TITLE

Pattern Matching Encryption, Strategic Equivalence of Range Voting and Approval Voting, and Statistical Robustness of Voting Rules

Name Emily Shen
University MIT

Key words:

VTP WORKING PAPER #109
February 2013

Pattern Matching Encryption, Strategic Equivalence of Range Voting and Approval Voting, and Statistical Robustness of Voting Rules

by

Emily Shen

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2013

Author . . . .
Department of Electrical Engineering and Computer Science
February 1, 2013

Certified by . . . .
Ronald L. Rivest
Viterbi Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by . . . .
Leslie A. Kolodziejski
Professor of Electrical Engineering
Chair, Department Committee on Graduate Theses

Pattern Matching Encryption, Strategic Equivalence of Range Voting and Approval Voting, and Statistical Robustness of Voting Rules

by

Emily Shen

Submitted to the Department of Electrical Engineering and Computer Science
on February 1, 2013, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

We present new results in the areas of cryptography and voting systems.

1. Pattern matching encryption: We present new, general definitions for queryable encryption schemes – encryption schemes that allow evaluation of private queries on encrypted data without performing full decryption. We construct an efficient queryable encryption scheme supporting pattern matching queries, based on suffix trees. Storage and communication complexity are comparable to those for (unencrypted) suffix trees. The construction is based only on symmetric-key primitives, so it is practical.

2. Strategic equivalence of range voting and approval voting: We study strategic voting in the context of range voting in a formal model. We show that under general conditions, as the number of voters becomes large, strategic range voting becomes equivalent to approval voting. We propose beta distributions as a new and interesting way to model voter's subjective information about other votes.

3. Statistical robustness of voting rules: We introduce a new notion called "statistical robustness" for voting rules: a voting rule is statistically robust if, for any profile of votes, the most likely winner of a sample of the profile is the winner of the complete profile. We show that plurality is the only interesting voting rule that is statistically robust; approval voting (perhaps surprisingly) and other common voting rules are not statistically robust.

Thesis Supervisor: Ronald L. Rivest
Title: Viterbi Professor of Electrical Engineering and Computer Science

Acknowledgments

First of all, I am deeply grateful to my advisor, Ron Rivest, for his patient, kind, and insightful support and guidance throughout my time at MIT. It has been an honor and a pleasure working with Ron.

I would also like to thank Melissa Chase, Shafi Goldwasser, and Ariel Procaccia, for kindly agreeing to serve as readers for my thesis and providing helpful feedback.

I am grateful to Melissa Chase and Josh Benaloh for being excellent mentors to me during my summers at Microsoft Research Redmond, and for their valuable contributions as collaborators on work in this thesis. I would also like to thank Ari Juels for hosting me for a summer at RSA Labs and being an inspirational mentor.

Thank you to the wonderful CSAIL admins Be Blackburn, Joanne Hanley, and Holly Jones for providing cheerful assistance (and snacks) over the years.

I am thankful to my parents, Eve and Yie-Der Shen, and my sister Helen Shen, for their constant support and encouragement. I thank Vincent Koon Kam King for always being there for me and having faith in me throughout this process. Thank you to all my friends for their support over the years.

Finally, I gratefully acknowledge financial support during my graduate studies by a Bell Labs Graduate Research Fellowship, and by the Center for Science of Information, an NSF Science and Technology Center, under grant agreement CCF-0939370.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis presents new results in two areas: cryptography and voting systems. In the area of cryptography, we construct an efficient pattern matching encryption scheme. In the area of voting systems, we study strategic voting in the context of range and approval voting. We also propose a new property for voting systems, called statistical robustness. We describe each of these three contributions in turn.

1.1 Pattern Matching Encryption

In traditional notions of encryption, a message is encrypted with a key to produce a ciphertext, and then only the owner of the corresponding secret key can decrypt the ciphertext. The decryption process is "all-or-nothing", in the sense that the secret key owner can decrypt the message, and anyone without the secret key can learn nothing about the message. However, modern applications of encryption often call for more expressive notions of encryption, where data can be encrypted in such a way that allows certain kinds of querying on the data, without decrypting the data.

In the first part of the thesis, we construct a pattern matching encryption scheme – an encryption scheme that supports pattern matching (substring search) queries. To motivate this work, we consider a scenario where a client has a large amount of sensitive genomic data. The client would like to outsource storage of the sensitive data, in encrypted form, to the cloud. Later, the client would like to make use of the data, for example by making queries to check whether certain cancer markers appear anywhere as substrings of the genomic data, without having to download and decrypt all of the data. The client would like that the server not learn too much information about the genomic data or about the search strings. In addition, the encryption and queries should be efficient. In our work, we specifically consider pattern matching queries – that is, given a pattern string p and a data string s, return all occurrences of p as a (contiguous) substring of s.

Related work. Searchable symmetric encryption (SSE) [19] encrypts data in a way that allows keyword searching, where the keywords are known ahead of time, and data is tagged with its keywords when it is encrypted. In our setting, the space of strings that can be searched for is all possible strings of length up to the length of the data; if a string were to be tagged with all of its substrings when it is encrypted, ciphertexts would grow quadratically (rather than linearly) with the size of the input.

Predicate encryption (also known as functional encryption [8]) allows the secret key owner to generate "tokens" for various predicates, so that a token for a predicate f can be evaluated on a ciphertext that is an encryption of m to determine whether $f(m)$ is satisfied. State-of-the-art predicate encryption schemes (e.g., [36, 50]) support inner-product queries; that is, f specifies a vector v, and $f(m) = 1$ if $\langle m, v \rangle = 0$. Applying an inner product predicate encryption scheme naively to construct a pattern matching encryption scheme, where the patterns can be of any length, would result in ciphertexts and query time that are $O(n^n)$, where n is the length of the string s, which is clearly impractical.

Fully homomorphic encryption (FHE), beginning with the breakthrough work of Gentry [30] and further developed in subsequent work, e.g., [11, 10, 31], allows one to evaluate any arbitrary circuit on encrypted data without being able to decrypt. FHE would solve the pattern matching encryption problem, but existing constructions are far from efficient enough to be practical.

The closest related work is that of structured encryption [15], which allows specific types of queries on encrypted data structures, by "translating" a data structure that allows efficient querying into an encrypted setting, in a way that the efficiency is comparable to that of the unencrypted setting. In constructing our pattern matching encryption scheme, we take a very similar approach.

Contribution. We formalize a paradigm called "queryable encryption", which generalizes previous definitions in the area. Queryable encryption describes encryption that allows private, efficient querying of the data. The definition is general and does not restrict the particular query functionality supported. Furthermore, it allows for some information "leakage" that the server may learn from the ciphertext and from the query protocol. The definition also allows the query protocol to be interactive. Leakage and interactivity can be helpful in achieving an efficient scheme. We define security using a simulation-based definition, which states precisely what the leakage consists of, and guarantees that nothing other than the specified leakage is learned by the server (adversary). (The idea of allowing leakage in this manner was seen previously in work on structured encryption [15] and to some extent in work on searchable encryption [19].) We give definitions of correctness and security against both honest-but-curious and malicious adversaries.

We construct a pattern matching encryption scheme that satisfies correctness and security against malicious adversaries. Encryption time is $O(\lambda n)$ and query time is $O(\lambda m + k)$, where λ is the security parameter, n is the length of the string that is encrypted, m is the length of the pattern string, and k is the number of occurrences of the pattern as a substring of the original string. The query protocol takes three rounds of communication between the client and the server. The pattern matching encryption scheme provides efficiency comparable to that of unencrypted pattern matching using suffix trees, a data structure that supports pattern matching with pre-processing time $O(n)$ and search time $O(m + k)$.

Our construction is based on suffix trees. We use standard symmetric-key primitives (symmetric-key encryption schemes, pseudorandom functions, and hash functions), as well as the following tools: (1) authenticated encryption [38, 6, 5], which guarantees that an adversary given encryptions of messages of its choice cannot generate new ciphertexts that it hasn't seen, (2) which-key-concealing encryption [1, 25], which guarantees that an adversary cannot tell whether two messages are encrypted under the same key or different keys, and (3)Rabin-Karp rolling hash [35], a hash that can be computed efficiently on a sliding window of input, reusing computation from the previous window. Since all the operations are based on symmetric-key primitives, the resulting construction is fairly practical.

1.2 Strategic Range and Approval Voting

In the next part of the thesis, we turn our attention to voting systems. A voting system, or voting rule, specifies what types of ballots or votes can be cast, and how to aggregate those ballots to determine an election outcome. We consider strategic behavior in the context of voting systems. Specifically, we consider two voting systems, range voting [53] and approval voting [13]. In range voting, each voter gives each alternative a score in some pre-specified range. In approval voting, each voter "approves" or "disapproves" of each candidate. Effectively, approval voting is a special case of range voting, where there are only two allowable scores. We are interested in how rational, strategic voters vote when using range voting and approval voting. Our main question is, when using range voting, when would a rational voter want to vote "approval-style" (i.e., giving each candidate either the minimum or the maximum allowed score)? Put another way, when is strategic range voting equivalent to approval voting?

Previous results by Smith and others (see Smith [52, 51] for a summary) showed that strategic range voting is equivalent to approval voting when the number of voters is large and under an assumption about the behavior of "near-tie" probabilities. However, they do not give a characterization of formal, general conditions under which this assumption may hold.

Contribution. We study strategic range voting in a very general model of voter's information about the other votes. We propose that a voter has independent probability density functions f_i for each alternative i, specifying the distribution of the average score given to that alternative. Given the f_i's, we give a method for a voter to determine an optimal (utility-maximizing) vote. We show that, as long as the f_i's are defined on the entire range of allowable scores, then as the number of voters grows large, a rational range voter will want to vote approval-style.

Next, we propose a concrete class of distributions that might reasonably be used to model a voter's information – beta distributions. A beta distribution models a posterior probability distribution on the average score for an alternative after seeing a given number of approvals and disapprovals from pre-election polls, given a uniform prior. We observe that an optimal strategy when using beta distributions to model voter information is not always sincere – that is, it might be in the voter's best interest to approve of alternative A_i and disapprove of alternative A_j even though she prefers A_j to A_i.

1.3 Statistical Robustness of Voting Rules

In the next part of the thesis, we introduce a property called *statistical robustness* for voting rules.

Statistical robustness asks the following question – if a random sample is drawn from the voter population in a given election, will the most likely election outcome from that sample be the same as the winner for the entire population? If so, we say that the voting system is statistically robust.

One way to think of statistical robustness is as a measure of "weather-proofness" – if the weather is bad, and some (randomly chosen) voters don't make it to the polls, is the election outcome likely to be affected? From this perspective, it is clear that statistical robustness is a desirable property for a voting system to satisfy.

Statistical robustness is relevant in the context of post-election auditing, which attempts to check that the election outcome is correct, i.e., a full hand recount would give the same outcome as the announced outcome. Some existing audit methods, known as ballot-polling auditing [43], work well for voting rules like plurality, where each ballot observed that is a vote for the reported winner can be used to increase the confidence that the reported winner is correct. However, it is not clear how such methods could be made to work for voting rules that are not statistically robust.

Contribution. We propose statistical robustness as an interesting and desirable property for a voting system. We give definitions for "full" statistical robustness, as well as parameterized definitions for various levels of statistical robustness, with respect to three different random sampling methods: binomial sampling, and sampling with and without replacement.

We show that plurality (and its complement, veto) and random ballot satisfy statistical robustness, while approval, range (score) voting, single-transferable vote (STV), plurality with runoff, Copeland, and maximin are not statistically robust with respect to one or more of the sampling methods. Furthermore, we show that any positional scoring rule whose score vector contains at least three distinct values is not statistically robust, with respect to sampling with or without replacement. It was somewhat surprising to find that approval voting is not robust, and that plurality is essentially the only interesting voting rule that is statistically robust.

1.4 Organization

This thesis is organized as follows. Chapter 2 describes the work on pattern matching encryption, which is joint work with Melissa Chase, with helpful input from Ron Rivest. Chapter 3 describes strategic range voting and approval voting, which is joint work with Josh Benaloh and Ron Rivest. Chapter 4 describes statistical robustness of voting rules, which is joint work with Ron Rivest.

Chapter 2

Pattern Matching Encryption

2.1 Introduction

In traditional symmetric-key encryption schemes, a user encrypts a message so that only the owner of the corresponding secret key can decrypt it. Decryption is "all-or-nothing"; that is, with the key one can decrypt the message completely, and without the key one learns nothing about the message. However, many settings, such as cloud storage, call for encryption schemes that support the evaluation of certain classes of queries on the data, without decrypting the data. A client may wish to store encrypted data on a cloud server, and then be able to issue queries on the data to the server in order to make use of the data without retrieving and decrypting the original ciphertext.

For example, suppose a medical research lab wants to store its subjects' genomic data using a cloud storage service. Due to the sensitive nature of genomic data, the data stored on the cloud must be encrypted. At the same time, the researchers need to be able to use and query the data efficiently, for example, by making short substring queries on the genomic data. Using a traditional encryption scheme, any global query on the data would require retrieving and decrypting the entire original data, negating many of the advantages of cloud storage. The owner of the data would like that the process of performing these queries not reveal much information to the server about the genomic data or the search strings.

Queryable encryption. In this chapter, we formalize a type of encryption that we call *queryable encryption*. A queryable encryption scheme allows for evaluation of some query functionality $\mathcal{F}$ that takes as input a message M and a query q and outputs an answer. A client encrypts a message M under a secret key and stores the ciphertext on a server. Then, using the secret key, the client can issue a query q by executing an interactive protocol with the server. At the end of this protocol, the client learns the value of $\mathcal{F}(M, q)$. For example, for pattern matching queries, a query q is a pattern string, the message M is a string, and $\mathcal{F}(M, q)$ returns the set of indices of all occurrences of q as a substring of M.

For security, we will think of the server as an adversary trying to learn information about the message and the queries. Ideally, we would like that an adversary that is given a ciphertext and that engages in query protocols for several queries learns nothing about the message or the queries. However, in order to achieve an efficient scheme, we will allow some limited information about the message and the queries to be revealed ("leaked") to the server through the ciphertext and the query protocol. We define notions of security that specify explicitly what information is leaked, and guarantee that an adversary learns nothing more than the specified leakage. The idea that it may be acceptable for a queryable encryption to leak some information to gain efficiency was seen previously in the case of structured encryption [15], and to some extent in the case of searchable encryption [19].

We define correctness and security within two adversarial models: honest-but-curious and malicious. In the honest-but-curious model, the adversary executes the protocol honestly but tries to learn information about the message and the queries along the way. In the malicious model, the adversary tries to learn information, possibly by not following the protocol honestly.

Pattern matching encryption. Next, we focus on constructing a pattern matching encryption scheme, that is, a queryable encryption scheme that support pattern matching queries – given a string s and a pattern string p, return all occurrences of p as a substring of s. In the genomic data application, for example, researchers may wish to query the database to determine whether a particular cancer marker sequence appears in any of the data.

For efficiency, our goal is for space and computation complexity to be comparable to that of evaluating pattern matching queries in the unencrypted setting. This means that general techniques such as fully homomorphic encryption [30, 11, 10, 31] and functional encryption [8, 36, 50] will not be practical. By focusing on the specific functionality of pattern matching queries, we are able to achieve a scheme with much better efficiency.

To construct a pattern matching encryption scheme, we use suffix trees, a data structure used to efficiently perform pattern matching on unencrypted data. We combine basic symmetric-key primitives to develop a method that allows traversal of select edges in a suffix tree in order to efficiently perform pattern matching on encrypted data, without revealing significant information about the string or the queries.

2.1.1 Related Work

Searchable encryption and structured encryption. We draw on related work on symmetric searchable encryption (SSE) [19] and its generalization to structured encryption [15].

These works take the approach of considering a specific type of query and identifying a data structure that allows efficient evaluation of those queries in an unencrypted setting. The construction then "translates" the data structure into an encrypted setting, so that the user can encrypt the data structure and send the server a "token" to evaluate a query on the encrypted structure. The translation is done in such a way that the efficiency is comparable to that of the unencrypted setting.

Since the server is processing the query, the server will be able to determine the memory access pattern of the queries, that is, which parts of memory have been accessed, and when the same memory block is accessed again.[1] The approach to

[1]Note that this is true even if we use fully homomorphic encryption (e.g., [30, 11, 10, 31]) or functional encryption [8, 36, 50].

security in SSE and structured encryption is to acknowledge that some information will be leaked because of the memory access pattern, but to clearly specify the leakage, and to guarantee that is the only information that the server can learn.

While the structured encryption and SSE approach is a natural one, there has been relatively little work in this area, and the kinds of data structures and queries supported are rather limited. In particular, existing results focus on encrypting index data structures to support lookup queries. In our work, we extend the approach to encrypting suffix trees to support pattern matching queries.

Predicate encryption and fully homomorphic encryption. Predicate encryption (a special case of functional encryption [8]) allows the secret key owner to generate "tokens" for various predicates, so that a token for a predicate f can be evaluated on a ciphertext that is an encryption of m to determine whether $f(m)$ is satisfied. State-of-the-art predicate encryption schemes (e.g., [36, 50]) support inner-product queries; that is, f specifies a vector v, and $f(m) = 1$ if $\langle m, v \rangle = 0$. Applying an inner product predicate encryption scheme naively to construct a pattern matching encryption scheme, where the patterns can be of any length, would result in ciphertexts and query time that are $O(n^n)$, where n is the length of the string s, which is clearly impractical.

Fully homomorphic encryption (FHE), beginning with the breakthrough work of Gentry [30] and further developed in subsequent work, e.g., [11, 10, 31], allows one to evaluate any arbitrary circuit on encrypted data without being able to decrypt. FHE would solve the pattern matching encryption problem, but existing constructions are far from being efficient enough to be practical.

Oblivious RAMs. The problem of leaking the memory access pattern is addressed in the work on Oblivious RAMs [46], which shows how to implement any query in a way that ensures that the memory access pattern is independent of the query. There has been significant process in making oblivious RAMs efficient; however, even the most efficient constructions to date (see, e.g., Stefanov et al. [54]) increase the amortized costs of processing a query by a factor of at least $\log n$, where n is the size of the stored data. In our setting, where we assume that the large size of the dataset may be one of the primary motivations for outsourcing storage, a $\log n$ overhead may be unacceptable.

Secure two-party computation of pattern matching. There have been several works on secure two-party or multiparty computation (e.g., [20, 45]) and specifically on secure pattern matching and other text processing in the two-party setting (see [44, 34, 29, 37, 27, 56]). This is an interesting line of work; however, our setting is rather different. In our setting, the client has outsourced storage of its encrypted data to a server, and then the client would like to query its data with a pattern string. The server does not have the data string in the clear; it is encrypted. Thus, even without considering the extra rounds of communication, we cannot directly apply secure two-party pattern matching protocols.

Memory delegation and integrity checking. We consider both honest-but-curious and malicious adversaries. In the case of malicious adversaries, we are concerned with detecting when the adversary misbehaves, i.e., deviates from the protocol. One way the adversary may misbehave is by returning something other than what was originally stored on the server. Along these lines, there is related work on memory delegation (e.g., [16]) and memory checking (e.g., [22]), verifiable computation (e.g., [7, 28]), integrity checking (e.g., [55]), and encrypted computation on untrusted programs (e.g., [26]); the theme of these works is retrieving and computing on data stored on an untrusted server. For our purposes, since we focus on the specific functionality of pattern matching encryption in order to achieve an efficient scheme using simple primitives, we do not need general purpose integrity checking techniques, which can be expensive or rely on more complex assumptions.

2.1.2 Our Results

We present a general definition of queryable encryption schemes. We give definitions of correctness and simulation-based definitions of security with leakage, against chosen query attacks by both honest-but-curious and malicious adversaries.

We then define pattern matching encryption as a special case of queryable encryption, and we construct a pattern matching encryption scheme and prove its correctness and security against malicious adversaries. The pattern matching encryption scheme is based on suffix trees. The encryption time and ciphertext size are $O(\lambda n)$, querying for a pattern takes time and communication complexity $O(\lambda m + k)$, where n is the length of the encrypted string, m is the length of the pattern, and k is the number of occurrences of the pattern as a substring of the encrypted string. The query protocol takes a constant number of rounds of communication. All operations are based only on symmetric-key primitives, so the resulting construction is practical.

2.2 Notation and Cryptographic Primitives

We begin with some notation and definitions of cryptographic primitives that we will use throughout the chapter.

2.2.1 Basic Notation

We write $x \overset{\text{R}}{\leftarrow} X$ to denote an element x being sampled uniformly at random from a finite set X, and $x \leftarrow A$ to denote the output x of an algorithm A. We write $x||y$ to refer to the concatenation of strings x and y, and $|x|$ to refer to the length of a string x. If $x = x_1 \ldots x_n$ is a string of n characters, and a and b are integers, $1 \leq a, b \leq n$, then $x[a..b]$ denotes the substring $x_a x_{a+1} \ldots x_b$. We sometimes use ϵ to denote the empty string. In other places ϵ will be used to denote a quantity that is negligible in the security parameter; the intended meaning of ϵ will be clear from context.

If T is a tuple of values with variable names $(a, b, \ldots)$, then $T.a, T.b, \ldots$ refer to the values in the tuple. If n is a positive integer, we use $[n]$ to denote the set $\{1, \ldots, n\}$.

If S is a set, $\mathcal{P}(S)$ is the corresponding power set, i.e., the set of all subsets of S.

We use λ to refer to the security parameter, and we assume all algorithms implicitly take λ as input. A function $\nu : \mathbb{N} \to \mathbb{N}$ is negligible in λ if for every positive polynomial $p(\cdot)$ there exists an integer $\lambda_p > 0$ such that for all $\lambda > \lambda_p$, $\nu(\lambda) < 1/p(\lambda)$. We let $\mathrm{negl}(\lambda)$ denote an unspecified negligible function in λ.

Following standard GMR notation [33], if $p(\cdot, \cdot, \ldots)$ is a predicate, the notation $\Pr[a \leftarrow A; b \leftarrow B; \ldots : p(a, b, \ldots)]$ denotes the probability that $p(a, b, \ldots)$ is true after $a \leftarrow A, b \leftarrow B, \ldots$ are executed in order. We write $\mathcal{A}^{\mathcal{O}}$ to represent that algorithm $\mathcal{A}$ can make oracle queries to algorithm $\mathcal{O}$. We will assume that adversaries are stateful algorithms; that is, an adversary $\mathcal{A}$ maintains state across multiple invocations by implicitly taking its previous state as input and outputting its updated state.

If f is a function with domain D, and $S \subseteq D$, then $f[S]$ denotes the image of S under f. If $F : \mathcal{K} \times D \to R$ is a family of functions from D to R, where $\mathcal{K}$, D, and R are finite sets, we write F_K for the function defined by $F_K(x) = F(K, x)$.

2.2.2 Pseudorandom Functions and Permutations

A pseudorandom function family (PRF) is a family F of functions such that no probabilistic polynomial-time (PPT) adversary can distinguish a function chosen randomly from F from a uniformly random function, except with negligible advantage.

Definition 2.2.1 (Pseudorandom Function Family)**.** Let D and R be finite sets, and let $F : \{0,1\}^\lambda \times D \to R$ be a family of functions. Let $\mathcal{R}$ denote the set of all possible functions $Z : D \to R$. F is a *pseudorandom function family (PRF)* if for all PPT adversaries A,

$$|\Pr[K \xleftarrow{\text{R}} \{0,1\}^\lambda : \mathcal{A}^{F_K}(1^\lambda) = 1] - \Pr[Z \xleftarrow{\text{R}} \mathcal{R} : \mathcal{A}^{Z}(1^\lambda) = 1]| \leq \mathrm{negl}(\lambda) \ .$$

Similarly, a pseudorandom permutation family (PRP) is a family of functions such that no PPT adversary can distinguish a function randomly chosen from F and a uniformly random permutation, except with negligible advantage.

Definition 2.2.2 (Pseudorandom Permutation Family)**.** Let D be a finite set, and let $F : \{0,1\}^\lambda \times D \to D$ be a family of functions. Let $\mathcal{P}$ denote the set of all possible permutations (one-to-one, onto functions) $P : D \to D$. F is a *pseudorandom permutation family (PRP)* if for all PPT adversaries A,

$$|\Pr[K \overset{\mathrm{R}}{\leftarrow} \{0,1\}^\lambda : \mathcal{A}^{F_K}(1^\lambda) = 1] - \Pr[P \overset{\mathrm{R}}{\leftarrow} \mathcal{P} : \mathcal{A}^{P}(1^\lambda) = 1]| \leq \mathrm{negl}(\lambda) \ .$$

2.2.3 ϵ-Almost-Universal Hash Functions

An ϵ-almost-universal hash function is a family $\mathcal{H}$ of hash functions such that, for any pair of distinct messages, the probability of a hash collision when the hash function is chosen randomly from $\mathcal{H}$ is at most ϵ.

Definition 2.2.3 (ϵ-Almost-Universal Hash Function)**.** Let U and B be finite sets, and let $H : \{0,1\}^\lambda \times U \to B$ be a family of hash functions. H is *ϵ-almost-universal* if for any $x, x' \in U$, $x \neq x'$,

$$\Pr[t \overset{\mathrm{R}}{\leftarrow} \{0,1\}^\lambda : H_t(x) = H_t(x')] \leq \epsilon \ .$$

Let us look at an example of a known ϵ-almost-universal hash construction, which we shall use later.

Example 2.2.4. [Polynomial hash] We view a message x as a sequence $(x_1, \ldots, x_n)$ of ℓ-bit strings. For any k in the finite field $\mathrm{GF}(2^\ell)$, the hash function $H_k(x)$ is defined as the evaluation of the polynomial p_x over $\mathrm{GF}(2^\ell)$ defined by coefficients $x_1, \ldots, x_n$, at the point k. That is, $H_k(x) = p_x(k) = \Sigma_{i=1}^{n} x_i k^{i-1}$, where all operations are in $\mathrm{GF}(2^\ell)$.

The hash function family defined above is ϵ-almost-universal, for $\epsilon = (n-1)/2^\ell$. To see this, suppose $H_k(x) = H_k(x')$ for some $x \neq x'$. Then $p_x(k) = p_{x'}(k)$. This means $p_{x-x'}(k) = \Sigma_{i=1}^{n}(x_i - x'_i)k^{i-1} = 0$, where at least one of $(x_i - x'_i)$ is not 0. Since $p_{x-x'}(\cdot)$ is a non-zero polynomial of degree at most $n-1$, it can have at most $n-1$ roots. The probability that a k chosen randomly from $\mathrm{GF}(2^\ell)$ will be one of the at most $n-1$ roots is at most $(n-1)/2^\ell$.

2.2.4 PRF Composed with Almost Universal Hashing

When computing a PRF on a long input, it can be more efficient to first hash the input down to a short string, and then apply the PRF to the hash output. If the hash function is ϵ-almost-universal for some negligible ϵ, then the resulting construction is still a PRF. This observation is due to Levin [42] and is known sometimes as Levin's trick.

The following theorem says that a PRF composed with an ϵ-almost-universal hash function, where ϵ is negligible, gives another PRF. A proof of this theorem has been given previously in [21]; we include a version of that proof here, for completeness.

Theorem 2.2.5. *Let p be some polynomial. Let $F : \{0,1\}^{\lambda} \times B \to R$ be a PRF, and let $H : \{0,1\}^{p(\lambda)} \times U \to B$ be an ϵ-almost-universal hash function for some $\epsilon = \text{negl}(\lambda)$. Then $F(H) : \{0,1\}^{\lambda+p(\lambda)} \times U \to R$, defined by $F_{K,t}(x) = F_K(H_t(x))$, is a PRF.*

Proof. Let $\mathcal{A}$ be an adversary attacking the PRF property of $F(H)$. We wish to show that $\mathcal{A}$'s advantage in distinguishing $F_K(H_t(\cdot))$ for random K, t from $Z(\cdot)$, where Z is a uniformly random function from U to R, is negl(λ). To do so, we first argue that $\mathcal{A}$'s advantage in distinguishing $F_K(H_t(\cdot))$ from $Y(H_t(\cdot))$, where Y is a uniformly random function from B to R, is negl(λ). We then argue that $\mathcal{A}$'s advantage in distinguishing $Y(H_t(\cdot))$ from $Z(\cdot)$ is negl(λ). Therefore, $\mathcal{A}$'s total advantage in distinguishing $F_K(H_t(\cdot))$ from $Z(\cdot)$ is negl(λ).

By the PRF property of F, we immediately have that $\mathcal{A}$'s advantage in distinguishing $F_K(H_t(\cdot))$ for a random K from $Y(H_t(\cdot))$ is at most negl(λ).

