctly, and ask whether or not it is possible to evaluate any functionality correctly assuming the existence of one small trusted primitive. We find that a very relevant notion in this setting is that of secure secret sharing that can detect which players have returned tampered shares to the reconstruction functionality.

Identifiable secret sharing, where the reconstruction function can correctly identify which players have returned tampered shares, can be realized only when a majority of the clients are honest [52, 47, 17, 59]. But without an honest majority, there is no way for the server to tell apart unmodified and modified shares. Indeed, $n/2$ cheaters can simulate a consistent sharing of an incorrect secret, which makes it impossible for the server to tell which of the two sets of consistent shares is correct. However, this does not rule out the alternative of allowing the server to inform each client (with negligible error probability) which shares have been modified assuming that this client is honest. We refer to this as locally-identifiable secret sharing (LISS). Note that except with negligible probability, each honest client will agree on which clients are corrupted and should be disqualified.

Settling for computational security, LISS can be realized via the use of digital signatures: the sharing procedure distributes to all clients the same public verification key $vk$, and
gives to each client a signature of its additive share of the secret using the corresponding signing key $sk$. Reconstruction proceeds by letting each client send to the server its original share, $vk$, and the signature on the share. The server can then identify definite cheaters as those who supply an inconsistent triplet, and partition the remaining clients according to the value of $vk$ they provide. In fact, such a computationally secure LISS scheme was implicitly used by Gordon et al. [33] in the context of defining a complete primitive for MPC. The possibility of an unconditionally secure construction remained open. This question is motivated not only by the goals of enhancing security and eliminating assumptions, but also by the potential efficiency advantages of information-theoretic techniques. This is especially significant in applications (such as those discussed below) where the share generation process is distributed between multiple players.

This work has also appeared at the IACR Theory of Cryptography Conference 2012 as *Identifying Cheaters without an Honest Majority* (Copyright IACR). This is joint work with Yuval Ishai and Rafail Ostrovsky and is presented here with their permission.

Our Results.

Constructions. Our main result is an affirmative answer to the above question: we present an unconditional construction of an $n$-out-of-$n$ LISS scheme whose security holds in the presence of an arbitrary number of corrupted players. More generally, we show how to efficiently transform any secret sharing scheme into one in which the reconstruction function reveals to every honest player of the identity of all shares that have been tampered with. In particular, all honest players agree on the same set of cheaters.

We also consider a weaker variant of LISS that we call *unanimously identifiable secret sharing* (UISS) in which only the latter agreement property is required. That is, if reconstruction fails, all honest players should agree on the same (non-empty) set of cheaters. This weaker primitive is easier to construct. (In fact, a construction of UISS is implicit in [62].)
In contrast to LISS, however, UISS does not guarantee that all cheaters are detected in the event that reconstruction fails.

**Applications.** We present several applications of the above primitives in the context of MPC without an honest majority. In the following, the term MPC refers to the special case of secure function evaluation, namely MPC of non-reactive (stateless) functionalities. We use poly and negl to represent polynomial and negligible functions, respectively, and κ denote a statistical security parameter. While we mainly consider statistical security, our results are also useful in the domain of computational security.

**Complete primitives for MPC.** It is well known that fully secure MPC (with fairness and guaranteed output delivery) is impossible to achieve in general without an honest majority [18]. This naturally raises the question of finding a minimal complete primitive that can be used to get around this limitation. Such a primitive is defined by a (stateless) deterministic functionality \( g \) mapping \( n \) inputs to \( n \) outputs, such that any \( n \)-party functionality \( f \) can be realized using a trusted instance of \( g \) initialized between every tuple of players that can supply input to it. The first such results characterized complete boolean primitives for MPC with security against a passive adversary [48, 46]. In the case of active adversaries, Fitzi et al. [25] presented a complete primitive for fully secure MPC whose computational complexity grows linearly with complexity of \( f \). This left open the question of finding a “simple” complete primitive, whose complexity does not depend on the complexity of \( f \). One such primitive was given by Gordon et al. [33] using digital signatures. We use UISS to get an unconditional variant of this result. In this variant, the complexity of \( g \) only grows with the output length of \( f \).

**Theorem.** There is a deterministic, polynomial-time computable functionality \( g \) with input and output size \( \text{poly}(n, \kappa, \beta) \) such that any \( n \)-party function \( f \) computed by a circuit of size \( \sigma \) and output length \( \beta \) can be realized with full statistical security (and \( 2^{-\kappa} \) simulation error) using \( \text{poly}(n, \sigma) \) calls to \( g \).
This result has an interesting interpretation in the context of a recent line of work on basing cryptography on tamper-proof hardware (see [44, 35] and references therein). In this line of work, several impossibility results in cryptography (including UC security, unconditional security, software protection and obfuscation) were circumvented by using tamper-proof hardware tokens. These works spent efforts on minimizing the size of the tokens, employing stateless (rather than stateful) tokens, and minimizing or eliminating cryptographic assumptions. The above result can be viewed as achieving all these goals simultaneously in the context of another major impossibility result: the impossibility of fully secure MPC without an honest majority. It implies that a small and stateless token, connected via secure channels to the $n$ players, suffices to unconditionally realize fully secure MPC. We note that connecting the same token to all players is necessary, as implied by the results of Fitzi et al. [25].

We also present other variants of the previous completeness theorem which rely on computational assumptions but still avoid the use of cryptography inside the primitive. These variants have the advantage of requiring only a small number of calls to the primitive (independent of the complexity of $f$).

Applications to partial fairness. A recent line of works studies the extent to which partial fairness can be achieved in MPC without an honest majority. Partial fairness can be defined by restricting the simulation error to be small (e.g., inverse polynomial) but not negligible [34]. We show that in partially fair protocols of Beimel et al.[5, 4] (extending previous two-party protocols of Moran et al. [55] and Gordon and Katz [34]), the use of a digital signature scheme can be replaced by a unanimously identifiable commitment scheme, a second primitive we define that can be used as a substitute for LISS in certain applications. This yields unconditional multiparty protocols for coin-flipping and MPC with partial fairness in the preprocessing model, namely assuming that players have offline access to correlated randomness. We note that trusted preprocessing does not trivialize the problem, because
the output needs to be unpredictable in the end of the preprocessing phase. In fact, the negative results on achieving full fairness apply to the preprocessing model as well. The preprocessing model does allow, however, to eliminate the assumptions of secure channels and broadcast, which can be implemented unconditionally in the preprocessing model [64].

The preprocessing phase can be realized either by a trusted offline dealer or via a distributed protocol (possibly employing additional parties for unconditional security). Even if one relies on a computationally secure protocol for distributing the preprocessing phase, the protocols we get have the advantage of making only a black-box use of the underlying cryptographic primitives, whereas the original protocols from [5, 4] make a non-black-box use of a one-way function.

In the case of coin-flipping, applying our primitive to the offline dealer protocol from [5] implies the following:

**Theorem** Assume preprocessing by a trusted off-line dealer. Fix constants $n$ and $t$ such that $t < 2n/3$. Then, for any $r$, there is an $r$-round $n$-party unconditionally secure coin-tossing protocol over point-to-point channels tolerating up to $t$ malicious players with bias $O(1/r)$.

Our results on MPC with partial fairness are obtained via a general technique for unconditionally upgrading security against fail-stop adversaries to security against malicious adversaries where the messages sent by the players are determined in the preprocessing stage.

**A negative result.** It is known that MPC without an honest majority can be realized unconditionally in the OT-hybrid model, provided that one settles for “security with abort” [45, 43]. That is, the adversary can decide whether to abort the protocol after learning the outputs of corrupted players but before the honest players receive their output. We show that such protocols cannot be strengthened so that all honest players agree on the identity of a corrupted player in the event that the protocol aborts, even if a broadcast primitive and trusted access to an arbitrary pairwise functionality is assumed. This is contrasted with the computational setting, in which this stronger notion of security can be realized under stan-
standard cryptographic assumptions [32]. Our negative result strengthens a previous negative result from [25], which shows that pairwise functionalities alone are not sufficient in general for fully secure $n$-party computation.

3.1 Preliminaries and Notation

Our communication model allows for authenticated point to point and broadcast channels unless specified otherwise. While we define our algorithms in terms of finite sets (with fixed input size) and fixed error rate, they can be implemented by uniform algorithms that are polynomial in the bit-length of the inputs, the number of parties and the statistical error $\kappa$ guaranteeing $\delta = 2^{-\kappa}$ error. The latter is the default convention whenever no concrete value of $\delta$ is specified.

To simplify notation and analysis we only consider non-adaptive adversaries, but all of our definitions and secret sharing proofs can be extended to the adaptive case. Throughout the paper, we denote the $n$ players in the protocol by $\{P_1, P_2, \ldots P_n\} = \mathcal{P}$, and sometimes identify a player name with its index. A collection of subsets $A$ of $\mathcal{P}$ will be called monotone if for any $B \in A$, if $B \subset C \subset \mathcal{P}$ then, $C \in A$.

Secret Sharing

**Definition (Access Structure)** An access structure is a monotone set $A \subset 2^\mathcal{P}$. In applications, this will be collection of sets authorized to reconstruct the secret. We will call a set $B \in A$ an authorized set with regard to the access structure $A$. If $A$ is implied by context, we will simply refer to $B \in A$ as an authorized set.

**Definition (Secret Sharing)** Let $S$ be a finite set with $|S| \geq 2$. A secret sharing scheme for an access structure $A$ is defined by a pair of algorithms $\text{Share}$ and $\text{Rec}$. $\text{Share}$ is a randomized map from $S \rightarrow S_1 \times S_2 \times \ldots S_n$ with $S_i$ the share domain of $P_i$ such that the
following guarantees hold:

- **(Correctness)** For any $B \in \mathbb{A}$ and $s \in S$:
  \[
  \Pr[\text{Rec}(B, \text{Share}(s)_B) = s] = 1.
  \]
  where $\text{Share}(s)_B$ corresponds to the restriction of $\text{Share}(s)$ to its entries in $B$.

- **(Secrecy)** Unauthorized sets do not gain any information about the underlying secret. Formally, if $C \not\in \mathbb{A}$, for any $s, s' \in S$ and collection $s_i \in S_i$ for $P_i \in C$:
  \[
  \Pr[\text{Share}(s)_C = (s_i)_{i \in C}] = \Pr[\text{Share}(s')_C = (s_i)_{i \in C}].
  \]

### Identifiable Secret Sharing.

An identifiable secret sharing scheme is a secret sharing scheme in which the reconstruction algorithm can identify all cheaters in the event that it fails to reconstruct the secret. The above guarantee should hold except with some failure probability $\delta$ as long as there are at most $t$ cheaters for an additional parameter $t$. In our definition we assume that the tampering is done by a single adversary who can observe the shares of a set $C$ of up to $t$ corrupted players and based on this information decide on how to tamper with their shares.

**Definition.** *(Identifiable Secret Sharing)* A secret sharing scheme realizing $\mathbb{A}$ is $(\delta, t)$-identifiable if for any (unbounded) adversary $A$ and any $s \in S$, the success probability of $A$ in the following game is less than $\delta$:

1. $(s_1, s_2, \ldots, s_n) \leftarrow \text{Share}(s)$;
2. $A$ outputs a set $C \subset [n]$ such that $|C| \leq t$ and receives $(s_j)_{j \in C}$;
3. $A$ outputs $(B, (s'_j)_{j \in C \cap B})$ where $B \in \mathbb{A}$;
4. $\text{Out} \leftarrow \text{Rec}(B, (t_j)_{j \in B})$ where $t_j = s'_j$ if $j \in C$ and $t_j = s_j$ otherwise.
A succeeds if for some \( j \in C \cap B \), \( s'_j \neq s_j \) and \( \text{Out} \neq (\bot, \{ P_i \in C \cap B : s'_i \neq s_i \}) \).

The first work on identifiable secret sharing is due to McEliece and Sarwate [52] who showed that Shamir’s \( k \)-threshold secret sharing scheme allows perfect identification if \( k + 2t \) players of which at most \( t \) are cheaters are involved in reconstruction. Several works consider various relaxations of identifiability [66, 11, 14] which suffice for some applications but are not suitable for MPC with a dishonest majority. There is also substantial work on the efficiency of identifiable secret sharing [47, 59, 17].

**Identifiability without an Honest Majority**

Identifiability is not possible with a dishonest majority for a simple reason: If half of the participants are dishonest they can run the sharing algorithm independently among themselves and return as their shares the output of the second run of the algorithm. This strategy makes it impossible for \( \text{Rec} \) to identify which half of the shares come from the first run of the \( \text{Share} \) algorithm and which come from the second since they are run independently. The statement requires a little more effort to prove as the adversary we describe above might not actually be changing the shares, however the essence of the proof is captured above. We present the full details of the proof below.

**Theorem. No identifiability with a dishonest majority** For any \( t, n, S, \mathbb{A} \) such that \( t \geq n/2, \ |S| \geq 2 \) and \( \mathbb{A} \neq \emptyset \), there is no \((1/4, t)\)-identifiable secret sharing scheme with secret space \( S \) and access structure \( \mathbb{A} \).

We begin with a few lemmas.

**Lemma.** Take any \((1/4, t)\)-identifiable secret sharing scheme with \( |S| \geq 2 \) and access structure \( \mathbb{A} \neq \emptyset \). For any \( \mathcal{P}_1 \subset \mathcal{P} \) such that there is a \( |\mathcal{P}_2| \leq t \) with \( \mathcal{P}_1 \cup \mathcal{P}_2 \in \mathbb{A} \) and,
\( P_1 \cap P_2 = \emptyset \), for any \( s_1 \neq s_2 \in S \):

\[
\Pr[\text{Share}(s_1)_{P_1} = \text{Share}(s_2)_{P_1}] \leq 1/4
\]

Assume to the contrary that the above was not true for \( s_1, s_2 \). Consider an adversary \( A \) interacting with an instance of security game initialized with \( s_1 \) that corrupts \( P_2 \) and proceeds by uniformly sampling the shares of \( P_2 \) from a uniform sharing of \( s_2 \) and initiates reconstruction on \( P_1 \cup P_2 \). In this case, the reconstruction function must always succeed in reconstructing the secret when the shares for \( P_1 \) that were generated during \( A \)'s second sharing (and are discarded) equals its shares in the original sharing. Notice that this happens with probability at least 1/4 by our assumption. Furthermore, whenever the original shares for \( P_1 \) and the shares that \( A \) generated are the same, the tuple that \( A \) generates for \( P_2 \) does not equal their original value since \( P_1 \cup P_2 \in A \) and this would imply a set of shares reconstructing to two different values.

Therefore, the reconstruction function is incorrect and does not identify a tampering \( A \) with probability at least 1/4, contradicting \((1/4, t)\)-identifiability.

We now begin with the proof of the main theorem. Let \( P_1 \) and \( P_2 \) be a non-trivial partition of \( P \) such that both partitions are of size \( \leq t \) and let \( s_1 \neq s_2 \) be two elements in \( S \). Let \( A_1 \) be the adversary that corrupts \( P_1 \) and replaces their shares by the shares returned to these players from a uniform sharing of \( s_1 \) and \( A_2 \) be the adversary that corrupts \( P_2 \) and replaces their shares with those returned by a uniform sharing of \( s_2 \). Both adversaries initiate reconstruction on \( P_1 \cup P_2 \).

