Privacy-Preserving Regular Expression Evaluation on Encrypted Data

Lei Wei

A dissertation submitted to the faculty of the University of North Carolina at Chapel Hill in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Computer Science.

Chapel Hill
2013

Approved by:

Christian Cachin

Philip Mackenzie

Fabian Monrose

Michael Reiter, Chair

Gene Tsudik

ABSTRACT

LEI WEI: Privacy-preserving Regular-Expression Evaluation on Encrypted Data
(Under the direction of Michael K. Reiter)

Motivated by the need to outsource file storage to untrusted clouds while still permitting controlled use of that data by authorized third parties, in this dissertation we present a family of protocols by which a client can evaluate a regular expression on an encrypted file stored at a server (the cloud), once authorized to do so by the file owner. We present a protocol that provably protects the privacy of the regular expression and the file contents from a malicious server and the privacy of the file contents (except for the evaluation result) from an honest-but-curious client. We then extend this protocol in two primary directions. In one direction, we develop a strengthened protocol that enables the client to detect any misbehavior of the server; in particular, the client can verify that the result of its regular-expression evaluation is based on the authentic file stored there by the data owner, and in this sense the file and evaluation result are authenticated to the client.

The second direction in which we extend our initial protocol is motivated by the vast adoption of resource-constrained mobile devices, and the fact that our protocols involve relatively intensive client-server interaction and computation on the searching client. We therefore investigate an alternative in which the client (e.g., via her mobile device) can submit her encrypted regular expression to a partially trusted proxy, which then interacts with the server hosting the encrypted data and reports the encrypted evaluation result to the client. Neither the search query nor the result is revealed to an honest-but-curious proxy or malicious server during the process. We demonstrate the practicality of the protocol by prototyping a system to perform regular-expression searches on encrypted emails and evaluate its performance using a real-world email dataset.

To my parents, and Shih.

ACKNOWLEDGEMENTS

The completion of this dissertation would not have been possible without the guidance of my advisor, Prof. Mike Reiter, whom I feel extremely fortunate to have worked with and owe all my gratitude to. Looking back 6 years ago when I accidentally stepped into this field as a newcomer, he has been instrumental in my personal development and helped shape who I am today. Throughout the years, I have been immensely impressed by and benefited from his masterful understanding of the field, endless source of wisdom, careful direction and advise, and sometimes brutal demands to lift me up to his high standards. There is still so much I can learn from him everyday, but I feel like it is time for me to carry with all these qualities learned from him going forward, hopefully to have a positive impact on others.

I would also like to thank Dr. Christian Cachin, Dr. Fabian Monrose, Dr. Philip Mackenzie and Dr. Gene Tsudik for serving on my dissertation committee. I am very grateful for all of them to take time from their busy schedules to hold meetings with me and providing invaluable feedbacks, especially considering the challenge of scheduling across 9 time zones.

Special thanks to Fabian, who is always available for encouragement, a fun chat or simply a game of foosball or pingpong to distract myself from the stress in graduate school.

I would also like to thank my friend, Hao Xu, for his generosity for providing many insightful discussions and assistance on fighting bugs in my code from time to time. I would also like to thank all my friends in the security lab, from whom I benefited through inspiring discussions, helpful critiques to my work and feedbacks that helped me improve my presentation skills.

My gratitude for my girlfriend, Shih Ku, who has accompanied and supported me throughout my time in graduate school, though not always physically close, but always close in heart.

Of course, I would not have reached to this point without the love and care from my parents. It was them who gave me all of my gifts, and who I can always count on no matter what happens.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

Outsourcing file storage to storage service providers (SSPs) and “clouds” can provide significant savings to file owners in terms of management costs and capital investments (e.g., [58]). However, because cloud storage can heighten the risk of file disclosure, prudent file owners encrypt their cloud-resident files to protect their confidentiality. This encryption introduces difficulties in managing access to these files by third parties, however. For example:

- Third-party service providers who are contracted to analyze files stored in the cloud generally cannot do so if the files are encrypted. For example, periodically “scanning” files to detect new malware, as is common today for PC platforms, cannot presently be performed on encrypted files by a third party.

- With some exceptions (see Chapter 2), third-party customers generally cannot search the files if they are encrypted. Searches on genome datasets, pharmaceutical databases, document corpora, or network logs are critical for research in various fields, but the privacy constraints of these datasets may mandate their encryption, particularly when stored in the cloud.

These difficulties are compounded when the third party views its queries on the files to be sensitive, as well. New malware signatures may be sensitive since releasing them enables attackers to design malware to evade them (e.g., [75]). Customers of datasets in numerous domains (e.g., pharmaceutical research) may view their research interests, and hence their queries, as private.

As a step toward resolving this tension among file protection, search access by authorized third parties, and privacy for third-party queries, in this dissertation we develop a family of protocols by which a third-party (called the “client”) can perform private searches on encrypted files (stored at the “server”), once it is authorized to do so by the file owner, with various security properties. The overall framework is demonstrated in Fig. 1.1. The type of searches that our protocols enable is