Next, to see that $\mathcal{A}$ cannot distinguish $Y(H_t(\cdot))$ for a random t from $Z(\cdot)$, let $x_1, \ldots, x_q$ be the queries $\mathcal{A}$ makes to its oracle. (Without loss of generality, assume $x_1, \ldots, x_q$ are distinct.) If all of the hashes $H_t(x_1), \ldots, H_t(x_q)$ are distinct, then $Y(H_t(\cdot))$ and $Z(\cdot)$ will both output q uniformly random, independent values, so $\mathcal{A}$ will not be able to distinguish the two functions.

Therefore, $\mathcal{A}$'s advantage in distinguishing $Y(H_t(\cdot))$ from $Z(\cdot)$ is at most the probability of a collision among $H_t(x_1), \ldots, H_t(x_q)$. Let X denote the event that a collision occurs among $H_t(x_1), \ldots, H_t(x_q)$. Since Y is a uniformly random function, each output of $Y(H_t(\cdot))$ is a uniformly random, independent value (independent of the input and of t), until and unless X occurs. Once X occurs, the subsequent outputs of $Y(H_t(\cdot))$ do not affect the probability of X. Therefore, to analyze the probability of X, we can think of $x_1, \ldots, x_q$ as being chosen before and independently of t.

There are at most q^2 pairs $i < j$, and by the ϵ-almost universality of H, for each pair there is at most an ϵ probability of a collision. Thus, the probability of X is at most $q^2\epsilon$, which is $\text{negl}(p(\lambda)) = \text{negl}(\lambda)$.

All together, $\mathcal{A}$'s distinguishing advantage is at most $\text{negl}(\lambda)$. □

2.2.5 Symmetric-Key Encryption

Definition 2.2.6 (Symmetric-Key Encryption)**.** A *symmetric* (or *symmetric-key*) *encryption scheme* consists of the following PPT algorithms.

Gen(1^λ)**:** The key generation algorithm takes a security parameter λ and generates a secret key K.

Enc(K, m)**:** The encryption algorithm takes a secret key K and a message m and returns a ciphertext c. Note that *Enc* will be randomized, but we omit the randomness as an explicit input.

Dec(K, c)**:** The decryption algorithm is a deterministic algorithm that takes a secret key K and a ciphertext c and returns a message m or a special symbol $\perp$.

Correctness. For correctness, we require that for all λ and for all m, letting $K \leftarrow Gen(1^\lambda)$, we have $Dec(K, Enc(K, m)) = m$.

CPA Security. We require *indistinguishability under chosen-plaintext attacks (IND-CPA)*, or *CPA security*, which is defined using the following game. First, the challenger runs $Gen(1^\lambda)$ to generate a secret key K, which is kept hidden from the adversary. Next, the adversary is allowed to make any number of queries to an encryption oracle $Enc(K, \cdot)$. The adversary then outputs two equal-length challenge messages m_0 and m_1 and receives a challenge ciphertext equal to $Enc(K, m_b)$ for a random choice of $b \in \{0, 1\}$. The adversary can make more queries to the encryption oracle. Finally, it outputs a guess b' of the bit b. The adversary wins the game if $b' = b$.

The adversary's advantage is the difference between the probability that it wins the game and 1/2 (from guessing randomly). CPA security says that no PPT adversary can win the above game with more than negligible advantage.

Definition 2.2.7 (CPA security)**.** A symmetric encryption scheme (Gen, Enc, Dec) is *CPA-secure* if for all PPT adversaries $\mathcal{A}$,

$$|\Pr[K \leftarrow Gen(1^\lambda); (m_0, m_1) \leftarrow \mathcal{A}^{Enc(K,\cdot)}(1^\lambda); b \overset{\text{R}}{\leftarrow} \{0,1\}; c \leftarrow Enc(K, m_b);$$
$$b' \leftarrow \mathcal{A}^{Enc(K,\cdot)}(c) : \ b' = b] - 1/2| \leq \text{negl}(\lambda) \ ,$$

where the two messages (m_0, m_1) output by $\mathcal{A}$ must be of equal length.

Which-Key Concealing. We will also require symmetric encryption schemes to satisfy a property called *which-key concealing*. The which-key concealing property was introduced by Abadi and Rogaway [1] and (under the name "key hiding") by Fischlin [25].

The which-key-concealing requirement says, roughly, that an adversary cannot tell whether ciphertexts are encrypted under the same key or different keys. More formally, which-key concealing is defined via a game, in which the adversary tries to distinguish between the following two experiments. In one experiment, $Gen(1^\lambda)$ is run twice, to generate two keys K and K'. The adversary is given a "left" oracle $Enc(K, \cdot)$ and a "right" oracle $Enc(K', \cdot)$, to both of which it is allowed to make any number of queries. The adversary then outputs a bit. The other experiment is the same, except that only one key K is generated, and both of the left and right oracles output $Enc(K, \cdot)$. The adversary's advantage is the difference between the probability that it outputs 1 in the two experiments. Which-key concealing says that no PPT adversary can win the above game with more than negligible advantage. Note that in order for an encryption scheme to be which-key-concealing, clearly it must be randomized.

Definition 2.2.8 (Which-Key Concealing)**.** A symmetric encryption scheme $(\mathit{Gen}, \mathit{Enc}, \mathit{Dec})$ is *which-key-concealing* if for all PPT adversaries $\mathcal{A}$,

$$|\Pr[K \leftarrow \mathit{Gen}(1^\lambda); K' \leftarrow \mathit{Gen}(1^\lambda); \mathcal{A}^{\mathit{Enc}(K,\cdot),\mathit{Enc}(K',\cdot)}(1^\lambda) = 1] - \Pr[K \leftarrow \mathit{Gen}(1^\lambda); \mathcal{A}^{\mathit{Enc}(K,\cdot),\mathit{Enc}(K,\cdot)}(1^\lambda) = 1]| \leq \mathrm{negl}(\lambda) \ .$$

Let us now see an example of a symmetric encryption scheme that is CPA-secure and which-key-concealing.

Example 2.2.9 ($\mathcal{E}_{\mathrm{xor}}$)**.** Let p be a polynomial, and let $F : \{0,1\}^\lambda \times \{0,1\}^\lambda \to \{0,1\}^{p(\lambda)}$ be a PRF. The encryption scheme $\mathcal{E}_{\mathrm{xor}}$ for message space $\mathcal{M} = \{0,1\}^{p(\lambda)}$ is defined as follows.

Gen(1^λ)**:** Let $K \xleftarrow{\mathrm{R}} \{0,1\}^\lambda$ be the secret key.

Enc(K, m)**:** Let $r \xleftarrow{\mathrm{R}} \{0,1\}^\lambda$, and output $c = (r, F_K(r) \oplus m)$.

Dec(K, c)**:** Let $c = (r, x)$. Output $m = x \oplus F_K(r)$.

Bellare et al. [4] proved that the above scheme is CPA-secure:

Theorem 2.2.10. *[4] If F is a PRF, then $\mathcal{E}_{\mathrm{xor}}$ is CPA-secure.*

We will prove that $\mathcal{E}_{\mathrm{xor}}$ is which-key-concealing.

Theorem 2.2.11. *If F is a PRF, then $\mathcal{E}_{\mathrm{xor}}$ is which-key-concealing.*

Proof. Let $\mathcal{A}$ be an adversary playing the which-key-concealing game.

We first replace $\mathcal{E}_{\mathrm{xor}}$ in the which-key-concealing game with a modified scheme $\mathcal{E}'_{\mathrm{xor}}$. $\mathcal{E}'_{\mathrm{xor}}$ is the same as $\mathcal{E}_{\mathrm{xor}}$, except that F_K is replaced with a uniformly random function $R : \{0,1\}^\lambda \to \{0,1\}^{p(\lambda)}$. By the PRF property of F, replacing $\mathcal{E}_{\mathrm{xor}}$ with $\mathcal{E}'_{\mathrm{xor}}$ can change $\mathcal{A}$'s advantage in the which-key-concealing game by at most a negligible quantity.

So, suppose $\mathcal{A}$ is playing the which-key-concealing game for $\mathcal{E}'_{\text{xor}}$. Suppose $\mathcal{A}$ makes a total of q queries, $m_1, \ldots, m_q$, to its encryption oracles, where each m_i is a query to either the left or the right oracle. Let (r_i, x_i) denote the answer to the ith query, and let $y_i = x_i \oplus m_i$.

If there are any i and j such that $r_i = r_j$ and m_i is a query to the left oracle while m_j is a query to the right oracle, then $\mathcal{A}$ will be able to distinguish whether the two oracles use the same key based on whether $y_i = y_j$. However, if all of $r_1, \ldots, r_q$ are distinct, then for any key the encryption algorithm will choose each y_i as a uniformly random, independent value, so $\mathcal{A}$ will gain no information about which experiment it is in and can do no better than a random guess.

Thus, $\mathcal{A}$'s advantage in winning the which-key-concealing game for $\mathcal{E}'_{\text{xor}}$ is at most the probability that any of $r_1, \ldots, r_q$ are equal, which is upper bounded by $q^2/2^\lambda$. Combining this with the negligible difference in $\mathcal{A}$'s advantage against $\mathcal{E}'_{\text{xor}}$ and against $\mathcal{E}_{\text{xor}}$, we have that $\mathcal{A}$'s advantage in winning the which-key-concealing game for $\mathcal{E}_{\text{xor}}$ is negligible. □

Ciphertext integrity and authenticated encryption. We will also sometimes require a symmetric encryption scheme to have a property called *ciphertext integrity*. The notions of ciphertext integrity and *authenticated encryption* (defined below) were introduced by [6, 38, 5]. Ciphertext integrity says, roughly, that an adversary given encryptions of messages of its choice cannot construct any new ciphertexts that decrypt successfully (i.e., decrypt to a value other than $\perp$).

Formally, ciphertext integrity is defined using the following game. First, the challenger runs $\mathit{Gen}(1^\lambda)$ to generate a secret key K, which is kept hidden from the adversary. The adversary then adaptively makes a polynomial number of queries, $m_1, \ldots, m_q$. To each query m_i the challenger responds by sending $c_i \leftarrow \mathit{Enc}(K, m_i)$ to the adversary. Finally, the adversary outputs a value c. The adversary wins the game if c is not among the previously received ciphertexts $\{c_1, \ldots, c_q\}$ and $\mathit{Dec}(K, c) \neq \perp$.

We define the advantage of an adversary $\mathcal{A}$ in attacking the ciphertext integrity of a symmetric encryption scheme as the probability that $\mathcal{A}$ wins the above game.

Definition 2.2.12 (Ciphertext integrity)**.** A symmetric encryption scheme (*Gen*, *Enc*, *Dec*) has ciphertext integrity if for all PPT adversaries $\mathcal{A}$, $\mathcal{A}$'s advantage in the above game is at most $\mathrm{negl}(\lambda)$.

Definition 2.2.13 (Authenticated encryption)**.** A symmetric encryption scheme is an *authenticated encryption scheme* if it has CPA security and ciphertext integrity.

Let us now see an example of an authenticated encryption scheme. One way to construct an authenticated encryption scheme is "encrypt-then-MAC" [5]. (A MAC is a message authentication code, the details of which we do not give here; instead, we will just use the fact that a PRF defines a secure MAC.) Using encrypt-then-MAC, one first encrypts a message m with a CPA-secure scheme to get a ciphertext c', and then computes a MAC of c' to get a tag t. The ciphertext is then $c = (c', t)$. The decryption algorithm verifies that t is a valid tag for c' and then decrypts c' using the CPA-secure scheme. In the following example, we apply encrypt-then-MAC to $\mathcal{E}_{\text{xor}}$ to obtain an authenticated encryption scheme $\mathcal{E}_{\text{xor-auth}}$ (which, like $\mathcal{E}_{\text{xor}}$, is also which-key-concealing).

Example 2.2.14 ($\mathcal{E}_{\text{xor-auth}}$)**.** Let p be a polynomial, and let $F : \{0,1\}^\lambda \times \{0,1\}^\lambda \to \{0,1\}^{p(\lambda)}$ and $G : \{0,1\}^\lambda \times \{0,1\}^{\lambda+p(\lambda)} \to \{0,1\}^\lambda$ be PRFs. The encryption scheme $\mathcal{E}_{\text{xor-auth}}$ for message space $\mathcal{M} = \{0,1\}^{p(\lambda)}$ is defined as follows.

Gen(1^λ)**:** Choose $K_1, K_2 \xleftarrow{\text{R}} \{0,1\}^\lambda$ and let $K = (K_1, K_2)$ be the secret key.

Enc(K, m)**:** Let $r \xleftarrow{\text{R}} \{0,1\}^\lambda$, and output $c = (r, x = F_{K_1}(r) \oplus m, t = G_{K_2}(r||x))$.

Dec(K, c)**:** Let $c = (r, x, t)$. If $t \neq G_{K_2}(r||x)$, output $\perp$. Otherwise, output $m = x \oplus F_{K_1}(r)$.

Theorem 2.2.15. $\mathcal{E}_{\text{xor-auth}}$ *is an authenticated encryption scheme.*

Proof. The theorem follows directly from the following facts: (1) G is a PRF and therefore defines a MAC, (2) $\mathcal{E}_{\text{xor}}$ is CPA-secure, and (3) applying encrypt-then-MAC to a CPA-secure scheme gives an authenticated encryption scheme [5]. □

$\mathcal{E}_{\text{xor-auth}}$ also retains the which-key-concealing property of $\mathcal{E}_{\text{xor}}$.

Theorem 2.2.16. *$\mathcal{E}_{\text{xor-auth}}$ is which-key-concealing.*

Proof. The proof is very similar to the which-key-concealing proof for $\mathcal{E}_{\text{xor}}$. We first replace $\mathcal{E}_{\text{xor-auth}}$ with a modified scheme $\mathcal{E}'_{\text{xor-auth}}$ in which F and G are replaced with random functions R_1 and R_2, respectively. By the PRF property of F and G, this changes $\mathcal{A}$'s advantage in winning the which-key-concealing game by at most a negligible quantity. Now, suppose $\mathcal{A}$ makes q encryption queries, $m_1, \ldots, m_q$. Let (r_i, x_i, t_i) denote the response to the ith query, and let $y_i = x_i \oplus m_i$. If all of $r_1, \ldots, r_q$ are distinct and all of $r_1||x_1, \ldots, r_q||x_q$ are distinct, then for any key the encryption algorithm will choose all of the y_i and t_i as uniformly random, independent values, so $\mathcal{A}$ will gain no information about which experiment it is in. But if all of the r_i are distinct, then so are all of the $r_i||x_i$. Therefore, $\mathcal{A}$'s advantage against $\mathcal{E}'_{\text{xor-auth}}$ is at most the probability that any of the r_i are equal, which is upper bounded by $q^2/2^\lambda$. All together, $\mathcal{A}$'s advantage against $\mathcal{E}_{\text{xor-auth}}$ is negligible. $\square$

2.3 Queryable Encryption

We now present the main definitions for our construction.

Definition 2.3.1. A *queryable encryption scheme* for a message space $\mathcal{M}$, a query space $\mathcal{Q}$, an answer space A, and query functionality $\mathcal{F} : \mathcal{M} \times \mathcal{Q} \to \mathsf{A}$, consists of the following probabilistic polynomial-time (PPT) algorithms.

$K \leftarrow \mathit{Gen}(1^\lambda)$: The key generation algorithm takes a security parameter 1^λ and generates a secret key K.

$CT \leftarrow \mathit{Enc}(K, M)$: The encryption algorithm takes a secret key K and a message $M \in \mathcal{M}$, and outputs a ciphertext CT.

$A \leftarrow \mathit{IssueQuery}(K, q) \leftrightarrow \mathit{AnswerQuery}(CT)$: The interactive algorithms *IssueQuery* and *AnswerQuery* compose a query protocol between a client and a server. The client takes as input the secret key K and a query q, and the server takes as input a ciphertext CT. At the end of the query protocol, the client outputs an answer $A \in \mathsf{A}$; the server has no output. A is a private output that is not seen by the server.

Correctness. For correctness we require the following property. For all $\lambda \in \mathbb{N}$, $q \in \mathcal{Q}$, $M \in \mathcal{M}$, let $K \leftarrow \mathit{Gen}(1^\lambda)$, $CT \leftarrow \mathit{Enc}(K, M)$, and $A \leftarrow \mathit{IssueQuery}(K, q) \leftrightarrow \mathit{AnswerQuery}(CT)$. Then $\Pr[A = \mathcal{F}(M, q)] = 1 - \text{negl}(\lambda)$.

This correctness property ensures correct output if all algorithms are executed honestly. (This is the usual way in which correctness is defined for similar types of schemes, such as searchable encryption, structured encryption, and functional encryption.) However, it does not say anything about the client's output if the server does not honestly execute *AnswerQuery*, for example.

Correctness against malicious adversaries. We also consider a stronger property, correctness against malicious adversaries, which says that the client's output will be correct if all algorithms are executed honestly, but the client will output $\perp$ if the server does not execute *AnswerQuery* honestly. Thus, the server may misbehave, but it cannot cause the client to unknowingly produce incorrect output.

More formally, correctness against malicious adversaries requires the following. For all PPT algorithms $\mathcal{A}$, for all $\lambda \in \mathbb{N}$, $q \in \mathcal{Q}$, $M \in \mathcal{M}$, let $K \leftarrow \mathit{Gen}(1^\lambda)$, $CT \leftarrow \mathit{Enc}(K, M)$, and $A \leftarrow \mathit{IssueQuery}(K, q) \leftrightarrow \mathcal{A}(CT)$. If $\mathcal{A}$ honestly executes *AnswerQuery*, then $\Pr[A = \mathcal{F}(M, q)] = 1 - \text{negl}(\lambda)$. If $\mathcal{A}$'s differs from *AnswerQuery* in its input-output behavior, then $\Pr[A = \perp] = 1 - \text{negl}(\lambda)$.

Discussion. Note that, although the above is called a queryable "encryption scheme", it does not include an explicit decryption algorithm, as the client might not ever intend to retrieve the entire original message. However, we could easily augment the functionality $\mathcal{F}$ with a query that returns the entire message.

Note also that typically we expect M to be quite large, while the representation of q and $F(M, q)$ are small, so we would like the query protocol to be efficient relative to the size of q and $F(M, q)$. Without such efficiency goals, designing a queryable encryption scheme would be trivial. *AnswerQuery* could return the entire ciphertext, and *IssueQuery* would then decrypt the ciphertext to get M and compute $\mathcal{F}(M, q)$ directly.

Our queryable encryption definition generalizes previous definitions of searchable encryption [19] and structured encryption [15], in the following ways.

Queryable encryption allows any general functionality $\mathcal{F}$. In contrast, the definition of searchable encryption is tied to the specific functionality of returning documents containing a requested keyword. Structured encryption is a generalization of searchable encryption, but the functionalities are restricted to return pointers to elements of an encrypted data structure. Since we allow general functionalities, our definition is similar to those of functional encryption. The main difference between queryable encryption and functional encryption is in the security requirements, which we will describe in the next section.

Also, queryable encryption allows the query protocol to be interactive. In searchable encryption, structured encryption, and functional encryption, the query protocol consists of two algorithms $TK \leftarrow \mathit{Token}(K, q)$ and $A \leftarrow \mathit{Query}(TK, CT)$. The client constructs a query token and sends it to the server, and the server uses the token and the ciphertext to compute the answer to the query, which it sends back to the client. We can think of these schemes has having a one-round interactive query protocol. Our definition allows for arbitrary interactive protocols, which may allow for better efficiency or privacy.

Because we allow the query protocol to be interactive, we do not need the server to actually learn the answer to the query. After the server's final message, the client may do some additional computation using its secret key to compute the answer. Since the server does not see the final answer, we are able to achieve stronger privacy guarantees.

2.4 Honest-but-Curious $(\mathcal{L}_1, \mathcal{L}_2)$-CQA2 Security

We first consider security in an honest-but-curious adversarial model. In this model, we assume that the server is honest (it executes the algorithms honestly), but curious (it can use any information it sees in the honest execution to learn what it can about the message and queries).

Ideally, we would like to be able to guarantee that ciphertexts and query protocols reveal nothing about the message or the queries. However, such a strict requirement will often make it very difficult to achieve an efficient scheme.

Therefore, we relax the security requirement somewhat so that some information may be revealed (leaked) to the server. The security definition will be parameterized by two "leakage" functions $\mathcal{L}_1, \mathcal{L}_2$. $\mathcal{L}_1(M)$ denotes the information about the message that is leaked by the ciphertext. For any j, $\mathcal{L}_2(M, q_1, \ldots, q_j)$ denotes the information about the message and all queries made so far that is leaked by the jth query.

We would like to ensure that the information specified by $\mathcal{L}_1$ and $\mathcal{L}_2$ is the only information that is leaked to the adversary, even if the adversary can choose the message that is encrypted and then adaptively choose the queries for which it executes a query protocol with the client. To capture this, our security definition considers a real experiment and an ideal experiment, and requires that the view of any adaptive adversary in the real experiment be simulatable given only the information specified by $\mathcal{L}_1$ and $\mathcal{L}_2$.

Following [15], we call the definition $(\mathcal{L}_1, \mathcal{L}_2)$-CQA2 security, where the name "CQA2" comes from "chosen query attack" and is somewhat analogous to CCA2 (chosen ciphertext attack) for symmetric encryption schemes, where an adversary can make decryption queries for chosen ciphertexts after receiving the challenge ciphertext.

Definition 2.4.1 (Honest-but-Curious $(\mathcal{L}_1, \mathcal{L}_2)$-CQA2 Security)**.** Let $\mathcal{E} = (\mathit{Gen}, \mathit{Enc}, \mathit{Query})$ be a queryable encryption scheme for a message space $\mathcal{M}$, a query space $\mathcal{Q}$, an answer space A, and query functionality $\mathcal{F} : \mathcal{M} \times \mathcal{Q} \rightarrow \mathsf{A}$. Let $\mathcal{L}_1$ and $\mathcal{L}_2$ be two functions.

Consider the following experiments involving an adversary $\mathcal{A}$ and a simulator $\mathcal{S}$:

Real$_{\mathcal{E},\mathcal{A}}(\lambda)$**:** The challenger begins by running $Gen(1^{\lambda})$ to generate a secret key K. The adversary $\mathcal{A}$ outputs a message M. The challenger runs $Enc(K, M)$ to generate a ciphertext CT, and sends CT to $\mathcal{A}$. The adversary adaptively chooses a polynomial number of queries, $q_1, \ldots, q_t$. For each query q_i, the challenger sends the adversary the view v_i of an honest server in the interactive protocol $IssueQuery(K, q_i) \leftrightarrow AnswerQuery(CT)$. Finally, $\mathcal{A}$ outputs a bit b, and b is output by the experiment.

Ideal$_{\mathcal{E},\mathcal{A},\mathcal{S}}(\lambda)$**:** First, $\mathcal{A}$ outputs a message M. The simulator $\mathcal{S}$ is given $\mathcal{L}_1(M)$ (not M itself), and outputs a value CT. The adversary adaptively chooses a polynomial number of queries, $q_1, \ldots, q_t$. For each query q_i, the simulator is given $\mathcal{L}_2(M, q_1, \ldots, q_i)$ (not q_i itself), and it outputs a simulated view v_i. Finally, $\mathcal{A}$ outputs a bit b, and b is output by the experiment.

We say that $\mathcal{E}$ is $(\mathcal{L}_1, \mathcal{L}_2)$-*CQA2 secure against honest-but-curious adversaries* if, for all PPT adversaries $\mathcal{A}$, there exists a simulator $\mathcal{S}$ such that

$$|\Pr[\mathbf{Real}_{\mathcal{E},\mathcal{A}}(\lambda) = 1] - \Pr[\mathbf{Ideal}_{\mathcal{E},\mathcal{A},\mathcal{S}}(\lambda) = 1]| \leq \mathrm{negl}(\lambda) \ .$$

Discussion. The above definition is based heavily on the definition for structured encryption [15] and generalizes it to interactive query protocols. It is also loosely related to simulation-based definitions for functional encryption [8], with one important difference: In our definition, we only consider a single ciphertext; security is not guaranteed to hold if multiple ciphertexts are encrypted under the same key. Note that security for only one ciphertext only makes sense in the symmetric-key setting, since in the public-key setting one can encrypt any number of messages with the public key. In our application, it will be reasonable to expect that each instantiation of the scheme will be used to encrypt only one message.

Although in some applications, it may be interesting and sufficient to model the server as an honest-but-curious adversary, often we will be interested in a stronger adversarial model. That is, we would like to ensure privacy against even a malicious adversary – one that does not execute its algorithms honestly. In the next section, we present a definition of security against malicious adversaries.

2.5 Malicious $(\mathcal{L}_1, \mathcal{L}_2)$-CQA2 Security

The definition of $(\mathcal{L}_1, \mathcal{L}_2)$-CQA2 security against malicious adversaries is similar to the one for honest-but-curious adversaries, except for the following two differences. First, for each query q_i, in the malicious game, the adversary interacts with either an honest challenger running $\mathit{IssueQuery}(K, q_i)$ in the real game, or the simulator given $\mathcal{L}_2(M, q_1, \ldots, q_i)$ in the ideal game. (In the honest-but-curious game, the adversary just received the view of an honest server in the interactive protocol $\mathit{IssueQuery}(K, q_i) \leftrightarrow \mathit{AnswerQuery}(CT)$.)

Second, at the end of the protocol for q_i, the adversary outputs the description of a function g_i of its choice. In the real game, the adversary receives $g_i(A_1, \ldots, A_i)$, where A_j is the private answer output by the client for query q_j. In the ideal game, the adversary receives $g_i(A'_1, \ldots, A'_i)$, where $A'_j = \perp$ if the client output $\perp$ in the query protocol for q_j; otherwise, $A'_j = F(M, q_j)$.

This last step of the game is necessary in the malicious game because the adversary may learn extra information based on the client's responses to the incorrectly formed messages from the adversary. The client's private output, although it is not sent to the server in the actual query protocol, can be thought of as a response to the last message sent by the adversary. We want to capture the notion that, even if the adversary were able to learn some function g_i of the client's private outputs $A_1, \ldots, A_i$, it would not learn more than $g_i(F(M, q_1), \ldots, F(M, q_i))$. However, for any $A_j = \perp$, in the evaluation of g_i, $F(M, q_j)$ is replaced by $\perp$.

Definition 2.5.1 (Malicious $(\mathcal{L}_1, \mathcal{L}_2)$-CQA2 security)**.** Let $\mathcal{E} = (\mathit{Gen}, \mathit{Enc}, \mathit{Query})$ be a queryable encryption scheme for a message space $\mathcal{M}$, a query space $\mathcal{Q}$, an answer space A, and query functionality $\mathcal{F} : \mathcal{M} \times \mathcal{Q} \to \mathsf{A}$. Let $\mathcal{L}_1$ and $\mathcal{L}_2$ be two functions.

Consider the following experiments involving an adversary $\mathcal{A}$ and a simulator $\mathcal{S}$.

Real'$_{\mathcal{E},\mathcal{A}}(\lambda)$**:** The challenger begins by running $Gen(1^\lambda)$ to generate a secret key K. The adversary $\mathcal{A}$ outputs a message M. The challenger runs $Enc(K, M)$ to generate a ciphertext CT, and sends CT to $\mathcal{A}$. The adversary adaptively makes a polynomial number of queries $q_1, \ldots, q_t$. For each query q_i, first $\mathcal{A}$ interacts with the challenger, which runs $IssueQuery(K, q_i)$. Let A_i be the challenger's private output from the protocol for q_i. Then $\mathcal{A}$ outputs a description of a function g_i, and it receives $h_i \leftarrow g_i(A_1, \ldots, A_i)$. Finally, $\mathcal{A}$ outputs a bit b.

Ideal'$_{\mathcal{E},\mathcal{A},\mathcal{S}}(\lambda)$**:** First, $\mathcal{A}$ outputs a message M. The simulator $\mathcal{S}$ is given $\mathcal{L}_1(M)$ (not M itself), and outputs a value CT. The adversary adaptively makes a polynomial number of queries $q_1, \ldots, q_t$. For each query q_i, first the simulator is given $\mathcal{L}_2(M, q_1, \ldots, q_i)$ (not q_i itself), and $\mathcal{A}$ interacts with the simulator. Then $\mathcal{A}$ outputs a description of a function g_i, and it receives $h_i \leftarrow g_i(A'_1, \ldots, A'_i)$, where $A'_i = \perp$ if the simulator output $\perp$ in the query protocol for q_i; otherwise, $A'_i = \mathcal{F}(M, q_i)$. Finally, $\mathcal{A}$ outputs a bit b.