Consider the output of the reconstruction function when it is returned a tuple of shares where the shares from \( P_1 \) are a uniform sharing of \( s_1 \) and \( P_2 \) come from independent sharings of \( s_2 \). Either the reconstruction function will return a subset of \( P_1 \) or a subset of \( P_2 \) with probability \( \leq 1/2 \). This implies that either when \( A_1 \) is attacking the functionality when the secret is \( s_2 \), it identifies a subset of \( P_1 \) as having cheated with probability \( \leq 1/2 \) or it identifies \( P_2 \) as having cheated when \( A_2 \) is attacking the scheme and the original secret is \( s_1 \).
with probability $\leq 1/2$. WLOG assume that the first case holds.

In this case, when $A_1$ is attacking the scheme where the initial secret is initialized to $s_2$ it is only identified as a cheater with probability $\leq 1/2$. However, by the lemma above, $A_1$ is actually tampering with probability $\geq 3/4$. Therefore the reconstruction function is incorrect with probability $\geq 1/4$ as desired.

3.2 Locally Identifiable Secret Sharing

We now give our relaxation of identifiable secret sharing that can be realized when arbitrarily many players may be corrupted. Informally, the guarantee we require is that if the reconstruction fails, the reconstruction algorithm outputs a tuple of players to each player $P_i$ with the guarantee that if $P_i$ is honest, the tuple returned to $P_i$ is precisely the players that tampered with their shares. While this is equivalent to identifiability from the point of view of the honest players, it allows us to circumvent the impossibility result presented above. Note that we define LISS as being a special type of secret sharing scheme, so the usual correctness and secrecy requirements should hold in addition to the requirements detailed below.

**Definition (Lists)** Throughout this paper when we refer to a list $L$ we refer to a subset of the players in the protocol ($L \subseteq \{P_1, P_2, \ldots, P_n\}$).

**Definition (LISS)** A secret sharing scheme realizing $A$ is locally $\delta$-identifiable if it satisfies the following requirements:

- **Unanimity:** For any adversary $A$ and $s \in S$, the probability of $A$’s success in the following game is at most $\delta$:

  1. $(s_1, s_2, \ldots, s_n) \leftarrow \text{Share}(s)$;

  2. $A$ outputs a set $C \subseteq [n]$ to corrupt and then receives $(s_i : i \in C)$;
3. A outputs \((B, (s'_j)_{j \in C \cap B})\) such that \(B \in \mathbb{A}\) and \(B \not\subseteq C\);

4. \(\text{Out} \leftarrow \text{Rec}(B, (t_j)_{j \in B})\) where \(t_j = s'_j\) if \(j \in C\) and \(t_j = s_j\) otherwise.

The adversary succeeds unless:

1. Reconstruction succeeds: \(\text{Out} = s\) or,

2. Each honest player’s list is the list of all cheaters: \(\text{Out} = (\bot, (L_j)_{j \in B})\) where for all \(j \in B \setminus C\), \(L_j = \{P_i \in C \cap B : s'_j \neq s_j\}\).

- The scheme has \textbf{Predictable Failures} (Defined below).

We briefly motivate the requirement of Predictable Failures before defining it. The problem to address is that the additional outputs \(L_j\), or even the event of not reconstructing the secret, may leak some information concerning the secret unless a separate guarantee is made. This can cause a problem in applications and therefore we must have a way to simulate the actions of \(\text{Rec}\) in the case of tampering. Note that this is a new issue not present in identifiable secret sharing: As the \(\text{Rec}\) function does not simply output a list of tampering players, there are no a-priori guarantees concerning the lists corresponding to dishonest players and therefore we must make requirements on them separately.

\textbf{Definition (Predictable Failures)} A secret sharing scheme has \(\delta\)-Predictable Failures if there is an algorithm \(S\text{Rec}\) such that for any adversary \(A\) and \(s \in S\), the probability of success in the following game is less than \(\delta\):

1. \((s_1, s_2, \ldots, s_n) \leftarrow \text{Share}(s)\);

2. \(A\) outputs a set \(C \subset [n]\) to corrupt and receives \((s_i)_{i \in C}\);

3. \(A\) outputs \((B, (s'_j)_{j \in C \cap B})\) such that \(B \in \mathbb{A}\) and \(B \not\subseteq C\);

4. \(S\text{Out} \leftarrow S\text{Rec}(C, B, (s_i)_{i \in C}, (s'_i)_{i \in C \cap B})\);
5. \( \text{Out} \leftarrow \text{Rec}(B, (t_j)_{j \in B}) \) where \( t_j = s'_j \) if \( j \in C \) and \( t_j = s_j \) otherwise.

\( A \) succeeds unless:

1. \( \text{SRec} \) correctly predicts success: \( \text{SOut} = \text{SUCCESS} \) and \( \text{Out} = s \) or,

2. \( \text{SRec} \) predicts the output of \( \text{Rec} \): \( \text{SOut} = \text{Out} \neq s \).

**Our Construction.**

Let \( (\text{Sh}, \text{Rc}) \) be a secret sharing scheme realizing access structure \( A \) with \( \text{Sh} : S \rightarrow \mathbb{F}^n \) where \( \mathbb{F} \) is a field. Let \( I_{n \times n} \) and \( 0_{n \times n} \) denote the identity and all zero matrix respectively. We use \( \mathbb{F}^{n \times n} \) to denote the set of all \( n \times n \) matrices with elements in \( \mathbb{F} \) and \( GL_n(\mathbb{F}) \) to denote the set of all such invertible matrices. For a matrix \( M \) we will use \( M(i,j) \) to denote the \((i,j)\) entry of \( M \). By default we assume vectors to be column vectors, we will use the notation \( \vec{a}^T \) when referring to a row vector. We use the notation \( \mathbb{F}^* \) to denote \( \mathbb{F} \setminus \{0\} \).

The intuition for our construction is that we will issue each player \( P_i \) a set of values \((\vec{a}_i^T, \vec{b}_i, u_i, v_i)\) so that for each \( P_i \), \( \vec{a}_i^T \vec{b}_i \) will be the user’s actual secret in the scheme while \( \vec{a}_i^T \vec{b}_j \) is going to be equal to an information checking quantity, validated by using \( u_i \) and \( v_j \).

**Share(\( s \)):**

1. Generate \((t_1, t_2, \ldots, t_n) \leftarrow \text{Sh}(s), u_i, v_i \leftarrow \mathbb{F}^* \) for all \( i \in [n] \);

2. Define \( C_0 \in \mathbb{F}^{n \times n} \) as
   \[
   \begin{cases}
   C_0(i,j) = u_i^{j+1}v_j^{i+1} + u_iv_j + 1 & \text{for } i \neq j; \\
   C_0(i,i) = t_i & \text{for } i \in [n].
   \end{cases}
   \]

3. Define \( C \) blockwise as:
   \[
   \begin{pmatrix}
   C_0 & I_{n \times n} \\
   I_{n \times n} & 0_{n \times n}
   \end{pmatrix};
   \]
4. Generate $B \triangleleft \mathcal{S} GL_{2n}(\mathbb{F})$ and define $A = CB^{-1}$;

5. Label row $i$ of $A$ as $\vec{a}^T_i$ and column $j$ of $B$ as $\vec{b}_j$;

6. Return $(s_i = (\vec{a}^T_i, \vec{b}_i, u_i, v_i))_{i \in [n]}$.

\textbf{Rec}(D, (\vec{a}^T_i, \vec{b}_i, u_i, v_i)_{i \in D}) with $D \in \mathbb{A}$, $\vec{a}^T_i, \vec{b}_i \in \mathbb{F}^{2n}$, $u_i, v_i \in \mathbb{F}^*$:

1. If for all $i \neq j$, $\vec{a}^T_i \vec{b}_j = u_i^{j+1} v_j^{i+1} + u_i v_j + 1$:
   - Set $\vec{a}^T_i \vec{b}_i = t_i$ for all $i \in B$;
   - Return $\text{Rec}(D, (t_i)_{i \in D})$.

2. Else, for all $i \in D$ set:

   \[ \mathcal{L}_i = \{ P_j : \vec{a}^T_j \vec{b}_i \neq u_i^{j+1} v_j^{i+1} + u_i v_j + 1 \text{ or } \vec{a}^T_j \vec{b}_i \neq u_j^{i+1} v_i^{j+1} + u_j v_i + 1 \} ; \]

3. Return $(\bot, (\mathcal{L}_i)_{i \in D})$.

\textbf{Theorem.} If $\delta > n^2(n+1)/(|\mathbb{F}| - 1)$, the scheme described above is a $\delta$-LISS scheme realizing $\mathbb{A}$ with secret space $S$ and share space $S_i = \mathbb{F}^{4n+2}$.

We quickly state a corollary for the efficiency of the scheme that the above theorem implies before presenting a proof of security.

\textbf{Corollary.} Suppose there is a secret sharing scheme which realizes an $n$-party access structure $\mathbb{A}$ with secret space $S$ and share length $\beta$. Then, for any $\delta > 0$ there is a $\delta$-LISS with the same $\mathbb{A}$ and $S$ whose share length is $O(n \log(n/\delta) + n\beta)$.
Proof of Security.

Proof of Secrecy.

For secrecy it is necessary to argue that the shares any set of non-authorized parties does not depend on the underlying secret. Notice that the only place in the scheme where $t_i$ is used is in the generation of the $i^{th}$ row of $A$. Therefore, a collection of shares for a group of players will be generated only using the $t_i$ values for that group. Since we assume the underlying $(\text{Sh}, \text{Rc})$ scheme has perfect secrecy, if this group is unauthorized, their shares in our constructed scheme also do not depend on the shared secret.

Proof of Local Identifiability.

Fix an arbitrary secret $s \in S$. Let $A_i, B_i, U_i, V_i$ be the random variables $\vec{a}_i^T, \vec{b}_i, u_i, v_i$ values on an execution of $\text{Share}(s)$. We will first show that for any view of the adversary, any tampering will cause reconstruction to fail with all honest parties. We will prove the statement when all but one party is corrupted as this will prove the general case by a union bound. In other words, if we let:

$$p_{i,j} = u_i^{j+1}v_i^{j+1} + u_jv_i + 1$$

**Theorem (Informal)** Take any $j \in [n]$ and let $T = [n] \setminus \{j\}$. Take any $\vec{a}_i, \vec{b}_i \in \mathbb{F}^{2n}$, $u_i, v_i \in \mathbb{F}$ for $i \in T$ and let $A_j, B_j, U_j, V_j$ denote the conditional distribution on the shares of player $j$ where the shares of the remaining players are fixed to these values. Then, for any $k \in T$ and $(\vec{a}_k', \vec{b}_k', u_k', v_k') \neq (\vec{a}_k, \vec{b}_k, u_k, v_k)$:

$$\Pr[p_{j,k}(U_j, v_k') = A_jb_k'] \land \Pr[p_{k,j}(u_k', V_j) = a_k'B_j] < \delta.$$  

This will imply that even by corrupting all but one player $j$, the attacker can not change any player $k$’s share without it being on $\mathcal{L}_j$ with probability less than $\delta$. By taking a union bound, this implies that by corrupting a subset, all tampered players will be on all honest
players lists with probability $n^2\delta$. Therefore, showing the above statement will imply the scheme is a $n^2\delta$ LISS scheme. Notice that what we’re showing is actually much stronger than is necessary as we’re saying no matter what element from the output distribution the adversary gets, his chance in successfully outputting a tampered value that will pass the honest player’s verification check is low.

We will first examine when either $\vec{b}'_k$ or $v'_k$ was modified from their original value. For the remainder of the proof fix $j, k \in [n]$ with $j \neq k$, $\{\alpha_i^T, b_i, u_i, v_i : i \in [n] \setminus \{j\}\} = Y$ such that $\Pr[\text{Share}(s)_T = Y] > 0$ where $T = [n] \setminus \{j\}$. For the rest of the section we will let $A_i, B_i, U_i$ and $V_i$ denote the distribution on the respective shares on a uniform execution of $\text{Share}(s)$.

**Lemma.** If $\vec{b}'_k \neq \vec{b}_k$ or $v'_k \neq v_k$:

$$\Pr \left[ A_j \vec{b}'_k = U_j^{k+1} v_j^{j+1} + U_j v'_k + 1 \bigg| \bigwedge_{i \in T} \left( (A_i = \alpha_i^T) \land (B_i = \vec{b}_i) \land (U_i = u_i) \land (V_i = v_i) \right) \right] < \frac{(n + 1)/|F^*|}{}. $$

**Proof.** There are two cases to consider; the first corresponds to when the $\vec{b}'_k$ value falls in the linear span of the other $\vec{b}_i$ values of the corrupted parties.

**Case 1.** $\vec{b}'_k \in \text{Span}(\{\vec{b}_i\}_{i \in T})$:

In this case write $\vec{b}'_k = \sum_{i \in [n] \setminus \{j\}} \gamma_i \vec{b}_i$ where $\gamma_i \in F$. Then:

$$A_j \vec{b}'_k = U_j^{k+1} v_j^{j+1} + U_j v'_k + 1 \iff \sum_{i \in T} \gamma_i \left( U_j v_i + 1 \right) = U_j^{k+1} v_j^{j+1} + U_j v'_k + 1$$

$$\iff \sum_{i \in T} \gamma_i \left( U_j v_i + 1 \right) - \left( U_j^{k+1} v_j^{j+1} + U_j v'_k + 1 \right) = 0.$$

Notice that unless $\gamma_i = 0$ for all $i \in T \setminus \{k\}$ the above defines a non-zero polynomial of degree at most $n + 1$ in $U_j$ (since $v_i \neq 0$ for all $i \in [n]$). Similarly, if $\gamma_i = 0$ for all $i \in T \setminus \{k\}$, unless $\gamma_k = 1$ the above still defines a non-zero polynomial of degree at most $n + 1$ by observing the constant coefficient. Furthermore, if $\gamma_i = 0$ for all $i \in T \setminus \{k\}$ and $\gamma_k = 1$ (which implies
\( \vec{b}_k = \vec{b}_k \) it still defines a non-zero polynomial of degree at most \( n + 1 \) unless \( v'_k = v_k \) (by observing the linear term). Therefore, if \( (\vec{b}'_k, v'_k) \neq (\vec{b}_k, v_k) \), the event in the theorem is satisfied if and only if the random variable \( U_j \) satisfies a fixed polynomial of degree at most \( n + 1 \).

We now claim that for any \( f \in \mathbb{F}^* \):

\[
\Pr \left[ U_j = f \left| \bigwedge_{i \in T} \left( A_i = \vec{a}_i^T \right) \land \left( B_i = \vec{b}_i \right) \land (U_i = u_i) \land (V_i = v_i) \right. \right] = |\mathbb{F}^*|^{-1}
\]

Notice that this will suffice to prove our theorem since if \( U_j \) are distributed uniformly even after all the values on the right are fixed, then the polynomial will be satisfied with probability at most \((n+1)|\mathbb{F}^*|^{-1}\). This equality follows from the fact that \( U_j \) is independent of the random variables \( \{A_i, B_i, U_i, V_i\}_{i \in T} \) since \( U_j \) is only during the generation of \( A_j \) (it only appears in the \( j^{th} \) row of \( C \) and \( A = CB^{-1} \) implies the \( j^{th} \) row of \( C \) does not affect the generation of any other rows of \( A \)).