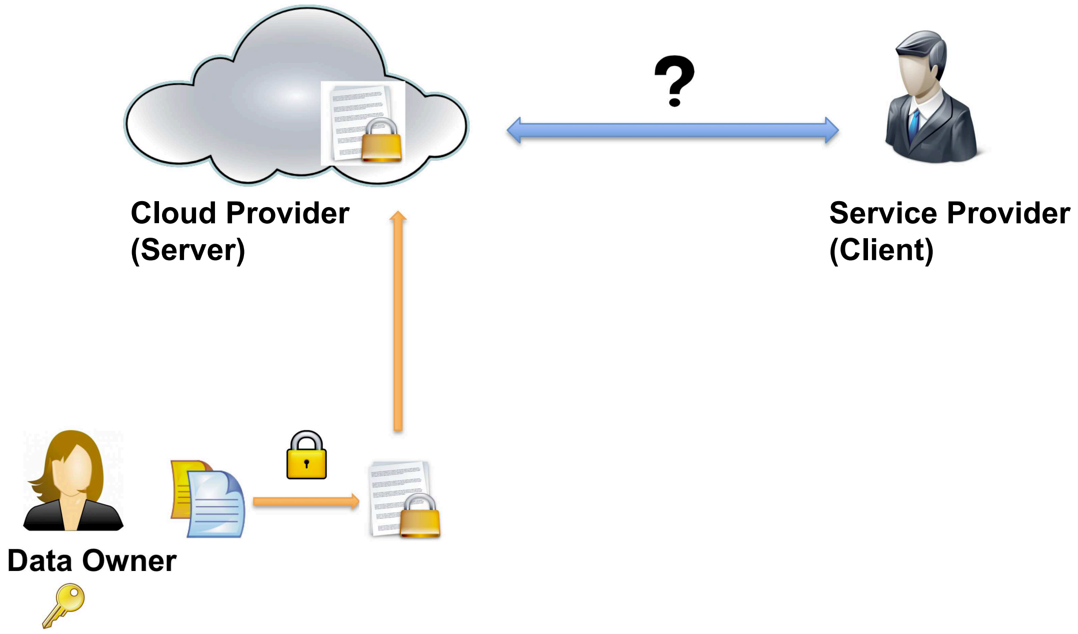


Figure 1.1: Overall framework: data owner stores encrypted data at the server, with which an authorized client performs private searches.

motivated by the scenarios above, which in many cases involve pattern matching a file against one or more regular expressions. Regular-expression searches are a widely adopted search primitive in many languages and programming frameworks[1] (e.g., see [38]). Multi-pattern string matching is especially common in analysis of content for malware (e.g., [64, 50]) and also is commonplace in searches on genome data, for example. In fact, there are now a number of available genome databases (e.g., [2, 5]) and accompanying tools for multi-pattern matching against them (e.g., [12]).

1.1 Third-Party Private DFA Evaluation on Encrypted Files in the Cloud

With the goal of improving privacy in such applications above, in Chapter 3 we develop novel protocols to evaluate a deterministic finite automaton (DFA) of the client's choice on the plaintext of the encrypted file and to return the final state to the client to indicate which, if any, of the patterns encoded in the DFA were matched. We stress that while there is much work on secure two-party

[1] To be more precise, the term "regular expression" is used in some frameworks in a way that follows but deviates somewhat from its original definition. Our system supports searches using regular expressions as originally defined, i.e., searches that can be expressed as deterministic finite automata [43].

computation including the specific case of private DFA evaluation on a private file, very few works have anticipated the possibility that the file is available only in encrypted form. This setting will become more common as data-storage outsourcing grows.

The security properties we prove for our protocols include privacy of the DFA and file contents against arbitrary server adversaries, and privacy of the file (except what is revealed by the evaluation result) against honest-but-curious client adversaries. Though our proofs are limited to only honest-but-curious client adversaries, we also provide heuristic justification for the security of our protocols against arbitrary client adversaries. Our protocols appear to be extensible with standard techniques to provably protect file privacy against arbitrary client adversaries, but we stop short of doing so in light of the substantially greater cost it would impose and our motivating scenarios involving third parties that the file owner must authorize and so presumably trusts to some extent. We do, however, discuss efficient heuristics to detect a misbehaving client or server that highlight new opportunities in the cloud storage setting.

A central observation that facilitates our protocols is that a DFA transition function can be encoded as a bivariate polynomial over the ring of an additively homomorphic encryption scheme with which the file characters are encrypted. In our protocols, the client, who has this polynomial as input, and the server, who has the encrypted file as input, obliviously perform DFA state transitions by jointly evaluating this polynomial. Neither party learns the current state at any point of the protocol execution; instead, they share the current state at each step, requiring that the polynomial be adapted in each round to accommodate this sharing.

We believe our protocols will be efficient enough for many practical scenarios. They support evaluation of any DFA over an alphabet Σ on any file consisting of ℓ symbols drawn from Σ, and require the file to be stored using ℓm ciphertexts where $m = |\Sigma|$. Since m is a multiplicative factor in the storage cost, our protocols are best suited small alphabets Σ, e.g., bits ($m = 2$), bytes ($m = 256$), alphanumeric characters ($m = 36$), or DNA nucleotides ($m = 4$ for “A”, “C”, “G”, and “T”). Specifically, in Chapter 3, we first present a protocol that leverages additively homomorphic encryption (e.g., [59]) and transmits $(nm+3)\ell+3$ ciphertexts to evaluate a DFA of n states. We then leverage additively homomorphic encryption that also supports *one* homomorphic multiplication of ciphertexts (e.g., [17]) to construct an improved protocol that transmits only $(n + m + 1)\ell + 3$ ciphertexts. Our techniques could also be utilized with fully homomorphic encryption to produce a noninteractive protocol with a communication cost of $O(nm)$ fully homomorphic ciphertexts and, in particular, that is independent of the file length ℓ.

1.2 Ensuring File Authenticity in Private DFA Evaluation on Encrypted Files in the Cloud

Even though our protocols provide provable privacy guarantees for both the DFA query and file content against arbitrarily malicious server adversaries, a malicious server could still try to tamper with the evaluation result by deviating from the protocol specification, or even input fraudulent encrypted files into the protocol to fool the client. Indeed, though the traditional notion of a protocol secure against an arbitrarily malicious adversary prevent any misbehaviors during protocol execution, it provides no guarantees on what *input* a malicious party may use in the protocol. Protocols for a third-party client to perform private searches on encrypted data in the cloud, while revealing nothing to the cloud server and nothing but the search result to the client, do exist for some types of searches (e.g., [67, 28, 73]). To our knowledge, however, none also enforce that the cloud server employs the data that the data owner stored at the cloud server.

Motivated by this, in Chapter 4 we present a strengthened protocol that allows the client to detect any misbehavior of the server, and in particular, to tell whether the server input the authentic encrypted file stored there by the data owner. In that sense, the authenticity of the file input by the server and the integrity of the computation result are both enforced. At the same time, the protocol provably protects the file contents (except for the result of the computation) from an honest-but-curious client (and heuristically from even a malicious client) and provably protects both the file contents and DFA from an arbitrarily malicious server. To our knowledge, our protocol is the first example of performing secure DFA computation on both encrypted *and authenticated* data.