We say that $\mathcal{E}$ is $(\mathcal{L}_1, \mathcal{L}_2)$-*CQA2 secure against malicious adversaries* if, for all PPT adversaries $\mathcal{A}$, there exists a simulator $\mathcal{S}$ such that

$$|\Pr[\textbf{Real'}_{\mathcal{E},\mathcal{A}}(\lambda) = 1] - \Pr[\textbf{Ideal'}_{\mathcal{E},\mathcal{A},\mathcal{S}}(\lambda) = 1]| \leq \text{negl}(\lambda) \ .$$

2.6 Pattern Matching Encryption

Definition 2.6.1 (Pattern matching encryption)**.** A *pattern matching encryption scheme* for an alphabet Σ is a queryable encryption scheme for:

- message space $\mathcal{M} = \Sigma^*$,
- query space $\mathcal{Q} = \Sigma^*$,
- answer space $\mathsf{A} = \mathcal{P}(\mathbb{N})$, and
- query functionality $\mathcal{F}$ where $\mathcal{F}(s, p)$ is the set of indices of all the occurrences of p as a substring of s. That is, $\mathcal{F}(s, p) = \{i | s[i..i + m - 1] = p\}$, where $m = |p|$.

2.7 Suffix Trees

Our goal is to construct a pattern matching scheme – a queryable encryption scheme that supports the functionality $\mathcal{F}$, where $\mathcal{F}(s, p)$ returns the indices of all occurrences of p as a substring of s.

We first look to pattern matching algorithms for unencrypted data. There are several known pattern matching algorithms [39, 9, 35, 2], varying in their preprocessing efficiency and query efficiency. Most of these algorithms have preprocessing time $O(m)$ and query time $O(n)$, where n is the length of the string s and m is the length of the pattern p. Pattern matching using suffix trees, however, has preprocessing time $O(n)$ and query time $O(m)$. This is ideal for our applications, where the client stores one string s encrypted on the server, and performs queries for many pattern strings p. Therefore, we will focus on pattern matching using suffix trees as the basis for our scheme.

A *suffix tree* for a string s of length n is defined as a tree such that the paths from the root to the leaves are in one-to-one correspondence with the n suffixes of s, edges spell non-empty strings, each internal node has at least two children, and any two edges edges coming out of a node start with different characters. For a suffix tree for a string s to exist, s must be prefix-free; if it is not, we can first append a special symbol \$ to make s prefix-free. Figure 2-1 shows a suffix tree for an example string, "cocoon".

Pattern matching using suffix trees uses the following important observation: a pattern p is a substring of s if and only if it is a prefix of some suffix of s. Thus, to search s for a pattern p, search for a path from the root of which p is a prefix.

For a string of length n, a suffix tree can be constructed in $O(n)$ time [57, 24]. It can be easily shown that a suffix tree has at most $2n$ nodes. However, if for each node we were to store the entire string spelled on the edge to that node, the total storage would be $O(n^2)$ in the worst case. (To see this, consider the suffix tree for the string $s_1 \ldots s_n$, where each s_i is unique. The suffix tree would contain a distinct edge for each of the n suffixes $s_1 \ldots s_n, s_2 \ldots s_n, \ldots s_n$; these suffixes have a total length

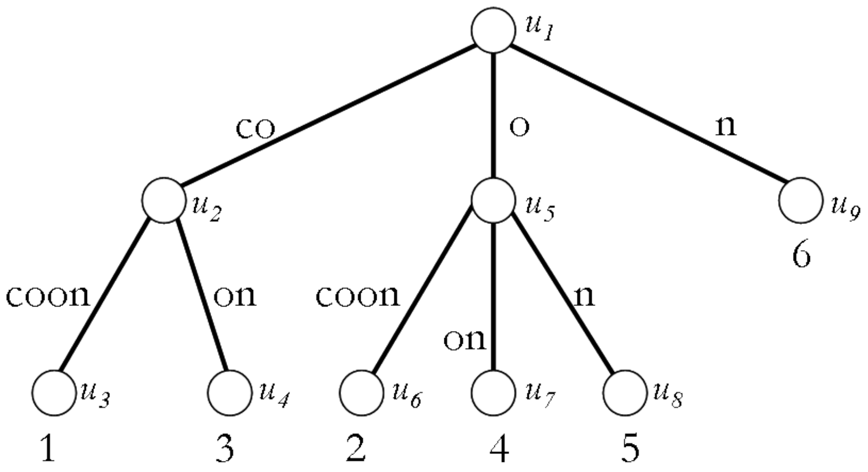

Figure 2-1: A suffix tree for the string $s =$"cocoon". We will use the node labels $u_1, \ldots, u_9$ later to explain how the pattern matching encryption scheme works.

$O(n^2)$.) To represent a suffix tree in $O(n)$ space, for each node u other than the root, one stores the start and end indices into s of the first occurrence of the substring on the edge to u. In addition, one stores the string s. Using this representation, a suffix tree takes $O(n)$ storage and can be used to search for any pattern p of length m in $O(m)$ time, and to return the indices of all occurrences of p in $O(m + k)$ time, where k is the number of occurrences.

We note a few observations about suffix trees that will be useful in our construction. For any node u, let $\rho(u)$ be the string spelled out on the path from the root to u. The string $\rho(u)$ uniquely identifies a node u in a suffix tree, i.e., no two distinct nodes u and u' have $\rho(u) = \rho(u')$. Let us also define $\hat{\rho}(u)$ to be the string spelled out on the path from the root to the parent of u, followed by the first character on the edge to u. Since no two edges coming out of a node start with the same character, $\hat{\rho}(u)$ also uniquely identifies u. Furthermore, the set of indices in s of occurrences of $\rho(u)$ is exactly the same as the set indices of occurrences of $\hat{\rho}(u)$.

For any given string s, a suffix tree is generally not unique (the children of each node may be ordered in any way). For the remainder of the chapter, we will assume that when a suffix tree is constructed, the children of every node are "ordered" lexicographically according to some canonical order of the alphabet. Thus, for a given string s, we talk about *the* unique suffix tree for s, and we can also talk about the ith child of a node (in a well-defined way), for example. In the example in Figure 2-1, the suffix tree for "cocoon" is constructed with respect to the alphabet ordering (c, o, n). In Figure 2-1, u_5 and u_6 are the first and second children, respectively, of u_2.

2.8 Notation

Before we describe our pattern matching encryption scheme, we introduce some helpful notation. Some of the notation will be relative to a string s and its suffix tree Tree_s, even though they are not explicit parameters.

- u: a node in Tree_s
- ϵ: the empty string
- $\mathrm{par}(u)$: the parent node of u. If u is the root, $\mathrm{par}(u)$ is undefined.
- $\mathrm{child}(u, i)$: the ith child node of u. If u has fewer than i children, $\mathrm{child}(u, i)$ is undefined.
- $\deg(u)$: the out-degree (number of children) of u
- $\rho(u)$: the string spelled on the path from the root to u. $\rho(u) = \epsilon$ if u is the root.
- $\hat{\rho}(u)$: For any non-root node u, $\hat{\rho}(u) = \rho(\mathrm{par}(u)) \| u_1$, where u_1 is the first character on the edge from $\mathrm{par}(u)$ to u. If u is the root, $\hat{\rho}(u) = \epsilon$.
- leaf_i: the ith leaf node in Tree_s, where the leaves are numbered 1 to n, from left to right
- len_u: the length of the string $\hat{\rho}(u)$
- ind_u: the index in s of the first occurrence of $\rho(u)$ (equivalently, of $\hat{\rho}(u)$) as a substring. That is, ind_u is the smallest i such that $\hat{\rho}(u) = s[i..i + \mathit{len}_u - 1]$. If $\rho(u) = \epsilon$, ind_u is defined to be 0.

- lpos_u: the position (between 1 and n) of the leftmost descendant of u. That is, $\mathit{leaf}_{\mathit{lpos}_u}$ is the leftmost leaf in the subtree rooted at u.

- num_u: the number of occurrences in s of $\rho(u)$ (equivalently, of $\hat{\rho}(u)$) as a substring. If $\rho(u) = \epsilon$, num_u is defined to be 0. Note that for non-root nodes u, num_u is equal to the number of leaves in the subtree rooted at u.

To illustrate the notation above, let us look at the suffix tree in Figure 2-1 for the string "cocoon". In this tree, we have $u_2 = \mathrm{par}(u_3), u_3 = \mathrm{child}(u_2, 1), \mathrm{deg}(u_2) = 2, \rho(u_3) = \text{"cocoon"}, \hat{\rho}(u_3) = \text{"coc"}, \mathit{leaf}_5 = u_8, \mathit{ind}_{u_2} = 1, \mathit{lpos}_{u_2} = 1, \mathit{num}_{u_2} = 2$.

2.9 Intuition

Before presenting the details of our construction we provide some intuition.

Our construction will make use of a dictionary, which is a data structure that stores key-value pairs (k, V), such that for any key k the corresponding value V can be retrieved efficiently (in constant time).

We will use a symmetric encryption scheme $\mathcal{E}_{\mathrm{SKE}}$, a PRF F, and a PRP P. The key generation algorithm will generate three keys K_D, K_C, K_L for $\mathcal{E}_{\mathrm{SKE}}$, and four keys K_1, K_2, K_3, K_4. (We will explain how the keys are used as we develop the intuition for the construction.)

First attempt. We first focus on constructing a queryable encryption scheme for a simpler functionality $\mathcal{F}'$, where $\mathcal{F}'(s, p)$ computes whether p occurs as a substring in s, and, if so, the index of the first occurrence in s of p. We will also only consider correctness and security against an honest-but-curious server for now.

As a first attempt, let $\mathcal{E}_{\mathrm{SKE}}$ be a CPA-secure symmetric encryption scheme, and encrypt a string $s = s_1 \dots s_n$ in the following way. First, construct the suffix tree Tree_s for s. Then construct a dictionary D, where for each node u in Tree_s, there is an entry with search key $F_{K_1}(\rho(u))$ and value $\mathcal{E}_{\mathrm{SKE}}.\mathit{Enc}(K_D, \mathit{ind}_u)$, and let the ciphertext consist of the dictionary D. Then, in the query protocol for a query p, the

node	key	value
u_1	$F_{K_1}(\epsilon)$	$\mathcal{E}_{\text{SKE}}.Enc(K_D, 0)$
u_2	$F_{K_1}(\text{“co”})$	$\mathcal{E}_{\text{SKE}}.Enc(K_D, 1)$
u_3	$F_{K_1}(\text{“cocoon”})$	$\mathcal{E}_{\text{SKE}}.Enc(K_D, 1)$
u_4	$F_{K_1}(\text{“coon”})$	$\mathcal{E}_{\text{SKE}}.Enc(K_D, 3)$
u_5	$F_{K_1}(\text{“o”})$	$\mathcal{E}_{\text{SKE}}.Enc(K_D, 2)$
u_6	$F_{K_1}(\text{“ocoon”})$	$\mathcal{E}_{\text{SKE}}.Enc(K_D, 2)$
u_7	$F_{K_1}(\text{“oon”})$	$\mathcal{E}_{\text{SKE}}.Enc(K_D, 4)$
u_8	$F_{K_1}(\text{“on”})$	$\mathcal{E}_{\text{SKE}}.Enc(K_D, 5)$
u_9	$F_{K_1}(\text{“n”})$	$\mathcal{E}_{\text{SKE}}.Enc(K_D, 6)$

Figure 2-2: The dictionary composing the ciphertext for the string “cocoon” in the “first attempt” scheme. Note that the node identifiers $u_1, \ldots, u_9$ are not a part of the dictionary; they are provided for the purpose of cross-referencing with the suffix tree in Figure 2-1.

client sends $F_{K_1}(p)$. The server then checks whether D contains an entry with search key $F_{K_1}(p)$. If so, it returns $D(F_{K_1}(p))$, which the client decrypts using K_D to get the index of the first occurrence in s of p.

For example, for our example string “cocoon”, the ciphertext in this first attempt would consist of the dictionary shown in Figure 2-2.

The obvious problem with this approach is that it only works for patterns that are substrings of s that end exactly at a node; it does not work for finding substrings of s that end partway down an edge.

Returning a possible match. To address this problem, we observe that we can uniquely identify each node u by $\hat{\rho}(u)$ instead of $\rho(u)$. Furthermore, if u is the last node (farthest from the root) for which any prefix of p equals $\hat{\rho}(u)$, then either p is not a substring of s, or p ends partway down the path to u and the indices in s of the occurrences of $\hat{\rho}(u)$ are the same as the indices in s of the occurrences of p.

In the dictionary D, we will now use $\hat{\rho}(u)$ instead of $\rho(u)$ as the search key for a node u. We will say that a prefix $p[1..i]$ is a *matching prefix* if $p[1..i] = \hat{\rho}(u)$ for some u, i.e., there is a dictionary entry corresponding to $p[1..i]$; otherwise, $p[1..i]$ is a *non-matching prefix*.

The ciphertext will also include an array C of character-wise encryptions of s, with $C[i] = \mathcal{E}_{\text{SKE}}.\mathit{Enc}(K_C, s_i)$. In the query protocol, the client will send $T_1, \ldots, T_m$, where $T_i = F_{K_1}(p[1..i])$. The server finds the entry $D(T_j)$, where $p[1..j]$ is the longest matching prefix of p. The server will return the encrypted index $\mathcal{E}_{\text{SKE}}.\mathit{Enc}(K_D, \mathit{ind})$ stored in $D(T_j)$. The client will then decrypt it to get ind, and request the server to send $C[\mathit{ind}], \ldots, C[\mathit{ind} + m - 1]$. The client can decrypt the result to check whether the decrypted string is equal to the pattern p and thus, whether p is a substring of s.

Returning all occurrences. We would like to return the indices of all occurrences of p in s, not just the first occurrence or a constant number of occurrences. However, in order to keep the ciphertext size $O(n)$, we need the storage for each node to remain a constant size. In a naive approach, in each dictionary entry we would store encryptions of indices of all of the occurrences of the corresponding string. However, this would take $O(n^2)$ storage in the worst case.

We will use the observation that the occurrences of the prefix associated with a node are exactly the occurrences of the strings associated with the leaves in the subtree rooted at that node. Each leaf corresponds to exactly one suffix. So, we construct a leaf array L of size n, with the leaves numbered 1 to n from left to right. Each element $L[i]$ stores an encryption of the index in s of the string on the path to the ith leaf. That is, $L[i] = \mathcal{E}_{\text{SKE}}.\mathit{Enc}(K_L, \mathit{ind}_{\mathit{leaf}_i})$. In the encrypted tuple in the dictionary entry for a node u we also store lpos_u, the leaf position of the leftmost leaf in the subtree rooted at u, and num_u, the number of occurrences of $\hat{\rho}(u)$. That is, the value in the dictionary entry for a node u is now $\mathcal{E}_{\text{SKE}}.\mathit{Enc}(K_D, (\mathit{ind}_u, \mathit{lpos}_u, \mathit{num}_u))$ instead of $\mathcal{E}_{\text{SKE}}.\mathit{Enc}(K_D, \mathit{ind}_u)$. The server will return $\mathcal{E}_{\text{SKE}}.\mathit{Enc}(K_D, (\mathit{ind}_u, \mathit{lpos}_u, \mathit{num}_u))$ for the last node u matched by a prefix of p. The client then decrypts to get $\mathit{ind}_u, \mathit{lpos}_u, \mathit{num}_u$, asks for $C[\mathit{ind}], \ldots, C[\mathit{ind} + m - 1]$, decrypts to determine whether p is a substring of s, and if so, asks for $L[\mathit{lpos}_u], \ldots, L[\mathit{lpos}_u + \mathit{num} - 1]$ to retrieve all occurrences of p in s.

Hiding common non-matching prefixes among queries. The scheme outlined so far works; it supports the desired pattern matching query functionality, against an honest-but-curious adversary. However, it leaks a lot of unnecessary information, so we will add a number of improvements to reduce the information that is leaked.

For any two queries p and p' whose first j prefixes are the same, the values $T_1, \ldots, T_j$ in the query protocol will be the same. Therefore, the server will learn that $p[1..j] = p[1..j']$, even though $p[1..j]$ may not be contained in s at all. Memory accesses will reveal to the server when two queries share a prefix $p[1..j]$ that is a matching prefix (i.e., contained in the dictionary), but we would like to hide when queries share non-matching prefixes.

In order to hide when queries share non-matching prefixes, we change each T_i to be an $\mathcal{E}_{\text{SKE}}$ encryption of $f_1^{(i)} = F_{K_1}(p[1..i])$ under the key $f_2^{(i)} = F_{K_2}(p[1..i])$. The dictionary entry for a node u will contain values $f_{2,i}$ for its children nodes, where $f_{2,i} = F_{K_2}(\hat{\rho}(\text{child}(u, i))$. Note that $f_{2,i}$ is the key used to encrypt T_j for any pattern p whose prefix $p[1..j]$ is equal to $\hat{\rho}(\text{child}(u, i))$. In the query protocol, the server starts at the root node, and after reaching any node, the server tries using each of the $f_{2,i}$ for that node to decrypt each of the next T_j's, until it either succeeds and reaches the next node or it reaches the end of the pattern.

Hiding node degrees, lexicographic order of children, number of nodes in suffix tree. Since the maximum degree of any node is the size d of the alphabet, we will hide the actual degree of each node by creating dummy random $f_{2,i}$ values so that there are d in total. In order to hide the lexicographic order of the children and hide which of the $f_{2,i}$ are dummy values, we store the $f_{2,i}$ in a random permuted order in the dictionary entry.

Similarly, since a suffix tree for a string of length n contains at most $2n$ nodes, we will hide the exact number N of nodes in the suffix tree by constructing $2n - N$ dummy entries in D. For each dummy entry, the search key is a random value f_1, and the value is $(f_{2,1}, \ldots, f_{2,d}, W)$, where $f_{2,1}, \ldots, f_{2,d}$ are random and W is an encryption of 0.

Hiding string indices and leaf positions. In order to hide the actual values of the string indices $\mathit{ind}, \ldots, \mathit{ind} + m - 1$ and the leaf positions $\mathit{lpos}, \ldots, \mathit{lpos} + \mathit{num} - 1$, we make use of a pseudorandom permutation family P of permutations $[n] \to [n]$. Instead of sending $(\mathit{ind}, \ldots, \mathit{ind} + m - 1)$, the client applies the permutation P_{K_3} to $\mathit{ind}, \ldots, \mathit{ind} + m - 1$ and outputs the resulting values in a randomly permuted order as $(x_1, \ldots, x_m)$. Similarly, instead of sending $(\mathit{lpos}, \ldots, \mathit{lpos} + \mathit{num} - 1)$, the client applies the permutation P_{K_4} to $\mathit{lpos}, \ldots, \mathit{lpos} + \mathit{num} - 1$ and outputs the resulting values in a randomly permuted order as $(y_1, \ldots, y_{\mathit{num}})$. Note that while the server does not learn the actual indices or leaf positions, it still learns when two queries ask for the same or overlapping indices or leaf positions.

Handling malicious adversaries. The scheme described so far satisfies correctness against an honest-but-curious adversary, but not a malicious adversary, since the client does not perform any checks to ensure that the server is sending correct messages. The scheme also would not satisfy security against a malicious adversary for reasonable leakage functions, since an adversary could potentially gain information by sending malformed or incorrect ciphertexts during the query protocol.

To handle a malicious adversary, we will require $\mathcal{E}_{\text{SKE}}$ to be an authenticated encryption scheme. Thus, an adversary will not be able to obtain the decryption of any ciphertext that is not part of the dictionary D or the arrays C or L. Also, we will add auxiliary information to the messages encrypted, to allow the client to check that any ciphertext returned by the adversary is the one expected by the honest algorithm. The client will be able to detect whether the server returned a ciphertext that is in D but not the correct one, for example.

Specifically, we will encrypt (s_i, i) instead of just s_i, so that the client can check that it is receiving the correct piece of the ciphertext. Similarly, we will encrypt $(\mathit{ind}_{\mathit{leaf}_i}, i)$ instead of just $\mathit{ind}_{\mathit{leaf}_i}$. For the dictionary entries, in addition to ind_u,num_u, and lpos_u, we will include $\mathit{len}_u, f_1(u), f_{2,1}(u), \ldots, f_{2,d}(u)$ in the tuple that is encrypted. The client can then check whether the W sent by the adversary corresponds to the longest matching prefix of p, by verifying that $F_{K_1}(p[1..\mathit{len}]) = f_1$, and that none of the $f_{2,1}, \ldots, f_{2,d}$ successfully decrypts any of the T_j for $j > len$.

2.10 Pattern Matching Scheme

Let $F : \{0,1\}^\lambda \times \{0,1\}^* \rightarrow \{0,1\}^\lambda$ be a PRF, and let $P : \{0,1\}^\lambda \times [n] \rightarrow [n]$ be a PRP. Let $\mathcal{E}_{\text{SKE}} = (Gen, Enc, Dec)$ be an authenticated, which-key-concealing symmetric encryption scheme. Our pattern matching encryption scheme $\mathcal{E}_{\text{PM}}$ for an alphabet Σ with $|\Sigma| = d$ is as follows.

***Gen*(1^λ):** Choose random strings $K_D, K_C, K_L, K_1, K_2, K_3, K_4 \overset{\text{R}}{\leftarrow} \{0,1\}^\lambda$.[2] The secret key is

$$K = (K_D, K_C, K_L, K_1, K_2, K_3, K_4).$$

***Enc*(K, s):** Let $s = s_1 \ldots s_n \in \Sigma^n$. Construct the suffix tree $Tree_s$ for s.

1. Construct a dictionary D as follows.

   For any node u, define $f_1(u) := F_{K_1}(\hat{\rho}(u))$ and $f_2(u) := F_{K_2}(\hat{\rho}(u))$.

   For each node u in $Tree_s$ (including the root and leaves), proceed as follows:

   - Choose a random permutation $\pi_u : [d] \rightarrow [d]$.
   - For $i = 1, \ldots, d$, let $f_{2,i}(u) = f_2(\text{child}(u, \pi_u(i))$ if $1 \leq \pi_u(i) \leq \deg(u)$; otherwise let $f_{2,i}(u) \overset{\text{R}}{\leftarrow} \{0,1\}^\lambda$.
   - Let $X_u = (ind_u, lpos_u, num_u, len_u, f_1(u), f_{2,1}(u), \ldots, f_{2,d}(u))$, and let $W_u = \mathcal{E}_{\text{SKE}}.Enc(K_D, X_u)$.
   - Store $V_u = (f_{2,1}(u), \ldots, f_{2,d}(u), W_u)$ with search key $\kappa_u = f_1(u)$ in D.

   Let N denote the number of nodes in $Tree_s$. Construct $2n - N$ dummy entries in D as follows. For each dummy entry, choose random strings

[2]We will assume for simplicity that $\mathcal{E}_{\text{SKE}}.Gen$ simply chooses a random key $k \overset{\text{R}}{\leftarrow} \{0,1\}^\lambda$, so throughout the construction we will use random values as $\mathcal{E}_{\text{SKE}}$ keys. To allow for general $\mathcal{E}_{\text{SKE}}.Gen$ algorithms, instead of using a random value r directly as a key, we could use a key generated by $\mathcal{E}_{\text{SKE}}.Gen$ with r providing $\mathcal{E}_{\text{SKE}}.Gen$'s random coins.

$f_1, f_{2,1}, \ldots, f_{2,d} \overset{\text{R}}{\leftarrow} \{0,1\}^\lambda$, and store $(f_{2,1}, \ldots, f_{2,d}, \mathcal{E}_{\text{SKE}}.Enc(K_D, 0))$ with search key f_1 in D.

2. Construct an array C as follows.

   For $i = 1, \ldots, n$, compute $c_i = \mathcal{E}_{\text{SKE}}.Enc(K_C, (s_i, i))$ and set $C[P_{K_3}(i)] = c_i$.

3. Construct an array L as follows.

   For $i = 1, \ldots, n$, compute $\ell_i = \mathcal{E}_{\text{SKE}}.Enc(K_L, (ind_{leaf_i}, i))$ and set $L[P_{K_4}(i)] = \ell_i$.

Output the ciphertext $CT = (D, C, L)$.

IssueQuery$(K, p) \leftrightarrow$ ***AnswerQuery***(CT)**:** The interactive query protocol, between a client with K and p and a server with CT, runs as follows.

Let $p = p_1 \ldots p_m \in \Sigma^m$, and let $CT = (D, C, L)$.

1. The client computes, for $i = 1, \ldots, m$,

$$f_1^{(i)} = F_{K_1}(p_1 \ldots p_i),\ f_2^{(i)} = F_{K_2}(p_1 \ldots p_i)\ ,$$

   and sets $T_i = \mathcal{E}_{\text{SKE}}.Enc(f_2^{(i)}, f_1^{(i)})$.

   The client sends the server $(T_1, \ldots, T_m)$.

2. The server proceeds as follows, maintaining variables $f_1, f_{2,1}, \ldots, f_{2,d}, W$.

   Initialize $(f_{2,1}, \ldots, f_{2,d}, W)$ to equal $D(F_{K_1}(\epsilon))$, where ϵ denotes the empty string.

```
For $i = 1, \ldots, m$ :
  For $j = 1, \ldots, d$:
    Let $f_1 \leftarrow \mathcal{E}_{\text{SKE}}.Dec(f_{2,j}, T_i)$. If $f_1 \neq \perp$, update $(f_{2,1}, \ldots, f_{2,d}, W)$ to
equal $D(f_1)$, and break (proceed to the next value of $i$). Otherwise, do
nothing.
```

   At the end, the server sends W to the client.

3. The client runs $X \leftarrow \mathcal{E}_{\text{SKE}}.Dec(K_D, W)$. If $X = \perp$, output $\perp$ and end the protocol. Otherwise, parse X as $(ind, lpos, num, len, f_1, f_{2,1}, \ldots, f_{2,d})$. Check whether $F_{K_1}(p[1..len]) = f_1$. If not, output $\perp$ and end the protocol. Otherwise, check whether $\mathcal{E}_{\text{SKE}}.Dec(f_{2,i}, T_j) = \perp$ for any $j \in \{len+1, \ldots, m\}$ and $i \in \{1, \ldots, d\}$. If so, output $\perp$ and end the protocol. If $ind = 0$, output $\emptyset$. Otherwise, choose a random permutation $\pi_1 : [m] \to [m]$. For $i = 1, \ldots, m$, let $x_{\pi_1(i)} = P_{K_3}(ind + i - 1)$. The client sends $(x_1, \ldots, x_m)$ to the server.

4. The server sets $C_i = C[x_i]$ for $i = 1, \ldots, m$ and sends $(C_1, \ldots, C_m)$ to the client.

5. For $i = 1, \ldots, m$, the client runs $Y \leftarrow \mathcal{E}_{\text{SKE}}.Dec(K_C, C_{\pi_1(i)})$. If $Y = \perp$, output $\perp$ and end the protocol. Otherwise, let the result be (p'_i, j). If $j \neq ind + i - 1$, output $\perp$. Otherwise, if $p'_1 \ldots p'_m \neq p$, then the client outputs $\emptyset$ as its answer and ends the protocol. Otherwise, the client chooses a random permutation $\pi_2 : [num] \to [num]$. For $i = 1, \ldots, num$, let $y_{\pi_2(i)} = P_{K_4}(lpos + i - 1)$. The client sends $(y_1, \ldots, y_{num})$ to the server.

6. The server sets $L_i = L[y_i]$ for $i = 1, \ldots, num$, and sends $(L_1, \ldots, L_{num})$ to the client.

7. For $i = 1, \ldots, num$, the client runs $\mathcal{E}_{\text{SKE}}.Dec(K_L, L_{\pi_2(i)})$. If the result is $\perp$, the client outputs $\perp$ as its answer. Otherwise, let the result be (a_i, j). If $j \neq lpos + i - 1$, output $\perp$. Otherwise, output the answer $A = \{a_1, \ldots, a_{num}\}$.

2.11 Efficiency

In analyzing the efficiency of our construction, we will assume data is stored in computer words that hold $\log n$ bits; therefore, we will treat values of size $O(\log n)$ as constant size.

We assume encryption and decryption using $\mathcal{E}_{\text{SKE}}$ take $O(\lambda)$ time. Also we assume the dictionary is implemented in such a way that dictionary lookups take constant time (using hash tables, for example).

Efficient batch implementation of PRFs. Assuming the evaluation of a PRF takes time linear in the length of its input, in a naive implementation of our scheme, computing the PRFs $f_1(u)$ and $f_2(u)$ for all nodes u would take $O(n^2)$. This is because even though there are only at most $2n$ nodes, the sum of the lengths of the strings $\hat{\rho}(u)$ associated with the nodes u can be $O(n^2)$. Similarly, computing the PRFs used for $T_1, \ldots, T_m$ would take $O(m^2)$ time.