**Case 2.** \( \vec{b}_k \not\in \text{Span}(\{\vec{b}_i\}_{i \in T}) \):

As a reminder if we define the event:

\[
F = \bigwedge_{i \in T} \left( (A_i = \vec{a}_i^T) \land (B_i = \vec{b}_i) \land (U_i = u_i) \land (V_i = v_i) \right)
\]

We are bounding the conditional probability:

\[
\Pr \left[ A_j \vec{b}_k = U_j^{k+1}v'^{j+1}_k + U_jv'_k + 1 \bigg| F \right]
\]

In fact, we will show the stronger statement by bounding this probability even after \( \vec{a}_i^T : i \in [n + 1, 2n] \) are fixed as well. Fix any \( \vec{a}_i^T \in \mathbb{F}^{2n} \) for \( i \in [n + 1, 2n] \) such that \( \{\vec{a}_i^T\}_{i \in [2n] \setminus \{j\}} \) is linearly independent. We will call:

\[
E = \bigwedge_{i \in T'} \left[ (A_i = \vec{a}_i^T) \right] \land \bigwedge_{i \in T} \left[ (B_i = \vec{b}_i) \land (U_i = u_i) \land (V_i = v_i) \right]
\]
where $T' = [2n] \setminus \{j\}$. We are bounding the conditional probability:

$$\Pr \left[ A_j \vec{b}_k = U_j^{k+1} v_j^{j+1} + U_j v'_k + 1 \big| E \right].$$

Notice $\Pr[A_j = \vec{a}_j^T | E]$ is non-zero only if $\vec{a}_j \notin \text{Span}(\{\vec{a}_i\}_{i \in T'})$ since $A$ is invertible with probability 1. Call $Z = \text{Span}(\{\vec{a}_i\}_{i \in T'})$, then, the above equals:

$$\sum_{\vec{a}_j^T \notin Z} \Pr \left[ \left( \vec{a}_j^T \vec{b}_k = U_j^{k+1} v_j^{j+1} + U_j v'_k + 1 \right) \land (A_j = \vec{a}_j) \land E \right] \Pr[E].$$

Now take $e \in \mathbb{F}^{2n}$ such that:

- $e^T \vec{b}_i = 0$ for all $i \in T$;
- $e^T \vec{b}_k = 1$.

Note such an $e$ exists since the $\vec{b}_i$ with $i \in T$ and $\vec{b}_k$ are linearly independent. Since $\vec{a}_j^T \notin Z$ there is at most one $f \in \mathbb{F}$ such that $\vec{a}_j^T + fe \in Z$ (if it was possible to write a linear combination to two such values, one can easily write $\vec{a}_j^T$ as a linear combination of elements in $Z$). Therefore, for any function $F$, the summation:

$$\sum_{\vec{a}_j^T \notin Z} \sum_{f: \vec{a}_j^T + fe \notin Z} F(\vec{a}_j^T + fe)$$

will count each vector $b \notin Z$ as an input to $F$ at least $|\mathbb{F}| - 1$ times (it will be counted once for every $b - fe$ which is also not in $Z$). We can therefore bound the previous expression above by:

$$\frac{1}{|\mathbb{F}| - 1} \sum_{\vec{a}_j^T \notin Z} \sum_{f: \vec{a}_j^T + fe \notin Z} \Pr \left[ \left( (\vec{a}_j + fe^T) \vec{b}_k = U_j^{k+1} v_j^{j+1} + U_j v'_k + 1 \right) \land (A_j = \vec{a}_j + fe) \land E \right] \Pr[E].$$
We now claim if $\vec{a}_j^T + fe \not\in Z$ and $\vec{a}_j^T \not\in Z$, for any $u \in \mathbb{F}$:

$$\Pr[(U_j = u) \land (A_j = \vec{a}_j^T + f e) \land E] = \Pr[(U_j = u) \land (A_j = \vec{a}_j^T) \land E].$$

To see this, consider an equivalent way of generating the matrices $A$, $B$ and $C$. Recall in the generation procedure above first $C$ is generated according to some distribution, $B$ is generated as a random invertible matrix and $A$ is set as $A = CB^{-1}$. Observe that this is equivalent to the scenario where $C$ is generated according to the usual algorithm, then $A$ is generated as a random invertible matrix and $B$ is set as $B = A^{-1}C$. Keeping the second method of generation in mind, we can see that the probabilities that the matrix $A$ takes one of the two invertible matrices even with $U_j$ set to a specific value are equal:

$$\Pr[(U_j = u) \land (A_j = \vec{a}_j^T) \land \bigwedge_{i \in T'} (A_i = \vec{a}_i^T)] = \Pr[(U_j = u) \land (A_j = \vec{a}_j^T + f e) \land \bigwedge_{i \in T'} (A_i = \vec{a}_i^T)].$$

Notice that if we also fix the $(U_i, V_i : i \in T)$ and $t_i$ values, the $B_i$ values are determined - either $B_i = b_i$ with probability 1 if $< a_r, b_i >$ is the value implied by the correspond index in $C$ for all $r \in [2n]$ - or it is 0 if one such equality does not hold. Notice if $(b_i : i \in T)$ satisfy all these expressions for $A_j = \vec{a}_j^T$ they also satisfy it for $A_j = \vec{a}_j^T + f e$ since $eb_i = 0$ for $i \in [n] \setminus \{j\}$. Therefore:

$$\Pr[(U_j = u) \land (A_j = \vec{a}_j^T) \land E] =$$

$$\Pr[(U_j = u) \land (A_j = \vec{a}_j^T + f e_T) \land E],$$

which proves the claim. Therefore, the probability we are analyzing is equal to:

$$\frac{1}{|\mathbb{F}| - 1} \sum_{\vec{a}_j^T \not\in Z} \sum_{f \vec{a}_j^T + fe \not\in Z} \frac{\Pr\left[\vec{a}_j \vec{b}_k + f = U_j^{k+1} v_{j+1} + U_j v_k + 1 \land (A_j = \vec{a}_j) \land E\right]}{\Pr[E]},$$

91
If we call the event $\vec{a}_j \vec{b}_k + f = U_j^k v_{j_k}^i + U_j^k v_{j_k}^i + 1$, $F_f$ we see that all $F_f$ are mutually exclusive (over the choice of $f$) and $\sum_f \Pr[F_f] = 1$. Therefore, we can bound the probability above by:

$$\frac{1}{|F| - 1} \sum_{\vec{a}_j \not\in Z} \frac{\Pr[(A_j = \vec{a}_j) \land E]}{\Pr[E]} = \frac{1}{|F| - 1}$$

We have just shown that if the adversary returns $\vec{b}_k, v_k'$ with $\vec{b}_k \neq \vec{b}_k'$ or $v_k \neq v_k'$, with probability at least $\frac{n+1}{|F| - 1}$, $\vec{a}_j \vec{b}_k \neq u_{j}^{k+1} v_{j}^{i+1} + u_{j} v_{j} + 1$ for $P_j$ uncorrupted, which implies that the reconstruction will not succeed and $P_k$ will be placed on $\mathcal{L}_j$ with high probability as desired.

Since the generation procedure is identically distributed to the procedure which generates $A$ invertible first, generates $C$ as before and sets $B = A^{-1}C$, an identical argument proves failure with the same probability bound if either the $\vec{a}_j^T$, $u_k$ values are tampered with. This implies that if $P_k$ tampers with his share and $P_j$ is honest, with probability at least $(n + 1)(|F| - 1)^{-1}$, $P_k$ will be in $\mathcal{L}_j$. By a union bound, we can conclude that all $\mathcal{L}_j$ with $P_j$ uncorrupted will be all of the parties that tamper with their shares with probability at least $n^2 (n + 1)(|F| - 1)^{-1}$ which proves local identifiability with the parameters given in the theorem.

**Proof of Predictable Failures**

By the above argument, if a dishonest party $P_j$ changes any part of his share, with probability $1 - \frac{n(n+1)}{|F| - 1}$ reconstruction will fail he will be on the list of all honest $P_i$. Whether or not another corrupted party will appear on $\mathcal{L}_j$ can be computed directly from the adversary’s knowledge of the shares both parties return for reconstruction. Therefore with probability at least $1 - \frac{n^2(n+1)}{|F| - 1}$ every corrupted party’s list will be returned as:

- If corrupted $P_j$’s share was tampered, every uncorrupted $P_i$ will be on the list as well as any other $P_i$ corrupted that the verification check $(\vec{a}_i^T \vec{b}_j = u_{i}^{j+1} v_{j}^{i+1} + u_{i} v_{j} + 1)$ fails
with.

- If corrupted $P_j$’s share was not tampered, only other $P_i$ corrupted that the verification check $(\vec{a}_i \vec{b}_j = u_{i+1}^j v_{i+1}^j + u_i v_j + 1)$ fails with will be on $\mathcal{L}_j$.

- For honest $P_j$, $\mathcal{L}_j = \{P_i : P_i$ tampered with its share.$\}$.

### 3.3 Relaxing Local Identifiability

In this section we define a new commitment primitive, *unanimously identifiable commitments* that can be used as a leaner substitute for LISS in certain applications. Additionally, we note that this commitment primitive implies a variant of LISS (called *unanimously identifiable secret sharing*) that can also be used in our applications to MPC. This black-box construction of UISS may fail in identifying all cheaters, but has some advantages such as reduced algebraic complexity and a modification to secure against rushing adversaries (detailed at the end of the next section). Furthermore, using UIC directly in certain applications is more efficient in applications where the full UISS guarantees are not needed.

**Unanimously Identifiable Commitments**

A *unanimously identifiable commitment* (UIC) scheme has a single player (called the sender) committed to a value $s \in S$ by having a trusted dealer send commitments $c_i$ to all other players in the protocol and decommitment information $d$ to the sender such that any tampering of the $d$ value will cause all honest players to either reconstruct the original secret or fail reconstruction simultaneously. As with standard commitments, $(c_i)_{i \in [n]}$ should leak no information concerning $s$.

**Definition.** (Unanimously Identifiable Commitments) A $\delta$-UIC scheme consists of a randomized algorithm $\text{Offline}$ and a deterministic algorithm $\text{Decommit}$ with the
following syntax:

1. **Offline**: $S \rightarrow C^n \times D$. Takes as input a secret $s \in S$ outputs $n$ commitments $c_1, c_2, \ldots, c_n$ and decommitment information $d$.

2. **Decommit**: $C \times D \rightarrow S \cup \{\bot\}$. Takes as input $c_i$ and the decommitment information $d$ and recreates the secret $s$ or outputs $\bot$ indicating failure.

Where the algorithms (Offline, Decommit) should satisfy:

- **Completeness.** For any $s \in S$, if $Pr[Offline(s) = (c_1, c_2, \ldots, c_n, d)] > 0$ then, Decommit$(c_i, d) = s$ for any $i \in [n]$.

- **Secrecy.** The values $c_1, c_2, \ldots, c_n$ reveal no information concerning $s$. Formally, for any $\bar{c} = (c_1, c_2, \ldots, c_n)$ and any $s, s' \in S$, the probability that the first $n$ values of Offline$(s)$ is $\bar{c}$ is equal to the probability that the first $n$ values of Offline$(s')$ is $\bar{c}$.

The next natural requirement of this commitment scheme is the binding property. A natural way to define this binding property would be to require that an adversary who corrupts the sender and arbitrarily many other players can not produce decommitment information that will cause an honest player’s commitment to decommit to something other than the original $s$ or $\bot$. However, in such a case, if the adversary produced decommitment information that worked with some of the honest players and not with others, it would break agreement between the honest players, which is vital to our applications.

An alternative formulation would be that if the corrupt sender changes his decommitment information at all, it would cause the commitments of all honest players to decommit to $\bot$. While this would suffice, it’s a stronger requirement than we need. Instead, we will only require that a dishonest sender can only cause decommitment with all users to succeed or fail *simultaneously*, thus retaining agreement between the honest players. On a more technical note, we also require that the event that decommitment succeeds not reveal any
information to the adversary that he would not have otherwise have access to, which we capture with our simulation based definition below. We define the additional requirements formally (we name the sender $Q$) below:

A UIC scheme is secure if in addition to completeness and secrecy, there exists simulators $W_1, W_2$ such that the two guarantees described below hold with probability at least $1 - \delta$ for any $A$. Consider the following experiment:

1. The adversary, $A$ outputs a set $T \subset [n] \cup \{Q\}$ of players to corrupt;
2. $(c_1, c_2, \ldots, c_n, d) \leftarrow \text{Offline}(s)$;
3. For all $i \in T \cap [n]$ send $c_i$ to the adversary, if $Q \in T$ send $d$ to the adversary;
4. If $Q \notin T$, set $\text{dec} = d$; otherwise, $\text{dec}$ is output by $A$;
5. For all $i \in T \cap [n]$, $A$ outputs $(c_i', i)$, fake commitment information for $P_i$.

The guarantees around this experiment are as follows:

- **Binding with Agreement on Abort.** $\text{Decommit}(c_i, \text{dec}) = s$ for all $P_i$ uncorrupted or $\text{Decommit}(c_i, \text{dec}) = \perp$ for all $P_i$ uncorrupted.

- **Simulatable Abort.** Let $V$ be the view of $A$ at the end of 5., then:
  1. If $A$ corrupted $Q$:
     
     $W_1(V)$ correctly predicts if $\text{Decommit}(c_i, \text{dec}) = \perp$ for all $i \in [n]$.
  2. Otherwise:
     
     $W_2(V, c_i')$ correctly predicts if $\text{Decommit}(c_i', d) = \perp$ for each $i \in T \cap [n]$.
A Construction.

Let $\mathbb{F}$ be a field. We now give a construction of a $\delta$-UIC scheme with $S = \mathbb{F}$, $C = \mathbb{F}^{n+2}$ and $D = \mathbb{F}^2$.

**Offline($s$):**

1. Generate $P(X)$, a random $n + 1$ degree polynomial over $\mathbb{F}$ such that $P(0) = s$;
2. For all $i \in [n]$ generate $x_i \leftarrow \mathbb{F}^*$ and let $y_i = P(x_i)$;
3. Set $c_i = (x_i, y_i)$ and $d = P(X)$. Return $((c_i)_{i \in [n]}, d)$.

**Decommit**($c_i = (x_i, y_i), d = P(X)$ of degree $n + 1$) : If $P(x_i) \neq y_i$ return $\perp$. Else, return $P(0)$.

**Theorem.** Let $|\mathbb{F}| > (n + 1)^2\delta^{-1} + 1$. The scheme described above is a $\delta$-UIC with $S = \mathbb{F}$, $C = \mathbb{F}^{n+2}$ and $D = \mathbb{F}^2$.

**Proof.** Completeness and Secrecy follow trivially from elementary properties of polynomial interpolation. We begin therefore with the remaining guarantees:

**Proof of Binding with Agreement on Abort.**

This guarantee is non-trivial only if $A$ corrupts $Q$. Let $T$ be the set corrupter by $A$ and $T' = T \cap [n]$. If the adversary who receives $d = P(X)$ returns a different polynomial $P'(X)$, notice for any honest player $P_i$, Decommit($((x_i, y_i), P'(X)) = \perp$ if and only if $P(x_i) \neq P'(x_i)$.