Traditionally, one needs to know the file content and the signature to verify the authenticity of a file, and so the main technical difficulty in our case is to ensure computation on authenticated (signed) data without disclosing the plaintext to either party. The most common approach one might first consider to solve this problem is to leverage zero-knowledge proof techniques. By asking the data owner to publish commitments of the file character signatures, the server might then prove that his input used in the protocol is consistent with the published commitments. In the ways we see to instantiate this intuition, however, it would require much higher computation and communication costs than our protocol. Instead, we introduce a new technique to enforce correct server behavior and the authenticity of the input on which it is allowed to operate, without relying on zero-knowledge proofs at all. At a high level, the protocol takes advantage of the verifiability of the computation result to check the correctness of the server behavior. The protocol is designed so that that legitimate outputs are encoded in a small space only known to the client, and any malicious behavior by the server will result in the final output lying outside this space, which is then easily detected by the client. We prove this property (in the random oracle model) and the privacy of both the file and the DFA against an arbitrarily malicious server. We also prove the privacy of the file (except for the result of the DFA evaluation) against an honest-but-curious client.

1.3 Toward Practical Encrypted Email That Supports Private, Regular-Expression Searches

As a practical application of private regular expression searches on encrypted data, in Chapter 5 we report a case study of a prototype implementation using our protocol to perform private regular expression searches on encrypted emails. In particular, the protocol developed there suffices to support the search options (including Boolean combinations) offered by the Thunderbird email client for the text and numeric email fields, for example. Our system is thus able to support range queries on the date field and various types of substring queries on the source, destination, and subject fields of emails.

Our attention on the application of searchable encrypted emails was drawn from the numerous designs of socalled *searchable encryption* schemes that have been proposed with it as one of the main application. Unfortunately, to our knowledge few have made it to practical use. We believe that this state of affairs is due in part due to inflexibility, in the sense that such schemes typically require the document creator to tag it by the keywords on which searches will be supported in the future. Even though one can imagine tagging and encrypting all the words in an document to allow for all searches on any word, anything from a typo in the document to different forms of word stemming will render the search results unsatisfactory. For example, a document tagged with the keyword "meetings" would not be returned in the search results for the queries "meet" or "meeting". A recent study of user email query patterns [40] showed that many queries that users create are only partial words, and so substring searching capability is important to provide an adequate user experience. Furthermore, very few searchable encryption schemes offer the capabilities of performing substring, conjunctive, disjunctive and range queries, and we are aware of none that offers all them at the same time.

Our protocol for regular-expression searching gains computational efficiency by using interaction, in fact requiring data transfer between the searching client and the server holding the ciphertext of a volume larger than the searchable ciphertext itself. This obviously begs the question of whether a more suitable solution would be to download each email to the client and decrypt it there, to be searched locally. With the widespread use of volume-priced networking (i.e., over cellular data plans), however, neither design is particularly appealing. So, we instead explore a different design in which the user (e.g., via her mobile device) submits her *encrypted* regular expression (or suitable representation thereof) to a proxy, which then interacts with the server hosting the encrypted data using our protocol. After this interaction, the proxy reports information back to the user that permits her to determine whether there was a match, so she can retrieve the file from the server in that case. We stress that the interaction between the user and the proxy is independent of the lengths and number of ciphertexts stored at the server, and that the proxy is untrusted for the privacy of the search or the file contents (provided that it does not collaborate maliciously with the server). So, for example, the proxy could be run in a cloud distinct from that where the server is run.

The task of constructing such a protocol to be efficient is, as we found, very challenging. Starting from a protocol that implements the above functionality, we detail a series of optimizations that resulted in an optimized protocol with more than an order of magnitude improvement in the performance. At a high level, the optimizations involve careful redesign of the protocol in order to take advantage of well known algebraic optimization techniques (e.g., preprocessing to optimize pairing operations) and a few novel algebraic techniques to reduce the online computational costs. After detailing these protocol optimizations, we then explore additional optimizations that leverage specifics of the email setting. These optimizations pertain to the specific regular expression alphabet that should be utilized for each type of searchable field (i.e., source email address, sender name, date, and subject).

Following these optimizations, we detail an implementation of our protocol and its performance when searching emails from a real-world email dataset. We show, for example, that our implementation incurs average latencies of 0.89 seconds per email for performing a 9-character substring search on the sender email address field, and 0.17 seconds per email for performing a range query spanning about 6 months on the email date field. These numbers were obtained from a proxy and server each having 8 physical cores with simultaneous multithreading enabled, yielding 16 logical cores. We also evaluate options for exploiting parallelism with our protocol, ranging from very coarse (i.e., one server thread and one proxy thread per server-proxy protocol instance, but running 16 protocols instances in parallel) to very fine (i.e., 16 server threads and 16 proxy threads in one protocol instance).

1.4 Contributions

In summary, the contributions of this dissertation are:

- We developed protocols (in Chapter 3) that enable a client having a private regular expression to evaluate on the encrypted file stored at a server, once authorized to do so by the file owner. Our protocols contribute over prior work by offering the protection of the privacy of the file content against both server and client. More precisely, the protocols protect privacy of the query and file content against arbitrarily malicious server adversaries and honest-but-curious client adversaries.

- In Chapter 4, we present a extension of the protocol developed in Chapter 3 so that, in addition to offering the security guarantees already provided by the original protocol, the client is able to detect any misbehavior of a server adversary. Furthermore, it can even tell whether the server input the authentic encrypted file stored there by the file owner during the protocol execution. Consequently, the input and the evaluation result are both authenticated to the client. To our knowledge, this is the first protocol published that considers secure computation on both encrypted and authenticated data in the context of DFA evaluation.

- To demonstrate the usefulness of private DFA evaluation in the real world, in Chapter 5 we prototype a system using our protocol to perform private regular expression searches on encrypted emails. Toward that goal, we developed another protocol, along with several optimizations that allows a user to securely delegate the search query to a proxy, which will interact with the server where the encrypted email is stored to perform the search and return the result to the user. The privacy of the query and the data are both protected against an arbitrarily malicious server and an honest-but-curious proxy. We provide extensive evaluations on the implementation and report its performance using a real world email data set.

CHAPTER 2

Related Work

In this chapter, we discuss research that is related to this dissertation. We discuss general techniques for secure computation in Section 2.1 and then protocols specifically tailored to private DFA evaluation in Section 2.2. Protocols specifically targeted other types of search functionality are discussed in Section 2.3. Previous work on research to ensure that authentic inputs are employed in secure computation protocols is discussed in Section 2.4, and some previous implementation efforts for searching on encrypted data are briefly surveyed in Section 2.5.

2.1 General Techniques for Secure Computation

The problem we study in this dissertation — i.e., privately evaluating a regular expression on the plaintext of an encrypted file stored at a server — could be implemented with general techniques for "computing on encrypted data" [63] or two-party secure computation [74, 37]. These general techniques tend to yield less efficient protocols than one designed for a specific purpose, and our case will be no exception. In particular, the former achieves computations non-interactively using fully homomorphic encryption, for which existing implementations [33, 71, 66, 69] are dramatically more costly than the techniques we use.