It turns out that we can take advantage of the way the strings we are applying the PRFs to are related, to speed up the batch implementation of the PRFs for all of the nodes of the tree. We will use two tools: the polynomial hash (defined in Example 2.2.4) and suffix links (described below).

To compute the PRF of a string x, we will first hash x to λ bits using the polynomial hash, and then apply the PRF (which takes time $O(\lambda)$ on the hashed input). In order to efficiently compute the hashes of all of the strings $\hat{\rho}(u)$, we use a trick that is used in the Rabin-Karp rolling hash (see Cormen et al. [18],e.g.). (A rolling hash is a hash function that can be computed efficiently on a sliding window of input; the hash of each window reuses computation from the previous window.) The Rabin-Karp hash is the polynomial hash, with each character of the string viewed as a coefficient of the polynomial applied to the random key of the hash. The key observation is that the polynomial hash H allows for constant-time computation of $H_k(x_1 \ldots x_n)$ from $H_k(x_2 \ldots x_n)$, and also of $H_k(x_1 \ldots x_n)$ from $H_k(x_1 \ldots x_{n-1})$. To see this, notice that $H_k(x_1 \ldots, x_n) = x_1 + k \cdot H_k(x_2 \ldots x_n)$, and $H_k(x_1 \ldots x_n) = H_k(x_1 \ldots x_{n-1}) + x_n k^{n-1}$.

Using this trick, for any string x of length ℓ, we can compute the hashes $H_k(x[1..i])$ for all $i = 1, \ldots, m$ in total time $O(\lambda m)$. Thus, the $T_1, \ldots, T_m$ can be computed in time $O(\lambda m)$.

In order to compute the hashes of $\hat{\rho}(u)$ for all nodes u in time $O(n)$, we need one more trick. Many efficient suffix tree construction algorithms include *suffix links*: Each non-leaf node u with associated string $\rho(u) = a||B$, where a is a single character, has a pointer called a suffix link pointing to the node u' whose associated string $\rho(u')$ is B.

It turns out that connecting the nodes in a suffix tree using the suffix links forms another tree, in which the parent of a node u is the node u' to which u's suffix link points. To see this, notice that each internal node has an outgoing suffix link, and each node's suffix link points to a node with a shorter associated string of one fewer character, so there can be no cycles.

Since $\hat{\rho}(u) = \rho(\text{par}(u))||u_1$, we can first compute the hashes of $\rho(u)$ for all non-leaf nodes u, and then compute $\hat{\rho}(u)$ for all nodes u in constant time from $\rho(\text{par}(u))$. To compute $\rho(u)$ for all nodes u, we traverse the tree formed by the suffix links, starting at the root, and compute the hash of $\rho(u)$ for each u using $\rho(u')$, where u' is u's parent in the suffix link tree. Each of these computations takes constant time, since $\rho(u)$ is the same as $\rho(u')$ but with one character appended to the front. Therefore, computing the hashes of $\rho(u)$ for all non-leaf nodes u (and thus, computing the hashes of $\hat{\rho}(u)$ for all nodes u) takes total time $O(n)$.

Encryption efficiency. Using the efficient batch implementation of PRFs suggested above, the PRFs $f_1(u)$ and $f_2(u)$ can be computed for all nodes u in the tree in total time $O(\lambda n)$. Therefore, the dictionary D of $2n$ entries can be computed in total time $O(\lambda n)$. The arrays C and L each have n elements and can be computed in time $O(\lambda n)$. Therefore, encryption takes time $O(\lambda n)$ and the ciphertext is of size $O(\lambda n)$.

Query protocol efficiency. In the query protocol, the client first computes $T_1, \ldots, T_m$. Using the efficient batch PRF implementation suggested above, computing the $f_1^{(i)}$ and $f_2^{(i)}$ for $i = 1, \ldots, m$ takes total time $O(m)$, and computing each $\mathcal{E}_{\text{SKE}}.Enc(f_2^{(i)}, f_1^{(i)})$ takes $O(\lambda)$ time, so the total time to compute $T_1, \ldots, T_m$ is $O(\lambda m)$.

To find W, the server performs at most md decryptions and dictionary lookups, which takes total time $O(\lambda m)$. The client then computes $x_1, \ldots, x_m$ and the server retrieves $C[x_1], \ldots, C[x_m]$, in time $O(m)$. If the answer is not $\emptyset$, the client then computes $y_1, \ldots, y_{num}$ and the server retrieves $L[y_1], \ldots, L[y_{num}]$ in time $O(num)$, in time $O(num)$. Thus, both the client and the server take computation time $O(\lambda m + num)$ in the query protocol. (Since we are computing an upper bound on the query computation time, we can ignore the possibility that the server cheats and the client aborts the protocol by outputting $\perp$.) The query protocol takes three rounds of communication, and the total size of the messages exchanged is $O(\lambda m + num)$.

2.12 Correctness Against Malicious Adversaries

We will show that $\mathcal{E}_{\mathrm{PM}}$ is correct against malicious adversaries.

Theorem 2.12.1. *If $\mathcal{E}_{\mathrm{SKE}}$ is an authenticated encryption scheme, then $\mathcal{E}_{\mathrm{PM}}$ is correct against malicious adversaries.*

Proof. It is fairly straightforward to see that if the adversary executes *AnswerQuery* honestly, then the client's output will be correct.

We will argue that for each of the places where $\mathcal{A}$ could output an incorrect value, the client will detect $\mathcal{A}$'s cheating and output $\perp$, with all but negligible probability.

Lemma 2.12.2. *If $\mathcal{E}_{\mathrm{SKE}}$ is an authenticated encryption scheme, then if an adversary $\mathcal{A}$ outputs an incorrect W in the query protocol, the client's response to W will be $\perp$, with all but negligible probability.*

Proof. In the protocol for a query p, the client runs $\mathcal{E}_{\mathrm{SKE}}.Dec(K_D, W)$ to get either $\perp$ or a tuple X, which it parses as $(ind, lpos, num, len, f_1, f_{2,1}, \ldots, f_{2,d})$. The client outputs $\perp$ if any of the following events occur:

- (Event W.1) $\mathcal{E}_{\mathrm{SKE}}.Dec(K_D, W) = \perp$, or
- (Event W.2) W decrypts successfully, but $f_1 \neq F_{K_1}(p[1..len])$, or
- (Event W.3) W decrypts successfully and $f_1 = F_{K_1}(p[1..len])$, but $\mathcal{E}_{\mathrm{SKE}}.Dec(f_{2,i}, T_j) \neq \perp$ for some $i \in \{1, \ldots, d\}$, $j > len$.

On the other hand, if the adversary cheats, then W is not the ciphertext in the dictionary entry $D(F_{K_1}(p[1..i]))$, where $p[1..i]$ is the longest matching prefix of p, which means one of the following events:

- (Event $W.1'$) W is not a ciphertext in D,
- (Event $W.2'$) W is a ciphertext in D but not for any prefix of p. That is, $W = D(\kappa)$ where κ is not equal to $F_{K_1}(p[1..i])$ for any i.
- (Event $W.3'$) W is a ciphertext in D for a prefix of p, but there is a longer matching prefix of p. That is, $W = D(F_{K_1}(p[1..i]))$ for some i, but there exists a $j > i$ such that there is an entry $D(F_{K_1}(p[1..j]))$.

We want to show that if the adversary cheats, then the client will output $\perp$.

If event $W.1'$ occurs, then we will show below that $W.1$ occurs with all but negligible probability, by the ciphertext integrity of $\mathcal{E}_{\text{SKE}}$.

If event $W.2'$ occurs, then event $W.2$ occurs with all but negligible probability upper bounded by $1/2^\lambda$, the probability that $F_{K_1}(p[1..len]) = f_1$ when f_1 is an independent, random value.

If event $W.3'$ occurs, then clearly $W.3$ also occurs.

It remains to show that event $W.1'$ implies event $W.1$ with all but negligible probability.

Suppose an adversary $\mathcal{A}$ causes event $W.1'$ but not event $W.1$. Then the W output by $\mathcal{A}$ is not among the ciphertexts in the dictionary, but $\mathcal{E}_{\text{SKE}}.Dec(K_D, W) \neq \perp$. Then we can use $\mathcal{A}$ to construct an algorithm $\mathcal{B}$ that breaks ciphertext integrity of $\mathcal{E}_{\text{SKE}}$. Algorithm $\mathcal{B}$ executes $\mathcal{E}_{\text{PM}}$ honestly, except that in the encryption algorithm, instead of generating each W_u as $\mathcal{E}_{\text{SKE}}.Enc(K_D, X_u)$, it queries its encryption oracle on X_u and uses the resulting ciphertext as c_j. Then, when $\mathcal{A}$ outputs W, $\mathcal{B}$ outputs W in the ciphertext integrity game. Note that $\mathcal{A}$'s view is the same as when it is interacting with the real scheme $\mathcal{E}_{\text{PM}}$. If W is not among the ciphertexts in D, but $\mathcal{E}_{\text{SKE}}.Dec(K_D, W) \neq \perp$, then $\mathcal{B}$ wins the ciphertext integrity game. Therefore, if $\mathcal{A}$ has probability ϵ of causing event $W.1'$ but not event $W.1$, $\mathcal{B}$ wins the ciphertext integrity game with the same probability ϵ. □

Lemma 2.12.3. *If $\mathcal{E}_{\text{SKE}}$ is an authenticated encryption scheme, then if an adversary $\mathcal{A}$ outputs incorrect $C_1, \ldots, C_m$ in the query protocol, the client's response to $C_1, \ldots, C_m$ will be $\perp$, with all but negligible probability.*

Proof. In the query protocol, for each i, the client outputs $\perp$ if either of the following events occur:

- (Event $C.1$) $\mathcal{E}_{\text{SKE}}.Dec(K_C, C_i) = \perp$, or
- (Event $C.2$) $\mathcal{E}_{\text{SKE}}.Dec(K_C, C_i) = (p_i', j)$ where j is not the correct index.

On the other hand, if the adversary cheats and outputs incorrect $C_1, \ldots, C_m$, then for some i, $C_i \neq C[x_i]$, which means either of the following events:

- (Event $C.1'$) C_i is not among $C[1], \ldots, C[n]$, or
- (Event $C.2'$) $C_i = C[k]$ where $k \neq x_i$.

We want to show that if the adversary cheats, then the client will output $\perp$.

For any i, if event $C.1'$ occurs, then we will show below that event $C.1$ occurs with all but negligible probability, by the ciphertext integrity of $\mathcal{E}_{\text{SKE}}$.

If event $C.2'$ occurs, then event $C.2$ occurs, since if $C_i = C[k]$ for some $k \neq x_i$, C_i will decrypt to (s_j, j) for an incorrect index j.

It remains to show that for any i event $C.1'$ implies event $C.1$, with all but negligible probability. Suppose an adversary $\mathcal{A}$ causes event $C.1'$ but not event $C.1$. Then C_i is not among $C[1], \ldots, C[n]$, but $\mathcal{E}_{\text{SKE}}.Dec(K_C, C_i) \neq \perp$. Then we can use $\mathcal{A}$ to construct an algorithm $\mathcal{B}$ that breaks ciphertext integrity of $\mathcal{E}_{\text{SKE}}$. $\mathcal{B}$ executes $\mathcal{E}_{\text{PM}}$ honestly, except that in the encryption algorithm, instead of generating each c_j as $\mathcal{E}_{\text{SKE}}.Enc(K_C, (s_j, j))$, it queries its encryption oracle on (s_j, j) and uses the resulting ciphertext as c_j. Then, when $\mathcal{A}$ outputs $C_1, \ldots, C_m$, $\mathcal{B}$ chooses a random $i' \xleftarrow{\text{R}} \{1, \ldots, m\}$ and outputs $C_{i'}$ in the ciphertext integrity game. Note that $\mathcal{A}$'s view is the same as when it is interacting with the real scheme $\mathcal{E}_{\text{PM}}$. If C_i is not among $C[1], \ldots, C[n]$, but $\mathcal{E}_{\text{SKE}}.Dec(K_C, C_i) \neq \perp$, then $\mathcal{B}$ wins the ciphertext integrity game if $i' = i$. Therefore, if $\mathcal{A}$ has probability ϵ of causing event $C.1'$ but not event $C.1$ for any i, $\mathcal{B}$ wins the ciphertext integrity game with probability at least ϵ/m. $\square$

Lemma 2.12.4. *If $\mathcal{E}_{\mathrm{SKE}}$ is an authenticated encryption scheme, then if an adversary $\mathcal{A}$ outputs incorrect $L_1, \dots, L_{num}$ in the query protocol, the client's response to $L_1, \dots, L_{num}$ will be $\perp$, with all but negligible probability.*

The proof is omitted, since it almost identical to the proof of Lemma 2.12.3.

We have shown that if an adversary $\mathcal{A}$ cheats when producing any of its outputs to the client, the client will output $\perp$ with all but negligible probability. Therefore, $\mathcal{E}_{\mathrm{PM}}$ is correct against malicious adversaries. □

2.13 Security

We now prove that our pattern matching encryption scheme satisfies malicious-$(\mathcal{L}_1, \mathcal{L}_2)$-CQA2 security for certain leakage functions $\mathcal{L}_1$ and $\mathcal{L}_2$.

2.13.1 Leakage

Before we describe the leakage of our scheme, we define some relevant notions.

We say that a query p *visits* a node u in the suffix tree Tree_s for s if $\hat{\rho}(u)$ is a prefix of p. For any j let p_j denote the jth query, and let $m_j = |p_j|$. Let n_j denote the number of nodes visited by the query for p_j in s, let $u_{j,i}$ denote the ith such node, and let $\mathit{len}_{j,i} = |\hat{\rho}(u_{j,i})|$. Let num_j denote the number of occurrences of p_j as a substring of s. Let ind_j denote the index ind in the ciphertext W returned by *AnswerQuery* for p_j. Note that ind_j is the index in s of the longest matching prefix of p_j, which is also the index in s of the longest prefix of p_j that is a substring of s. Let lpos_j denote the leaf index lpos in the ciphertext W returned by *AnswerQuery* for p_j. If p_j is a substring of s, lpos_j is equal to the position (between 1 and n, from left to right) of the leftmost leaf ℓ for which p_j is a prefix of $\hat{\rho}(\ell)$.

The query prefix pattern for a query p_j tells which of the previous queries $p_1, \dots, p_{j-1}$ visited each of the nodes visited by p_j.

Definition 2.13.1 (Query prefix pattern)**.** The *query prefix pattern* $\mathrm{QP}(s, p_1, \dots, p_j)$ is a sequence of length n_j, where the ith element is a list list_i of indices $j' < j$ such that the j'th query also visited $u_{j,i}$.

The index intersection pattern for a query p_j essentially tells when any of the indices $\mathit{ind}_j, \ldots, \mathit{ind}_j + m_j - 1$ are equal to or overlap with any of the indices $\mathit{ind}_i, \ldots, \mathit{ind}_i + m_i - 1$ for any previous queries p_i.

Definition 2.13.2 (Index intersection pattern)**.** The *index intersection pattern* $\mathrm{IP}(s, p_1, \ldots, p_j)$ is a sequence of length j, where the ith element is equal to $r_1[\{\mathit{ind}_i, \ldots, \mathit{ind}_i + m_i - 1\}]$ for a fixed random permutation $r_1 : [n] \to [n]$.

The leaf intersection pattern for a query p_j essentially tells when any of the leaf positions $\mathit{lpos}_j, \ldots, \mathit{lpos}_j + \mathit{num}_j - 1$ are equal to or overlap with any of the leaf positions $\mathit{lpos}_i, \ldots, \mathit{lpos}_i + \mathit{num}_i - 1$ for any previous queries p_i.

Definition 2.13.3 (Leaf intersection pattern)**.** The *leaf intersection pattern* $\mathrm{LP}(s, p_1, \ldots, p_j)$ is a sequence of length j, where the ith element is equal to $r_2[\{\mathit{lpos}_i, \ldots, \mathit{lpos}_i + \mathit{num}_i - 1\}]$ for a fixed random permutation $r_2 : [n] \to [n]$.

The leakage of the scheme $\mathcal{E}_{\mathrm{PM}}$ is as follows. $\mathcal{L}_1(s)$ is just $n = |s|$. $\mathcal{L}_2(s, p_1, \ldots, p_j)$ consists of

$$(m_j = |p_j|,\ \{\mathit{len}_{j,i}\}_{i=1}^{n_j},\ \mathrm{QP}(s, p_1, \ldots, p_j),\ \mathrm{IP}(s, p_1, \ldots, p_j),\ \mathrm{LP}(s, p_1, \ldots, p_j))\ .$$

For example, consider the string s “cocoon” (whose suffix tree is shown in Figure 2-1) and a sequence of three queries, $p_1 =$ “co”, $p_2 =$ “coco”, and $p_3 =$ “cocoa”. Then the leakage $\mathcal{L}_1(s)$ is $n = 6$.

The query for “co” visits node u_2, the retrieved indices into s are $1, 2$, and the retrieved leaf positions are $1, 2$. The query for “coco” visits nodes u_2 and u_3, the indices retrieved are $1, 2, 3, 4$, and the leaf positions retrieved are 1. The query for “cocoa” visits nodes u_2 and u_3, the indices retrieved are $1, 2, 3, 4, 5$, and no leaf positions are retrieved (because there is not a match).

Thus, the leakage $\mathcal{L}_2(s, p_1, p_2, p_3)$ consists of:

- the lengths $2, 4, 5$ of the patterns,

- the query prefix pattern, which says that p_1, p_2, p_3 visited the same first node, and then p_2 and p_3 visited the same second node,

- the index intersection pattern, which says that two of the indices returned for p_2 are the same as the two indices returned for p_1, and four of the indices returned for p_3 are the same as the four indices returned for p_2, and

- the leaf intersection pattern, which says that the leaf returned for p_2 is one of the two leaves returned for p_1.

2.13.2 Malicious $(\mathcal{L}_1, \mathcal{L}_2)$-CQA2 Security

Theorem 2.13.4. *Let $\mathcal{L}_1$ and $\mathcal{L}_2$ be defined as in Section 2.13.1. If F is a PRF, P is a PRP, and $\mathcal{E}_{\text{SKE}}$ is a CPA-secure, key-private symmetric-key encryption scheme, then the pattern matching encryption scheme $\mathcal{E}_{\text{PM}}$ satisfies malicious $(\mathcal{L}_1, \mathcal{L}_2)$-CQA2 security.*

Proof. We define a simulator $\mathcal{S}$ that works as follows. $\mathcal{S}$ first chooses random keys $K_D, K_C, K_L \xleftarrow{R} \{0,1\}^\lambda$.

Ciphertext. Given $\mathcal{L}_1(s) = n$, $\mathcal{S}$ constructs a simulated ciphertext as follows.

1. Construct a dictionary D as follows. For $i = 1, \ldots, 2n$, choose fresh random values $\kappa_i, f_{2,1}, \ldots, f_{2,d}, \xleftarrow{R} \{0,1\}^\lambda$, and store $V_i = (f_{2,1}, \ldots, f_{2,d}, W = \mathcal{E}_{\text{SKE}}.Enc(K_D, 0))$ with search key κ_i in D.

2. Choose an arbitrary element $\sigma_0 \in \Sigma$. Construct an array C, where $C[i] = \mathcal{E}_{\text{SKE}}.Enc(K_C, (\sigma_0, 0))$ for $i = 1, \ldots, n$.

3. Construct an array L, where $L[i] = \mathcal{E}_{\text{SKE}}.Enc(K_L, 0)$ for $i = 1, \ldots, n$.

Output $CT = (D, C, L)$.

Tables. In order to simulate the query protocol, $\mathcal{S}$ will need to do some bookkeeping.

$\mathcal{S}$ will maintain two tables $\mathcal{T}_1$ and $\mathcal{T}_2$, both initially empty. $\mathcal{T}_1$ contains all currently defined tuples (i, j, κ) such that the entry in D with search key κ represents the jth node visited by the ith query. We write $\mathcal{T}_1(i, j) = \kappa$ if (i, j, κ) is an entry in $\mathcal{T}_1$.

$\mathcal{T}_2$ contains all currently defined tuples $(\kappa, f_2, flag, flag_1, \ldots, flag_d)$, where for the node u represented by the entry $D(\kappa)$, $\kappa = f_1(u)$, $f_2 = f_2(u)$, $flag$ indicates whether u has been visited by any query, and $flag_i$ indicates whether child$(u, \pi_u(i))$ has been visited. The value of each flag is either "visited" or "unvisited". We write $\mathcal{T}_2(\kappa) = (f_2, flag, flag_1, \ldots, flag_d)$ if $(\kappa, f_2, flag, flag_1, \ldots, flag_d)$ is an entry in $\mathcal{T}_2$.

Choose an arbitrary entry (κ^*, V^*) in D to represent the root node of $Tree_s$. In $\mathcal{T}_2(\kappa)$, set all flags to "unvisited" and set $f_2 = 0$. (The f_2 for the root node will never be used, so it is fine to set it to 0.) Define $\mathcal{T}_1(i, 0) = \kappa^*$ for any i.

Query protocols. For the jth token query p_j, $\mathcal{S}$ is given $\mathcal{L}_2(s, p_1, \ldots, p_j)$, which consists of $m_j = |p_j|$, $\{len_{j,i}\}_{i=1}^{n_j}$, QP$(s, p_1, \ldots, p_j)$, IP$(s, p_1, \ldots, p_j)$, and LP$(s, p_1, \ldots, p_j)$.

For $t = 1, \ldots, n_j$, if $list_t =$ QP$(p_j, s)[t]$ is non-empty (i.e., the node $u_{j,t}$ was visited by a previous query), let j' be one of the indices in $list_t$. Let $\kappa = \mathcal{T}_1(j', t)$ and let $(f_2, flag, flag_1, \ldots, flag_d) = \mathcal{T}_2(\kappa)$. $T_{len_{j,t}} = \mathcal{E}_{\text{SKE}}.Enc(f_2, \kappa)$. Set $\mathcal{T}_1(j, t) = \kappa$.

If instead $list_t$ is empty, choose a random unused entry (κ, V) in D to represent the node $u_{j,t}$, and set $\mathcal{T}_1(j, t) = \kappa$. Let $\kappa' = \mathcal{T}_1(j, t-1)$ and let $(f_2, flag, flag_1, \ldots, flag_d) = \mathcal{T}_2(\kappa')$. Choose a random $i \in \{1, \ldots, d\}$ such that $flag_i$ is "unvisited", and set $flag_i$ to "visited". Let $f_{2,i}$ be $D(\kappa').f_{2,i}$. Set $T_{len_t} = \mathcal{E}_{\text{SKE}}.Enc(f_{2,i}, \kappa)$, set $\mathcal{T}_2(\kappa).f_2 = f_{2,i}$, set $\mathcal{T}_2(\kappa).flag$ to "visited", and set $\mathcal{T}_2(\kappa).flag_i$ to "unvisited" for $i = 1, \ldots, d$.

For any $i \neq len_t$ for any $t = 1, \ldots, n_j$, choose a random $f_2 \overset{\text{R}}{\leftarrow} \{0, 1\}^\lambda$, and let $T_i = \mathcal{E}_{\text{SKE}}.Enc(f_2, 0)$.

Send $(T_1, \ldots, T_m)$ to the adversary.

Upon receiving a W from the adversary, check whether $W = D(\mathcal{T}_1(j, n_j)).W$. If not, output $\perp$. Otherwise, let $(x_1, \ldots, x_m)$ be a random ordering of the elements of the set IP$(p_j, s)[j]$, and send $(x_1, \ldots, x_m)$ to the adversary.

Upon receiving $C_1, \ldots, C_m$ from the adversary, check whether $C_i = C[x_i]$ for each i. If not, output $\perp$. Otherwise, let $(y_1, \ldots, y_{num})$ be a random ordering of the elements of $\mathrm{LP}(p_j, s)[j]$, and send $(y_1, \ldots, y_{num})$ to the adversary.

Upon receiving $L_1, \ldots, L_{num}$ from the adversary, check whether $L_i = L[y_i]$ for each i. If not, output $\perp$.

This concludes the description of the simulator $\mathcal{S}$.

Sequence of games. We now show that the real and ideal experiments are indistinguishable by any PPT adversary $\mathcal{A}$ except with negligible probability. To do this, we consider a sequence of games $G_0, \ldots, G_{16}$ that gradually transform the real experiment into the ideal experiment. We will show that each game is indistinguishable from the previous one, except with negligible probability.

Game G_0. This game corresponds to an execution of the real experiment, namely,

- The challenger begins by running $\mathit{Gen}(1^\lambda)$ to generate a key K.
- The adversary $\mathcal{A}$ outputs a string s and receives $CT \leftarrow \mathit{Enc}(K, s)$ from the challenger.
- $\mathcal{A}$ adaptively chooses patterns $p_1, \ldots, p_q$. For each p_i, $\mathcal{A}$ first interacts with the challenger, who is running $\mathit{IssueQuery}(K, p_i)$ honestly. Then $\mathcal{A}$ outputs a description of a function g_i, and receives $g_i(A_1, \ldots, A_i)$ from the challenger, where A_i is the challenger's private output from the interactive protocol for p_i.

Game G_1. This game is the same as G_0, except that in G_1 the challenger is replaced by a simulator that does not generate keys K_1, K_2 and replaces F_{K_1} and F_{K_2} with random functions. Specifically, the simulator maintains tables R_1, R_2, initially empty. Whenever the challenger in G_0 computes $F_{K_i}(x)$ for some x, the simulator uses $R_i(x)$ if it is defined; otherwise, it chooses a random value from $\{0,1\}^\lambda$, stores it as $R_i(x)$, and uses that value.

A hybrid argument shows that G_1 is indistinguishable from G_0 by the PRF property of F.

Lemma 2.13.5. *If F is a PRF, then G_0 and G_1 are indistinguishable, except with negligible probability.*

Proof. We consider a hybrid game H_1. H_1 is the same as G_0 except that it uses R_1 in place of F_{K_1}.

Suppose an adversary $\mathcal{A}$ can distinguish G_0 from H_1. Then we can construct an algorithm $\mathcal{B}$ that attacks the PRF property of F with the same advantage. $\mathcal{B}$ acts as $\mathcal{A}$'s challenger in G_0, except that whenever there is a call to $F_{K_1}(x)$, $\mathcal{B}$ queries its oracle on x. When $\mathcal{A}$ outputs a guess bit, $\mathcal{B}$ outputs the same guess bit. If $\mathcal{B}$'s oracle is a function from F, $\mathcal{A}$'s view will be the same as in game G_0, while if it is a random function, $\mathcal{A}$'s view will be the same as in game H_1. Thus, $\mathcal{B}$ answers its challenge correctly whenever $\mathcal{A}$ does, and breaks the PRF property of F with the same advantage that $\mathcal{A}$ distinguishes games G_0 and H_1.

A similar argument shows that games H_1 and G_1 are indistinguishable by the PRF property of F. Thus, we conclude that G_0 and G_1 are indistinguishable. □

Game G_2**.** This game is the same as G_1, except that in G_2 the simulator does not generate keys K_3, K_4 and replaces P_{K_3} and P_{K_4} with random permutations. Specifically, the simulator maintains tables R_3 and R_4, initially empty. Whenever the simulator in G_1 computes $P_{K_i}(x)$ for some x, the simulator in G_2 uses $R_i(x)$, if it is defined; otherwise, it chooses a random value in $[n]$ that has not yet been defined as $R_i(y)$ for any y, and uses that value.

A hybrid argument similar to the one used for G_0 and G_1 shows that G_1 and G_2 are indistinguishable by the PRP property of P.

Lemma 2.13.6. *If P is a PRP, then G_2 and G_1 are indistinguishable, except with negligible probability.*

Proof. We consider a hybrid game H_1. Game H_1 is the same as G_1 except that it uses R_3 in place of P_{K_3}.

Suppose $\mathcal{A}$ can distinguish G_0 from H_1. Then we can construct an algorithm $\mathcal{B}$ that attacks the PRF property of F with the same advantage. $\mathcal{B}$ acts as $\mathcal{A}$'s challenger in G_1, except that whenever there is a call to $P_{K_3}(x)$, $\mathcal{B}$ queries its oracle on x. When $\mathcal{A}$ outputs a guess bit, $\mathcal{B}$ outputs the same guess bit. If $\mathcal{B}$'s oracle is a function from P, $\mathcal{A}$'s view will be the same as in game G_1, while if it is a random permutation, $\mathcal{A}$'s view will be the same as in game H_1. Thus, $\mathcal{B}$ answers its challenge correctly whenever $\mathcal{A}$ does, and breaks the PRP property of P with the same advantage that $\mathcal{A}$ distinguishes games G_1 and H_1.