As $P$ and $P'$ can only agree on $n + 1$ values and the adversary has no information on the value of $x_i$ this implies that Decommit($((x_i, y_i), P'(X)) = \perp$ with probability $1 - (n + 1)/|\mathbb{F}^*|$. By a union bound, this implies that this will be the case for all honest $P_i$ with probability at least $1 - n(n + 1)/|\mathbb{F}^*|$. 

96
Proof of Simulatable Abort.

We will first deal with the case where $A$ has corrupted $Q$. In this case, by the same argument as above, if the adversary changes $P(X) \rightarrow P'(X)$ then $\text{Decommit}(c_i, \text{dec}) = \perp$ for all uncorrupted $P_i$ with high probability, so defining $W_1$ to correctly predict whether $\text{Decommit}(c_i, \text{dec}) = \perp$ for honest $P_i$ can be accomplished by simply seeing if $P(X)$ was tampered.

Next, let us examine the case where $A$ has not corrupted $Q$. In this case, we define $W_2$ as follows. On input $V, c'_i = (x'_i, y'_i)$, $W_2$ will check whether or not the tuple $(x'_i, y'_i) = c_j$ for any $P_j \in T$. It will predict that $\text{Decommit}(c'_i, d) = \perp$ if and only if $(x'_i, y'_i) \neq c_j$ for any $P_j \in T$. Notice that when $(x'_i, y'_i) = c_j$ for some $P_j \in T$ then $W_2$ is always correct as $P(x'_i) = P(x_j) = y_j = y'_i$. Similarly if $(x'_i, y'_i)$ is not a $c_j$ value, then $W_2$ will be correct unless $P(x'_i) = y'_i$. Note that the adversary has no information concerning $P(X)$ except for the values it takes for at most $n$ points. As $P(X)$ is degree $n + 1$, for any other $x'_i$ value, the probability that $P(x'_i) = y'_i$ for any fixed $x'_i, y'_i$ is $|F|^{-1}$ proving that $W_2$ is correct on each of these predictions as well with probability $1 - |F|^{-1}$.

Related Concepts. In our applications, we mainly use UIC as a substitute for digital signatures. There are some other unconditional notions that have also been introduced for similar purposes (such pseudosignatures [63], distributed commitments [24] and IC signatures [19, 62]). While the construction itself is not novel (for example, it is used in [62]), the property that all of the honest players accept or reject the same commitment is crucial to our applications and differs from the guarantees placed on the other primitives.

3.4 Unanimously Identifiable Secret Sharing

In this section we formally define UISS and provide the reduction to UISS from our commitment primitive. Additionally, we give some benefits of constructing the UISS primitive
in this fashion rather than using our direct LISS construction at the end of the section.

**Definition (UISS)** A secret sharing scheme over $A$ is $\delta$-unanimously identifiable if:

1. **Unanimity:** For any adversary $A$, its probability of success in the following game is at most $\delta$ for any $s \in S$:

   1. $A$ outputs a set $T \subset [n]$ to corrupt;
   2. $(s_1, s_2, \ldots, s_n) \leftarrow \text{Share}(s)$;
   3. $A$ receives $(s_i : i \in T)$;
   4. $A$ outputs $(B, (s'_j)_{j \in T \cap B})$ such that $\{P_i : i \in B\}$ is in $A$ and is not completely corrupted;
   5. $\text{Out} \leftarrow \text{Rec}(B, (t_j)_{j \in B})$ where $t_j = s'_j$ if $j \in T$ and $t_j = s_j$ otherwise.

The adversary succeeds unless:

1. Reconstruction succeeds: $\text{Out} = s$ or,
2. Each honest party’s list is the same list of corrupted parties: $\text{Out} = (\mathcal{L}_j)_{j \in B}$ where for all $j,k \in B \setminus T$, $\mathcal{L}_j = \mathcal{L}_k \subset T \cap B$ and for some $j \in B$, $\mathcal{L}_j \neq \emptyset$.

2. The scheme has Predictable Failures.

**UISS from Unanimously Identifiable Commitments**

Let $(\text{Offline}, \text{Decommit})$ be a $\delta'$-unanimously identifiable commitment scheme, $(\text{Sh}, \text{Rc})$ be a secret sharing scheme over the access structure $A$ where $\text{Sh} : S \to T^n$ and $\text{Offline} : T \to C^n \times D$. We will give bounds $\delta'$ should satisfy after giving the construction in order for the resulting UISS scheme to be $\delta$-unanimously identifiable.

$\text{Share}(s)$:
1. Generate \((t_1, t_2, \ldots, t_n) \leftarrow \text{Sh}(s)\);

2. For all \(i \in [n]\), generate \((c_{1,i}, c_{2,i}, \ldots, c_{n,i}, d_i) \leftarrow \text{Offline}(t_i)\);

3. Set \(v_i = (\{c_{i,j}\}_{j \in [n]}, d_i), s_i = (t_i, v_i)\) for \(i \in [n]\) and return \((s_i)_{i \in [n]}\).

\text{Rec}(s_i : i \in B)\) where \(\{P_i : i \in B\} \in A:\)

1. Parse \(s_i = (t_i, (\{c_{i,j}\}_{j \in [n]}, d_i))\) for all \(j \in [n]\);

2. If for all \(i, j \in B\), \(\text{Decommit}(c_{i,j}, d_j) = t_j \neq \perp\):
   - Return \(s \leftarrow \text{Rc}(t_i : i \in B)\)

3. If for some \(i, j \in B\), \(\text{Decommit}(c_{i,j}, d_j) = \perp\):
   - For all \(i \in B\) set \(\mathcal{M}_i = \{j \in B : \text{Decommit}(c_{i,j}, d_j) = \perp\}\). Next set, \(\mathcal{L}_i = \{P_j : \mathcal{M}_j \neq \mathcal{M}_i\}\) and return \((\mathcal{L}_i)_{i \in B}\).

Notice that with a slight modification the \(t_i\) component of \(s_i\) is actually unnecessary since it can be reconstructed as \(\text{Decommit}(c_{i,i}, d_i)\) at the start of the \text{Rec} protocol but we include this factor in the definition and analysis in order to provide the leanest analysis. Using our previous unanimously identifiable commitment construction, this allows us to conclude:

**Theorem.** If \(\delta' < \delta n^{-2}\), the scheme described above is a \(\delta\)-UISS scheme over \(A\) with secret space \(S\) and share space \(S_i = C^n \times D\) for all \(i \in [n]\).

**Proof of Secrecy.** Notice that the only party who receives the value \(d_i\) is party \(P_i\) and the \(c_{i,j}\) values do not depend on the underlying \(t_i\). Therefore, any set of \(k-1\) parties’ shares have no information concerning the \(t_i\) values of for \(i\) not in the set - Therefore secrecy of the underlying secret sharing scheme (\(\text{Sh, Rc}\)) implies secrecy.
Proof of Unanimous Identifiability. By the guarantee of Binding with Agreement on Abort, if $P_j$ returns tampered $(t'_i, (\{c'_{i,j}\}_{j \in [n]}, d'_i))$ to reconstruction - With probability $1 - \delta/n$ either $\text{Decommit}(c_{k,i}, d'_i) = \bot$ for all $P_k$ uncorrupted or $\text{Decommit}(c_{k,i}, d'_i) = s_i$ for all $P_k$ uncorrupted. As long as either event holds for all corrupted $P_j$, the adversary fails in the Unanimous Identifiability game - which will happen with probability at least $1 - \delta$ by a union bound.

Proof of Predictable Failure. We define the function $\text{SRec}$ as follows:

Label the set of corrupted parties $T$, the set to be reconstructed on $B$ and the shares $P_i$, with $i \in T$ receives as $(t_i, (\{c_{i,j}\}_{j \in [n]}, d_i))$ and that a party in $T \cap B$ returns as $(t'_i, (\{c'_{i,j}\}_{j \in [n]}, d'_i))$ with each coming from the appropriate vector space.

- To determine if reconstruction succeeds:
  - For all $i \in T \cap B$ such that $d'_i \neq d_i$:
    1. Set $V_i = (T, (c_{j,i})_{j \in T}, d_i, d'_i)$;
    2. Set $D_i = \{i \in T \cap B : W_1(V_i) = 1\}$. Recall $W_1(V_i) = 1$ implies all honest parties will reject $d'_i$ with high probability.
  - For all $i \in B \setminus T$:
    1. Set $V'_i$ as $(T, (c_{j,i})_{j \in T})$;
    2. Set $E_i = \{j \in T \cap B : W_2(V'_i, c'_{i,j}) = 1\}$. Recall $W_2(V'_i, c'_{i,j}) = 1$ implies honest party $P_i$ will reject $c'_{i,j}$ with high probability.

- If $D_i = \emptyset$ for all $i \in T \cap B$ and $E_i = \emptyset$ for all $i \in B \setminus T$ return SUCCESS. Else:
  - For $i \in T$, set $L_i = E_i$. For $i \in B \setminus T$, set $L_i = \{P_j : P_i \in D_j\}$.

Notice that if the $W_1$ and $W_2$ algorithms correctly predict the behavior of $\text{Decommit}$, the
lists will be correct. Since the algorithms are called at most \( n^2 \) times, by a union bound, they will be correct with probability at least 
\[ 1 - \delta n^{-2} \times n^2 = 1 - \delta \] as desired.

For \( q > -\log(\delta n^{-2}) + \log(n) + 1 \), \( F = GF(2^q) \) and \( \delta' = \delta n^{-2} \), our previous unanimously identifiable commitment construction is a \( \delta' \)-unanimously identifiable commitment scheme with (using the notation from this section), \( T = F, C = F^{n+3} \) and \( D = F^{n+2} \). If we interpret the output of the internal secret sharing scheme as an element in \( F \) (where \( q \) is taken to be at least as many bits as the share-length), we conclude:

**Theorem.** For any \( \delta > 0 \), \( S \) and \( A \) where (Share, Rec) is a secret sharing scheme with secret space \( S \) over \( A \) with share length \( \beta \), there is a \( \delta \)-UISS scheme over \( A \) with secret space \( S \) and share length \( O(n^2 \log(\delta^{-1}n) + n^2\beta) \).

### 3.4.1 Rushing Adversaries

A rushing adversary is one that takes advantage of the fact that synchronicity is difficult to realize in applications. A rushing adversary is allowed to observe all communication directed to it (or sent publicly) in a given step of a protocol and only submit its own output after all other parties have submitted theirs. Consider a scheme where instead of submitting their shares to a central implementation of Rec all parties are expected to simultaneously broadcast their shares. Clearly secrecy can not be maintained against a rushing adversary since it can observe \( k - 1 \) honest shares and refuse to submit his own. Similarly, unanimity can not be maintained since an adversary who observes all other shares can construct his own so that reconstruction succeeds to a different secret. However, consider instead a setting where instead of a reconstruction algorithm taking all shares as input and returning a list to every party, the parties broadcast their shares and at the end either reconstruct the secret or internally output a list \( L \) of dishonest parties who deviated from the protocol. This is actually possible with a slight modification to our algorithm precisely because of the underlying unanimously identifiable commitment scheme.
The **Share** phase will remain unchanged whereas the broadcast phase will consist of every party broadcasting their \((t_i, d_i)\) values. Notice that in this scheme the adversary will not have access to any of the \(c_{j,i}\) values if \(P_j\) is uncorrupted. However, any chance to tampering with the \(d_i\) value (or modify the \(t_i\) value without modifying \(d_i\)) will result in \(\text{Decommit}(c_{j,i}, d_i) = \bot\) for all honest \(P_j\). This allows once again list \(L_j\) to be created as all \(P_i\) such that \(\text{Decommit}(c_{j,i}, d_i) = \bot\), satisfying all previous security requirements on list simulatability and unanimity amongst honest parties.

### 3.5 Applications

A secure multiparty computation (MPC) protocol allows a set of players to compute a function evaluated on their individual inputs while revealing no information other than the output of the function. We assume familiarity with (standalone) MPC throughout this section and refer the reader to [12] for formal definitions.

**Model of Computation**

By default, we consider static, computationally unbounded adversaries who may corrupt up to \(t\) of the \(n\) parties \((t = n\) by default). We consider both *active* adversaries, who may arbitrarily control the corrupted players, *passive* adversaries, who can only observe the internal state of corrupted players, and *fail-stop* adversaries who behave like passive adversaries except that they can make corrupted players stop sending messages. Our network model is synchronous with point-to-point channels and a broadcast channel.

The security of an MPC protocol with respect to an ideal functionality \(f\) is defined by comparing a real world execution of the protocol to an *ideal model* execution where a trusted party evaluates \(f\). By default, we refer to *statistical* security, where the statistical advantage of distinguishing between the real world and the ideal model execution is bounded by \(2^{-\kappa}\).
for a statistical security parameter \( \kappa \). We will only consider the case of secure function evaluation, in which \( f \) is stateless. We will mostly consider fully secure MPC in which the ideal model adversary cannot prevent the trusted party from sending the outputs of \( f \) to the honest players. Full security cannot be achieved even for some simple functionalities such as coin-flipping [18] without an honest majority or other assumptions we will discuss. This impossibility holds even with trusted preprocessing; however, in the latter model the assumptions of secure point-to-point channels and a broadcast primitive are unnecessary as they can be implemented unconditionally [64].

**Complete Primitives for MPC**

An \( n \)-party functionality \( g \) is called a complete primitive for \( n \)-party MPC if it is possible to securely realize any \( n \)-party functionality \( f \) in the \( g \)-hybrid model, namely by using ideal calls to \( g \). Here we consider security against an active adversary who may corrupt an arbitrary number of players.

In prior works, such primitives either depend on the complexity of the function being evaluated [25] or rely on cryptographic assumptions [33]. It remained open to construct an unconditionally complete primitive whose complexity is independent of the complexity of the evaluated function \( f \). In the following section, we show how to construct such a primitive. Our contribution can be seen as identifying a cryptographic LISS scheme implicitly present in the construction of Gordon et al. [33] and replacing it with an unconditional construction. We now present their scheme and identify the implicit LISS scheme.

**A Complete Primitive for Computational MPC [33]**

We present the complete primitive FCR presented in [33]. Since we believe it will be more instructive to see the primitive in action, we begin by describing preprocessing before defining the primitive. Before calling FCR to compute a functionality \( F \), the \( n \) parties perform secure
computation with abort for a related $F'$. $F'(x_1, \ldots, x_n)$ computes:

$$\text{key} = \bigoplus_{i=1}^{n} \text{key}_i \text{ and } o = \text{Enc}_{\text{key}}(F(x_1, \ldots, x_n))$$

where $\text{Enc}$ is either a one-time-pad or semantically secure symmetric key encryption scheme (the implication of either choice is outlined in the theorem statement below) and $\text{key}_i$ with $i \in [n-1]$ are chosen at random of the same length as $\text{key}$ and $\text{key}$ is a uniformly generated key for the scheme implied by the choice of $\text{Enc}$. It then generates a key pair ($sk, vk$) for a cryptographic signature scheme and computes $\sigma_i = \text{Sign}(sk, (\text{key}_i, i))$ for every $i \in [1, n]$. The output of $F'(x_1, \ldots, x_n)$ to party $i$ is $(o, vk, \text{key}_i, \sigma_i)$. If a party aborts during this phase the remaining parties substitute a default input for the aborting party and begin again.