The latter utilizes a "garbled circuit" construction that is of size linear in the circuit representation of the function to be computed. Despite progress on practical implementations of this technique [54, 11, 60], this limitation renders it substantially more communication-intensive for the problem we consider. In particular, in Chapter 5 we study the setting in which a user wishes to outsource her (encrypted) search query to a partially trusted proxy machine; the proxy will interact with the server hosting the encrypted data and report the encrypted results back to the user. One requirement of the protocol is that the communication between the user and the proxy should be minimized. Using our protocol, the communication cost in the direction from the user to the proxy is only dependent on the size of the search query, and is independent of the number and size of the file ciphertexts. We are unaware of how to achieve this property using garbled circuits, however. Since the garbled circuit and its inputs are "unreusable" across different runs of the protocol, the user would need to provide a number of inputs (in this case, encrypted queries) to the proxy that equals to the number of files to be searched. Furthermore, the fact that in our construction, the user-generated encrypted query can be used an unlimited number of times enables a subscription service such that the proxy holds the encrypted query and periodically informs the user of the arrival of matched emails, without any further communication from the user to the proxy. Again, we are unaware of how to implement this functionality using generic garbled-circuit techniques.

An ingredient of our protocols in Chapter 3 and Chapter 5 is a two-party sharing of the data owner's file-decryption key between a client holding the search query and the server holding the encrypted file. By two-party secret-sharing the file-decryption key and using this to compute on encrypted data, our protocols are related to those of Choi et al. [24]. This work developed a protocol based on garbled circuits by which two parties can evaluate a general function after a private decryption key has been shared between them. This protocol can be used to solve the problem we propose, but inherits the aforementioned limitations of garbled circuits.

2.2 Specialized Protocols for DFA Evaluation

Two-party private DFA evaluation, in which a server has a file and a client has a DFA to evaluate on that file, has been a topic of recent focus. To our knowledge, Troncoso-Pastoriza et al. [70] were the first to present such a protocol, which they proved secure in the honest-but-curious setting. Frikken [31] presented a protocol for the same setting that improved on the round complexity and computational costs. Gennaro et al. [32] gave a two-party DFA evaluation protocol that they proved secure against arbitrary adversaries. Our work differs from these in that in all of our protocols, the file is made available to the parties only in ciphertext form, and in our protocols in Chapter 5, even the DFA is made available only in ciphertext form to a proxy to which the user delegates the search to, so that the proxy need not be trusted with the privacy of the search query. In this respect, the protocol of Blanton and Aliasgari [13] is relevant; they adapted the Troncoso-Pasoriza et al. protocol to an "outsourcing" model in which the client and server secret-share the DFA and file, respectively, between two additional hosts that interactively evaluate the DFA on the file without reconstructing either one. While our protocols utilize secret sharing, as well — in our case, of the file owner's file-decryption key — our protocol shares much less data and does not share the client's DFA (or thus require two parties between which to share it) at all. Furthermore, their protocol does not support the asymmetric encryption of the file, which in the encrypted email application we consider in Chapter 5 is the predominant method for preparing a private email for its intended recipient.

2.3 Specialized Protocols for Searching on Encrypted Data

Specialized protocols for performing searches on encrypted files or database relations have also been developed. For example, searchable encryption [67, 36, 16, 22, 28, 6, 19, 9, 49, 62, 47] enables a party holding a file-decryption key to search for attribute values in the ciphertext file stored at an untrusted server. These techniques have been generalized to support more complex queries, notably conjunctive [19], disjunctive [49] and range queries [65] and inner products [49]. Searchable encryption schemes typically achieve non-interactive queries on encrypted files, in part by attaching "tag" information to the ciphertext of each file to enable the query operation. However, broadening the supported search attributes typically requires expanding the tags, and so the sizes of the tags are determined by the richness of the supported queries. In contrast, in our work the file ciphertexts are independent of the DFA(s) to be evaluated (assuming a fixed alphabet Σ over which the DFAs are defined), and the computation is performed interactively between the two parties.

Richer forms of pattern-matching and search (though still not encompassing DFA evaluation) have also been studied in the two-party setting, e.g., by Jha et al. [45], Hazay and Lindell [41], Katz and Malka [48], and Hazay and Toft [42]. Again, these works input the plaintext file to one party and so do not directly apply to our setting.

Still other works have explored storing database relations at an untrusted server in a form that hides sensitive attributes or associations between attributes while supporting rich queries, e.g., range queries [44, 23] or SQL queries [39, 25]. The security properties offered by these techniques are usually heuristic, without formal definitions and proofs, and we are unaware of any designed to support DFA searches.

2.4 Input Authenticity in Secure Computation

Most work in the area of secure computation generally does not consider the authenticity of the inputs to the protocol. Indeed, the standard definition of security against arbitrarily malicious adversaries for general two-party protocols provides no restrictions on what input a malicious party may use in the protocol as long as he does not deviate from the protocol. The protocol we present in Chapter 4 allows the client to tell whether the server actually uses the authentic encrypted data of the data owner as input, in addition to the ability to detect any misbehavior by the server. In this sense, our protocol provides an authenticated evaluation result to the client. To our knowledge, ours is the first protocol to consider secure computation on authenticated data in the context of private DFA evaluation. The main area of specialized protocols in which input authenticity has previously been treated has been private intersection of certified sets [20, 27, 26, 68], in which the set elements of each party much be certified by a trusted third party for use in performing the intersection.

2.5 Implementations of Systems That Allow Searching on Encrypted Data

There have been some implementation efforts to prototype systems that support search operations on encrypted data. Kamara et al. [46] implemented a encrypted storage system using a symmetric searchable encryption scheme that supports keyword search. Waters et al. [72] built a searchable encrypted audit log system using an asymmetric searchable encryption scheme. Popa et al. [61] presented a scheme supporting SQL queries on encrypted data and provided an implementation of an encrypted database using that scheme. The system we developed, described in Chapter 5, enables private regular expression searches in an encrypted email system. To our knowledge, ours is the first implementation supporting private regular expression queries on encrypted files.