A similar argument shows that games H_1 and G_2 are indistinguishable by the PRP property of P. Thus, we conclude that G_1 and G_2 are indistinguishable.

□

Game G_3. This is the same as G_2, except that we modify the simulator as follows. For any query, when the simulator receives $C_1, \ldots, C_m$ from the adversary in response to indices $x_1, \ldots, x_m$, the simulator's decision whether to output $\perp$ is not based on the decryptions of $C_1, \ldots, C_m$. Instead, it outputs $\perp$ if $C_i \neq C[x_i]$ for any i. Otherwise, the simulator proceeds as in G_2.

We argue that games G_3 and G_2 are indistinguishable by the ciphertext integrity of $\mathcal{E}_{\mathrm{SKE}}$.

Lemma 2.13.7. *If $\mathcal{E}_{\mathrm{SKE}}$ has ciphertext integrity, then G_2 and G_3 are indistinguishable, except with negligible probability.*

Proof. We analyze the cases in which G_2 and G_3 each output $\perp$ in response to $C_1, \ldots, C_m$.

For each i, G_2 outputs $\perp$ if either of the following events occur:

- (Event $C.1$) $\mathcal{E}_{\mathrm{SKE}}.Dec(K_C, C_i) = \perp$, or
- (Event $C.2$) $\mathcal{E}_{\mathrm{SKE}}.Dec(K_C, C_i) = (p'_i, j)$ where j is not the correct index.

For each i, G_3 outputs $\perp$ if $C_i \neq C[x_i]$, which happens if either of the following events occur:

- (Event $C.1'$) C_i is not among $C[1], \ldots, C[n]$, or
- (Event $C.2'$) $C_i = C[k]$ where $k \neq x_i$.

If G_3 outputs $\perp$ for some i then G_2 outputs $\perp$ except with negligible probability, as we already showed by ciphertext integrity of $\mathcal{E}_{\mathrm{SKE}}$ in Lemma 2.12.3 in the proof of correctness of $\mathcal{E}_{\mathrm{PM}}$ against malicious adversaries.

If G_2 outputs $\perp$, if event $C.1$ occurred, then $C.1'$ also occurred, since C_i will decrypt successfully if it is one of $C[1], \ldots, C[n]$. If event $C.2$ occurred, then either $C.1'$ or $C.2'$ occurred, since C_i will decrypt to the correct value if $C_i = C[x_i]$. Therefore, if G_2 outputs $\perp$ for some i, so does G_3.

Thus, G_2 and G_3 are indistinguishable except with negligible probability. □

Game G_4**.** This game is the same as G_3, except for the following differences. The simulator does not decrypt the $C_1, \ldots, C_m$ from the adversary. For any query p, instead of deciding whether to output $\emptyset$ based on the decryptions of $C_1, \ldots, C_m$, the simulator outputs $\emptyset$ if p is not a substring of s. Otherwise, the simulator proceeds as in G_3.

As we showed in Lemmas 2.12.2 and 2.12.3, if the adversary does not send the correct W, the client will respond with $\perp$, and if the adversary does not send the correct $C_1, \ldots, C_m$, the client will also respond with $\perp$. Therefore, if the simulator has not yet output $\perp$ when it is deciding whether to output $\emptyset$, then $C_1, \ldots, C_m$ are necessarily the correct ciphertexts, and the decryptions $p'_1, \ldots, p'_m$ computed in G_3 match p if and only if p is a substring of s. Therefore, G_3 and G_4 are indistinguishable.

Game G_5 . This game is the same as G_4, except that in G_5, for $i = 1, \ldots, n$, instead of setting $c_i = \mathcal{E}_{\mathrm{SKE}}.Enc(K_C, (s_i, i))$, the simulator sets $c_i = \mathcal{E}_{\mathrm{SKE}}.Enc(K_C, (\sigma_0, 0))$, where σ_0 is an arbitrary element of Σ.

Note that in both G_4 and G_5, K_C is hidden and the c_i's are never decrypted. A hybrid argument shows that games G_4 and G_5 are indistinguishable by CPA security of $\mathcal{E}_{\mathrm{SKE}}$.

Lemma 2.13.8. *If $\mathcal{E}_{\text{SKE}}$ is a CPA-secure encryption scheme, then G_4 and G_5 are indistinguishable, except with negligible probability.*

Proof. We show this via a series of $n+1$ hybrid games $H_0, \ldots, H_n$. Let σ_0 be an arbitrary element of Σ. In H_i, during the encryption phase, for $i' \leq i$, the simulator computes $c_{i'}$ as $\mathcal{E}_{\text{SKE}}.Enc(K_C, (\sigma_0, 0))$. For $i' > i$, it computes $c_{i'}$ as $\mathcal{E}_{\text{SKE}}.Enc(K_C, (s_{i'}, i'))$. The rest of the game proceeds as in G_4. Note that $H_0 = G_4$ and $H_n = G_5$.

If there is an adversary $\mathcal{A}$ that can distinguish H_{i-1} from H_i for any $i \in \{1 \ldots n\}$, then we can construct an algorithm $\mathcal{B}$ that attacks the CPA security of $\mathcal{E}_{\text{SKE}}$ with the same advantage.

$\mathcal{B}$ acts as the simulator in H_{i-1}, with the following exceptions. During the encryption phase, for $i' < i$, $\mathcal{B}$ generates $c_{i'}$ by querying the encryption oracle on $(\sigma_0, 0)$, and for $i' > i$, $\mathcal{B}$ generates $c_{i'}$ by querying the encryption oracle on $(s_{i'}, i')$. $\mathcal{B}$ outputs $(s_i, i), (\sigma_0, 0)$ as its challenge, and uses the challenge ciphertext as c_i.

Now, if $\mathcal{B}$'s challenger returns an encryption of (s_i, i), then $\mathcal{A}$'s view will be the same as in H_{i-1}, while if the challenger returns an encryption of $(\sigma_0, 0)$, then $\mathcal{A}$'s view will be the same as in H_i. Thus, $\mathcal{B}$ answers its challenge correctly whenever $\mathcal{A}$ does, and breaks the CPA security of $\mathcal{E}_{\text{SKE}}$ with the same advantage that $\mathcal{A}$ distinguishes games H_{i-1} and H_i.

Since there are a polynomial number of games $H_0, \ldots, H_n$, we conclude that $H_0 = G_4$ and $H_n = G_5$ are indistinguishable. □

Game G_6**.** This game is the same as G_5, except that we eliminate the use of the random permutation R_3, in the following way. For $i = 1, \ldots, n$, the simulator set $C[i] = c_i$ instead of $C[R_3(i)] = c_i$, where $c_i = \mathcal{E}_{\text{SKE}}.Enc(K_C, (\sigma_0, 0))$. Furthermore, for any query p_j, the simulator is given an additional input $\text{IP}(s, p_1, \ldots, p_j)$ (as defined in Section 2.13.1). To generate $(x_1, \ldots, x_m)$ in the query protocol, the simulator outputs a random ordering of the elements in $\text{IP}(s, p_1, \ldots, p_j)[j]$.

Since each c_i is an encryption under K_C of $(\sigma_0, 0)$, it does not matter whether the c_i's are permuted in C; if we permute the c_i's or not, the result is indistinguishable. After we eliminate the use of R_3 in generating C, R_3 is only used by the simulator to compute $(x_1, \ldots, x_m)$. Thus, we can replace the computation of $(x_1, \ldots, x_m)$ for each query p_j with a random ordering of the elements of $\mathrm{IP}(s, p_1, \ldots, p_j)[j]$, and the result will be indistinguishable.

Game G_7**.** This is the same as G_6, except that we modify the simulator as follows. For any query, when the simulator receives $L_1, \ldots, L_{num}$ from the adversary in response to indices $y_1, \ldots, y_{num}$, the simulator's decision whether to output $\perp$ is not based on the decryptions of the $L_1, \ldots, L_{num}$; instead, it outputs $\perp$ if $L_i \neq L[y_i]$ for any i; otherwise, it proceeds to compute the answer A as in G_6.

A hybrid argument shows that games G_6 and G_7 are indistinguishable by the ciphertext integrity of $\mathcal{E}_{\mathrm{SKE}}$.

Lemma 2.13.9. *If $\mathcal{E}_{\mathrm{SKE}}$ has ciphertext integrity, then G_6 and G_7 are indistinguishable, except with negligible probability.*

The proof is omitted since it is nearly identical to the proof for G_2 and G_3.

Game G_8**.** This game is the same as G_7, except for the following differences. The simulator does not decrypt the $L_1, \ldots, L_{num}$ from the adversary. For any query p_j, instead of computing the answer A_j using the decryptions of $L_1, \ldots, L_{num}$, if A_j has not already been set to $\perp$ or $\emptyset$, the simulator sets $A_j = \mathcal{F}(s, p_j)$.

As we showed in Lemmas 2.12.2, 2.12.3, and 2.12.4, if any of the W, $C_1, \ldots, C_m$ or $L_1, \ldots, L_{num}$ from the adversary are incorrect, the client will respond to the incorrect message with $\perp$. Therefore, if the simulator has not yet output $\perp$ when it is computing A_j, then the adversary has executed *AnswerQuery* honestly, and $A_j = \mathcal{F}(s, p_j)$ (by correctness of $\mathcal{E}_{\mathrm{PM}}$). Therefore, G_7 and G_8 are indistinguishable.

Game G_9**.** This game is the same as G_8, except that in G_9, for each $i = 1, \ldots, n$, the simulator generates each ℓ_i as $\mathcal{E}_{\mathrm{SKE}}.Enc(K_L, 0)$ instead of $\mathcal{E}_{\mathrm{SKE}}.Enc(K_L, (ind_{leaf_i}, i))$.

A hybrid argument shows that G_8 and G_9 are indistinguishable by the CPA security of $\mathcal{E}_{\text{SKE}}$.

Lemma 2.13.10. *If $\mathcal{E}_{\text{SKE}}$ is a CPA-secure encryption scheme, then G_8 and G_9 are indistinguishable, except with negligible probability.*

The proof is omitted since it is nearly identical to the proof for G_4 and G_5.

Game G_{10}. This game is the same as G_9, except that we eliminate the use of the random permutation R_4, in the following way. For $i = 1, \ldots, n$, the simulator set $L[i] = \ell_i$ instead of $L[R_4(i)] = \ell_i$, where $\ell_i = \mathcal{E}_{\text{SKE}}.Enc(K_L, 0)$. Furthermore, for any query p_j, the simulator is given an additional input $\text{LP}(s, p_1, \ldots, p_j)$ (as defined in Section 2.13.1). To generate $(y_1, \ldots, y_{num})$ in the query protocol, the simulator outputs a random ordering of the elements in $\text{LP}(s, p_1, \ldots, p_j)[j]$.

The argument for game G_{10} is analogous to the one for game G_6. Since each ℓ_i is an encryption under K_L of 0, it does not matter whether the ℓ_i's are permuted in L; if we permute the ℓ_i's or not, the result is indistinguishable. After we eliminate the use of R_4 in generating L, R_4 is only used by the simulator to compute $(y_1, \ldots, y_{num})$. Thus, we can replace the computation of $(y_1, \ldots, y_{num})$ for each query p_j with a random ordering of the elements of $\text{LP}(s, p_1, \ldots, p_j)[j]$, and the result will be indistinguishable.

Game G_{11}. This is the same as G_{10}, except that we modify the simulator as follows. For any query, when the simulator receives a W from the adversary in response to $T_1, \ldots, T_m$, the simulator's decision whether to output $\perp$ will not based on the decryption of W. Instead, it will output $\perp$ if W is not the ciphertext in the dictionary entry $D(R_1(p[1..i]))$, where $p[1..i]$ is the longest matching prefix of p. Otherwise, the simulator proceeds as in G_{10}.

We argue that games G_{10} and G_{11} are indistinguishable by the ciphertext integrity of $\mathcal{E}_{\text{SKE}}$.

Lemma 2.13.11. *If $\mathcal{E}_{\text{SKE}}$ has ciphertext integrity, then G_{10} and G_{11} are indistinguishable, except with negligible probability.*

Proof. We analyze the cases in which G_{10} and G_{11} each output $\perp$ in response to a W.

G_{10} runs $\mathcal{E}_{\mathrm{SKE}}.\mathit{Dec}(K_D, W)$ to get either $\perp$ or a tuple X, which it parses as $(\mathit{ind}, \mathit{lpos}, \mathit{num}, \mathit{len}, f_1, f_{2,1}, \ldots, f_{2,d})$. G_{10} outputs $\perp$ if any of the following events occur:

- (Event $L.1$) $\mathcal{E}_{\mathrm{SKE}}.\mathit{Dec}(K_D, W) = \perp$, or
- (Event $L.2$) W decrypts successfully, but $f_1 \neq R_1(p[1..\mathit{len}])$, or
- (Event $L.3$) W decrypts successfully and $f_1 = R_1(p[1..\mathit{len}])$, but $\mathcal{E}_{\mathrm{SKE}}.\mathit{Dec}(f_{2,i}, T_j) \neq \perp$ for some $i \in \{1, \ldots, d\}$, $j > \mathit{len}$.

G_{11} outputs $\perp$ if W is not the ciphertext in the dictionary entry $D(R_1(p[1..i]))$, where $p[1..i]$ is the longest matching prefix of p, which is the case if any of the following events occur:

- (Event $L.1'$) W is not a ciphertext in D,
- (Event $L.2'$) W is a ciphertext in D but not for any prefix of p. That is, $W = D(\kappa)$ where κ is not equal to $R_1(p[1..i])$ for any i.
- (Event $L.3'$) W is a ciphertext in D for a prefix of p, but there is a longer matching prefix of p. That is, $W = D(R_1(p[1..i]))$ for some i, but there exists a $j > i$ such that there is an entry $D(R_1(p[1..j]))$.

If G_{11} outputs $\perp$ in response to W for any query, then G_{10} also outputs $\perp$ except with negligible probability, as we already showed by ciphertext integrity of $\mathcal{E}_{\mathrm{SKE}}$ in Lemma 2.12.2 in the proof of correctness of $\mathcal{E}_{\mathrm{PM}}$ against malicious adversaries.

If G_{10} outputs $\perp$, then G_{11} also outputs $\perp$, since if W is the ciphertext in $D(R_1(p[1..i]))$, then W will decrypt successfully, with $f_1 = R_1(p[1..\mathit{len}])$, and $\mathcal{E}_{\mathrm{SKE}}.\mathit{Dec}(f_{2,k}, T_j) = \perp$ for all $k \in \{1, \ldots, d\}, j > i$.

Thus, G_{10} and G_{11} are indistinguishable except with negligible probability. $\square$

Game G_{12}**.** This is the same as G_{11}, except that the simulator in G_{12} does not decrypt the W from the adversary in the query protocol.

Since the simulator in G_{11} no longer uses any values from the decryption of W, G_{12} is indistinguishable from G_{11}.

Game G_{13}**.** This is the same as G_{12}, except that in G_{13}, for each node u the simulator generates W_u as $\mathcal{E}_{\text{SKE}}.Enc(K_D, 0)$ instead of $\mathcal{E}_{\text{SKE}}.Enc(K_D, X_u)$.

A hybrid argument shows that G_{12} and G_{13} are indistinguishable by the CPA security of $\mathcal{E}_{\text{SKE}}$.

Lemma 2.13.12. *If* $\mathcal{E}_{\text{SKE}}$ *is a CPA-secure encryption scheme, then* G_{12} *and* G_{13} *are indistinguishable, except with negligible probability.*

The proof is omitted since it is nearly identical to the proof for G_4 and G_5.

Game G_{14}**.** This is the same as game G_{13}, except that in the query protocol, for any non-matching prefix $p[1..i]$, the simulator replaces T_i with an encryption under a fresh random key. That is, for any query p, for any prefix $p[1..i]$, $i = 1, \ldots, m$, if $p[1..i]$ is a non-matching prefix, the simulator chooses a fresh random value r and sets $T_i \leftarrow \mathcal{E}_{\text{SKE}}.Enc(r, R_1(p[1..i]))$; otherwise, it sets $T_i \leftarrow \mathcal{E}_{\text{SKE}}.Enc(R_2(p[1..i]), R_1(p[1..i]))$ as in game G_{13}.

For any k and i, let p_k denote the kth query, and let $T_{k,i}$ denote the T_i produced by the simulator for the kth query. The only way an adversary $\mathcal{A}$ may be able to tell apart G_{13} and G_{14} is if two queries share a non-matching prefix; that is, there exist i, j, j' such that $j \neq j'$ and $p_j[1..i] = p_{j'}[1..i]$. In this case, G_{14} will use different encryption keys to generate $T_{i,j}$ and $T_{i,j'}$, while G_{13} will use the same key. Note that the decryption keys for $T_{i,j}$ and $T_{i,j'}$ will never be revealed to $\mathcal{A}$ in either game. Thus, a hybrid argument shows that G_{13} and G_{14} are indistinguishable by the which-key-concealing property of $\mathcal{E}_{\text{SKE}}$.

Lemma 2.13.13. *If* $\mathcal{E}_{\text{SKE}}$ *is a which-key-concealing encryption scheme, then games* G_{13} *and* G_{14} *are indistinguishable, except with negligible probability.*

Proof. Suppose there exists an adversary $\mathcal{A}$ that can distinguish G_{13} and G_{14}. Let q_{max} be an upper bound on the number of queries $\mathcal{A}$ chooses, and let m_{max} be an upper bound on the length of $\mathcal{A}$'s queries, where q_{max} and m_{max} are polynomial in λ.

Consider the following sequence of $q_{max}(m_{max}+1)$ hybrid games. For each $i \in \{0, \ldots, m_{max}\}, j \in \{1, \ldots, q_{max}\}$, game $H_{i,j}$ is the same as G_{13}, with the following exceptions.

- For $j' < j$, for each i', if $p_{j'}[1..i']$ is non-matching, choose a fresh random value r and set $T_{j',i'} \leftarrow \mathcal{E}_{\text{SKE}}.Enc(r, R_1(p_{j'}[1..i]))$, as in game G_{14}.
- For the jth query p_j, for $i' \leq i$, if $p_j[1..i']$ is non-matching, again choose a fresh random value r and set $T_{j,i'} \leftarrow \mathcal{E}_{\text{SKE}}.Enc(r, R_1(p_j[1..i']))$, as in game G_{14}.

Note that $H_{0,1} = G_{13}$ and $H_{m_{max},q_{max}} = G_{14}$.

Now, we argue that if there is an adversary $\mathcal{A}$ that can distinguish $H_{i-1,j}$ from $H_{i,j}$ for any $i \in \{1, \ldots, m_{max}\}, j \in \{1, \ldots, q_{max}\}$, then we can construct an algorithm $\mathcal{B}$ that attacks the which-key-concealing property of $\mathcal{E}_{\text{SKE}}$ with the same advantage.

$\mathcal{B}$ will act as the simulator in $H_{i-1,j}$, with the following exception. If $p_j[1..i]$ is non-matching, $\mathcal{B}$ first queries its left encryption oracle on $R_1(p_j[1..i])$ and sets $T_{j,i}$ to the resulting ciphertext. $\mathcal{B}$ then remembers $p_j[1..i]$, and for any later queries $p_{j'}$ that share the prefix $p_j[1..i]$, $\mathcal{B}$ queries its right oracle on $R_1(p_j[1..i])$ and uses the resulting ciphertext as $T_{j',i}$. Otherwise, $\mathcal{B}$ proceeds as in $H_{i-1,j}$.

Now, if both of $\mathcal{B}$'s encryption oracles are for the same key, then $T_{j,i}$ and the $T_{j',i}$ for all future queries p'_j that share the prefix $p_j[1..i]$ will be encrypted under the same random key, and $\mathcal{A}$'s view will be the same as in $H_{i-1,j}$. On the other hand, if the two encryption oracles are for different keys, then $T_{j,i}$ will have been generated using a different random key from that used to generate $T_{j',i}$ for all future queries $p_{j'}$ that share the prefix $p_j[1..i]$, and $\mathcal{A}$'s view will be the same as in $H_{i,j}$.

Note that if $p_j[1..i]$ is a matching prefix, so $\mathcal{B}$ does not output a challenge, then $H_{i-1,j}$ and $H_{i,j}$ are identical, so $\mathcal{A}$'s view is the same as in both $H_{i-1,j}$ and $H_{i,j}$. Thus, $H_{i-1,j}$ and $H_{i,j}$ are indistinguishable by the key hiding property of $\mathcal{E}_{\text{SKE}}$.

We can show by a very similar reduction that games $H_{m_{max},j}$ and $H_{1,j+1}$ are indistinguishable. Since there are a polynomial number of hybrid games, we conclude then that games $H_{0,1} = G_{13}$ and $H_{m_{max},q_{max}} = G_{14}$ are indistinguishable. □

Game G_{15}. This is the same as game G_{14}, except that in the query protocol for any pattern p, for any non-matching prefix $p[1..i]$, the simulator replaces T_i with an encryption of 0. That is, for any query p, for any prefix $p[1..i]$, $i = 1, \ldots, m$, if $p[1..i]$ is non-matching, the simulator chooses a fresh random value r and sets $T_i \leftarrow \mathcal{E}_{\text{SKE}}.Enc(r, 0)$; otherwise, it sets $T_i \leftarrow \mathcal{E}_{\text{SKE}}.Enc(r, R_1(p[1..i]))$ as in game G_{14}.

The only way an adversary $\mathcal{A}$ may be able to tell apart G_{14} and G_{15} is if a prefix $p_j[1..i]$ is non-matching. In this case, in G_{14}, $T_{j,i}$ will be an encryption of 0, while in G_{15}, $T_{j,i}$ will be an encryption of $R_1(p_j[1..i])$. The decryption key for $T_{j,i}$ will never be revealed to $\mathcal{A}$ in either game. Thus, a hybrid argument shows that games G_{14} and G_{15} are indistinguishable by the CPA security of $\mathcal{E}_{\text{SKE}}$.

Lemma 2.13.14. *If $\mathcal{E}_{\text{SKE}}$ is a CPA-secure encryption scheme, then games G_{14} and G_{15} are indistinguishable, except with negligible probability.*

Proof. Suppose there exists an adversary $\mathcal{A}$ that can distinguish G_{14} and G_{15}. Let q_{max} be an upper bound on the number of queries that $\mathcal{A}$ chooses, and let m_{max} be an upper bound on the length of $\mathcal{A}$'s queries, where q_{max} and m_{max} are polynomial in λ.

We consider a sequence of $q_{max}(m_{max} + 1)$ hybrid games. For each $i \in \{0, \ldots, m_{max}\}$, $j \in \{1, \ldots, q_{max}\}$, game $H_{i,j}$ is the same as G_{14}, with the following exceptions.

- For $j' < j$, for each i', if $p_{j'}[1..i']$ is non-matching, choose a fresh random value r and set $T_{j',i'} \leftarrow \mathcal{E}_{\mathrm{SKE}}.\mathit{Enc}(r, 0)$, as in game G_{15}.
- For the jth query p_j, for $i' \leq i$, if $p_j[1..i']$ is non-matching, again choose a fresh random value r and set $T_{j,i} \leftarrow \mathcal{E}_{\mathrm{SKE}}.\mathit{Enc}(r, 0)$ as in game G_{15}.

Note that $H_{0,1} = G_{14}$ and $H_{m_{max},q_{max}} = G_{15}$.

Now, we argue that if there is an adversary $\mathcal{A}$ that can distinguish $H_{i-1,j}$ from $H_{i,j}$ for any $i \in \{1, \ldots, m_{max}\}, j \in \{1, \ldots, q_{max}\}$, then we can construct an algorithm $\mathcal{B}$ that attacks the CPA security of $\mathcal{E}_{\mathrm{SKE}}$ with the same advantage.

$\mathcal{B}$ acts as the simulator in $H_{i-1,j}$, with the following exception. If $p_j[1..i]$ is non-matching, it chooses a fresh random value r and outputs $r, 0$ as its challenge in the CPA-security game, and sets $T_{i,j}$ to be the resulting ciphertext. Otherwise, $\mathcal{B}$ proceeds as in $H_{i-1,j}$. Note that in both $H_{i-1,j}$ and $H_{i,j}$, the random key r is used to encrypt only one ciphertext.

Now, if the CPA-security challenger gave $\mathcal{B}$ an encryption of r, then $\mathcal{A}$'s view will be the same as in $H_{i-1,j}$. On the other hand if the CPA-security challenger returned an encryption of 0, then $\mathcal{A}$'s view will be the same as in $H_{i,j}$.

Note that if $p_j[1..i]$ is a matching prefix, so $\mathcal{B}$ does not produce a challenge, then $H_{i-1,j}$ and $H_{i,j}$ are identical, so $\mathcal{A}$'s view is the same as in both $H_{i-1,j}$ and $H_{i,j}$. Thus, $H_{i-1,j}$ and $H_{i,j}$ are indistinguishable by the CPA security of $\mathcal{E}_{\mathrm{SKE}}$.

We can show by a very similar reduction that games $H_{m_{max},j}$ and $H_{1,j+1}$ are indistinguishable. Since there are a polynomial number of hybrid games, we conclude then that games $H_{0,1} = G_{14}$ and $H_{m_{max},q_{max}} = G_{15}$ are indistinguishable. □

Game G_{16}. This is the final game, which corresponds to an execution of the ideal experiment. In G_{16}, the simulator is replaced with the simulator $\mathcal{S}$ defined above.

The differences between G_{15} and G_{16} are as follows. In G_{16}, the simulator no longer uses the string s when creating the dictionary D, and for each query p, it no longer uses p when creating $T_1, \ldots, T_m$. When constructing D, whenever the simulator in G_{15} generates a value by applying a random function to a string, $\mathcal{S}$ generates a fresh random value without using the string. Note that all of the $\hat{\rho}(u)$ strings used in D are unique, so $\mathcal{S}$ does not need to ensure consistency between any of the random values. Then, for any query p_j, for each matching prefix $p_j[1..i]$, $\mathcal{S}$ constructs T_i to be consistent with D and with prefix queries using the query prefix pattern $\mathrm{QP}(s, p_1, \ldots, p_j)$. While the simulator in G_{15} associates entries in D to strings when it first constructs D, $\mathcal{S}$ associates entries in D to strings as it answers each new query. However, both simulators produce identical views.

□

2.14 Conclusion

We presented a definition of queryable encryption schemes and defined security against both honest-but-curious and malicious adversaries making chosen query attacks. Our security definitions are parameterized by leakage functions that specify the information that is revealed about the message and the queries by the ciphertext and the query protocols.

We constructed an efficient pattern matching scheme – a queryable encryption scheme that supports finding all occurrences of a pattern p in an encrypted string s. Our approach is based on suffix trees. Our construction uses only basic symmetric-key primitives (pseudorandom functions and permutations and an authenticated, which-key-concealing encryption scheme). The ciphertext size and encryption time are $O(\lambda n)$ and query time and message size are $O(\lambda m + k)$, where λ is the security parameter, n is the length of the string, m is the length of the pattern, and k is the number of occurrences of the pattern. Querying requires only 3 rounds of communication.

While we have given a formal characterization of the leakage of our pattern matching scheme, it is an open problem to analyze the practical cost of the leakage. Given the leakage from several “typical” queries, what can a server infer about the message and the queries? For some applications, the efficiency may be worth the leakage tradeoff, especially in applications where current practice does not use encryption at all.

Chapter 3

Strategic Range Voting and Approval Voting

3.1 Introduction

In this chapter, we study two voting systems: range voting and approval voting. In range voting, studied in detail by Smith [53], the voter gives each alternative a score in some fixed range (e.g., between 0 and 1). In approval voting, studied in detail by Brams and Fishburn [13], the voter "approves" or "disapproves" each alternative. That is, the voter gives each alternative a score of either 0 or 1. Range voting allows several (or infinitely many) possible scores, while approval voting allows only two scores; thus, approval voting can be thought of as a special case of range voting. Range voting provides added expressiveness over approval voting. Thus, it is natural to ask when and whether this added expressiveness is beneficial to a rational (strategic) voter. That is, when would a strategic range voter, with some model of information about the other votes, want to give some alternatives intermediate scores, and when would she want to give each alternative either the maximum or the minimum score (voting "approval-style")? This question was studied previously by Warren Smith and others (see Smith's summary [51]). In this chapter, we study this question more formally and generally.

We first review related results on strategic voting in some simple models of voter information. We then show, for a more general model of information, that in the limit as the number of voters grows large, a rational range voter, maximizing her expected utility, will want to vote approval-style. Thus, in some sense, in most cases the added expressiveness of range voting is not beneficial to a rational voter, and in fact a voter using the intermediate scores in the range is not voting in her best interest.