The FCR primitive is then called on the values $(vk, \text{key}_i, \sigma_i)$. The output of FCR is as follows 1:

- If not all parties submit the same value for $vk$ then FCR partitions the parties according to the values of $vk$ they submitted and returns these partitions. Each partition repeats the protocol amongst itself, inserting default values as the inputs of the parties not in the partition.

- If $P_i$ submits $(vk, \text{key}_i, \sigma_i)$ such that $\text{Verify}(vk, (\text{key}_i, i), \sigma_i) = \text{FALSE}$ (in other words, $\sigma_i$ is not a signature of $(\text{key}_i, i)$ under $sk$), FCR identifies party $i$ as disqualifies party $i$. The remaining parties repeat the protocol amongst themselves with a default input for $P_i$.

- If all parties submit valid signatures under the same key $vk$, FCR returns $\text{key} = \bigoplus_{i=1}^{n} \text{key}_i$.

1Note that in [33] the value signed is actually $\text{key}_i$ instead of $(\text{key}_i, i)$. Their construction is vulnerable to either share exchanging or duplication, we present the corrected version.
Theorem. (FCR is Complete for Fairness [33]) Suppose an oblivious transfer protocol exists. There is a $\alpha$-bit primitive complete for computational fully secure multi-party computation where 1. With no additional assumption, $\alpha$ is the output size of the function being evaluated; 2. If one-way-functions exist, $\alpha$ is polynomial in the security parameter.

Note that in their construction, when a subset of parties repeat the protocol, there is an implicit assumption that these parties amongst themselves re-initialize a new primitive that will take input only from this subset of the parties. Since it’s difficult to see how this model can be realized in the real-world, we do not make this assumption in our construction. This adds in the additional challenge that each party must declare the other parties it is currently running the protocol with.

Unconditional Primitive. The first primitive we present is complete for statistically secure MPC and its complexity depends only on the output length of the evaluated functionality $f$. We describe our primitive as having three modes, meaning that on each round every player will give an additional input saying which mode of the primitive it is supplying an input for. If a given player is not supplying input to a certain mode, its input is set to a default value in this mode and it is returned no output from this mode. We begin with an informal description of how the primitive works before defining it in complete formality.

- **FCR$_1$** - Takes as input a bit from a player, runs an $n$-out-of-$n$ UISS sharing algorithm on this bit and distributes the shares amongst all players.

- **FCR$_2$** - Takes as input two tuples of shares from the UISS scheme. Internally, the primitive reconstructs the underlying secrets of each, evaluates the NAND of the two secrets, and re-shares the output using the UISS scheme. If reconstruction fails, the functionality will use the lists $L_j$ output by the UISS scheme to partition the players:

$^2$Note that the construction also uses cryptographic signatures, which are implied by OT.
If any player is on his own list, the functionality declares this player is disqualified and his input is replaced by a default value by all players. If not, the functionality outputs a proper partition of the players so that all honest players remain in the same partition. It achieves this by letting $P_i$ and $P_j$ remain in the same partition if $L_i = L_j$.

- $FCR^1_3$ - Takes as input $\beta$ separate tuples of UISS shares, where $\beta$ is an output length parameter. The functionality either reconstructs each secret and broadcasts all the reconstructed bits, or, if some reconstruction fails, partitions the players as in the previous mode using the first instance of failed reconstruction.

Note that while the first two primitives are randomized, they can be made deterministic by using a standard reduction: the internal randomness can be securely emulated by taking the XOR of shares contributed by all players.

Using the above primitive, one can securely evaluate any boolean circuit $C$, which consists of NAND gates and has $\beta$ output bits, in the following way. The players first use $FCR^1_1$ to share each of their input bits. After this phase is completed, for each gate in $C$ the players use $FCR^1_2$ to evaluate shares of the value of each internal value in $C$. Finally, the players feed the shares of the output values to $FCR^1_3$ and receive the outputs of $C$.

Notice that any deviation from the above protocol will result in all honest players identifying the same set of cheaters by the UISS guarantee, allowing them to restart the computation internally while eliminating at least one cheater. Additionally, by these guarantees, the partitions can be simulated given only the views of the corrupted players. Defining the three modes of operation as one primitive that can be called on only some partition of the players requires some additional technical steps to fit in with our model of one trusted primitive. We now begin the formal description of the primitive.

At each step, the party $P_i$ will also be supplying the primitive a list of players $M_i$ it is currently running the protocol with. A set of players $R \subset P$ (with lists $(M_i : P_i \in R)$) will
be called consistent if:

- Each party in \( R \) agrees it is running the protocol with all of \( R \): \( M_i = R \) for all \( P_i \in R \);

If a given subset of parties is consistent, the scheme will attempt to run the reconstruction algorithm and use the UISS guarantee to further partition them if not possible. If a party \( P_i \) is not the part of a consistent set, this implies that either \( P_i \) is not supplying the \( M_i \) value assigned by the protocol or a corrupted \( P_j \in M_i \) has changed its \( M_j \) value. In such a case it will actually be possible to further refine the partitioning since the honest \( P_j \) will not modify the \( M_j \) values assigned to them by the primitive.

**Definition.** The primitive \( FCR_1^1 \) is defined as follows where \((Share_j,Rec_j)\) is a \( j\)-out-of-\( j\) \( \delta \)-UISS scheme. The scheme has three modes:

1. \( FCR_1^1((b_1,M_1),(b_2,M_2),\ldots,(b_n,M_n)) \): This mode takes the input bits of the parties sequentially and secret shares them amongst the other parties they are currently executing the protocol with.
   - For \( R \subset \mathcal{P} \) consistent:
     - Generate \((s_j)_{j \in R} \leftarrow Share_{|R|}(b_i)\) where \( b \in \{0,1\} \) and send \((i,R,s_j)\) to \( P_j \in R \)
   - For all \( P_i \) not in a consistent set, let \( M'_i = \{P_j : P_j \in M_i \land M_j = M_i\} \) and return \( M'_i \) to \( P_i \). We call this process refining \( P_i \)'s partition and will reuse it in the future. As described below, the players in \( P_i \)'s new partition will now re-start computation internally by inserting default inputs for players not in this partition and updating their \( M_i \) value to \( M'_i \).

2. \( FCR_2^1((s_1,t_1,M_1),(s_2,t_2,M_2),\ldots,(s_n,t_n,M_n)) \):
   - For \( R \subset \mathcal{P} \) consistent:
     - Run \( Rec_{|R|}(s_i : i \in R) \rightarrow Out \) and \( Rec_{|R|}(t_i : i \in R) \rightarrow Out' \).
- If either reconstruction fails, return $M'_i$ for $i \in R$ as in $FCR^1_i$ based on $(L_i : i \in R)$: $P_i$ and $P_j$ in $R$ remain in the same partition if any only if $L_i = L_j$ and $P_j \notin L_i$. Formally, $M'_i = \{P_j : P_j \in M_i \land L_i = L_j \land P_j \notin L_i\}$.
- Otherwise, compute $z = \text{Out NAND Out}'$; $(z_i : i \in R) \leftarrow \text{Share}_{R}(z)$ and return $z_i$ to $P_i$.

- For all $P_i$ not in a consistent set, refine $P_i$’s partition, similarly to above.

3. $FCR^1_3(((s_{1,i})_{i \in [\alpha]}, M_1), ((s_{2,i})_{i \in [\alpha]}, M_2), \ldots, ((s_{n,i})_{i \in [\alpha]}, M_n))$:

- For $R \subset P$ consistent:
  - $\text{Rec}_{R}(s_{j,i} : j \in R) \rightarrow \text{Out}_i$ for $i \in [n]$;
  - If any reconstruction fails, return $M'_i$ for $i \in R$ as in $FCR^1_2$ based on $(L_i : i \in R)$;
  - Otherwise return $s = \text{Out}_1 \circ \text{Out}_2 \circ \text{Out}_3 \ldots \text{Out}_\alpha$ to all $P_i \in R$.

- For all $P_i$ not in a consistent set, refine $P_i$’s partition.

The scheme begins with $M_i = [n]$ set for all $P_i \in P$ - Anytime the primitive is called if a party receives an $M_i$ output from the primitive, it updates its $M_i$ value with the one output by the primitive and restarts the protocol with the updated $M_i$ value. Notice that if at any point reconstruction fails we non-trivially partition the players in the partition that submitted the shares such that all honest players remain in the same partition.

To use this primitive to securely evaluate $f$, first each party calls $FCR^1_2$ on each bit of their input which is then shared to all parties. For each NAND gate of $f$, $FCR^1_2$ then allows all parties to return their shares for a given bit internal to the circuit and have the primitive securely share the output bit. Finally, $FCR^1_3$ takes as input shares of all the output bits, reconstructs them and returns their concatenation. Notice that since this primitive calls $\text{Share}$ it is randomized - However, since it take input from all $n$ players we
can modify it to be deterministic if we let it accept from each party its usual input as well as a random string. The randomness for the primitive will then be determined by XOR'ing the randomness supplied by all the players. As long as at least one party involved is honest, his randomness will be correctly distributed, allowing the randomness of the primitive to be simulated deterministically. If at any point the primitive instead re-partitions the players the protocol is re-run among each partition with default values substituted for the players not in the partition.

**Theorem.** There is a deterministic, polynomial-time computable functionality $g$ with input and output size $\text{poly}(n, \kappa, \beta)$ such that any functionality $f$ computed by a circuit of size $\sigma$ and output length $\beta$ can be realized with full statistical security (and $2^{-\kappa}$ simulation error) using $\text{poly}(n, \kappa, \sigma)$ calls to $g$.

Moreover, we can reduce the algebraic complexity of either of our primitives by applying the technique of randomizing polynomials [3]: For any finite field $\mathbb{F}$, there exists a primitive $g$ as above for either theorem with algebraic degree 3 over $\mathbb{F}$.

**Reducing the Number of Calls.** Our second primitive improves on efficiency over the first by requiring fewer calls, but requires a preprocessing phase which is implemented using an MPC with identifiability on aborts (in other words, if the protocol fails then all honest players agree on the identity of a corrupted player.) Settling for computational security, such a protocol can be based on the existence of (two-party) oblivious transfer [32].

**The Second Construction.** We now describe our second primitive where the number of invocations is independent of the complexity of the function to be evaluated. The primitive requires a preprocessing phase which is implemented using a multi-party computation protocol with cheat detection (in other words, if the protocol fails at least one corrupt party is identified) such as [32], since the output is in a secret shared form fairness is actually implied by such a protocol (see [33] for details). As mentioned, this construction is achieved
with the assumption of oblivious transfer but avoids the use of cryptography in the primitive. We again begin with an informal description before providing the formal functionality.

The protocol for $f$ begins by the having the players run an MPC protocol as above to compute UISS-shares of the output of $f$. In case the preliminary MPC protocol fails, all players disqualify the player that caused the abort and restart the protocol by using a default value as the input of disqualified players.

- $FCR^2$ takes as input a tuple of UISS shares for a $\beta$-bit secret and reconstructs the secret. In case reconstruction succeeds the primitive returns the reconstructed value to all players. If reconstruction fails, the primitive outputs a partition of the players by the lists output by the UISS scheme as in $FCR^1$.

The protocol for $f$ proceeds by repeatedly interleaving the preliminary (computational) MPC with calls to $FCR^2$ until an output value is successfully reconstructed by the latter. Each failure results in the honest players disqualifying at least one corrupted player. As before, in each point of the protocol all honest players agree on the identity of disqualified players. We now provide the full description:

**Definition** The primitive $FCR^2$ is defined as follows where $(Share_j, Rec_j)$ is a $2^{-\kappa} j$-out-of-$j$ UISS scheme from $S \rightarrow \prod_{i \in [1,j]} C_i$ for all $j \in [1,n]$:

- **On input** $((s_1, M_1), (s_2, M_2), \ldots, (s_n, M_n))$:

- **For all** $R \subset \mathcal{P}$ consistent:
  - Run $Rec_{\mid R\mid}((s_i)_{i \in R}) \rightarrow Out$;
  - If $Out = s$, implying reconstruction succeeded, return $s$ to all $P_i \in R$;
  - If $Out = (L_i)_{i \in R}$, partition the parties according to their lists as in $FCR^1$

- **For all** $P_i$ not in a consistent set, refine $P_i$’s partition.
The preprocessing stage begins by each player who has not yet received a secret broadcasting its $M_i$ value. The players will then partition themselves based on these values - A player $P_i$ will be in the same partition as $P_j$ if $M_i = M_j$ and $P_i \in M_i$ and $P_j \in M_i$. The preprocessing phase is run independently for each partition to evaluate $(s_j : j \in M_i) \leftarrow \text{Share}_{|M_i|}(\text{key})$ where $\text{key}$ is a uniformly generated key to the semantically secure symmetric key encryption scheme $(\text{Enc}, \text{Dec})$ (whose existence follows from our assumption of OT [42]) and returns to $P_i$ in the partition $(s_i, \text{Enc}_{\text{key}}(F(x_1, x_2, \ldots, x_n)))$ where the $x_i$ values are chosen by the players $P_i$ if $P_i$ is in this partition or set to a default value otherwise. Note that implementing the preprocessing phase by using a secure-with-abort MPC with dishonest majority is possible [32] using oblivious transfer so while cryptographic assumptions may be needed in the protocol, none are needed for the complete primitive. If the preprocessing phase fails for a partition, the parties use the cheater-detection property of the protocol to identify a dishonest player and disqualify him, removing them from their $M_i$ values and re-starting the preprocessing phase. If the preprocessing phase succeeds for a partition, it then uses the primitive defined to attempt to reconstruct the output $F(x_1, x_2, \ldots, x_n)$. Notice by calling the primitive, it will either succeed in doing so, or further refine its partition with high probability.

Since each time the primitive is called either the honest players are able to reconstruct the secret or they eliminate the same dishonest parties from their partition we can bound the number of calls of the primitive by $n$ before the honest players have successfully evaluated the function and conclude:

**Theorem.** Suppose an oblivious transfer protocol exists. Then there is a deterministic, polynomial-time computable functionality $g$ with input and output size $\text{poly}(n, \kappa)$ such that any polynomial time computable $f$ (with arbitrary input and output size) can be realized with full computational security (and $\text{negl}(\lambda) + 2^{-\kappa}$ simulation error) using at most $n$ calls to $g$. 
**All Subsets Variant.** As mentioned, in the worst case the above protocol requires \( n \) separate calls to \( FCR^2 \). In cases where the number of players is small this can be improved by having the protocol executed initially between all subsets of players simultaneously. In this case, if the reconstruction fails, the functionality can then partition the players and use the protocol execution according to each partition to get the output of this partition without having them re-run the protocol (this also introduces a \( 2^n \) factor in the simulation error).

**A Negative Result.** As both primitives we give take input from all \( n \) players; a natural question to ask is if any pairwise functionality would suffice to achieve fair computation. This was shown impossible by Fitzi et al. [25] but another natural question is whether or not the relaxed notion of identifiability with abort where in the ideal world, the adversary may receive the output and cause the protocol to abort, but in this case all honest parties isolate a single dishonest player is possible. We show that in fact, no pairwise functionality is complete for identifiability with abort in Section 3.7.