CHAPTER 3

Third-Party Private DFA Evaluation on Encrypted Files in the Cloud

Motivated by the need to outsource file storage to untrusted clouds while still permitting limited use of that data by third parties, in this chapter, we present practical protocols by which a client can evaluate a DFA on an encrypted file stored at a cloud server, once authorized to do so by the file owner. Our protocols provably protect the privacy of the DFA and the file contents from a malicious server and the privacy of the file contents (except for the result of the evaluation) from an honest-but-curious client (and, heuristically, from a malicious client). We introduce our main protocol in Section 3.2 and an improved protocol in Section 3.3. We further present simple techniques to detect client or server misbehavior in Section 3.4. Before that, we first define the studied problem in Section 3.1.

3.1 Problem Description

A deterministic finite automaton M is a tuple $\langle Q, \Sigma, \delta, q_{\mathsf{init}}\rangle$ where Q is a set of $|Q| = n$ *states*; Σ is a set (*alphabet*) of $|\Sigma| = m$ *symbols*; $\delta : Q \times \Sigma \to Q$ is a transition function; and q_{init} is the initial state. (A DFA can also specify a function $\Delta : Q \to \{0, 1\}$, for which $\Delta(q) = 1$ indicates that q is an accepting state. We will discuss extensions of our protocols to this case.)

Our goal is to enable a client holding a DFA M to interact with a server holding the ciphertext of a file to evaluate M on the file plaintext. More specifically, the client should output the final state to which the file plaintext drives the DFA; i.e., if the plaintext file is a sequence $\langle\sigma_k\rangle_{k\in[\ell]}$ where $[\ell]$ denotes the set $\{0, 1, \ldots, \ell-1\}$ and where each $\sigma_k \in \Sigma$, then the client should output $\delta(\ldots\delta(\delta(q_{\mathsf{init}}, \sigma_0), \sigma_1), \ldots, \sigma_{\ell-1})$. We also permit the client to learn the file length ℓ and the server to learn both ℓ

and the number of states n in the client's DFA.[1] The client should learn nothing else about the file, however, and the server should learn nothing else about the file or the client's DFA.

Because the file exists in the system only in encrypted form, some private-key information must be injected into the protocol to enable a DFA to be evaluated on the file plaintext. Since (only) the data owner holds the private key, one approach would be to involve the data owner in the protocol. However, in keeping with the goals of cloud outsourcing, our protocols require the data owner only to authorize the client to perform DFA evaluations with the server — but not to participate in those evaluations herself. In our protocols, this authorization occurs by the data owner sharing the private file-decryption key between the client and server. As a result, a client and server that collude could pool their information to decrypt the file. Here we assume no such collusion, however, for two reasons. First, we are primarily motivated by scenarios in which the client represents a partially trusted service provider or customer, and so even if the cloud server were to be compromised, we presume this party would not be the cause. So, we prove security against only a client or server acting in isolation and with primary attention to only an honest-but-curious client (though we also heuristically justify the security of our protocol against an arbitrary client). Second, even without sharing the file decryption key between the client and server, the functionality offered by our protocol (i.e., evaluating a DFA on the file) would enable a colluding client and server to evaluate arbitrary (and arbitrarily many) DFAs on the file, eventually permitting its decryption anyway. The only defense against collusion that we see would be to involve the data owner in the protocol; again, we do not explore this possibility here.

Another potential form of collusion that we do not explicitly consider here is collusion between the data owner and the server, presumably to learn the DFA used by the client. In our protocol, however, the protection of DFA privacy does not depend on the security of the data owner's file-decryption key. Since the data owner is not involved in the protocol, it does not offer the server any additional leverage in learning the client's DFA.

[1] Since exposing the final state reduces file entropy by $\log_2 n$ bits, presumably the server should learn n so as to monitor for excessive exposure or to charge for the information learned by the client. Moreover, the client can arbitrarily inflate n by adding unreachable states. As such, we consider disclosing n to the server to be practically necessary but of little threat to the client. This stands in contrast to some other two-party private computation scenarios in which input-size hiding has been a priority, e.g., private set intersection [8].

Our protocols do not retrieve the file based on the DFA evaluation results, e.g., in a way that hides from the server what file is being retrieved. However, once the client learns the final state of the DFA evaluation, it can employ various techniques to retrieve the file privately (e.g., [35]). Moreover, some of our motivating scenarios in Chapter 1, e.g., malware scans of cloud-resident files by a third party, may not require file retrieval but only that matches be reported to the file owner.

3.2 A Secure DFA Evaluation Protocol

In this section we present a protocol that meets the goals described in Section 3.1. We give the construction in Section 3.2.1, and then we define and prove security against server and client adversaries in Section 3.2.2 and Section 3.2.3, respectively.

3.2.1 Construction

Let "$\leftarrow$" denote assignment and "$s \overset{\$}{\leftarrow} S$" denote the assignment to s of a randomly chosen element of set S. Let κ denote a security parameter.

Encryption scheme Our scheme is built using an additively homomorphic encryption scheme with plaintext space $\mathbb{R}$ where $\langle \mathbb{R}, +_{\mathbb{R}}, \cdot_{\mathbb{R}} \rangle$ denotes a commutative ring. Specifically, an encryption scheme $\mathcal{E}$ includes algorithms Gen, Enc, and Dec where: Gen is a randomized algorithm that on input 1^κ outputs a public-key/private-key pair $(pk, sk) \leftarrow \mathsf{Gen}(1^\kappa)$; Enc is a randomized algorithm that on input public key pk and plaintext $m \in \mathbb{R}$ (where $\mathbb{R}$ can be determined as a function of pk) produces a ciphertext $c \leftarrow \mathsf{Enc}_{pk}(m)$, where $c \in C_{pk}$ and C_{pk} is the ciphertext space determined by pk; and Dec is a deterministic algorithm that on input a private key sk and ciphertext $c \in C_{pk}$ produces a plaintext $m \leftarrow \mathsf{Dec}_{sk}(c)$ where $m \in \mathbb{R}$. In addition, $\mathcal{E}$ supports an operation $+_{pk}$ on ciphertexts such that for any public-key/private-key pair (pk, sk), $\mathsf{Dec}_{sk}(\mathsf{Enc}_{pk}(m_1) +_{pk} \mathsf{Enc}_{pk}(m_2)) = m_1 +_{\mathbb{R}} m_2$. Using $+_{pk}$, it is possible to implement $\cdot_{pk}$ for which $\mathsf{Dec}_{sk}(m_2 \cdot_{pk} \mathsf{Enc}_{pk}(m_1)) = m_1 \cdot_{\mathbb{R}} m_2$.