Next, we propose a more concrete class of functions, beta distributions, as a reasonable and interesting model for voter information. A beta distribution can be used to model a posterior probability distribution on the average score for each alternative after seeing a given number of approvals and disapprovals from pre-election polls, given a uniform prior.

The rest of this chapter is organized as follows. In Section 3.2 we review notation and definitions. In Section 3.3 we describe some related known results about strategic range and approval voting. In Section 3.4 we present our general model for voter information; in Section 3.5 we describe optimal range voting in this model, showing that it can be achieved by approval-style voting. Section 3.6 describes beta distributions as a way to model information about other votes. Sections 3.7 gives conclusions and open problems.

3.2 Preliminaries

Ballots, profiles, and alternatives. We consider an election which uses a profile $P = (B_1, \ldots, B_n)$ containing n ballots (a.k.a. votes) to determine a winner from a set $\mathcal{A} = \{A_1, \ldots, A_m\}$ of m alternatives.

Range voting and approval voting. For both range voting and approval voting, a ballot $\mathbf{B}_i$ is a vector of m numbers (called "scores"):

$$\mathbf{B}_i = (B_{i,1}, B_{i,2}, \ldots, B_{i,m}) \ .$$

Each component $B_{i,j}$ is the level of support the ballot $\mathbf{B}_i$ expresses for alternative A_j; larger numbers indicate greater support.

In range voting, each score $B_{i,j}$ is a number in the range from some b_{min} up to some b_{max}. The scores may be allowed to be any real number in the interval $[b_{min}, b_{max}]$, or they may be restricted to integers in that range, depending on the version of range voting that is used. We will let the scores be any real number in $[b_{min}, b_{max}]$. Without loss of generality, we let $b_{min} = 0$ and $b_{max} = 1$.

In approval voting, each score $B_{i,j}$ is either 0 or 1. A ballot $\mathbf{B_i}$ indicates approval of those alternatives A_j for which $B_{i,j} = 1$ and disapproval of those alternatives A_j for which $B_{i,j} = 0$. Approval voting can be thought of as a special case of range voting with only two allowed scores.

For both range and approval voting, a winning alternative is one with maximum total score, summed over the votes cast. That is, for $j = 1, \ldots, m$, let $X_j = \Sigma_{i=1}^{n} B_{i,j}$. Then the winner is one of the alternatives A_j for which $X_j = \max\{X_1, \ldots, X_m\}$. We assume ties are broken uniformly at random (although other tie-breaking rules could be used).

Clearly, a voter can ignore the additional expressive freedom allowed by range voting, and give each alternative a score of either 0 or 1, effectively reducing the range voting ballot to just an approval voting ballot, i.e., voting *"approval-style"*. (For range voting, we will sometimes use "approve" to mean "give the maximum score (1) to" and "disapprove" to mean "give the minimum score (0) to".)

Our main question here is: in a range voting election, when would a voter want to vote approval-style?

Other $n-1$ voters. We will consider a single voter Alice who is determining how best to vote. For notational purposes, we will assume Alice is the last of the n voters and casts ballot $\mathbf{B}_n$. Let $X'_j = \Sigma_{i=1}^{n-1} B_{i,j}$ denote the sum of the scores given to alternative A_j by the other $n-1$ voters.

The actual order in which the voters cast their ballots is unimportant. We just need that the other votes do not depend on Alice's vote. Although we say Alice is the last voter, Alice does not see the previous votes, and instead she has only some partial information or beliefs (the modeling of which we will discuss later) about the other votes.

Utilities. We make use of utility functions, as developed in the utility theory of von Neumann and Morgenstern [58]. We assume that Alice has a utility u_i for the outcome that A_i is the winner, for each i. Let $\mathbf{u} = (u_1, u_2, \ldots, u_m)$. Alice prefers alternative A_j to alternative A_k if $u_j > u_k$. In game-theoretic terms, u_{ij} is the payoff to voter i if alternative A_j wins.

We assume that Alice's utilities are a function of only the selected winner; no utility is derived from other factors such as the actual number of votes any alternative receives, the act of voting "sincerely", etc.

We assume that Alice is rational, meaning that she votes in a way that maximizes her expected utility (over Alice's uncertainty about the other votes and any randomness used to break ties) of the alternative selected as the winner of the election. We call a ballot *optimal* for Alice if it maximizes her expected utility of the winner of the election.

We only consider the utilities of a single voter, Alice, so we do not have any issues that may arise with interpersonal comparison of utility.

Three or more alternatives. We assume there are three or more alternatives ($m > 3$). Otherwise, the optimal voting strategy is trivial, regardless of what information is known about the other votes. If there is only one alternative, all ballots are optimal. If there are only two alternatives A_1 and A_2, if $u_1 = u_2$ (i.e., Alice is indifferent between A_1 and A_2), then all ballots are optimal. Otherwise (if there are only two alternatives A_1 and A_2 and $u_1 \neq u_2$), without loss of generality suppose $u_1 > u_2$. Then, a ballot that gives the maximum score 1 to A_1 and the minimum score 0 to A_2 is optimal.

Sincerity. In addition to the question of whether a rational range voter's best strategy can be achieved by approval-style voting, we are also interested in whether the best strategy is *sincere*. We say that a range voting ballot $\mathbf{b} = (b_1, \ldots, b_m)$ is sincere if, for any two alternatives A_i and A_j, if $u_i \geq u_j$, then $b_i \geq b_j$.

Brams and Fishburn [13] first noted that for approval voting, when there are only $m = 3$ alternatives, a voter's optimal strategy can always be achieved by a sincere ballot, since it can only help the voter to approve her most preferred candidate and disapprove her least preferred candidate. The same argument applies to range voting – an optimal strategy can always include a score of 1 for the most preferred candidate and a score of 0 for the least preferred candidate.

3.3 Strategic Range Voting: Known Results

Complete information. First, we observe that if Alice has complete information and knows the totals X'_i from the other voters exactly, then it is simple to determine an optimal strategy for Alice – essentially, give the maximum score 1 to her most preferred alternative that can be made to win, and give the minimum score 0 to every other alternative.

That is, if the winner is already determined by the other votes, i.e., the difference between the highest and second highest totals among $X'_1, \ldots, X'_m$ is greater than 1, then Alice's vote cannot change the outcome, and all possible ballots are optimal.

Otherwise, for each $j = 1, \ldots, m$, let $\mathbf{b_j}$ be the ballot that gives a score of 1 to alternative A_j and 0 to every other alternative. Alice computes her expected utility for the election winner if she casts ballot $\mathbf{b_j}$, for each j, and casts the one (or any of the ones) that maximizes her expected utility. (The only reason this is an *expected* utility is because there may be a tie for winner that will be broken randomly; otherwise, Alice's utility is known, since we are in a "complete information" scenario.)

If there are no alternatives A_j that have a score X'_j that is exactly 1 less than the maximum of $X'_1, \ldots, X'_m$, then the strategy given above is just to give a score of 1 to her most preferred alternative among the ones whose score from the other voters is within 1 of the leader.

The approval-style strategy given above may be one among an infinite number of optimal strategies, since the range of allowable scores is continuous.

Example 3.3.1. As an example, suppose there are 4 alternatives $A_1, \ldots, A_4$, 10 voters, and the scores from the first 9 voters are $X'_1 = 4.0$, $X'_2 = 6.2$, $X'_3 = 5.7$, and $X'_4 = 5.1$. Let Alice's utilities be $u_1 = 30$, $u_2 = 15$, $u_3 = 18$, and $u_4 = 25$.

Then an optimal strategy for Alice is to assign a score of 1 to A_3 and 0 to every other alternative, since A_2 and A_3 are the only alternatives that can win, and Alice prefers A_2 to A_3.

Modifying the above example slightly, we get an example involving ties.

Example 3.3.2. Suppose Alice's utilities are again $u_1 = 30$, $u_2 = 15$, $u_3 = 18$, and $u_4 = 25$, but the scores from the other voters are $X'_1 = 4.0$, $X'_2 = 6.2$, $X'_3 = 5.7$, and $X'_4 = 5.2$.

Then it is now optimal for Alice to assign a score of 1 to A_4 and 0 to every other alternative, since doing so creates a tie for the winner between A_2 and A_4, so Alice achieves an expected utility of $(u_2 + u_4)/2 = (15 + 25)/2 = 20$. Giving the max score to A_3 instead would make A_3 the outright winner and Alice would achieve a utility of 18.

Zero information. At the opposite extreme of full information, we can consider the case where Alice has "zero" information about the other votes. More precisely, Alice assumes that every other voter gives every alternative a score that is uniformly random in $[0, 1]$. (Note that this is only one among many possible ways to define "zero" information; we don't claim that this is the only reasonable way to model a voter having no information.)

Warren Smith [52] gives an analysis of strategic range voting in the zero-information scenario. Smith assumes that the number of voters n is large and that the probability of a pairwise "near tie" for winner between any two candidates A_i and A_j being broken in favor of A_i by Alice's ballot $\mathbf{B_n}$ is proportional to $\max(B_{n,i} - B_{n,j}, 0)$.

In more detail, a "near tie" is defined as follows:

Definition 3.3.3. We say there is a *near tie* between A_i and A_j if, based on the total scores from the other voters, either A_i or A_j can be the winner with the addition of Alice's vote (depending on how she votes), and no other candidate can win.

So, A_i and A_j are in a near tie for winner if $|X'_i - X'_j| \leq 1$ and $X'_k + 1 < min(X'_i, X'_j)$ for all $k \neq i, j$. The probability that the winner if Alice were to abstain (or cast a ballot of all zeros) is A_j and the winner with the addition of Alice's ballot is A_i is assumed to be proportional to the score difference $B_{n,i} - B_{n,j}$ (if $B_{n,i} > B_{n,j}$; 0 otherwise). Roughly, this property can be interpreted as saying that, when the totals X'_i and X'_j are within 1 of each other, the distribution of the difference $X'_i - X'_j$ is uniform.

The "proportionality property" described above is assumed to be a consequence of the number of voters being large. However, why the proportionality assumption should follow from the number of voters being large is not discussed, although it may seem true intuitively. In our general model in Section 3.4, we will prove this consequence formally.

The final assumption of Smith is that the probabilities of ties for winner between three or more candidates are small enough that they can be ignored.

In the zero information model, since the scores for each alternative are completely random, every pair of distinct alternatives A_i and A_j is equally like to be in a pairwise near tie for winner. Smith [52] proves that when all pairwise ties are equally likely, under the assumptions given above, a range voter's optimal strategy is to vote approval-style using a method known as *mean-based thresholding*: give the maximum score 1 to every alternative A_i for which u_i is at least the average of the utilities for all the alternatives, and give the minimum score 0 to every other alternative. Brams and Fishburn [13] originally proved optimality of mean-based thresholding in an analogous model for *approval voting*. Note that mean-based thresholding is a sincere strategy; if a voter approves of A_i then she will also approve of any alternative A_j for which $u_j > u_i$.

To illustrate mean-based thresholding for range voting, let us look at an example.

Example 3.3.4. Suppose Alice's utilities are again $u_1 = 30$, $u_2 = 15$, $u_3 = 18$, and $u_4 = 25$. Then the average utility is $(30+15+18+25)/4 = 22$, so Alice's mean-based thresholding strategy is to give a score of 1 to A_1 and A_4 and give a score of 0 to A_2 and A_3.

Below we give an adaptation of Smith's proof [52] of the optimality of mean-based thresholding.

Theorem 3.3.5. *Suppose all pairwise near ties are equally likely, ties between three or more alternatives can be ignored, and the probability of Alice's ballot $\boldsymbol{B}_n$ breaking a pairwise near tie (as defined in Definition 3.3.3) between any two alternatives A_i and A_j in favor of A_i is proportional to* $\max(B_{n,i} - B_{n,j}, 0)$. *Then mean-based thresholding is an optimal range voting strategy for Alice.*

Proof. For any i and j, let p_{ij} denote the probability of a near tie for winner between A_i and A_j.

For any given alternative A_i, suppose we fix the scores on Alice's ballot for all the other alternatives A_j, $j \neq i$. Then, for A_i, suppose Alice starts with a score of 0 and then considers increasing her score for A_i by some amount q. The resulting change Δ in Alice's expected utility is proportional to

$$\Sigma_{j\neq i} p_{ij} \cdot q \cdot (u_i - u_j),$$

since Alice gains u_i and loses u_j if the increase in A_i's score causes A_i to win when A_j was the winner otherwise.

The near-tie probability p_{ij} is the same for all pairs i, j, so Δ is proportional to $q\Sigma_{j\neq i}(u_i - u_j)$. Thus, if $\Sigma_{j\neq i}(u_i - u_j) \geq 0$, we want to set $q = 1$ (i.e., give A_i a score of 1), and otherwise we want $q = 0$ (i.e., give A_i a score of 0.

Simplifying, we get:

$$\begin{array}{rcl} \Sigma_{j\neq i}(u_i - u_j) & \geq & 0 \\ (n-1)u_i & > & \Sigma_{j\neq i} u_j \\ n u_i & > & \Sigma_j u_j \\ u_i & > & (\Sigma_j u_j)/n \ . \end{array}$$

Thus, Alice should approve of all alternatives with utility at least equal to her average utility for the alternatives, and disapprove of all other alternatives. □

Smith [52] also studies several other voting strategies in the zero-information model empirically, using computer simulations, for various numbers of voters.

Laslier's Leader Rule Before presenting our model, we describe related work of Laslier [40] on strategic approval voting. Laslier analyzes how a rational voter should vote when the number of other voters is large, in the following model. It is assumed that Alice knows exactly how many of the other voters (theoretically) approve each alternative (from pre-election polls, for example), but the actual number of votes each alternative gets in the election will be perturbed somewhat by a "trembling ballot" model. In the trembling ballot model, for each voter and for each alternative, there is a small, fixed probability $\epsilon > 0$ that the voter's vote for that alternative will not be recorded (equivalently, that the vote will be recorded as a 0 (disapproval) for that alternative). These "recording mistakes" happen independently for each combination of voter and alternative, regardless of how the voter votes for other alternatives.

Furthermore, it is assumed that Alice believes that ties between three or more candidates can be ignored. Note that this is a behavioral assumption, not a mathematical conclusion from the model; in fact, as Laslier mentions, it is not the case that three-way ties become negligible in the trembling ballot model as the number of voters grows large. However, it is reasonable for voters to believe that three-way ties can be ignored and to vote accordingly.

In the trembling ballot model, Laslier proves that a voter's optimal strategy is given by what is known as the "leader rule": Let A_1 be the alternative with the highest announced score (from pre-election polls), and let A_2 be the alternative with the second highest announced score. Then the leader rule says to approve of every alternative A_j that the voter prefers to A_1 ($u_j > u_1$), and approve of A_1 if the voter prefers it to A_2 ($u_1 > u_2$).

The leader rule is simple to understand, and it is also a sincere strategy. However, the trembling ballot model is somewhat restrictive, as the voter is assumed to have exact information about the other votes; the only uncertainty in the election comes from the recording mistakes that occur with small probability. Furthermore, the voter ignores three-way ties, even though mathematically they cannot necessarily be ignored.

3.4 Our Model

We now present a general way to model a voter's information about the other votes. We will prove formally that under general conditions, a range voter should vote approval-style, as the number of voters grows large.

3.4.1 Notation

We first define some notation that will be useful for our analysis. For any ballot $\mathbf{B}_i = (B_{i,1}, \ldots, B_{i,m})$, let $\overline{B_{i,j}} = B_{i,j}/n$ for each j be "normalized scores", and let $\overline{\mathbf{B}_i} = (\overline{B_{i,1}}, \ldots, \overline{B_{i,m}})$ be the corresponding "normalized ballot".

Let $\overline{X_j} = X_j/n = \Sigma_{i=1}^{n} \overline{B_{i,j}}$ be the average of the scores given to alternative A_j, or the sum of the normalized scores given to alternative A_j. Note that for approval voting, $\overline{X_j}$ is the fraction of voters that approve A_j.

In our analysis we will work with the normalized ballots $\overline{\mathbf{B_i}}$ instead of the original ballots $\mathbf{B_i}$. Clearly, determining the winner using the normalized ballots $\overline{\mathbf{B_i}}$ and the average scores $\overline{X_j}$ is equivalent to using the original ballots $\mathbf{B_i}$ and total scores X_j. A winning alternative A_j is one for which $\overline{X_j} = \max\{\overline{X_1}, \ldots, \overline{X_m}\}$.

Let $\overline{X'_j} = \Sigma_{i=1}^{n-1} \overline{B_{i,j}}$ denote the sum of the normalized scores given to alternative A_j by the other $n-1$ voters. (Note that $\overline{X'_j}$ is not the same as the average of the scores given to A_j by the other $n-1$ voters, since the scores are divided by n, not $n-1$.)

3.4.2 Modeling Information about Other Votes

We model Alice's information about the other votes in a general way. Alice assumes that the sums of the normalized scores from the other $n-1$ voters, $\overline{X'_1}, \ldots, \overline{X'_m}$, are independent, and that each $\overline{X'_i}$ is distributed according to a given probability density function (PDF) f_i and cumulative probability distribution function (CDF) F_i.

For $i = 1, \ldots, m$, let $f_i(x)$ be the PDF for $\overline{X_i'}$, and let $F_i(x)$ be the CDF for $\overline{X_i'}$:

$$F_i(x) = \Pr\{\overline{X_i'} \leq x\} \tag{3.1}$$

$$f_i(x) = \frac{dF_i(x)}{dx} \tag{3.2}$$

$$F_i(x) = \int_0^x f_i(y)\ dy \tag{3.3}$$

We assume that f_i and F_i are well-defined over the entire interval $[0, 1]$. We do not make any other restrictions on f_i or F_i. Thus, our model is quite general.

Note that Alice's information about the other votes may not necessarily be accurate; this is fine for our purposes. Alice's information reflects her expectations about how the other voters will vote, and these are the expectations that Alice will use to determine an optimal ballot to cast. We are interested in how to determine such an optimal ballot, and when optimal range voting can or cannot be achieved through an approval-style ballot.

3.4.3 Maximum of Random Variables

Let $\overline{X_*'}$ denote the maximum of the $\overline{X_i'}$'s, and let $\overline{X_{*-i}'}$ denote the maximum of all of the $\overline{X_j'}$'s *other than* $\overline{X_i'}$.

Let $F_*(x)$ denote the CDF for $\overline{X_*'}$ and let $f_*(x)$ denote the corresponding PDF. Similarly, let F_{*-i} denote the CDF for $\overline{X_{*-i}'}$ and let $f_{*-i}(x)$ denote the corresponding the PDF. Also, let $F_{*-i,j}$ denote the CDF for $\overline{X_{*-i,j}'}$, where $\overline{X_{*-i,j}'}$ is the maximum of all the $\overline{X_k'}$'s other than $\overline{X_i'}$ and $\overline{X_j'}$.

Then

$$F_*(x) = \prod_{i=1}^{m} F_i(x) \tag{3.4}$$

since $\overline{X_*'} \leq x$ if and only if each $\overline{X_i'} \leq x$, using independence.

As a consequence we get that

$$\begin{aligned} f_*(x) &= \frac{dF_*(x)}{dx} && (3.5) \\ &= \sum_i f_i(x) \prod_{j \neq i} F_j(x) && (3.6) \\ &= \sum_i f_i(x) F_{*-i}(x) \ . && (3.7) \end{aligned}$$

Similarly for $F_{*-i}(x)$:

$$F_{*-i}(x) = \prod_{j \neq i} F_j(x) \tag{3.8}$$

$$\begin{aligned} f_{*-i}(x) &= \frac{dF_{*-i}(x)}{dx} && (3.9) \\ &= \sum_{j \neq i} f_j(x) \prod_{k \neq i,j} F_k(x) && (3.10) \\ &= \sum_{j \neq i} f_j(x) F_{*-i,j}(x) \ . && (3.11) \end{aligned}$$

The probability that $\overline{X_i'}$ is the maximum of the m variables $\overline{X_1'}, \ldots, \overline{X_m'}$ is

$$\int_0^1 f_i(x) F_{*-i}(x) \ dx \tag{3.12}$$

Roughly speaking, we sum the probability for each x that $\overline{X_i'} = x$ and that all other $\overline{X_j'}$ are not larger than x.

3.5 Utility of a given ballot

Let $\mathbf{b} = (b_1, b_2, \ldots, b_m)$ denote a given range voting ballot, where each b_i is in the continuous interval $[0, 1]$. Let $\overline{\mathbf{b}} = (\overline{b_1}, \overline{b_2}, \ldots, \overline{b_m})$ be the corresponding normalized ballot, where each $\overline{b_i} = b_i/n$ and is in $[0, 1/n]$.

What is the expected utility for Alice of casting a given ballot $\mathbf{b}$?

The probability that A_i wins, if Alice casts ballot $\mathbf{b}$, is

$$\int_0^1 f_i(x) \left(\prod_{j \neq i} F_j(x + \overline{b_i} - \overline{b_j}) \right) \, dx \tag{3.13}$$

Here $f_i(x)$ is the probability density of A_i receiving a total of x from the normalized votes of the other voters, and $F_j(x+\overline{b_i}-\overline{b_j})$ is the probability that $X'_j+\overline{b_j} \leq X'_i+\overline{b_i}$.

Then Alice's overall expected utility of casting ballot $\mathbf{b}$ is the sum, over each alternative, of the utility for that alternative times the probability of that alternative winning,

$$U(\overline{\mathbf{b}}) = \sum_{k=1}^{m} u_k \int_0^1 f_k(x) \left(\prod_{j \neq k} F_j(x + \overline{b_k} - \overline{b_j}) \right) \, dx \tag{3.14}$$

For any given point (normalized ballot) $\overline{\mathbf{b_0}}$, if U is differentiable at $\overline{\mathbf{b_0}}$, then by the definition of differentiability, we have

$$U(\overline{\mathbf{b}}) = U(\overline{\mathbf{b_0}}) + \nabla U(\overline{\mathbf{b_0}}) \cdot (\overline{\mathbf{b}} - \overline{\mathbf{b_0}}) + R(\overline{\mathbf{b}}), \tag{3.15}$$

where $R(\overline{\mathbf{b}})$ is a remainder term such that $\lim_{\overline{\mathbf{b}} \to \overline{\mathbf{b_0}}} \frac{|R(\overline{\mathbf{b}})|}{||\overline{\mathbf{b}}-\overline{\mathbf{b_0}}||} = 0$, and ∇U is the gradient of the function U.

Letting $\overline{\mathbf{b_0}} = \mathbf{0}$, we get

$$U(\overline{\mathbf{b}}) = U(\mathbf{0}) + \nabla U(\mathbf{0}) \cdot \overline{\mathbf{b}} + o(1/n) \ . \tag{3.16}$$

That is,

$$U(\overline{\mathbf{b}}) = U(\mathbf{0}) + \sum_i c_i \overline{b_i} + o(1/n) \tag{3.17}$$

where

$$c_i = \frac{\partial U}{\partial \overline{b_i}} \Big|_{\overline{\mathbf{b}}=0} \ . \tag{3.18}$$

Since the f_i's are assumed to be well-defined everywhere on the interval $[0,1]$, U is differentiable everywhere on $[0,1/n]^m$ (where differentiability at the endpoints is one-sided). In particular, U is (one-sided) differentiable at $\mathbf{0}$.

The partial derivative $\frac{\partial U}{\partial \overline{b_i}}$ is equal to the following:

$$\begin{aligned}\frac{\partial U}{\partial \overline{b_i}} = u_i \int_0^1 f_i(x) \left[\sum_{\ell \neq i} f_\ell(x + \overline{b_i} - \overline{b_\ell}) \prod_{j \neq i, j \neq \ell} F_j(x + \overline{b_i} - \overline{b_j}) \right] \, dx \\ + \sum_{k \neq i} u_k \int_0^1 f_k(x)(-f_i(x + \overline{b_k} - \overline{b_i})) \prod_{j \neq i, j \neq k} F_j(x + \overline{b_k} - \overline{b_j}) \, dx\end{aligned} \tag{3.19}$$

Evaluating $\frac{\partial U}{\partial \overline{b_i}}$ at $\overline{\mathbf{b}} = \mathbf{0}$, we get

$$\begin{aligned} c_i &= u_i \int_0^1 f_i(x) \left[\sum_{\ell \neq i} f_\ell(x) \prod_{j \neq i, \ell} F_j(x) \right] \, dx \\ &+ \sum_{k \neq i} u_k \int_0^1 f_k(x)(-f_i(x)) \prod_{j \neq i, k} F_j(x) \, dx \\ &= \sum_{k \neq i} (u_i - u_k) \int_0^1 f_i(x) f_k(x) \prod_{j \neq i, j \neq k} F_j(x) \, dx \end{aligned}$$

Thus, $c_i = \frac{\partial U}{\partial \overline{b_i}}\Big|_{\overline{\mathbf{b}}=0} = \sum_{k \neq i} (u_i - u_k) \tau_{ik}$, where

$$\tau_{ik} = \int_0^1 f_i(x) f_k(x) \prod_{j \neq i, j \neq k} F_j(x) \, dx$$

is related to the approximate probability that A_i and A_k are in a near tie for winner. More precisely, $\tau_{ik} \overline{b_i}$ is the approximate probability that adding a normalized score of b_i (instead of 0) to A_i's total causes A_i to be the winner when otherwise A_k would be the winner.

Since the remainder term $R(\overline{\mathbf{b}})$ is $o(1/n)$, as the number of voters grows large ($n \to \infty$), the remainder term is dominated, and the utility $U(\overline{\mathbf{b}})$ of a normalized ballot $\mathbf{b}$ is well-approximated by

$$U(\mathbf{0}) + \sum_i c_i \overline{b_i}.$$

Then, when the number of voters is large, the best range voting ballot for Alice will be $(b_1, \ldots, b_m)$, where

$$b_i = \begin{cases} 1 & \text{if } \; c_i > 0 \\ 0 & \text{otherwise.} \end{cases} \tag{3.20}$$

Remarks. It is important to note that as we vary the number of voters, the probability density functions f_i (and the corresponding cumulative distribution functions F_i) stay fixed. That is, we fix the f_i summarizing Alice's beliefs about how the average scores from the other voters for each alternative are distributed; then, in the limit as the number of voters gets large, the linear approximation is a good approximation, and Alice's best range voting ballot is an approval-style ballot as given above.

Also, while we showed that an optimal strategy can be achieved by an approval-style ballot, we cannot conclude anything about whether such a ballot is sincere or not in general. In fact, as we will see in Section 3.6, for some choices of f_i's and u_i's, the optimal ballot computed as above is not sincere.

Deriving previous assumptions from our model. We showed above that $\tau_{ik}\overline{b_i}$ is the approximate probability that adding a normalized score of b_i to A_i makes A_i the winner when otherwise A_k would be the winner. Thus, we have formally derived under general conditions the assumption used by Smith [52] that the probability of an individual voter's ballot breaking a pairwise tie for winner is proportional to the difference in her scores for the two alternatives.

The assumption that Smith used that ties among three or more candidates can be ignored can also be derived from our general model. In the expression for the utility of a ballot, terms involving three-way ties are captured by second order partial derivatives. Applying the definition of differentiability as in Equation 3.15 we see that higher order partial derivative terms can be ignored (as the number of voters approaches ∞) because they add up to $o(1/n)$.

The fact that three-way tie probabilities are not negligible in Laslier's model does not contradict the previous observations. In the trembling ballot model, the probability density functions f_i are not fixed as n varies, so we cannot apply the same analysis.

Deriving mean-based thresholding from our model. It is straightforward to see that when all pairwise ties are assumed to be equally probable, then τ_{ik} is the same for all pairs i, k, so in the limit as $n \to \infty$, an individual voter's best strategy is to give a score of 1 to those alternatives A_i for which $\Sigma_{k \neq i}(u_i - u_k) > 0$. This is exactly the same condition as mean-based thresholding.

3.6 Beta Distributions

In this section, we propose beta distributions as a somewhat natural and realistic model for a voter's subjective information about the other votes. Beta distributions are a common choice for a Bayesian prior distribution [23]. To the best of our knowledge, beta distributions have not been considered in the context of range and approval voting.

The beta distribution is a continuous probability distribution over $[0, 1]$, parameterized by two "shape" parameters, α and β.

If X is drawn from the beta distribution with parameters α and β (denoted $\text{Beta}(\alpha, \beta)$), then its density function is

$$f(x; \alpha, \beta) = \frac{x^{\alpha-1}(1-x)^{\beta-1}}{B(\alpha, \beta)} \tag{3.21}$$

where

$$B(\alpha, \beta) = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)} \tag{3.22}$$

and

$$\Gamma(n) = (n-1)! \tag{3.23}$$

for integer values of n.