**Partial Fairness with preprocessing**

In this section we briefly sketch how the unanimously identifiable commitments (UIC) primitive can be used with the partially fair MPC protocols of Beimel et al. [5, 4] to eliminate the assumption of cryptographic signatures in the presence of trusted preprocessing.

**Construction with an Off-Line Dealer.** The MPC protocols from [5, 4] achieve unconditional security against fail-stop adversaries (with a non-negligible error) given a trusted preprocessing phase in which a dealer sends some secret information to each player. This information contains the messages each player should send during the protocol, but the choice of which message is sent may depend on the (public) identities of the players that aborted up to this point. To upgrade the security of such a protocol to hold against active adversaries, Beimel et al. rely on digital signatures to ensure that players do not deviate from their designated messages to upgrade security against fail-stop security to security against
active adversaries (this is combined with another step to eliminate trusted preprocessing not applicable to our result). Our observation is that one could instead rely on the UIC primitive by having the dealer give to the player who should send a message the decommitment information for this message and to all other players the corresponding commitments. Then, if a corrupted player attempts to modify this decommitment information, all honest players will recognize this simultaneously and continue the execution as if this player had aborted.

Note that when considering general MPC in this model (rather than coin-flipping), it may be useful to allow the preprocessing stage to depend on the players’ inputs. We refer to such a preprocessing phase as input dependent preprocessing. Since we require the outputs of the protocol to be unpredictable in the end of the preprocessing phase,\(^3\) input dependent preprocessing cannot be used to trivially solve the problem by simply delivering the outputs of \(f\) to the players.

**Active security with preprocessing**

We now give our main result on upgrading fail-stop security in the case where the users’ inputs are determined before the preprocessing stage. We then provide our variant allowing inputs as long as the domain and number of users remain small.

**Theorem.** Let \(\mathcal{P}\) be an \(r\)-round protocol with input dependent preprocessing, which realizes \(F\) with \(\epsilon\)-security against fail-stop adversaries who can corrupt up to \(t\) players. Furthermore, suppose that the online phase of \(\mathcal{P}\) has the following structure: in each round, each player sends a subset of the messages it had received in the preprocessing phase, where the identity of this subset can be computed publicly from the pattern of aborts up to this round. Then, there is a protocol \(\mathcal{P}'\) with the same features of \(\mathcal{P}\) except that it is \((\epsilon + 2^{-\kappa})\)-secure against active adversaries.

\(^3\)More precisely, security in the preprocessing model requires the ability to simulate the adversary’s view in the preprocessing phase before invoking the ideal functionality.
**Proof.** Modifying $\mathcal{P}$ to $\mathcal{P}'$ is straightforward: Each $M$ in $P_i$’s round $j$ message set (the set of messages that $P_i$ will choose its $j^{th}$ round message from) in $\mathcal{P}$ will be committed to by a unanimously identifiable commitment scheme $\text{Offline}(M) \rightarrow (d, c_1, c_2, \ldots, c_{n-1})$ where each $c_i$ is distributed to one of the additional players while $P_j$ (the player to send the message) gets $d$. If in the protocol $\mathcal{P}$, $P_j$ was supposed to broadcast $M$, it will instead broadcast the decommitment information $d$. Each player will then attempt to reconstruct the original message using $\text{Decommit}(d, c_i)$ - If the process succeeds the player will proceed as if $P_j$ had broadcast this message in $\mathcal{P}$. If not, it will treat this player as having aborted. Notice that by the **Binding with Agreement on Abort** property, with high probability all honest users will classify a player as having aborted together.

Let $A$ be an adversary for $\mathcal{P}'$, we will show how to simulate its view through an adversary $B$ in $\mathcal{P}$. When $A$ corrupts a player in $\mathcal{P}'$, $B$ will corrupt the same players in $\mathcal{P}$ - After $B$ receives the message tuples $\mathcal{M}$ for the corrupted players, our simulator will generate $\text{Offline}(M) \rightarrow (d, c_1, \ldots, c_{n-1})$, distribute $d$ to the player $M$ was assigned to and the corresponding commitments to the other corrupted players. For every $M$ that would be sent to an honest player in $\mathcal{P}$ the simulator generates $\text{Offline}(0) \rightarrow (d, c_1, c_2, \ldots, c_{n-1})$ and distributes the appropriate commitments to the remaining players - recall by the secrecy requirement this sequence of commitments can be explained as being commitments to any value with the appropriate choice of $d$.

During the execution of the protocol, if $A$ ever modifies a $d$ value it is to broadcast, the simulator will run the $W_1$ simulator to see if this value would cause the honest players to abort, if so, $B$ aborts this player from the protocol. For all messages $M$ broadcast by non-corrupted players, the simulator (which we assume is computationally unbounded) will find decommitment information value $d$ which causes $\text{Decommit}(d, c) \rightarrow M$ for all of the corrupted players’ $c$ values that correspond to this message. Since the fully secure computation phase can be simulated similarly, since it allows only a minority of the players in it to be corrupted,
this allows the view of $A$ to be completely simulated by an adversary corrupting $\mathcal{P}'$, implying that the view of the adversary can be simulated with probability of error at most $\epsilon$ from the ideal functionality. Furthermore through a similar transformation it is possible to simulate the views of the honest parties in $\mathcal{P}'$ interacting with $A$ to the honest parties in $\mathcal{P}$, giving full $(\epsilon + 2^{-\kappa})$-security where the $2^{-\kappa}$ factor is introduced due to simulation error in the UISS scheme.

It can be observed that the protocols with an offline dealer in [5, 4] satisfy the conditions of the previous theorem. We now present the two main results we can derive by applying our above result to their constructions. Using the above result with the coin-flipping protocol with preprocessing implicit in the construction from [5], we get the following corollary.

**Theorem.** Assume preprocessing by a trusted off-line dealer. Fix constants $n$ and $t$ such that $t < 2n/3$. Then, for any $r$, there is an $r$-round $n$-party unconditionally secure coin-flipping protocol over point-to-point channels tolerating up to $t$ malicious players with bias $O(1/r)$.

**Allowing inputs.** Adapting our scheme to allow users to have individual inputs is straightforward when $d^n$ is kept polynomially bounded where $d$ is an upper bound on the size of the input domain for each player. For this, the preprocessing authority will run the preprocessing phase while running over all possible inputs of the players. However, instead of labeling the inputs of each player in the preprocessing information, it will generate a random permutation $p_i$ over the domain of each player $P_i$ and label the preprocessing information corresponding to input set $(i_1, i_2, \ldots, i_n)$ as $(p_1(i_1), p_2(i_2), \ldots, p_n(i_n))$ and attach $p_i$ to the preprocessing information sent to $P_i$. There is then an additional round at the beginning of the protocol where each player $P_i$ broadcasts $p_i(x_i)$ for its actual input $x_i$ and the players use the preprocessing information labeled by the broadcast values for their computation - The additional preprocessing values given to the adversary do not hinder simulation as we require the preprocessing information be simulatable given only the view of the adversary. This
allows us to conclude the same result as Theorem 3.5 where input dependent preprocessing is replaced with \(d^n\) being polynomial in \(\kappa\).

In particular, combining this result with the general MPC protocol from [4] with preprocessing allows us to conclude:

**Theorem.** Assume preprocessing by a trusted off-line dealer. Let \(n\) and \(t\) be constants such that \(n/2 \leq t < 2n/3\) and \(\mathcal{F}\) be a deterministic \(n\)-party functionality with input domain bounded by a polynomial \(d(\kappa)\) for each player. Then, for any polynomial \(p(\kappa)\), there is a polynomial-time \(r\)-round 1/p secure protocol for \(\mathcal{F}\) which tolerates up to \(t\) corrupt players with \(r = pd^{n/2t}\).

### 3.6 Limitations of Pairwise Authentication

In this section we would like to informally ask whether the method of pairwise authentication (as used by Rabin and Ben-Or in [66]) can be used to achieve unanimous identifiability, as this is the most natural approach to consider. Note that this section is meant more as intuition, showing that a scheme does not exist that utilizes the framework in a ‘black-box’ fashion.

We model the framework in the context of secret sharing with a scheme with two levels of secret sharing. The first level consists of a secret \(s\) being shared in an \(n\)-out-of-\(n\) secret sharing scheme (we call this the top level sharing). In addition to each player \(P_i\) receiving his share from the top level scheme - the player will also receive verification information for each other player’s secret in the form of an 2-out-of-2 \(\epsilon\)-non-malleable secret share [33] (that detects any modification in a user’s returned share as long as not all players are corrupted) of that player’s secret (note that this is a stronger notion than the allocation in [66] since such a scheme will detect any tampering with high probability). We define this secret sharing scheme below:
Share(s):

1. Share \( s \) additively as \( s = s_1 \oplus s_2 \oplus \ldots \oplus s_n \);

2. For each \((i, j) \in [n] \times [n] \) with \( i \neq j \) set \( \text{NMSS.Share}(s_i) \rightarrow (t_{i,j}, u_{i,j}) \);

3. Return to \( P_i : (t_{i,j}, (u_{j,i} \forall j \neq i)) \).

**Theorem.** There is no reconstruction function \( S.\text{Rec}(s) \) such that \((S.\text{Share}, S.\text{Rec})\) is a UISS for \( n \) players with \( n \geq 3 \).

**Proof.** For notational simplicity, assume that the share space of each player in \( \text{NMSS} \) is equal (call it \( S \)) and both the first and second indices of \( \text{NMSS.Share}(s) \) for any \( s \) are uniformly distributed over \( S \) (though clearly they are not independent) for notational simplicity, though the analysis holds for the more general case as well. We will show the statement for \( n = 3 \), the remaining cases follow identically. Assume to the contrary that \((S.\text{Share}, S.\text{Rec})\) is a UISS scheme.

Consider the adversarial strategy where \( A \) corrupts \( P_1 \) and \( P_2 \) and proceeds as follows: \( A \) replaces \( s_1 \) with randomly chosen \( s'_1 \) and \( t_{1,2}, u_{1,2} \) with newly chosen shares from \( \text{NMSS.Share}(s'_1) \) and \( t_{1,3} \) and \( t_{3,1} \) with uniformly chosen shares from \( S \). By our assumption that \( \text{NMSS} \) has perfect secrecy, all information concerning the original secret is lost in this transformation and therefore, we can assume with probability at least \( p = 1 - \epsilon - |S|^{-1} \) the reconstruction function \( \text{Rec} \) fails in reconstruction and outputs a tuple of lists \((L_1, L_2, L_3)\). Call the distribution of all shares returned in this adversarial strategy when the original secret \( s \) is chosen at random from the secret space \( S, \mathbb{D} \).

First, notice that the distribution \( \mathbb{D} \) is identical in the case where \( s'_1 \) is not chosen at random, but is instead kept as \( s'_1 = s_1 \) (recall we assume \( s \) is chosen uniformly). Then, \( \mathbb{D} \) is equal to the distribution where \( s \) is additively shared into \( s_1 \oplus s_2 \oplus s_3 \) and for all
$(i,j) \notin \{(1,3),(3,1)\}$ \(t_{i,j}, u_{i,j} \leftarrow \text{NMSS.Share}(s_i)\) and otherwise \(t_{i,j} \leftarrow \$\) \(S\) and \(u_{i,j} \leftarrow \$\) \(S\). It is now easy to observe that the distribution \(D\) can be simulated by the adversary corrupting only \(P_1\) and re-randomizing its shares \(t_{1,3}\) and \(u_{3,1}\) - or in the case where \(P_3\) is corrupted and \(t_{3,1}\) and \(u_{1,3}\) are re-randomized. Notice this implies that \(L_2\) can not consist of only the corrupted party with probability greater than \(1/2\). An identical argument holds when \(n > 3\).

However, it may be the case that we were simply being too naive by using a simple additive secret sharing scheme at the top level. We will now investigate the problem when we replace the top secret sharing with \(\text{NMSS.Share}(s_i) \rightarrow (s_1, s_2, \ldots, s_n)\). Call this modified secret sharing scheme \(T.\text{Share}\). Surprisingly we find that this does give a good UISS scheme if and only if \(n \leq 4\).

**Theorem.** There is an absolute constant \(C\) such that for any \(\delta > Cn^2\epsilon\), there is a reconstruction function \(T.\text{Rec}\) such that \((T.\text{Share}, T.\text{Rec})\) is a \(\delta\)-UISS scheme for \(n = 3\) or \(n = 4\).

**Proof.** We will show the statement for the more difficult case, \(n = 4\) - the argument is similar for \(n = 3\). With probability \(1 - n^2\epsilon\) we can assume that for all honest \(P_i\) if the adversary corrupts \(P_j\) and tampers with a share \(u_{i,j}\) or \(t_{j,i}\) the \(\text{NMSS.Rec}\) functionality will fail for this player. We will say that two players are in conflict if the lower level reconstruction between them fails, or it succeeds to a value different than the top level secret supplied to the reconstruction functionality.

If the top level reconstruction functionality fails, we can assume one player \(P_i\) has tampered with his top level share - which will cause all honest players to be in conflict with \(P_i\) with high probability. We can always assume that no two honest players are in conflict with each other - though a dishonest player who has not tampered with his share can choose to be in conflict with any arbitrary subset of honest players by modifying his lower level share.

Notice that if any one player is in conflict with the remaining three, either the remaining three players are all dishonest or the single player is. Call the player in conflict with the
other three $P_i$. In this case, it’s simple to verify that outputting the lists $L_j = \{P_i\}$ for all $j \neq i$ and $L_i = \{P_j\}$ for some $j \neq i$ satisfies the UISS requirement.

Second, consider the case where one player is in conflict with two, but no player is in conflict with three. Let $P_i$ be the player that is in conflict with two (call then $P_k$ and $P_l$) - Recall that we’re assuming that the player who tampered his share is in conflict with all honest players.

First, we address the case where $P_i$ is in conflict with two players and no other player is. In this case, the reconstruction function will set $L_j = \{P_i\}$ for all $j \neq i$ and $L_i = \{P_j\}$ for arbitrary $j \neq i$. Notice that for this conflict pattern to occur, either $P_i$ is the only dishonest player who tampered with his share, or there are three dishonest players (if the tampering player was not $P_i$ and there were at most two dishonest players, another player would be in conflict with the two honest players). In both cases, we can observe that this pattern on lists satisfies the UISS requirement.

Second, consider the case where $P_i$ is in conflict with $P_k$ and $P_l$ and the remaining player $P_j$ is also in conflict with two players. If $P_j$ is in conflict with $P_k$ and $P_l$, one can verify the lists $L_j = L_i = \{P_k\}$ and $L_k = L_l = \{P_i\}$ satisfies the requirement. Second, consider the case where $P_k$ (wlog $P_l$) is in conflict with two players. If $P_k$ is in conflict with (in addition to $P_l$), $P_i$, this implies the remaining $P_j$ is not in conflict with any player and therefore as long as $L_a = \{P_j\}$ for all $a \neq j$, the requirement will be satisfied (an honest $P_j$ would be in conflict with the tampering player). The final subcase is where $P_k$ is in conflict with (in addition to $P_i$), $P_j$. In this case it can be verified that setting $L_i = L_j = \{P_k\}$ and $L_k = L_l = \{P_i\}$ satisfies the requirement (since the list of each player is a player they are in conflict with, the requirement obviously holds).