We also require $\mathcal{E}$ to support two-party decryption. Specifically, we assume there is an efficient randomized algorithm Share that on input a private key sk outputs shares $(sk_1, sk_2) \leftarrow \mathsf{Share}(sk)$, and that there are efficient deterministic algorithms Dec^1 and Dec^2 such that $\mathsf{Dec}_{sk}(c) = \mathsf{Dec}^2_{sk_2}(c, \mathsf{Dec}^1_{sk_1}(c))$.

An example of an encryption scheme $\mathcal{E}$ that meets the above requirements is due to Paillier [59] with modifications by Damgård and Jurik [29]; we henceforth refer to this scheme as "Pai". In this scheme, the ring $\mathbb{R}$ is $\mathbb{Z}_N$ where $N = pp'$ and p, p' are primes, and the ciphertext space C_{pk} is $\mathbb{Z}^*_{N^2}$.

We use $\sum^{pk}$ to denote summation using $+_{pk}$; $\sum^{\mathbb{R}}$ to denote summation using $+_{\mathbb{R}}$; and $\prod^{\mathbb{R}}$ to denote the product using $\cdot_{\mathbb{R}}$ of a sequence. For any operation op, we use t_{op} to denote the time required to perform op; e.g., t_{Dec} is the time to perform a Dec operation.

Encoding δ in a Bivariate Polynomial over $\mathbb{R}$ A second ingredient for our protocol is a method for encoding a DFA $\langle Q, \Sigma, \delta, q_{\mathsf{init}}\rangle$, and specifically the transition function δ, as a bivariate polynomial $f(x, y)$ over $\mathbb{R}$ where x is the variable representing a DFA state and y is the variable representing an input symbol. That is, if we treat each state $q \in Q$ and each $\sigma \in \Sigma$ as distinct elements of $\mathbb{R}$, then we would like $f(q, \sigma) = \delta(q, \sigma)$. We can achieve this by choosing f to be the interpolation polynomial

$$f(x, y) = \sum^{\mathbb{R}}_{\sigma \in \Sigma} \left(f_\sigma(x) \cdot_{\mathbb{R}} \Lambda_\sigma(y)\right) \quad \text{where} \quad \Lambda_\sigma(y) = \prod^{\mathbb{R}}_{\substack{\sigma' \in \Sigma \\ \sigma' \neq \sigma}} \frac{y -_{\mathbb{R}} \sigma'}{\sigma -_{\mathbb{R}} \sigma'} \tag{3.1}$$

is a Lagrange basis polynomial and $f_\sigma(q) = \delta(q, \sigma)$ for each $q \in Q$. Note that $\Lambda_\sigma(\sigma) = 1$ and $\Lambda_\sigma(\sigma') = 0$ for any $\sigma' \in \Sigma \setminus \{\sigma\}$.

Calculating Eqn. 3.1 requires taking multiplicative inverses in $\mathbb{R}$. While not every element of a ring has a multiplicative inverse in the ring, fortunately the ring $\mathbb{Z}_N$ used in Paillier encryption, for example, has negligibly few elements with no inverses, and so there is little risk of encountering an element with no inverse. Using Eqn. 3.1, we can calculate coefficients $\langle \lambda_{\sigma j}\rangle_{j \in [m]}$ so that $\Lambda_\sigma(y) = \sum^{\mathbb{R}\, m-1}_{j=0} \lambda_{\sigma j} \cdot_{\mathbb{R}} y^j$. For our algorithm descriptions, we encapsulate this calculation in the procedure $\langle \lambda_{\sigma j}\rangle_{\sigma \in \Sigma, j \in [m]} \leftarrow \mathsf{Lagrange}(\Sigma)$.

Each f_σ needed to compute $f(x, y)$ can again be determined as a Lagrange interpolating polynomial and then expressed as $f_\sigma(x) = \sum^{\mathbb{R}\, n-1}_{i=0} a_{\sigma i} \cdot_{\mathbb{R}} x^i$. In our pseudocode, we encapsulate this calculation as $\langle a_{\sigma i}\rangle_{\sigma \in \Sigma, i \in [n]} \leftarrow \mathsf{ToPoly}(Q, \Sigma, \delta)$.

Protocol steps Our protocol, denoted $\Pi_1(\mathcal{E})$, is shown in Fig. 3.1. Pseudocode for the client is aligned on the left of the figure and labeled c101–c116; the server pseudocode is on the right of the figure and labeled s101–s112; and messages exchanged between them are aligned in the center and labeled m101–m106. The client receives as input a public key pk under which the file (at the server) is encrypted; a share sk_1 of the private key sk corresponding to pk; another public key pk'; and the DFA $\langle Q, \Sigma, \delta, q_{\mathsf{init}} \rangle$. The server receives as input the public key pk; a share sk_2 of the private key sk; the alphabet Σ; and ciphertexts $c_{kj} \leftarrow \mathsf{Enc}_{pk}((\sigma_k)^j)$ of the k-th file symbol $\sigma_{k,}$ for each $j \in [m]$ and for each $k \in [\ell]$ where ℓ denotes the file length in symbols. We assume that sk_1 and sk_2 were generated as $(sk_1, sk_2) \leftarrow \mathsf{Share}(sk)$. Note that no information about sk' (the private key corresponding to pk') is given to either party, and so pk' ciphertexts (ρ created in c107 and c115 and sent in m103 and m105, respectively) are indecipherable and ignored in the protocol. These ciphertexts are included to simplify the proof of privacy against client adversaries (Section 3.2.3) and can be elided in practice. We do not discuss these values further in this section.

The protocol is structured as matching **for** loops executed by the client (c105–c113) and server (s103–s111). The client begins the k-th loop iteration with an encryption α of the current DFA state after being blinded by a random injection $\pi_1 : Q \rightarrow \mathbb{R}$ it chose in the $(k-1)$-th loop at line c109 (or, if $k = 0$, then in line c103), where $\mathsf{Injs}(Q \rightarrow \mathbb{R})$ denotes the set of injections from Q to $\mathbb{R}$. The client uses its share sk_1 of sk to create the "partial decryption" β of α (c106) and sends α, β to the server (m103). The server uses its share sk_2 to complete the decryption of α to obtain the blinded state γ (s104). We stress that because γ is blinded by π_1, γ reveals no information about the current DFA state to the server. The server then computes, for each $\sigma \in \Sigma$ (s105), a value Ψ_σ such that $\Lambda_\sigma(\sigma_k) = \mathsf{Dec}_{sk}(\Psi_\sigma)$ (s106) by utilizing coefficients $\langle \lambda_{\sigma j} \rangle_{\sigma \in \Sigma, j \in [m]}$ output from Lagrange (s102). The server then returns (in m104) values $\langle \mu_{\sigma i} \rangle_{\sigma \in \Sigma, i \in [n]}$ created so that $\mathsf{Dec}_{sk}(\mu_{\sigma i}) = \gamma^i \cdot_{\mathbb{R}} \Lambda_\sigma(\sigma_k)$ (s108).