The cumulative distribution function $F(x; \alpha, \beta)$ is given by

$$F(x; \alpha, \beta) = \frac{B(x; \alpha, \beta)}{B(\alpha, \beta)} \tag{3.24}$$

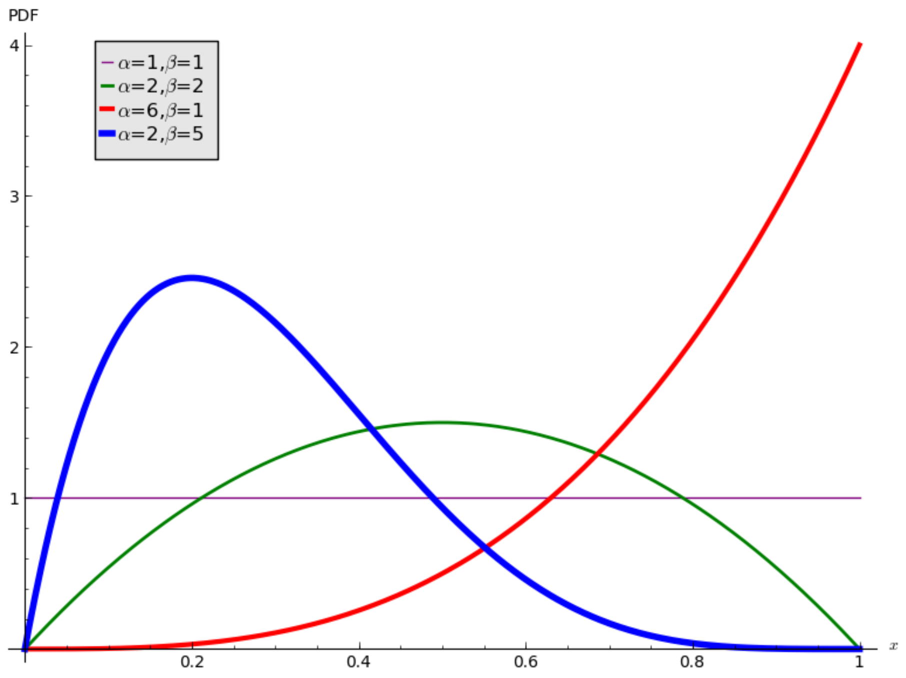

Figure 3-1: Beta probability density function (PDF) for several values of α and β.

where

$$B(x;\alpha,\beta) = \int_0^x t^{\alpha-1}(1-t)^{\beta-1}\,dt. \tag{3.25}$$

The beta distribution Beta(α,β) represents the Bayesian posterior distribution of a probability p after observing $\alpha-1$ successes and $\beta-1$ failures in $\alpha+\beta-2$ Bernoulli trials with unknown probability p, with a uniform prior (see, e.g., Evans et al. [23], Chapter 5). The distribution Beta(1,1) is the uniform distribution.

We propose using a beta distribution to represent the distribution of the average score (between 0 and 1) given to an alternative by the other votes, or in an approval voting setting, the fraction of voters approving of an alternative.

In an approval voting setting, we could imagine that pre-election polls announce the number of voters approving of each candidate out of some random sample of voters. Thus, if in a sample of 20 voters there were 5 voters approving of alternative A_1, we could model f_1 and F_1 using the beta distribution with $\alpha = 6$ and $\beta = 16$.

The integrals can be evaluated explicitly, e.g. using Sage, for fixed values of m, α, and β.

Figure 3-1 shows beta PDFs for some sample values of α and β.

Example 3.6.1. For example, suppose we are using approval voting. Let $m = 4$ and suppose a pre-election poll was conducted on a sample of 5 voters. In this sample, 2 voters approved A_1, 1 voter approved A_2, 4 voters approved A_3, and 5 voters approved A_4. Then we would model Alice's information about the how the other voters will vote as follows: $f_1(x) = f(x; 3, 4)$, $f_2(x) = f(x; 2, 5)$, $f_3(x) = f(x; 5, 2)$, and $f_4(x) = f(x; 6, 1)$. Suppose Alice's utilities are $u_1 = 0$, $u_2 = 5$, $u_3 = 7$, and $u_4 = 10$. Then, computing the c_i's from Section 3.5 using Sage, we get $c_1 = -1.47$, $c_2 = -0.101$, $c_3 = -4.33$, and $c_4 = 5.90$. Therefore, in the limit as the number of voters grows large, Alice's best strategy is to approve of only A_4.

When modeling a voter's information using beta distributions, the optimal strategy is not always sincere, as shown in the following example.

Example 3.6.2. Suppose $m = 6$, and Alice's information about the other voters is modeled by $f_1(x) = f(x; 15, 10)$, $f_2(x) = f(x; 13, 8)$, $f_3(x) = f(x; 13, 19)$, $f_4(x) = f(x; 3, 4)$, $f_5(x) = f(x; 6, 2)$, $f_6(x) = f(x; 15, 16)$. Suppose Alice's utilities are $\mathbf{u} = (0, 5, 9, 10, 16, 22)$. Then we get $c_1 = -15.4$, $c_2 = -9.72$, $c_3 = 0.0206$, $c_4 = -0.0347$, $c_5 = 22.1$, $c_6 = 3.13$, indicating that the optimal strategy in the limit as $n \to \infty$ is to approve alternatives A_3, A_5, A_6 and disapprove alternatives A_1, A_2, A_4. This is not a sincere strategy for Alice, since it approves of A_3 and disapproves of A_4, when Alice prefers A_4 to A_3 (her utilities are $u_3 = 9$ and $u_4 = 10$).

By examining the tie densities τ_{ik}, we can roughly interpret the situation as follows: a pairwise tie between A_3 and A_4 is much more likely than any other pairwise tie, and in that pairwise tie Alice would like A_4 to win; therefore, she should disapprove A_3 and approve A_4.

3.7 Conclusions and Open Questions

In this chapter, we studied strategic voting in the context of range voting, and asked when a voter's best strategy can be achieved by an approval vote. We reviewed known results on strategic range voting and approval voting in different models. We then proposed a general way of modeling a range voter's information about other votes – using an independent probability distribution for the average score each alternative receives from other voters. In this model, we showed that in the limit as the number of voters grows large, a voter's optimal strategy (computable by evaluating integrals using, e.g., Sage) can be achieved by an approval-style vote. This result may be interpreted to mean that, in some sense, when the number of voters is large, the added expressiveness of range voting over approval voting is not usually useful; in fact, using the added expressiveness of range voting would lower a voter's expected utility for the election outcome.

More concretely, we proposed beta distributions as a plausible way to model a voter's information, especially in an approval voting setting when there is pre-election poll data from which a voter may form her opinions about how others will actually vote. Even without pre-election approval polls, a voter may be able to provide an α and a β representing her subjective prior information for each alternative. We observed that with beta distributions, the optimal range or approval vote even in the limit of a large number of voters is not always sincere. An interesting open question is to formulate a set of clean conditions under which the optimal range or approval vote is or is not sincere.

Chapter 4

Statistical Robustness of Voting Rules

4.1 Introduction

It is well known that polling a sample of voters before an election may yield useful information about the likely outcome of the election, if the sample is large enough and the voters respond honestly.

It is less well known that the effectiveness of a sample in predicting an election outcome also depends on the voting rule (social choice function) used.

In this chapter, we introduce a notion of *"statistical robustness"* for voting rules. We say a voting rule is *statistically robust* if for any profile (a set of ballots) the winner of any random sample of that profile is most likely to be the same as the (most likely) winner for the complete profile. While the sample result may be "noisy" due to sample variations, if the voting rule is statistically robust the most common winner(s) for a sample will be the same as the winner(s) of the complete profile.

To coin some amusing terminology, we might say that a statistically robust voting rule is "weather-resistant"—you expect to get the same election outcome if the election day weather is sunny (when all voters show up at the polls) as you get on a rainy day (when only some fraction of the voters show up). We assume here that the chance of a voter showing up on a rainy day is independent of her preferences.

We consider the property of being statistically robust a desirable one for a voting rule, and thus consider lack of such statistical robustness somewhat of a defect in voting rules. In general, we consider a voting rule to be somewhat defective if applying the voting rule to a sample of the ballots may give misleading guidance regarding the likely winner for the entire profile.

If a voting rule is not statistically robust, then if for any reason some ballots are lost and not counted in an election, the election outcome may very likely change, even if the lost ballots are random and not maliciously chosen.

Another reason statistical robustness may be desirable is for post-election auditing. "Ballot-polling auditing" [43] attempts to confirm the result of an election by examining randomly sampled ballots (similar to an exit poll, except the audit is polling ballots, not voters), until there is strong statistical evidence that the reported outcome is correct. Ballot-polling auditing methods are currently designed for plurality voting, where each ballot that is a vote for the reported winner increases the confidence that the reported outcome is correct. However, for a voting rule that is not statistically robust, the result of a sample is not necessarily a good indication of the result of the entire election. It is unclear how ballot-polling auditing would work for voting rules that are not statistically robust.

We note that Bayesian auditing [49], a recently proposed audit method, does not restrict the voting rule to be one that is statistically robust. In a Bayesian audit, ballots are randomly selected and examined until the computed probability that the reported winner is the actual winner of the entire profile exceeds some threshold. The winning probabilities are computed by using a posterior distribution given the sample and a prior to generate likely ways of "completing" the sample to a full profile, and using the voting rule to determine the winner of each completion. Thus, there is no assumption that the winner of a sample is a good prediction of the winner of the entire profile.

Similarly, in an AI system that combines the recommendations of expert subsystems according to some aggregation rule, it may be of interest to know whether aggregating the recommendations of a sample of the experts is most likely to yield the same result as aggregating the recommendations of all experts. In some situations, some experts may have transient faults or be otherwise temporarily unavailable (in a manner independent of their recommendations) so that only a sample of recommendations is available for aggregation.

Related work. Since our definition is new, there is little or no directly related previous work. The closest work may be that of Walsh and Xia [59], who study various "lot-based" voting rules with respect to their computational resistance to strategic voting. In their terminology, a voting rule of the form "Lottery-Then-X" (a.k.a. "LotThenX") first takes a random sample of the ballots, and then applies voting rule X (where X may be plurality, Borda, etc.) to the sample. Their work is not concerned, as ours is, with the fidelity of the sample winner to the winner for the complete profile. Amar [3] proposes actual use of the "random ballot" method. Procaccia et al. [48] study a related but different notion of "robustness" that models the effect of voter errors; that is, robustness describes the resistance of a voting rule to some number of changes or "faults" in the votes. Another line of related work studies voting rules as maximum likelihood estimators [60, 17, 47], where it is assumed that there is a "true" ranking of the alternatives and the votes are noisy estimates of this ranking; then, the goal of a voting rule is to produce a ranking that is most likely to be the true ranking, based on a sample drawn from some noise model.

Our results. We define the notion of statistical robustness, with respect to three sampling methods: sampling without replacement, sampling with replacement, and binomial sampling. We then determine whether several voting rules are statistically robust. We show that plurality, veto, and random ballot are statistically robust, with respect to all three sampling methods. We show that other common voting rules – approval voting, single transferable vote (STV), Borda, Copeland, and Maximin – are not statistically robust, with respect to one or more of the above sampling methods. Furthermore, we show that any positional scoring rule whose score vector contains at least three distinct values (i.e., any positional scoring rule that is not equivalent to "approve t" for some t) is not statistically robust, with respect to sampling with or without replacement.

The rest of this chapter is organized as follows. Section 4.2 introduces notation and the voting rules we consider. We define the notion of statistical robustness for a voting rule in Section 4.3, determine whether several familiar voting rules are statistically robust in Section 4.4, and close with some discussion and open questions in Section 4.5.

4.2 Preliminaries

Ballots, Profiles, Alternatives. Assume a profile $P = (B_1, B_2, \ldots, B_n)$ containing n ballots will be used to determine a single winner from a set $\mathcal{A} = \{A_1, A_2, \ldots, A_m\}$ of m alternatives. There are various possibilities for the form of a ballot; the form of a ballot must be compatible with the voting rule used. We may view a profile as either a sequence or a multiset; it may contain repeated items (identical ballots).

Voting rules. Assume that a voting rule (social choice function) f maps profiles to a single outcome (one of the alternatives): for any profile P, $f(P)$ produces the *winner* for the profile P.

We allow f to be randomized, in order for ties to be handled reasonably. Our definition could alternatively have allowed f to output the *set* of tied winners; we prefer allowing randomization, so that f always outputs a single alternative. In our analysis, however, we do consider the set $\mathbf{ML}(f(P))$ of most likely winners for a given profile.

Thus, we say that A is a "most likely winner" of P if no other alternative is more likely to be $f(P)$. There may be several most likely winners of a profile P. For most profiles and most voting rules, however, we expect f to act deterministically, so there is a single most likely winner.

Often the social choice function f will be *neutral*—symmetric with respect to the alternatives—so that changing the names of the alternatives won't change the outcome distribution of f on any profile. While there is nothing in our definition that requires that f be neutral, we shall restrict attention in this chapter to neutral social choice functions. Thus, in particular, we will suppose that a tie-breaking rule used by any f in this chapter will not depend on the names of the alternatives; it will pick one of the tied alternatives uniformly at random.

We do assume that social-choice function f is *anonymous*—symmetric with respect to voters: reordering the ballots of a profile leaves the outcome unchanged.

We will consider the following voting rules. (For more details on voting rules, see Brams and Fishburn [14], for example.)

The following voting rules are preferential voting rules; that is, each ballot B_i gives a linear order $A_{i1} \succ A_{i2} \succ \ldots \succ A_{im}$, indicating that alternative A_{i1} is preferred to A_{i2}, which is preferred to A_{i3}, and so on. (In the rest of the chapter, we will omit the $\succ$ symbols and just write $A_{i1}A_{i2}\ldots A_{im}$, for example.)

For a preferential voting rule, a "pairwise election" between two alternatives A_i and A_j compares the number $n_{i,j}$ of ballots that rank A_i above A_j to the number $n_{j,i}$ of ballots that rank A_j to A_i. Then the difference $n_{i,j} - n_{j,i}$ is A_i's score in the pairwise election, and similarly, $n_{j,i} - n_{i,j}$ is A_j's score in the pairwise election. If $n_{i,j} - n_{j,i} > 0$, then A_i wins the pairwise election.

- A *positional scoring rule* is defined by a vector $\vec{\alpha} = \langle \alpha_1, \ldots, \alpha_m \rangle$; we assume $\alpha_i \geq \alpha_j$ for $i \leq j$. We will also assume $\alpha_i \geq 0$ for $i = 1, \ldots, m$, although this is not necessary.

  Alternative A_i gets α_j points for every ballot that ranks alternative A_i in the jth position. The winner is the alternative that receives the most points.

  Some examples of positional scoring rules are:

  Plurality: $\vec{\alpha} = \langle 1, 0, \ldots, 0 \rangle$

  Veto: $\vec{\alpha} = \langle 1, \ldots, 1, 0 \rangle$

  Borda: $\vec{\alpha} = \langle m-1, m-2, \ldots, 0 \rangle$

- *Single-transferable vote (STV)* (also known as *instant-runoff voting (IRV)*): The election proceeds in m rounds. In each round, each ballot is counted as a vote for its highest-ranked alternative that has not yet been eliminated, and the alternative with the fewest votes is eliminated. The winner of the election is the last alternative remaining.

- *Plurality with runoff*: The winner is the winner of the pairwise election between the two alternatives that receive the most first-choice votes.

- *Copeland*: The winner is an alternative that maximizes the number of alternatives it beats in pairwise elections.

- *Maximin*: The winner is an alternative whose lowest score in any pairwise election against another alternative is the greatest among all the alternatives.

We also consider the following voting rules that are not preferential:

- *Score voting* (also known as *range voting*): Each allowable ballot type is associated with a vector that specifies a score for each alternative. The winner is the alternative that maximizes its total score.

- *Approval* [12, 41]: Each ballot gives a score of 1 or 0 to each alternative. The winner is an alternative whose total score is maximized.

- *Random ballot* [32] (also known as *random dictator*): A single ballot is selected uniformly at random from the profile, and the alternative named on the selected ballot is the winner of the election. Note that random ballot may also be thought of as a preferential voting rule, in which a random ballot is selected, and the alternative ranked first on that selected ballot is the winner of the election.

4.3 Sampling and Statistical Robustness

Sampling. The profile P is the universe from which the sample will be drawn.

We define a *sampling process* to be a randomized function G that takes as input a profile P of size n and an integer parameter k ($1 \leq k \leq n$) and produces as output a sample S of P of expected size k, where S is a subset (or sub-multiset) of P.

We consider three kinds of sampling:

- *Sampling without replacement.* Here $G_{WOR}(P, k)$ produces a set S of size exactly k chosen uniformly without replacement from P.
- *Sampling with replacement.* Here $G_{WR}(P, k)$ produces a multiset S of size exactly k chosen uniformly with replacement from P.
- *Binomial sampling.* Here $G_{BIN}(P, k)$ produces a sample S of expected size k by including each ballot in P in the sample S independently with probability $p = k/n$.

We write $f(G(P, k))$ to denote the output of the voting rule f on the sample output by $G(P, k)$; note that f may be randomized, to break ties, for example.

Statistically Robust Voting Rules. We now give our main definitions.

Definition 4.3.1. If X is a discrete random variable (or more generally, some function whose range is a finite set), we let $\mathbf{ML}(X)$ denote the set of values that X takes with maximum probability. That is,

$$\mathbf{ML}(X) = \{x \mid \Pr(X = x) \text{ is maximum}\}$$

denotes the set of "most likely" possibilities for the value of X.

For any (possibly randomized) voting rule f and profile P, we will call $f(P)$ a random variable, even though it is not real-valued (it outputs an alternative A_i). Then, $\mathbf{ML}(f(P))$ contains the "most likely winner(s)" for voting rule f and profile P; typically this will contain just a single alternative. Similarly, $\mathbf{ML}(f(G(P, k)))$ contains the most likely winner(s) of a sample of expected size k. Note that $\mathbf{ML}(f(P))$ involves randomization only within f (if any, presumably to break ties), whereas $\mathbf{ML}(f(G(P, k)))$ also involves the randomization of sampling by G.

Definition 4.3.2. We say that a social choice function f is *statistically robust* for sampling rule G if for any profile P of size n and for any sample size $k \in \{1, 2, ..., n\}$,

$$\mathbf{ML}(f(G(P, k))) = \mathbf{ML}(f(P)) \ .$$

That is, an alternative is a most likely winner for a sample of size k if and only if it is a most likely winner for the entire profile P.

We will also sometimes parameterize the definition by the sample size k and talk about "k-statistical robustness".

Definition 4.3.3. We say that a social choice function f is *k-statistically robust* for sampling rule G if for any profile P of size $n > k$,

$$\mathbf{ML}(f(G(P, k))) = \mathbf{ML}(f(P)) \ .$$

When we talk about statistical robustness without reference to a specific sample size, we mean statistical robustness for *all* sample sizes.

Note that these definitions work smoothly with ties: if the original profile P was tied (i.e., there is more than one most likely winner of P), then the definition requires that all most likely winners of P have maximum probability of being a winner in a sample (and that no other alternatives will have such maximum probability).

Having a statistically robust voting rule is something like having an "unbiased estimator" in classical statistics. However, we are not interested in estimating some linear combination of the individual elements (as with classical statistics), but rather in knowing which alternative is most likely (i.e., which is the winner), a computation that may be a highly nonlinear function of the ballots.

A simple plurality example. Suppose we have a plurality election with 10 votes: 6 for A_1, 3 for A_2, and 1 for A_3. We try all three sampling methods, all possible values of k, and see how often each alternative is a winner in 1000 trials; Figure 4-1 reports the results, illustrating the statistical robustness of plurality voting, a fact we

G_{WOR}			
k	A_1	A_2	A_3
1	594	303	103
2	625	258	117
3	727	217	56
4	794	206	0
5	838	162	0
6	868	132	0
7	920	80	0
8	1000	0	0
9	1000	0	0
10	1000	0	0

G_{WR}			
k	A_1	A_2	A_3
1	597	299	104
2	569	325	106
3	676	260	64
4	718	256	26
5	749	219	32
6	764	212	24
7	804	181	15
8	818	171	11
9	842	146	12
10	847	150	3

G_{BIN}			
k	A_1	A_2	A_3
1	507	315	178
2	619	277	104
3	698	235	67
4	763	212	25
5	822	161	17
6	879	117	4
7	930	70	0
8	973	27	0
9	993	7	0
10	1000	0	0

Figure 4-1: Plurality voting with three sampling schemes on a profile P with ten votes: 6 for A_1, 3 for A_2, and 1 for A_3. 1000 trials were run for each sample size, with expected sample sizes running from $k = 1$ to $k = 10$. The entry indicates how many trials each alternative won, with ties broken by uniform random selection. (The perhaps surprisingly large value for A_3 of 178 for G_{BIN} results from the likelihood of an empty sample when $k = 1$; such ties are broken randomly.) Note that $ML(G(P, k)) = A_1$ for all three sampling methods G and all sample sizes k.

prove in Section 4.4.4.

We will show that plurality is statistically robust under all three sampling methods.

We will use the fact that statistical robustness for sampling without replacement implies statistical robustness for binomial sampling.

Theorem 4.3.4. *If a voting rule f is statistically robust for sampling without replacement, then f is statistically robust under binomial sampling.*

Proof. The probability that an alternative A_i wins in a sample produced by binomial sampling is the sum over all possible sample sizes k of the probability that the sample size is k times the probability that A_i wins in a uniform random sample of size k.

When binomial sampling returns an empty sample, then, with a neutral tie-breaking rule, every alternative is equally likely to be the winner. For non-empty samples, by the assumption of statistical robustness for sampling without replacement, for any $k > 0$, $f(P)$ is the most likely winner of a uniform random sample of size k.

Therefore, $f(P)$ is the most likely winner of a sample produced by binomial sampling for any positive probability p. □

4.4 Statistical Robustness of Various Voting Rules

In this section, we analyze whether various voting rules are statistically robust.

4.4.1 Unanimous Preferential Voting Rules

It turns out that for most preferential voting rules f, if f is 1-statistically robust for sampling with or without replacement, then f must be the plurality voting rule (or random ballot).

The following theorem was observed by the anonymous reviewers of AAAI-12.

Theorem 4.4.1. *Suppose f is a preferential voting rule with the following property, which we call* single unanimity*: if P consists of a single ballot (linear order) $A_{i_1} \dots A_{i_m}$, then $f(P) = A_{i_1}$. Then if f is* 1*-statistically robust for sampling with or without replacement, then for any profile P the set of most likely winners* $\mathbf{ML}(f(P))$ *is equal to the set of plurality winners* $\mathbf{ML}(plur(P))$*, where $plur$ denotes the plurality voting rule.*

Proof. If f is 1-statistically robust for sampling with or without replacement, then for any profile P, $\mathbf{ML}(f(G(P,1))) = \mathbf{ML}(f(P))$, where G is G_{WOR} or G_{WR}. For any sample of P of size 1, the winner of that sample will be the alternative listed at the top of that ballot. Therefore, the most likely winner of a sample of size 1 is the alternative that is listed at the top of the ballot the most times, which is by definition the plurality winner. In the case of a plurality tie, the set of most likely winners of a sample of size 1 is equal to the set of most likely plurality winners (the alternatives that are tied for the most first-choice votes). Thus, $\mathbf{ML}(f(G(P,1)) = \mathbf{ML}(plur(P))$. Therefore, $\mathbf{ML}(f(P)) = \mathbf{ML}(plur(P))$. □

Note that the standard property of *unanimity* is sufficient (but not necessary) for the single unanimity property used above. (A voting rule f is unanimous if, when P

consists of ballots that are all the same linear order, $f(P)$ selects the alternative at the top of that linear order as the winner.)

All of the preferential voting rules listed in Section 4.2, except veto (and any other positional scoring rule for which $\alpha_1 = \alpha_2$), are unanimous. (Veto and other positional scoring rules for which $\alpha_1 = \alpha_2$ are not unanimous because for a profile consisting of a single ballot, there will be a tie between at least the first two alternatives listed on that ballot, so the winner will not necessarily be the first alternative listed on the ballot.)

Thus, we have the following corollary.

Corollary 1. The following voting rules are not 1-statistically robust: STV, plurality with runoff, Copeland, Maximin, and Borda (and any positional scoring rule defined by $\langle \alpha_1, \ldots, \alpha_m \rangle$ for which $\alpha_1 > \alpha_2$).

Note that Theorem 4.4.1 says that the most likely winner of P under f is the plurality winner of P. This is slightly different from saying that f is the same as plurality. For example, f could be the random ballot rule; the most likely winner of a profile P under the random ballot method is the same as the plurality winner of P. Or, f could be even be some kind of "hybrid" that sometimes performs plurality and other times performs random ballot. For practical purposes, we will interpret Theorem 4.4.1 to say that any (singly) unanimous preferential voting rule that is 1-statistically robust for sampling with and without replacement must be either plurality or random ballot.

However, the theorem does not actually say whether plurality or random ballot themselves are statistically robust. We will prove that plurality and random ballot are in fact statistically robust. The theorem covers statistical robustness (for samples of size 1) for sampling with and without replacement, but not binomial sampling. In the following subsections we will give some non-robustness results for various unanimous preferential voting rules under binomial sampling.

4.4.2 Random Ballot

Theorem 4.4.2. *The random ballot method is statistically robust for each of the sampling methods G_{WR}, G_{WOR}, and G_{BIN}.*

Proof. When the random ballot method is applied to the entire profile or to a sample obtained using any of the three sampling methods, each ballot is equally likely to be chosen as the one to name the winner. $\square$

4.4.3 Score Voting

Theorem 4.4.3. *Score voting is not statistically robust for any of the three sampling methods G_{WR},G_{WOR}, and G_{BIN}.*

Proof. By means of a counterexample. Consider the following profile:

(1) $A_1 : 100, \quad A_2 : 0$

(99) $A_1 : 0, \quad A_2 : 1$

There is one vote that gives scores of 100 for A_1 and 0 for A_2, and 99 votes that gives scores of 0 for A_1 and 1 for A_2. A_1 wins the complete profile, since it has a total score of 100, while A_2 has a total score of 99.

Under binomial sampling with probability p, A_1 wins with probability about p — that is, with about the probability A_1's vote is included in the sample. (The probability is not exactly p because the binomial sampling may produce an empty sample, in which case A_1 and A_2 will be equally likely to be selected as the winner.)

For $p < 1/2$, A_2 wins more than half the time; thus score voting is not robust under binomial sampling.

Similarly, for sampling with or without replacement for small sample sizes k, the probability that the one ballot with a score of 100 for A_1 will be included in the sample will be small, so A_2 will be more likely to win the sample. Thus, score voting is not robust under sampling with or without replacement. $\square$

4.4.4 Plurality

Throughout this section, we let n_i denote the number of votes alternative A_i receives, with $\sum_i n_i = n$.

We prove that plurality voting is statistically robust for all three sampling methods.

Theorem 4.4.4. *Plurality voting is statistically robust, with sampling without replacement.*

Proof. Assume $n_1 > n_2 \geq \ldots \geq n_m$, so A_1 is the unique winner of the complete profile. (The proof below can easily be adapted to show that plurality is statistically robust when the complete profile has a tie for the winner.)

Let $K = (k_1, k_2, ..., k_m)$ denote the number of votes for the various alternatives within the sample of size k.

Let $\binom{a}{b}$ denote the binomial coefficient "a choose b", equal to $a!/(b!(a-b)!)$. There are $\binom{n}{k}$ ways to choose a sample of size k from the profile of size n.

The probability of a given configuration K is equal to

$$\Pr(K) = (\prod_{i=1}^{m} \binom{n_i}{k_i})/\binom{n}{k}.$$

Let $\gamma(i)$ denote the probability that A_i wins the election, and let $\gamma(i, k_{max})$ denote the probability that A_i receives k_{max} votes and wins the election.

Then $\gamma(i) = \sum_{k_{max}} \gamma(i, k_{max})$, and $\gamma(i, k_{max}) = \sum_{K \in \mathcal{K}} \Pr(K)/\text{Tied}(K)$, where $\mathcal{K}$ is the set of configurations K such that $k_i = k_{max}$ and $k_j \leq k_{max}$ for all $j \neq i$, and $\text{Tied}(K)$ is the number of alternatives tied for the maximum score in K. Note that $\text{Tied}(K)$ is the total number of tied alternatives; typically, $\text{Tied}(K)$ will be 1 (when there is a unique winner and no tie). The above equation depends on the tie-breaking rule being neutral.