The final case is when all players are in conflict with at most one other player. If there is one honest player, $P_i$, he will be in conflict with the tampering player, but in this subcase,
no other player will. This implies that the only way this situation can arise is when three of the four players are corrupted. In this case setting $L_i = P_j$ for any $i \neq j$ for each $i$ will satisfy the requirement as the honest player’s list is forced to contain a dishonest player since all others are dishonest.

**Theorem.** There is no reconstruction function $T.\text{Rec}(s)$ such that $(T.\text{Share}, T.\text{Rec})$ is a UISS scheme for $n \geq 5$.

**Proof.** Consider the case where $n = 5$, the statement follows for larger $n$ by extending the pattern identically to more players. We reiterate the information the non-malleable shares allow us to conclude:

- If reconstruction fails, at least one player has modified his top level share,
- All honest players will be in conflict with all players that modified their top level shares,
- No two honest players are in conflict with each other.

However, a dishonest player can make any honest player be in conflict with him by modifying his lower level share corresponding to this player. We will use the notation $P_i$ verifies the share of $P_j$ to mean that the lower level sharing between $P_i$ and $P_j$ succeeds. Consider the following pattern of verification, we will show that this can be achieved by multiple patterns of corruption such that any way the UISS reconstruction function attempts to use this pattern to locally-identify the cheating players will be inconsistent with one of the corruption patterns, implying there is no way for this UISS scheme to locally-identify the cheating players against arbitrary corruption patterns.
Consider the possible ways the above verification pattern could happen. It would certainly be possible if $P_1$ and $P_2$ were honest and only $P_4$ tampered with his top level share (recall in this case $P_5$ can cause verification with any party to succeed or fail as he desires). Therefore, $P_1$ and $P_2$ should not be on each others’ lists and $\mathcal{L}_1 = \mathcal{L}_2$. Similarly, it would be possible if $P_2$ and $P_3$ were honest and only $P_5$ tampered with his top level share and therefore $P_2$ and $P_3$ should not be on each others’ lists and $\mathcal{L}_2 = \mathcal{L}_3$. If both $P_3$ and $P_4$ were honest, the above verification patterns would again be possible if $P_1$ tampered with his top level share. Finally, if $P_4$ and $P_5$ were honest, the above pattern would be possible if $P_2$ tampered with his top level share. Therefore, an adversary who chooses one of the four above scenarios and forces the above corruption pattern will cause the UISS reconstruction function to fail with at least some constant probability (which may depend on $n$ when the argument is extended for general $n$).

### 3.7 On the Limits of Identifiability.

It is known that fully secure MPC with a dishonest majority is not possible even when all players are given access to a trusted pairwise functionality [25]. In this section, we ask if a relaxed notion of identifiability on aborts is possible where an adversary may be able to cause the functionality to abort after receiving his output, but if he does, the honest players all unanimously identify this player as a cheater and can disqualify him, assuming only a trusted pairwise primitive and a broadcast channel. Our answer is negative, that although
this model suffices for security with aborts, it is not sufficient for identifiability as well.

We begin by describing our model: The protocol is initialized with each player having a local input $x_i$ and randomness $r_i$. The protocol then continues in synchronous rounds: In each round $k$, each player $P_i$ broadcasts a single message $m^k_i$ and sends a message $m^k_{i,j}$ to the $\{i,j\}$-pairwise functionality for every $j \neq i$. Each message can depend on $x_i, r_i$ and on messages received in previous rounds.

Each pairwise functionality $f_{i,j}$ with $i < j$ then computes $(y^k_{i,j}, y^k_{j,i}) = f_{i,j}(m^k_{i,j}, m^k_{j,i})$ and sends $y^k_{i,j}$ to $P_i$ and $y^k_{j,i}$ to $P_j$. For this section, we will be proving an impossibility result for any pairwise functionality, and can therefore without loss of generality assume that pairwise secure communication channels are a part of $f_{i,j}$ and do not account for them separately. The OT-Hybrid model falls as a special case by defining $f_{i,j}$ to implement two oblivious transfer instances between $P_i$ and $P_j$, where in each instance a different player is the sender.

**Definition.** *(View)* The view of a player $P_i$ in an execution of a protocol $\mathcal{P}$ (that lasts for $q$ rounds) is defined as $(r_i, x_i, \{(m^k_i, y^k_{i,j}) : j \in [n] \setminus \{i\} : k \in [q]\})$, i.e. the player’s internal randomness and the messages it receives during the protocol. Notice that if the player is following the protocol, what messages it sent in each round can be computed from this set. The random variable $V_i(\vec{x}, \mathcal{P})$ denotes the random variable defined by the view of $P_i$ when the inputs are fixed to $\vec{x}$ during a uniform execution of $\mathcal{P}$ (we will drop $\mathcal{P}$ from this notation if clear from context). To refer to a specific view when the internal randomness of the players are fixed to $r_1, r_2, r_3$ we use the notation $V_i(\vec{x}|r_1, r_2, r_3)$.

For our negative result, we will show that this model can’t be achieved with simulation error $\epsilon$ for any $\epsilon$ that vanishes as the security parameter $\kappa$ goes to infinity. We will use the terms *almost always, almost never* or *almost identically distributed* if the implied error in the distribution vanishes with $\kappa$ (for *almost identically distributed* we mean that the statistical distance vanishes as $\kappa \to \infty$). We remind the reader at this point of the definition of a protocol being secure with unanimous abort. In our definition, we only describe the IDEAL
functionality that the protocol is simulating, for a precise definition see [31].

**Definition. (Security with Unanimous Abort)** A protocol $\mathcal{P}$ for computing a functionality $F$ is said to be secure with unanimous abort if the outputs of an adversary $A$ and honest players can be simulated by their counterparts in an ideal protocol $\text{Ideal}$ with ideal functionality $I_F$ described as follows:

1. Before the computation begins, $A$ chooses a set $C \subset \{P_1, P_2, \ldots, P_n\}$ to corrupt, and receives the input of each $P_i \in C$,
2. Each uncorrupted $P_i$ sends its input $x_i$ to $I_F$ and $A$ sends an input $x_j$ for each corrupted $P_j$ to $I_F$,
3. $I_F$ computes $F(x_1, \ldots, x_n) \rightarrow (y_1, \ldots, y_n)$,
4. $I_F$ sends $y_j$ to $A$ for each corrupted $P_j$,
5. $A$ sends a bit $b$ to $I_F$,
6. If $b = 0$, $I_F$ sends $y_i$ to $P_i$ for each honest $P_i$,
7. If $b = 1$, $I_F$ sends $\perp$ to $P_i$ for each honest $P_i$.

In line with the rest of our work, we consider the problem of identifiability on abort: While it may not be possible to prevent the adversary from withholding the outputs to the honest parties, perhaps we can force the identity of a corrupted party to be revealed if such a withholding occurs. In this section we will show that while pairwise functionalities are sufficient for unanimous abort (by realizing the OT-Hybrid model, for instance), they do not suffice for identifiability.

**Definition. (Security with Identifiable Abort)** A protocol $\mathcal{P}$ for computing a function $F$ is said to be secure with identifiable abort if the outputs of an adversary $A$ and
honest players can be simulated by their counterparts in an ideal protocol IDEAL with ideal functionality $I_F$ described as follows:

Steps 1. through 6. are identical to the **Security with Unanimous Abort** definition.

7. If $b = 1$, $A$ additionally outputs $d \in \{0, 1\}$ such that $P_d \in C$ and $I_F$ sends $(\bot, P_d)$ to $P_i$ for each honest $P_i$. Each such $P_i$ returns $(\bot, P_d)$ as their output from the protocol.

If at the end of the protocol, $P_i$ outputs $(\bot, P_d)$ we say that $P_i$ has aborted and that $P_i$ identified $P_d$ as a cheater. We now define a 3-party functionality $F$ such that no protocol $\mathcal{P}$ can securely evaluate $F$ with identifiable abort given access to any pairwise primitive $f$ when up to two players may be corrupted. Note we can assume the pairwise functionality $f$ is deterministic without loss of generality by allowing each player to supply $f_{i,j}$ a random string and $f_{i,j}$ using the XOR of these strings as its internal randomness. In this section we let $<,>$ denote the inner product over $\mathbb{F}_2$.

Inputs to $F$:

- $P_1$: $(\vec{b}_0, \vec{b}_1) \in \{0, 1\}^2 \times \{0, 1\}^2$
- $P_2$: $\vec{c} \in \{0, 1\}^2$
- $P_3$: $d \in \{0, 1\}$

Outputs:

- $P_1$ has no output.
- $P_2$: $(<\vec{b}_0, \vec{c}>, <\vec{b}_1, \vec{c}>)$
- $P_3$: $\vec{b}_d$

**Two Adversaries.** For our analysis it will be useful to define two adversaries that can attack an execution of $\mathcal{P}$. For notational simplicity we will assume $f_{i,j}$ always accepts the distinguished symbol $\bot$ as input from each player signaling that this player will not give an input to $f_{i,j}$ this round. The first adversary $A_1$ corrupts $P_1$ at the beginning of the protocol and begins executing $\mathcal{P}$ normally with $P_1$’s original input with the following exception: Whenever it is supposed to give an input $x_1$ to $f_{1,3}$ it will instead give $\bot$, and instead of using the output of $f_{1,3}$ at the end of the round, it will continue executing $\mathcal{P}$ as if it had received $f^1_{1,3}(x_1, \bot)$ as the output of $f_{1,3}$ (where $f_{i,j}^{(k)}$ denotes the output sent to player $P_k$ by the functionality). Informally, this adversary will supply only $\bot$ as its input to $f_{1,3}$ while simultaneously pretending $P_3$ is doing exactly the same on its end. We define $A_3$ similarly by
corrupting $P_3$ where $P_3$ is now supplying $\bot$ instead of $x_3$ as its input to $f_{1,3}$ and pretending to receive $f_{1,3}^3(\bot, x_3)$ as its output from $f_{1,3}$.

**The Modified Protocol.** Let $\mathcal{P}'$ be a modification of $\mathcal{P}$ as follows: Consider an arbitrary round $j$, and denote the input $P_1$ supplies to $f_{1,3}$ by $x_1$. $P_1$ will discard the output returned by $f_{1,3}$ and follow $\mathcal{P}$ as if it was returned $f_{1,3}^1(x_1, \bot)$ by $f_{1,3}$. Similarly, if $P_3$ is gave $x_3$ as its input, it will also discard the value returned by $f_{1,3}$ and continue following $\mathcal{P}$ as if it was returned $f_{1,3}^3(\bot, x_3)$. Note both protocols have an implicit statistical security parameter $\kappa$ that we will refer to in our following analysis. Since all players are following the instructions of $\mathcal{P}$ with only a few modifications while executing $\mathcal{P}'$, their instructions might tell them to identify a cheater even when there is no adversarial corruption. Recall that we define an event as happening almost never if the probability of its occurrence vanishes with $\kappa$.

**Lemma.** *(Abort Lemma)* For every input sequence $\vec{x} = (x_1, x_2, x_3)$, and for any $1 \leq i \leq 3$, player $P_i$ almost never aborts in $\mathcal{P}'$.

**Proof.** We begin with showing the statement for $P_2$. Assume that $P_2$ identifies a cheater with probability $\epsilon$ during an execution of $\mathcal{P}'$. Consider now the adversary $A_1$ attacking $\mathcal{P}$. Notice that the view of $P_2$ is distributed identically when either all players are executing $\mathcal{P}'$ or when $A_1$ is attacking $\mathcal{P}$ when the inputs of the players are unchanged. However, the view of $P_2$ is distributed identically when $A_3$ is attacking $\mathcal{P}$. If $P_2$ identified a cheater during an execution of $\mathcal{P}'$ it would have a $1/2$ probability of incorrectly identifying an honest party as a cheater when a third adversary $A^*$ that randomly either behaved as $A_1$ or $A_3$ was to attack $\mathcal{P}$. Since this behavior is inconsistent with the behavior of $P_2$ in the ideal world, the probability of this event must vanish as $\kappa$ is taken sufficiently large.

Next we will next show the statement for $P_1$. Notice we have shown that if $A_3$ attacks $\mathcal{P}$ then $P_2$ almost never aborts. Since in the ideal model honest parties abort together, $P_1$
must almost never abort as well. An symmetric argument also proves the statement for $P_3$.

**Lemma.** (Output Lemma) Let $\Omega_i(\vec{x})$, be the output returned by player $i$ in $P'$ when the inputs of the players are fixed to $\vec{x}$. For any $\vec{x}$, $\Omega_i(\vec{x}) = F_i(\vec{x})$ almost always.

**Proof.** We will first show the statement for $\Omega_2$. Recall that the output of $P_2$ in $P'$ is distributed identically with the output of $P_2$ when $A_3$ is attacking $P$. We can assume that $A_3$’s counterpart in the ideal world, $A^*_3$ almost always corrupts $P_3$, chooses its input according to some distribution $D(x_3)$ and sets $b = 0$, meaning the honest players do not abort by Lemma (Abort Lemma)\(^4\). Then, the output of $P_2$ in the ideal world execution with $A^*_3$ is almost always $F_2(x_1, x_2, D(x_3)) = F_2(\vec{x})$ (the output of $P_2$ in $F$ does not depend on $x_3$). Therefore, $P_2$ almost always outputs $F_2(\vec{x})$ when $A_3$ is attacking $P$ and the players give inputs $\vec{x}$. The statement follows since this is equivalent from the point of view of $P_2$ as when all three players follow $P'$ with the same inputs.

We will now show the statement for $\Omega_1$. Once again consider $A_3$ attacking $P$. We again have that $A^*_3$ almost always corrupts $P_3$, replaces its input with $D(x_3)$ and sets $b = 0$. In this case, the output of $P_1$ when $A_3$ is attacking $P$ on inputs $\vec{x}$ is distributed as $F_1(x_1, x_2, D(x_3)) = \Omega_1(\vec{x})$ as desired.

It remains to show the statement for $\Omega_3$. This time we will consider $A_1$ attacking $P$. Then the ideal world counterpart $A^*_1$ corrupts $P_1$, replaces the input with a distribution $D(x_1)$ and sets $b = 0$ almost always. Assume $D(x_1) = x' \neq x_1$ with probability $\epsilon$ that does not vanish with $\kappa$. Take $x_2$ such that $< x_1, x_2 > \neq < x', x_2 >$. Then, $\Omega_2(x_1, x_2, x_3) = F_2(D(x_1), x_2, x_3) \neq F_2(x_1, x_2, x_3)$ with probability that does not vanish with $\kappa$. This contradicts the previous analysis for $\Omega_2$.

In the following analysis, in referring to the view of a player, we will be referring to its view in $P'$ unless specified otherwise. We will refer to this as $V_i(\vec{x})$ or $V_i(\vec{x}|r_1, r_2, r_3)$.