Meanwhile, the client selects a new random injection $\pi_1 \overset{\$}{\leftarrow} \mathsf{Injs}(Q \rightarrow \mathbb{R})$ (c109). The client then constructs a new DFA transition function δ' reflecting the injection it chose in the last round (now denoted π_0, see line c108) and the new injection π_1 it chose for this round. Specifically, it creates a new DFA state transition function δ' defined as $\delta'(q, \sigma) = \pi_1(\delta(\pi_0^{-1}(q), \sigma))$ for all $\sigma \in \Sigma$ and $q \in \pi_0(Q)$ where $\pi_0(Q) = \{\pi_0(q)\}_{q \in Q}$; we denote this step as $\delta' \leftarrow \mathsf{Blind}(\delta, \pi_0, \pi_1)$ in line c110. That is, δ' "undoes" the previous injection π_0, applies δ, and then applies the new injection π_1. The client then interpolates a bivariate polynomial $f(x, y)$ such that $f(q, \sigma) = \delta'(q, \sigma)$ in line c111, using the algorithm described previously. The client then uses these coefficients and

```
client$(pk, sk_1, pk', \langle Q, \Sigma, \delta, q_{\mathsf{init}} \rangle)$          server$(pk, sk_2, \Sigma, \langle c_{kj} \rangle_{k \in [\ell], j \in [m]})$

c101. $n \leftarrow |Q|, m \leftarrow |\Sigma|$                          s101. $m \leftarrow |\Sigma|$
c102. $\pi_0 \leftarrow \mathbf{I}$                                      s102. $\langle \lambda_{\sigma j} \rangle_{\sigma \in \Sigma, j \in [m]}$
c103. $\pi_1 \overset{\$}{\leftarrow}$ Injs$(Q \to \mathbb{R})$                              $\leftarrow$ Lagrange$(\Sigma)$
c104. $\alpha \leftarrow \mathsf{Enc}_{pk}(\pi_1(q_{\mathsf{init}}))$
                      m101. $\xrightarrow{n}$
                      m102. $\xleftarrow{\ell}$
c105. for $k \leftarrow 0 \ldots \ell - 1$                              s103. for $k \leftarrow 0 \ldots \ell - 1$
c106.   $\beta \leftarrow \mathsf{Dec}^1_{sk_1}(\alpha)$
c107.   $\rho \leftarrow \mathsf{Enc}_{pk'}(\pi_1)$
                      m103. $\xrightarrow{\alpha, \beta, \rho}$
                                                                  s104.   $\gamma \leftarrow \mathsf{Dec}^2_{sk_2}(\alpha, \beta)$
c108.   $\pi_0 \leftarrow \pi_1$                                         s105.   for $\sigma \in \Sigma$
c109.   $\pi_1 \overset{\$}{\leftarrow}$ Injs$(Q \to \mathbb{R})$                       s106.     $\Psi_\sigma \leftarrow \sum_{j=0}^{m-1}{}_{pk} \lambda_{\sigma j} \cdot_{pk} c_{kj}$
c110.   $\delta' \leftarrow$ Blind$(\delta, \pi_0, \pi_1)$                       s107.     for $i \in [n]$
c111.   $\langle a_{\sigma i} \rangle_{\sigma \in \Sigma, i \in [n]}$                          s108.       $\mu_{\sigma i} \leftarrow \gamma^i \cdot_{pk} \Psi_\sigma$
          $\leftarrow$ ToPoly$(Q, \Sigma, \delta')$                        s109.     endfor
                                                                  s110.   endfor
                      m104. $\xleftarrow{\langle \mu_{\sigma i} \rangle_{\sigma \in \Sigma, i \in [n]}}$
c112.   $\alpha \leftarrow \sum_{\sigma \in \Sigma}{}_{pk} \sum_{i=0}^{n-1}{}_{pk} a_{\sigma i} \cdot_{pk} \mu_{\sigma i}$
c113. endfor                                                      s111. endfor
c114. $\beta \leftarrow \mathsf{Dec}^1_{sk_1}(\alpha)$
c115. $\rho \leftarrow \mathsf{Enc}_{pk'}(\pi_1)$
                      m105. $\xrightarrow{\alpha, \beta, \rho}$
                                                                  s112. $\gamma^* \leftarrow \mathsf{Dec}^2_{sk_2}(\alpha, \beta)$
                      m106. $\xleftarrow{\gamma^*}$
c116. return $\pi_1^{-1}(\gamma^*)$
```

Figure 3.1: Protocol $\Pi_1(\mathcal{E})$, described in Section 3.2

$\langle \mu_{\sigma i} \rangle_{\sigma \in \Sigma, i \in [n]}$ sent from the server (message m103) to assemble a ciphertext α of the new DFA state under the injection π_1 (c112).

After ℓ loop iterations, the client interacts with the server once more to decrypt the final state. It sends α and its partial decryption β to the server (m105), for which the server completes the decryption (s112) and returns the result (m106).

Protocol $\Pi_1(\mathcal{E})$ can be modified to return only a binary indication of whether the DFA's final state is an accepting one, if the DFA specifies a function Δ indicating whether a state is an accepting state. Specifically, the client can construct a polynomial $F(x)$ such that $F(q) = 1$ if $\Delta(q) = 1$ and $F(q) = 0$ otherwise, for $q \in Q$. Then, rather than interacting with the server to decrypt the final state, the client can interact with the server once to evaluate $F(x)$ on the (unknown) final state and again to decrypt this result.

For brevity, Fig. 3.1 omits numerous checks that the client and server should perform to confirm that the values each receives are well-formed. For example, the client should confirm that $\mu_{\sigma i} \in C_{pk}$ for each $\sigma \in \Sigma$ and $i \in [n]$, upon receiving these in m104. The server should similarly confirm the well-formedness of the values it receives.