For any k_{max}, consider now a particular configuration K used in computing $\gamma(1, k_{max})$: $K = (k_1, k_2, ..., k_m)$, where $k_1 = k_{max}$ and $k_i \leq k_{max}$ for $i > 1$. Then $\Pr(K) = (\prod_{i=1}^{m} \binom{n_i}{k_i})/\binom{n}{k}$.

Now consider the corresponding configuration K' used in computing $\gamma(2, k_{max})$, where k_1 and k_2 are switched: $K' = (k_2, k_1, k_3, ..., k_m)$. Then
$\Pr(K') = (\binom{n_1}{k_2}\binom{n_2}{k_1}\prod_{i=3}^{m}\binom{n_i}{k_i})/\binom{n}{k}$.

By Lemma 4.4.5 below, we have $\Pr(K) > \Pr(K')$.

Each configuration K' used in computing $\gamma(2, k_{max})$ has such a corresponding configuration K used in computing $\gamma(1, k_{max})$. Thus, $\gamma(1, k_{max}) > \gamma(2, k_{max})$.

Since $\gamma(1, k_{max}) > \gamma(2, k_{max})$ for any k_{max}, we have that $\gamma(1) > \gamma(2)$; that is, A_1 is more likely to be the winner of the sample than A_2.

By a similar argument, for every $i > 1$, $\gamma(1, k_{max}) \geq \gamma(i, k_{max})$ for any k_{max}, so $\gamma(1) > \gamma(i)$. Therefore, A_1 is the most likely to win the sample. □

Lemma 4.4.5. *If $n_1 > n_2$, $k_1 > k_2$, $n_1 \geq k_1$, and $n_2 \geq k_2$, then $\binom{n_1}{k_1}\binom{n_2}{k_2} > \binom{n_1}{k_2}\binom{n_2}{k_1}$.*

Proof. We wish to show that

$$\binom{n_1}{k_1}\binom{n_2}{k_2} > \binom{n_1}{k_2}\binom{n_2}{k_1}. \tag{4.1}$$

If $n_2 < k_1$, then $\binom{n_1}{k_2}\binom{n_2}{k_1} = 0$, so (4.1) is trivially true.

If $n_2 \geq k_1$, then we can rewrite (4.1) as $\binom{n_1}{k_1}/\binom{n_2}{k_1} > \binom{n_1}{k_2}/\binom{n_2}{k_2}$. So it suffices to show that for $n_1 > n_2$, $\binom{n_1}{k}/\binom{n_2}{k}$ is increasing with k, which is easily verified. □

Theorem 4.4.6. *Plurality voting is statistically robust, under binomial sampling.*

Proof. Follows from Theorems 4.4.4 and 4.3.4. □

Theorem 4.4.7. *Plurality is statistically robust, under sampling with replacement.*

Proof. The proof follows the same structure as for sampling without replacement. Again, assume $n_1 > n_2 \geq \ldots \geq n_m$.

For each configuration K used in computing $\gamma(1, k_{max})$ and the corresponding configuration K' used in computing $\gamma(2, k_{max})$, we show that $\Pr(K) > \Pr(K')$.

Under sampling with replacement, the probability of a configuration $K = (k_1, \ldots, k_m)$ is equal to

$$\Pr(K) = \binom{k}{k_1, \ldots, k_m}\prod_{i=1}^{m}\left(\frac{n_i}{n}\right)^{k_i},$$

where $\binom{k}{k_1,\ldots,k_m}$ denotes the multinomial coefficient, equal to $k!/(k_1!\ldots k_m!)$.

For any configuration K used in computing $\gamma(1, k_{max})$, consider the corresponding configuration K', obtained by swapping k_1 and k_2, used in computing in $\gamma(2, k_{max})$: $K' = (k_2, k_1, k_3, \ldots, k_m)$. Then

$$\Pr(K') = \binom{k}{k_1, \ldots, k_m} \left(\frac{n_2}{n}\right)^{k_1} \left(\frac{n_1}{n}\right)^{k_2} \prod_{i=3}^{m} \left(\frac{n_i}{n}\right)^{k_i}.$$

So $\Pr(K) = (n_1/n_2)^{(k_1-k_2)} \Pr(K')$. If $n_1 > n_2$ and $k_1 > k_2$, then $\Pr(K) > \Pr(K')$. If $n_1 > n_2$ and $k_1 = k_2$, then $\Pr(K) = \Pr(K')$. Thus, $\gamma(1, k_{max}) > \gamma(2, k_{max})$ for every k_{max}, and therefore, A_1 is more likely than A_2 to win a sample without replacement.

By a similar argument, for every $i > 1$, $\gamma(1, k_{max}) \geq \gamma(i, k_{max})$ for any k_{max}, so $\gamma(1) > \gamma(i)$. Therefore, A_1 is most likely to win the sample. □

4.4.5 Veto

Theorem 4.4.8. *Veto is statistically robust.*

Proof. Each ballot can be thought of as a vote for the least-preferred alternative; the winner is the alternative who receives the *fewest* votes.

For plurality, we showed that the alternative who receives the *most* votes in the complete profile is the most likely to receive the most votes for a random sample. By symmetry, the same arguments can be used to show that the alternative who receives the *fewest* votes in the complete profile is the most likely to receive the fewest votes in a random sample. Thus, the winner of a veto election is the most likely to win in a random sample. □

4.4.6 Approval Voting

We had conjectured that statistical robustness would hold for approval voting, thus distinguishing it from voting rules with more complex behavior, such as STV. Somewhat surprisingly, approval voting is not statistically robust.

Theorem 4.4.9. *Approval voting is* not *k-statistically robust for any k, for any of the three sampling methods G_{WR}, G_{WOR}, G_{BIN}.*

Proof. Proof by counterexample. Consider the following profile P:

(r) $\{A_1\}$

(r) $\{A_2, A_3\}$

There are 3 alternatives A_1, A_2, A_3. There are r ballots that approve of A_1 only and r ballots that approve of A_2 and A_3. Each alternative receives r votes, so each wins the election with probability 1/3.

For a sample of any size k, we can express the sample as (k_1, k_2), where k_1 is the number of ballots for A_1 and k_2 is the number of ballots for $\{A_2, A_3\}$, and $k_1+k_2 = k$. If $k_1 = k_2$, then all three alternatives A_1, A_2, and A_3 are equally likely to win. If $k_1 > k_2$, then A_1 wins, and if $k_2 > k_1$, then A_2 and A_3 each win with probability 1/2. Since the profile P contains an equal number of ballots of the two types, the probability that $k_1 > k_2$ is equal to the probability that $k_2 > k_1$. Therefore, A_1 is more likely than either A_2 or A_3 to win. Thus, approval voting is not k-statistically robust for any k under sampling with or without replacement.

Similarly, for binomial sampling for any probability p, although the sample size k is not fixed, we can again write any sample as (k_1, k_2), and using the same argument as above, we have that A_1 is more likely than either A_2 or A_3 to win. Thus, approval voting is not robust under binomial sampling. □

Intuitively, the non-robustness in the above example comes from the correlations in the votes for the various alternatives. Whenever A_2 gets a vote, A_3 also gets a vote, so A_2 can only ever win in a tie with A_3, while A_1 can win outright.

Note that the example above also shows that for approval voting there does not even exist a threshold τ such that for any sample of size k at least τ, approval voting is k-statistically robust

4.4.7 Plurality with Runoff

By Theorem 4.4.1, we know that plurality with runoff is not 1-statistically robust for sampling with or without replacement. Here we will show that plurality with runoff is not statistically robust for binomial sampling for any probability p.

Theorem 4.4.10. *Plurality with runoff is not statistically robust for binomial sampling for any probability p.*

Proof. Proof by counterexample. Choose r to be a large integer, and consider the following profile P:

(r)	$A_1\ A_3\ A_2$
(r)	$A_2\ A_3\ A_1$
$(r-1)$	$A_3\ A_1\ A_2$

In this profile, A_1 and A_2 have the most first-choice votes, so the runoff is between A_1 and A_2. In the pairwise election, A_1 receives $2r-1$ votes, while A_2 receives r votes, so A_1 wins.

Suppose we perform binomial sampling with probability p. Let n_i be the number of ballots in the sample that list A_i first. Each n_i is a binomial random variable, with mean rp for A_1 and A_2, and $(r-1)p$ for A_3.

Note that n_1 and n_2 are identically distributed, and for any fixed p, as r gets large, n_3 becomes distributed nearly identically to n_1 and n_2.

For simplicity, assume that two of the alternatives strictly have the most first-choice votes (i.e., there is no tie that needs to be broken to determine which two alternatives go into the runoff). (Since the n_i's are nearly identically distributed for large r, by symmetry the following arguments can be easily adapted to the case of ties.)

If n_1 is the minimum of the n_i's, then the runoff is between A_2 and A_3. A_2 receives n_2 votes and A_3 receives $n_3 + n_1$ votes, so A_3 is likely to win. Let p_{runoff} denote A_3's probability of winning in the case of a runoff between A_2 and A_3.

If n_2 is the minimum of the n_i's, then the runoff is between A_1 and A_3. A_1 receives n_1 votes and A_3 receives n_3+n_2 votes, so A_3 is likely to win. (A_3 wins with probability p_{runoff}.)

If n_3 is the minimum of the n_i's, then the runoff is between A_1 and A_2. A_1 receives $n_1 + n_3$ votes, and A_2 receives n_2 votes, so A_1 is likely to win. When r is large so the n_i's are distributed nearly identically, A_1 wins with probability about p_{runoff}.

For large r, since the three possible runoffs are almost equally likely, A_3 is about twice as likely as A_1 to win when the sample is selected using binomial sampling with any fixed probability p. □

4.4.8 Single Transferable Vote (STV)

By Theorem 4.4.1, we know that STV is not 1-statistically robust for sampling with or without replacement.

Here we show that STV is not statistically robust for binomial sampling.

Theorem 4.4.11. *STV is not statistically robust for binomial sampling for any probability p.*

Proof. Plurality with runoff and STV are identical when there are three (or fewer) candidates, so the counterexample given in the proof of Theorem 4.4.10 is also a counterexample for STV, showing that STV is not statistically robust for binomial sampling for any probability p.

Here, we give another counterexample, for an arbitrary number of candidates m.

We construct a profile for which the winner is very unlikely to be the winner in any smaller sample. Choose m (the number of alternatives) and r (a "replication factor") both as large integers.

The profile will consist of $n = mr$ ballots:

$(r+1)$	$A_1\ A_m\ \dots$
(r)	$A_2\ A_m\ A_1\ \dots$
(r)	$A_3\ A_m\ A_1\ \dots$
...	
(r)	$A_{m-1}\ A_m\ A_1\ \dots$
$(r-1)$	$A_m\ A_1\ \dots$

where the specified alternatives appear at the front of the ballots, and "…" indicates that the order of the other lower-ranked alternatives is irrelevant.

In this profile, A_m is eliminated first, then $A_2, \ldots, A_{m-1}$ in some order, until A_1 wins.

Suppose now that binomial sampling is performed, with each ballot retained with probability p. Let n_i be the number of ballots retained that list A_i first. Each n_i is a binomial random variable with mean $(r+1)p$ (for A_1), rp (for $A_2, \ldots, A_{m-1}$), and $(r-1)p$ (for A_m).

Claim 1: The probability that $n_m = 0$ goes to 0, for any fixed p, as $r \to \infty$.

Claim 2: The probability that there exists an i, $1 \leq i < m$, such that $n_i < n_m$, goes to 1 as $m, r \to \infty$.

Note that as r gets large, then n_i and n_m are very nearly identically distributed, so the probability that $n_i < n_m$ goes to 1/2. As r and m get large, the probability that *some* n_i will be smaller than n_m goes to 1.

Thus, in any sample $G_{BIN}(P, k)$, we expect to see some A_i other than A_m eliminated first. Since all of A_i's votes then go to A_m, A_m will with high probability never be eliminated and will be the winner. □

4.4.9 Positional Scoring Rules

By Theorem 4.4.1, we know that any positional scoring rule defined by $\langle \alpha_1, \ldots, \alpha_m \rangle$ where $\alpha_1 > \alpha_2$ is not 1-statistically robust. Here, we consider the case where $\alpha_1 = \alpha_2$ (so the rule is not unanimous), but the score vector consists of at least three distinct values.

Theorem 4.4.12. *Let $\vec{\alpha} = \langle \alpha_1, \ldots, \alpha_m \rangle$ be any positional scoring rule with integer α_i's such that $\alpha_1 > \alpha_i > \alpha_m$ for some $1 < i < m$. Then the positional scoring rule defined by $\vec{\alpha}$ is not 1-statistically robust.*

Proof. We will show that a counterexample exists for any $\vec{\alpha}$ for which $\alpha_1 > \alpha_i > \alpha_m$ for some i.

We construct a profile as follows. Start with r copies of each of the $m!$ possible ballots, for some large r. Clearly, for this profile, all m alternatives are tied, and each alternative wins the election with equal probability, $1/m$.

We will show that this profile can be "tweaked" so that in the resulting profile, all m alternatives are again tied and each win with equal probability. However, the number of first-choice votes will no longer be equal for all alternatives, so for a sample of size 1, the alternatives will not all win with equal probability.

The "tweak" is performed as follows. Take a single ballot type b (i.e., a permutation of the alternatives) that has A_1 in position 1 on the ballot, A_2 in position i, and A_3 in position m. Consider the 6 ballot types obtained by permuting A_1, A_2, A_3 within b (while keeping the other alternatives' positions fixed). We will change the number of ballots of each of these 6 types by $\delta_1, \ldots, \delta_6$ (the δ_i's may be positive, negative, or zero).

That is, starting from a ballot type $A_1 \ldots A_2 \ldots A_3$, we will change the counts of the following 6 ballot types:

$$\begin{array}{lll}
A_1 \ldots A_2 \ldots A_3 & \text{by} & \delta_1 \\
A_1 \ldots A_3 \ldots A_2 & \text{by} & \delta_2 \\
A_2 \ldots A_1 \ldots A_3 & \text{by} & \delta_3 \\
A_2 \ldots A_3 \ldots A_1 & \text{by} & \delta_4 \\
A_3 \ldots A_1 \ldots A_2 & \text{by} & \delta_5 \\
A_3 \ldots A_2 \ldots A_1 & \text{by} & \delta_6
\end{array}$$

where the "..." parts are the same for all 6 ballot types.

In order to keep the scores of $A_4, \ldots, A_m$ unchanged, we require $\delta_1 + \ldots + \delta_6 = 0$.

Next, in order to keep the scores of $A_1, \ldots, A_3$ unchanged, we write one equation for each of the three alternatives:

$$(\delta_1 + \delta_2)\alpha_1 + (\delta_3 + \delta_5)\alpha_i + (\delta_4 + \delta_6)\alpha_m = 0,$$

$$(\delta_3 + \delta_4)\alpha_1 + (\delta_1 + \delta_6)\alpha_i + (\delta_2 + \delta_5)\alpha_m = 0,$$

$$(\delta_5 + \delta_6)\alpha_1 + (\delta_2 + \delta_4)\alpha_i + (\delta_1 + \delta_3)\alpha_m = 0.$$

Finally, to ensure that the number of first-choice votes changes (so that the probability of winning a sample of size 1 changes) for at least one of A_1, A_2, A_3, we add an additional equation, $\delta_1 + \delta_2 = 1$, for example.

The 5 equations above in 6 variables will always be satisfiable with integer δ_i's. (The δ_i's might be fractional values, but then we will be able to scale all the δ_i's by some multiplicative factor to get integer solutions.) We can choose the replication factor r to be large enough so that the numbers of each ballot type are non-negative. Thus, there always exists a counterexample to statistical robustness as long as $\alpha_1 > \alpha_i > \alpha_m$. □

4.5 Discussion and Open Questions

We have introduced and motivated a new property for voting rules, statistical robustness, and provided some results on the statistical robustness of several well-known voting rules.

Many interesting open problems and conjectures remain, some of which are given below.

It may be surprising that plurality (and its complement, veto) and random ballot are the only interesting voting rules that appear to be statistically robust. Being statistically robust seems to be a somewhat fragile property, and a small amount of nonlinearity appears to destroy it.

For example, even plurality with weighted ballots (which one might have in an expert system with different experts having different weights) is not statistically robust: this is effectively the same as score voting.

Open Problem 1. Are some voting rules k-statistically robust for large enough sample sizes k? Many of our non-robustness results are for $k = 1$. It would be of interest to determine, for each voting rule and each kind of sampling, for which values of k statistical robustness holds.

Note that we showed for approval voting that there does not exist a threshold k' for which k-statistical robustness holds for $k > k'$, for any of the three sampling methods.

Conjecture 1. Show that plurality and veto are the only statistically robust voting rules among those where each ballot "approves t" for some fixed t, for all three sampling methods.

Conjecture 2. Show that a voting rule cannot be statistically robust if the number of distinct meaningfully-different ballot types is greater than m, the number of alternatives, for all three sampling methods.

Conjecture 3. Show that a voting rule cannot be robust if there are two profiles P and P' that have the same total score vectors, but which generate different distributions when sampled, for all three sampling methods.

Open Problem 2. Determine how best to use the information contained in a sample of ballots to predict the overall election outcome (where the entire profile is unknown), for each voting rule that is not statistically robust. (There may be something better to do than merely applying the voting rule to the sample.) This type of prediction may be useful for non-Bayesian post-election audit methods.

Open Problem 3. The lottery-based voting rules studied by Walsh and Xia [59] of the form "LotThenX" seem plausible alternatives for statistically robust voting rules, since their first step is to take a sample of the profile. Determine which, if any, LotThenX voting rules are statistically robust.

Bibliography

[1] Martin Abadi and Phillip Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). *Journal of Cryptology*, 15(2):103–127, 2002.

[2] Alfred V. Aho and Margaret J. Corasick. Efficient string matching: An aid to bibliographic search. *Comm. ACM*, 18(6):333–340, June 1975.

[3] Akhil Reed Amar. Choosing representatives by lottery voting. *Yale Law Journal*, 93:1283–1308, June 1984.

[4] Mihir Bellare, Anand Desai, Eron Jokipii, and Phillip Rogaway. A concrete security treatment of symmetric encryption: Analysis of the DES modes of operation. In *Proceedings of the 38th Annual IEEE Symposium on Foundations of Computer Science (FOCS '97)*, pages 394–403. IEEE Press, 1997.

[5] Mihir Bellare and Chanathip Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In Tatsuaki Okamoto, editor, *Advances in Cryptology – ASIACRYPT '00*, volume 1976 of *Lecture Notes in Computer Science*, pages 531–545. Springer, 2000.

[6] Mihir Bellare and Phillip Rogaway. Encode-then-encipher encryption: How to exploit nonces or redundancy in plaintexts for efficient cryptography. In Tatsuaki Okamoto, editor, *Advances in Cryptology – ASIACRYPT '00*, volume 1976 of *Lecture Notes in Computer Science*, pages 317–330. Springer, 2000.

[7] Siavosh Benabbas, Rosario Gennaro, and Yevgeniy Vahlis. Verifiable delegation of computation over large datasets. In Phillip Rogaway, editor, *Advances in Cryptology - CRYPTO '11*, volume 6841 of *Lecture Notes in Computer Science*, pages 111–131. Springer, 2011.

[8] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In Yuval Ishai, editor, *Theory of Cryptography Conference (TCC '11)*, volume 6597 of *Lecture Notes in Computer Science*, pages 253–273. Springer, 2011.

[9] Robert S. Boyer and J. Strother Moore. A fast string searching algorithm. *Comm. ACM*, 20(10):762–772, 1977.

[10] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. In *3rd Innovations in Theoretical Computer Science Conference (ITCS '12)*, 2012.

[11] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In *52nd Annual IEEE Symposium on. Foundations of Computer Science (FOCS '11)*, 2011.

[12] Steven J. Brams and Peter C. Fishburn. Approval voting. *American Political Science Review*, 72(3):831–847, 1978.

[13] Steven J. Brams and Peter C. Fishburn. *Approval Voting*. Birkhauser, 1983.

[14] Steven J. Brams and Peter C. Fishburn. Voting procedures. In Kenneth J. Arrow, Amartya K. Sen, and Kotaro Suzumura, editors, *Handbook of Social Choice and Welfare*, chapter 4. North-Holland, 2002.

[15] Melissa Chase and Seny Kamara. Structured encryption and controlled disclosure. In Masayuki Abe, editor, *Advances in Cryptology – ASIACRYPT '10*, volume 6477 of *Lecture Notes in Computer Science*, pages 577–594. Springer, 2010.

[16] Kai-Min Chung, Yael Tauman Kalai, Feng-Hao Liu, and Ran Raz. Memory delegation. In Phillip Rogaway, editor, *Advances in Cryptology - CRYPTO '11*, volume 6841 of *Lecture Notes in Computer Science*, pages 151–168. Springer, 2011.

[17] Vincent Conitzer, Matthew Rognlie, and Lirong Xia. Preference functions that score rankings and maximum likelihood estimation. In *Twenty-First International Joint Conference on Artificial Intelligence (IJCAI-09)*, pages 109–115, 2009.

[18] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, 3rd edition, 2009.

[19] Reza Curtmola, Juan Garay, Seny Kamara, and Rafail Ostrovsky. Searchable symmetric encryption: Improved definitions and efficient constructions. In *ACM Conference on Computer and Communications Security (CCS '06)*, pages 79–88. ACM, 2006.

[20] Ivan Damgard, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Rei Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology - CRYPTO '12*, volume 7417 of *Lecture Notes in Computer Science*, pages 643–662. Springer, 2012.

[21] Yevgeniy Dodis. Lecture notes, introduction to cryptography. `http://www.cs.nyu.edu/courses/spring12/CSCI-GA.3210-001/lect/lecture10.pdf`.

[22] Cynthia Dwork, Moni Naor, Guy N. Rothblum, and Vinod Vaikuntanathan. How efficient can memory checking be? In *Theory of Cryptography Conference (TCC '09)*, volume 5444 of *Lecture Notes in Computer Science*, pages 503–520, 2009.

[23] Merran Evans, Nicholas Hastings, and Brian Peacock. *Statistical Distributions*. Wiley, 3rd edition, 2000.

[24] Martin Farach. Optimal suffix tree construction with large alphabets. In *Proceedings of the 38th Annual IEEE Symposium on Foundations of Computer Science (FOCS '97)*, pages 137–143. IEEE Press, 1997.

[25] Marc Fischlin. Pseudorandom function tribe ensembles based on one-way permutations: Improvements and applications. In *Advances in Cryptology – EUROCRYPT '99*.

[26] Christopher Fletcher, Marten van Dijk, and Srinivas Devadas. A secure processor architecture for encrypted computation on untrusted programs. In *ACM Scalable Trusted Computing Workshop (STC)*, 2012.

[27] Keith B. Frikken. Practical private DNA string searching and matching through efficient oblivious automata evaluation. In *23rd Annual IFIP WG 11.3 Working Conference on Data and Applications Security (DBSec '09)*, pages 81–94, 2009.

[28] Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In Tal Rabin, editor, *Advances in Cryptology CRYPTO 2010*, volume 6223 of *Lecture Notes in Computer Science*, pages 465–482. Springer, 2010.

[29] Rosario Gennaro, Carmit Hazay, and Jeffrey S. Sorensen. Text search protocols with simulation based security. In *Public Key Cryptography*, pages 332–350, 2010.

[30] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *ACM Symposium on Theory of Computing (STOC '09)*, pages 169–178. ACM Press, 2009.

[31] Craig Gentry, Shai Halevi, and Nigel P. Smart. Homomorphic evaluation of the AES circuit. In Rei Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology - CRYPTO '12*, volume 7417 of *Lecture Notes in Computer Science*, pages 850–867. Springer, 2012.

[32] Allan Gibbard. Manipulation of schemes that mix voting with chance. *Econometrica*, 45(3):665–681, April 1977.

[33] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, April 1988.

[34] Carmit Hazay and Yehuda Lindell. Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries. *J. Cryptology*, 23(3):422–456, 2010.

[35] Richard M. Karp and Michael O. Rabin. Efficient randomized pattern-matching algorithms. *IBM Journal of Research and Development*, 31(2):249–260, March 1987.

[36] J. Katz, A. Sahai, and B. Waters. Predicate encryption supporting disjunctions, polynomial equations, and inner products. In *Advances in Cryptology - EUROCRYPT 2008*, volume 4965 of *Lecture Notes in Computer Science*, pages 146–162. Springer, 2008.

[37] Jonathan Katz and Lior Malka. Secure text processing with applications to private DNA matching. In *ACM Conference on Computer and Communications Security (CCS '10)*, pages 485–492, 2010.

[38] Jonathan Katz and Moti Yung. Unforgeable encryption and chosen ciphertext secure modes of operation. In Bruce Schneier, editor, *FSE 2000*, volume 1978 of *Lecture Notes in Computer Science*, pages 284–299. Springer, 2001.

[39] Donald E. Knuth, James H. Morris Jr., and Vaughan R. Pratt. Fast pattern matching in strings. *SIAM Journal on Computing*, 6(2):323–350, 1977.

[40] Jean-François Laslier. The leader rule: A model of strategic approval voting in a large electorate. *Journal of Theoretical Politics*, 21(1):113–126, January 2009.

[41] Jean-François Laslier and M. Remzi Sanver, editors. *Handbook of Approval Voting*. Studies in Choice and Welfare. Springer, 2010.

[42] Leonid A. Levin. One-way functions and pseudorandom generators. *Combinatorica*, 7(4):357–363, 1987.

[43] Mark Lindeman and Philip B. Stark. A gentle introduction to risk-limiting audits. *IEEE Security and Privacy*, 10:42–49, 2012. See `http://statistics.berkeley.edu/~stark/Preprints/gentle12.pdf`.

[44] Payman Mohassel, Salman Niksefat, Saeed Sadeghian, and Babak Sadeghiyan. An efficient protocol for oblivious DFA evaluation and applications. IACR Cryptology ePrint Archive, `http://eprint.iacr.org/2011/434`.

[45] Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, and Sai Sheshank Burra. A new approach to practical active-secure two-party computation. In Rei Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology - CRYPTO '12*, volume 7417 of *Lecture Notes in Computer Science*, pages 681–700. Springer, 2012.

[46] Rafail Ostrovsky. *Software protection and simulation on oblivious RAMs*. PhD thesis, MIT, 1992.

[47] Ariel D. Procaccia, Sashank J. Reddi, and Nisarg Shah. A maximum likelihood approach for selecting sets of alternatives. In *UAI-12: Proc. 28th Conference on Uncertainty in Artificial Intelligence*, pages 695–704, 2012.

[48] Ariel D. Procaccia, Jeffrey S. Rosenschein, and Gal A. Kaminka. On the robustness of preference aggregation in noisy environments. In *Proceedings AAMAS'07*, pages 416–422, 2007.

[49] Ronald L. Rivest and Emily Shen. A Bayesian method for auditing elections. In J. Alex Halderman and Olivier Pereira, editors, *Proceedings 2012 EVT/WOTE Conference*, 2012.

[50] Emily Shen, Elaine Shi, and Brent Waters. Predicate privacy in encryption systems. In *Theory of Cryptography Conference (TCC '09)*, pages 457–473, Berlin, Heidelberg, 2009. Springer-Verlag.

[51] Warren Smith. Range voting optimal strategy theorem. `http://www.rangevoting.org/RVstrat7.html`.

[52] Warren Smith. Range voting strategy experiments - honesty isn't a bad strategy. `http://www.rangevoting.org/RVstrat3.html`.

[53] Warren Smith. Range voting.org – center for range voting. `http://rangevoting.org/`.

[54] Emil Stefanov, Elaine Shi, and Dawn Song. Toward practical oblivious RAM. In *Network and Distributed System Security Symposium (NDSS)*, 2012.

[55] Emil Stefanov, Marten van Dijk, Ari Juels, and Alina Oprea. Iris: A scalable cloud file system with efficient integrity checks. In *Annual Computer Security Applications Conference (ACSAC)*, pages 229–238, 2012.

[56] Juan Ramón Troncoso-Pastoriza, Stefan Katzenbeisser, and Mehmet Utku Celik. Privacy preserving error resilient DNA searching through oblivious automata. In *ACM Conference on Computer and Communications Security (CCS '07)*, pages 519–528, 2007.

[57] Esko Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14(3):249–260, 1995.

[58] John von Neumann and Oskar Morgenstern. *Theory of Games and Economic Behavior*. Princeton University Press, 1944.

[59] Toby Walsh and Lirong Xia. Lot-based voting rules. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS '12)*, pages 603–610, 2012.

[60] H. P. Young. Condorcet's theory of voting. *American Political Science Review*, 82(4):1231–1244, December 1988.