\(^4\)Note we can assume that the ideal world counterpart corrupts the same player as $A_3$ since the input of $P_3$ is not revealed to any other player during the protocol - The only way that $A_3$’s knowledge of $P_3$’s input in the real world can be simulated is by having $A^*_3$ corrupt the same player.
**Lemma. (View of P_2)** Fix any \( \vec{b}_0, \vec{b}_1, \vec{b}_1', \vec{c} \leftarrow \{0, 1\}^2 \) such that \( < \vec{b}_1, \vec{c} > = < \vec{b}_1', \vec{c} > \) and let \( \vec{x} = ((\vec{b}_0, \vec{b}_1), \vec{c}, 0) \) and \( \vec{x}' = ((\vec{b}_0, \vec{b}_1'), \vec{c}, 0) \). Then, \( V_2(\vec{x}) \) and \( V_2(\vec{x}') \) are almost identically distributed.

**Proof.** If \( V_2(\vec{x}) \) is statistical distance more than \( \epsilon \) from \( V_2(\vec{x}') \) there is a distinguisher, who given the view of \( P_2 \) in \( P' \) that can distinguish between when the input of \( P_1 \) is set to \( (\vec{b}_0, \vec{b}_1) \) or \( (\vec{b}_0, \vec{b}_1') \) with advantage at least \( \epsilon/2 \). Observe that an adversary \( A'_3 \) that acts as \( A_3 \) but additionally corrupts \( P_2 \) at the beginning of the protocol will simulate an execution of \( P' \) and have access to the view of \( P_2 \). This implies that \( A'_3 \) can distinguish between when the second input of \( P_1 \) is \( \vec{b}_1 \) and \( \vec{b}_1' \) with probability at least \( 1/2 + \epsilon/2 \) when the inputs of \( P_2 \) and \( P_3 \) are fixed to 0 and \( \vec{c} \).

We now claim \( A'_3 \) can not be simulated by an adversary in the ideal model. Let the inputs to \( P_2 \) and \( P_3 \) be fixed to \( \vec{c} \) and 0 respectively while the input to \( P_1 \) is chosen uniformly at random (call the distribution on the first and second vectors of the input of \( P_1 \), \( B_0 \) and \( B_1 \)). Then, \( A'_3 \) can almost always recover the precise value for \( B_0 \) and \( < B_1, \vec{c} > \). Additionally, when \( < B_1, \vec{c} > = < \vec{b}_1, \vec{c} > \) it can distinguish whether \( B_1 = \vec{b}_1 \) or \( B_1 = \vec{b}_1' \) with probability \( \epsilon/2 \). Therefore, \( A'_3 \) can guess the exact value for both \( B_0 \) and \( B_1 \) with probability at least \( 1/2 + \epsilon/2 - \nu \) for some \( \nu \) that vanishes with \( \kappa \). Observe that any adversary in the ideal world can only guess both inputs correctly with probability at most \( 1/2 \) plus a negligible factor; therefore \( \epsilon \) must vanish with \( \kappa \).

The symmetric statement holds in the case where the input of \( P_3 \) is set to 1:

**Lemma.** Fix any \( \vec{b}_0, \vec{b}_0', \vec{b}_1, \vec{c} \leftarrow \{0, 1\}^2 \) such that \( < \vec{b}_0, \vec{c} > = < \vec{b}_0', \vec{c} > \) and let \( \vec{x} = ((\vec{b}_0, \vec{b}_1), \vec{c}, 1) \) and \( \vec{x}' = ((\vec{b}_0', \vec{b}_1), \vec{c}, 1) \). Then, \( V_2(\vec{x}) \) and \( V_2(\vec{x}') \) are almost identically distributed.

**Lemma.** Fix any \( \vec{b}_0, \vec{b}_1, \vec{c} \leftarrow \{0, 1\}^2 \) and let \( \vec{x} = ((\vec{b}_0, \vec{b}_1), \vec{c}, 0) \) and \( \vec{x}' = ((\vec{b}_0, \vec{b}_1), \vec{c}, 1) \). Then, \( V_2(\vec{x}) \) and \( V_2(\vec{x}') \) are almost identically distributed.

**Proof.** If the above indistinguishability condition did not hold, a modified \( A_1 \) that also
corrupts $P_2$ would be able to distinguish between the two inputs of $P_3$ with advantage $\epsilon$, violating security.

**Corollary.** Fix any $\vec{b}_0, \vec{b}_1, \vec{b}'_0, \vec{b}'_1, \vec{c}$ such that $< \vec{b}_0, \vec{c} > = < \vec{b}'_0, \vec{c} >$ and $< \vec{b}_1, \vec{c} > = < \vec{b}'_1, \vec{c} >$, let $\vec{x} = ((\vec{b}_0, \vec{b}_1), \vec{c}, 0)$ and $\vec{x}' = ((\vec{b}'_0, \vec{b}'_1), \vec{c}, 1)$. Then, $V_2(\vec{x})$ and $V_2(\vec{x}')$ are almost identically distributed.

**Proof.** We will show the statement for $d = 0$, the other case follows identically. Letting $\equiv$ denote two distributions being almost identically distributed, combining the two previous lemmas, we get:

$$V_2((\vec{b}_0, \vec{b}_1), \vec{c}, 0) \equiv V_2((\vec{b}_0, \vec{b}'_1), \vec{c}, 0) \equiv V_2((\vec{b}_0, \vec{b}'_1), \vec{c}, 1) \equiv V_2((\vec{b}'_0, \vec{b}_1), \vec{c}, 1) \equiv V_2((\vec{b}'_0, \vec{b}'_1), \vec{c}, 0).$$

We call the randomness sets that $P_1$, $P_2$ and $P_3$ draw from in $\mathcal{P}$ (which is the same as for $\mathcal{P}'$), $R_1, R_2$ and $R_3$ respectively. For notational simplicity we will assume that the randomness of each player is chosen uniformly from their randomness set.

**Lemma.** Take any input $\vec{x}$ and $(r_1, r_2, r_3) \leftarrow R_1 \times R_2 \times R_3$. If there exist $r'_1, r'_3 \in R_1 \times R_3$ such that $V_2(w| r_1, r_2, r_3) = V_2(w| r'_1, r_2, r'_3)$ (call this view $T$) then,

$$V_2(w| r'_1, r_2, r_3) = T \quad \text{and} \quad V_2(w| r_1, r_2, r'_3) = T.$$

**Proof.** Observe that $P_1$ and $P_3$ do not share any private communication in the execution of $\mathcal{P}'$. We will argue first that $V_2(x| r'_1, r_2, r_3) = T$, the other direction follows symmetrically. We will show the statement recursively, notice that if the statement holds at the beginning of round $i$, this implies that not only the inputs to $P_2$ (from the pairwise functionality or the broadcast channel) are the same when the randomness is set to either $r'_1, r_2, r_3$ or $r_1, r_2, r_3$ but also that the inputs to $P_3$ also remain unchanged (since the messages $P_3$ receives each round can be computed from the view of $P_2$ up to that point and its internal randomness since $P_1$ and $P_3$ do not share communication that is not also received by $P_2$). This implies that its inputs to the secure functionalities, and broadcast channel will be the same in both
instances in round $i$ since its randomness and received messages are unchanged. Similarly, the inputs to $P_1$ will be the same as if the randomness was initialized to $r'_1, r_2, r'_3$ which implies that its outputs from the pairwise functionalities and the broadcast channel, as well as its inputs to the secure functionality will be unchanged in either instance. Therefore the messages received by $P_2$ in the next round will be consistent with $T$.

**Corollary.** For any $T$ such that $T = V_2(\vec{x}|r_1, r_2, r_3)$ for some $\vec{x}, r_1, r_2, r_3$ there exist sets $R_1(\vec{x}, T)$ and $R_3(\vec{x}, T)$ such that $T = V_2(\vec{x}|z_1, r_2, z_3)$ if and only if $z_1 \in R_1(\vec{x}, T)$ and $z_3 \in R_3(\vec{x}, T)$.

**Lemma.** (View of $P_3$) Let $\vec{x}, \vec{x}'$ be two input triples that only differ in the input to $P_1$. Then, for any $T$, $R_3(\vec{x}, T) = R_3(\vec{x}', T)$.

**Proof.** By the same argument as above, if $V_2(\vec{x}|r_1, r_2, r_3) = T$ and $V_2(\vec{x}'|r'_1, r_2, r'_3) = T$ then, $V_2(\vec{x}|r_1, r_2, r'_3) = T$ which proves the statement.

**Definition (ε-Balanced Transcript)** A transcript $T$ is said to be $\epsilon$-balanced for $\vec{x}, \vec{x}'$ if:

$$(1 + \epsilon)^{-1}|R_1(\vec{x}', T)| < |R_1(\vec{x}, T)| < (1 + \epsilon)|R_1(\vec{x}', T)|.$$ 

**Lemma.** Take any constant $\epsilon \in (0, 1)$, $\vec{x} = (\vec{b}_0, \vec{b}_1, \vec{c}, \vec{d})$, $\vec{x}' = (\vec{b}_0', \vec{b}_1', \vec{c}, \vec{d})$ such that $< \vec{c}, \vec{b}_0 >= < \vec{c}, \vec{b}_0'>$ and $< \vec{c}, \vec{b}_1 >= < \vec{c}, \vec{b}_1'>$. $T = V_2(x')$ is almost always $\epsilon$-balanced for $\vec{x}, \vec{x}'$.

**Proof.** For any $T$ that is not $\epsilon$-balanced:

$$\Pr[V_2(x) = T] > (1 + \epsilon)\Pr[V_2(x') = T] \quad \text{or} \quad (1 + \epsilon)\Pr[V_2(x) = T] < \Pr[V_2(x') = T]$$

Since $\epsilon \in (0, 1)$ notice:

$$(1 + \epsilon)^{-1} = 1 - \epsilon/(1 + \epsilon) \leq 1 - .5\epsilon.$$
This implies that in either case $|\Pr[V_2(x) = T] - \Pr[V_2(x') = T]| \geq .5\epsilon \Pr[V_2(x) = T]$.

Assume that the statement does not hold. When we evaluate the statistical distance between the distribution of $V_2(x)$ and $V_2(x')$ we get (we will use $\epsilon B$ to denote the set of possible outputs of $V_2(x)$ that are $\epsilon$-balanced):

$$
\sum_T |\Pr[V_2(x) = T] - \Pr[V_2(x') = T]| \geq \sum_{T \not\in \epsilon B} |\Pr[V_2(x) = T] - \Pr[V_2(x') = T]| \\
\geq \sum_{T \not\in \epsilon B} .5\epsilon \Pr[V_2(x) = T] \\
\geq .5(1 - \epsilon/2)\epsilon > \epsilon/4.
$$

which contradicts the Lemma (View of $P_2$).

The above argument will now contradict the fact that $P_3$ is supposed to be able to distinguish between when the inputs are set to be $\vec{x}$ or $\vec{x}'$ (if $b_d \neq b_d'$). We will now define $Q_3$ to be the process that takes the view of $P_3$ and returns its final output from the protocol.

Notice that since $P_1$ and $P_3$ do not share any pairwise communication channel, we can define a related functionality $Q'_3$ that takes as input the input and initial randomness of $P_3$ and the view of $P_2$ that also returns the output of $P_3$ which will agree with $Q_3$ on any execution of $\mathcal{P}'$ (any message $P_3$ receives can be computed from the view of $P_2$ and $P_3$’s initial state).

Rewriting Lemma (Output Lemma) in terms of $Q'_3$ when the input of $P_3$ is set to 0:

**Corollary.** For any $\vec{x} = ((b_0, b_1), c, 0)$, when $(r_1, r_2, r_3) \leftarrow R_1 \times R_2 \times R_3$,

$$Q'_3(V_2(\vec{x}|r_1, r_2, r_3), r_3, 0) = b_0$$

almost always.

Let $\alpha_{\vec{x}}(b_0)$ denote the probability that $Q'_3(V_2(\vec{x}|r_1, r_2, r_3), r_3, 0) = b_0$ for arbitrary $\vec{x}$ when $r_1, r_2, r_3$ are chosen uniformly at random. Let $\rho_3$ denote the random variable defined by choosing $r_3$ from $R_3$. Let $\vec{x}' = ((\vec{b}_0, \vec{b}_1), c, 0)$ such that $<\vec{b}_0', c> = <\vec{b}_0, c>$ (in this case
note that for any \( T, R_3(\vec{x}, T) = R_3(\vec{x}', T) \) by Lemma (View of \( P_3 \)) and as such we will only refer to it as \( R_3(T) \). Let \( \epsilon T \) denote the set of all \( \epsilon \)-balanced transcripts (for \( V_2 \)) for \( \vec{x} \) and \( \vec{x}' \). Then, for any \( \epsilon \in (0, 1) \) and \( \kappa \) sufficiently large:

\[
\alpha_{\vec{x}}(\vec{b}_0) = \sum_T \Pr[V_2(\vec{x}) = T] \Pr[Q'_3(T, \rho_3) = \vec{b}_0 | V_2(\vec{x}) = T] \\
\leq \sum_{\epsilon T} \Pr[V_2(\vec{x}) = T] \Pr[Q'_3(T, \rho_3) = \vec{b}_0 | V_2(\vec{x}) = T] + \epsilon \\
= \sum_{\epsilon T} \Pr[V_2(\vec{x}) = T \land Q'_3(T, \rho_3) = \vec{b}_0] + \epsilon \\
\leq (|R_1||R_2||R_3|)^{-1} \sum_{T \in \epsilon B} |R_1(\vec{x}, T)||\{r_3 \in R_3(T) : Q'_3(T, r_3) = \vec{b}_0\}| + \epsilon \\
\leq (|R_1||R_2||R_3|)^{-1} \sum_{T \in \epsilon B} |R_1(\vec{x}'', T)||\{r_3 \in R_3(T) : Q'_3(T, r_3) = \vec{b}_0\}| + 2\epsilon \\
\leq (|R_1||R_2||R_3|)^{-1} \sum_{T \in \epsilon B} |R_1(\vec{x}'', T)|(|R_3(T)| - |\{r_3 \in R_3(T) : Q'_3(T, r_3) = \vec{b}_0\}|) + 2\epsilon \\
\leq (|R_1||R_2||R_3|)^{-1} \times \\
\left[ \sum_T |R_1(\vec{x}'', T)||R_3(T)| - \sum_{T \in \epsilon B} |R_1(\vec{x}'', T)||\{r_3 \in R_3(T) : Q'_3(T, r_3) = \vec{b}_0\}| \right] + 2\epsilon \\
\leq 1 - \alpha_{\vec{x}'}(\vec{b}_0') + 4\epsilon
\]

Which implies that either \( \alpha_{\vec{x}}(\vec{b}_0) \) or \( \alpha_{\vec{x}'}(\vec{b}_0') \) does not exceed \( 1/2 + 2\epsilon \). This contradicts Lemma (OutputLemma) as it implies that either on a uniform execution of \( P' \) with inputs fixed to \( x \), \( P_3 \) does not output \( \vec{b}_0 \) with probability greater than \( 1/2 + \epsilon \) or on a uniform execution with inputs fixed to \( x' \), \( P_3 \) does not output \( \vec{b}_0 \) with probability greater than \( 1/2 + \epsilon \).
References


[18] R. Cleve. Limits on the security of coin flips when half the processors are faulty (extended abstract). In *STOC ’86*, pages 364–369. ACM.