An Alternative Using Fully Homomorphic Encryption Our technique of encoding the DFA transition function δ using a bivariate polynomial $f(x, y)$ over $\mathbb{R}$ could also be used with fully homomorphic encryption [33, 71] to create a noninteractive protocol. The client could encrypt each coefficient $a_{\sigma i}$ of f under the public key pk and send these ciphertexts to the server, enabling the server to perform computations c112 by itself. At the end, the server could send a half decrypted final state back to the client, who would complete the decryption to obtain the result. This protocol achieves communication costs of $O(nm)$, which is independent of the file length. That said, existing fully homomorphic schemes are far less efficient than additively homomorphic schemes, and so the resulting protocol will be less communication-efficient than $\Pi_1(\mathcal{E})$ for many practical file lengths and DFA sizes.

3.2.2 Security Against Server Adversaries

In this section we show that the server, by executing this protocol (even arbitrarily maliciously), gains no advantage in either determining the DFA the client is evaluating or the plaintext of the file in its possession. That is, we show only the *privacy* of the file and DFA inputs against server adversaries. In this section, we are not concerned with showing that a client can detect server misbehavior, a property often called *correctness*. $\Pi_1(\mathcal{E})$ could be augmented using standard tools to enforce correctness, with an impact on performance; we do not explore this here. Instead, in Section 3.4 we describe novel extensions to $\Pi_1(\mathcal{E})$ that could be used to detect server misbehavior.

We formalize our claims against server compromise by defining two separate server adversaries. The first server adversary $S = (S_1, S_2)$ attacks the DFA $M = \langle Q, \Sigma, \delta, q_{\mathsf{init}}\rangle$ held by the client, as described in experiment $\mathbf{Expt}^{\texttt{s-dfa}}_{\Pi_1(\mathcal{E})}$ in Fig. 3.2. S_1 first generates a file $\langle\sigma_k\rangle_{k\in[\ell]}$ and two DFAs M_0, M_1. (Note that we use, e.g., "$M_0.Q$" and "$M_1.Q$" to disambiguate their state sets.) S_2 then receives the ciphertexts $\langle c_{kj}\rangle_{k\in[\ell],j\in[m]}$ of its file, information ϕ created for it by S_1, and oracle access to $\mathsf{clientOr}(pk, sk_1, pk', M_b)$ for b chosen randomly.

```
Experiment $\mathbf{Expt}^{\texttt{s-dfa}}_{\Pi_1(\mathcal{E})}(S_1, S_2)$
    $(pk, sk) \leftarrow \mathsf{Gen}(1^\kappa)$
    $(sk_1, sk_2) \leftarrow \mathsf{Share}(sk)$
    $(pk', sk') \leftarrow \mathsf{Gen}(1^\kappa)$
    $(\ell, \langle\sigma_k\rangle_{k\in[\ell]}, M_0, M_1, \phi) \leftarrow S_1(pk, sk_2)$
    $\mathbf{if}\ M_0.Q \neq M_1.Q\ \mathbf{or}\ M_0.\Sigma \neq M_1.\Sigma$
        $\mathbf{then\ return}\ 0$
    $b \overset{\$}{\leftarrow} \{0,1\}$
    $m \leftarrow |M_b.\Sigma|$
    $\mathbf{for}\ k \in [\ell], j \in [m]$
        $c_{kj} \leftarrow \mathsf{Enc}_{pk}((\sigma_k)^j)$
    $b' \leftarrow S_2^{\mathsf{clientOr}(pk, sk_1, pk', M_b)}(\phi, \langle c_{kj}\rangle_{k\in[\ell],j\in[m]})$
    $\mathbf{if}\ b' = b$
        $\mathbf{then\ return}\ 1$
        $\mathbf{else\ return}\ 0$
```

Figure 3.2: Experiments for proving DFA privacy of $\Pi_1(\mathcal{E})$ against server adversaries

clientOr responds to queries from S_2 as follows, ignoring malformed queries. The first query (say, consisting of simply "start") causes clientOr to begin the protocol; clientOr responds with a message of the form n (i.e., of the form of m101). The second invocation by S_2 must include a single integer ℓ (i.e., of the form of m102); clientOr responds with a message of the form α, β, ρ, i.e., three values as in m103. The next $\ell - 1$ queries by S_2 must contain nm elements of C_{pk}, i.e., $\langle\mu_{\sigma i}\rangle_{\sigma\in\Sigma,i\in[n]}$ as in m104, to which clientOr responds with three values as in message m103. The next query to clientOr again must contain nm elements of C_{pk} as in m104, to which clientOr

```
Experiment $\mathbf{Expt}^{\texttt{ind-cpa}}_{\mathcal{E}}(U)$
    $(\hat{pk}, \hat{sk}) \leftarrow \mathsf{Gen}(1^{\kappa})$
    $\hat{b} \xleftarrow{\$} \{0,1\}$
    $\hat{b}' \leftarrow U^{\mathsf{Enc}^{\hat{b}}_{\hat{pk}}(\cdot,\cdot)}(\hat{pk})$
    $\mathbf{if}\ \hat{b}' = \hat{b}$
        $\mathbf{then\ return}\ 1$
        $\mathbf{else\ return}\ 0$
```

Figure 3.3: Experiment for IND-CPA security

responds with three values as in $\mathsf{m105}$. The next (and last) query by S_2 can consist simply of a value in $\mathbb{R}$, as in message $\mathsf{m106}$.

Eventually S_2 outputs a bit b', and $\mathbf{Expt}^{\texttt{s-dfa}}_{\Pi_1(\mathcal{E})}(S) = 1$ only if $b' = b$. We say the *advantage* of S is $\mathbf{Adv}^{\texttt{s-dfa}}_{\Pi_1(\mathcal{E})}(S) = 2 \cdot \mathbb{P}\left(\mathbf{Expt}^{\texttt{s-dfa}}_{\Pi_1(\mathcal{E})}(S) = 1\right) - 1$ and define $\mathbf{Adv}^{\texttt{s-dfa}}_{\Pi_1(\mathcal{E})}(t, \ell, n, m) = \max_S \mathbf{Adv}^{\texttt{s-dfa}}_{\Pi_1(\mathcal{E})}(S)$ where the maximum is taken over all adversaries S taking time t and selecting a file of length ℓ and DFAs containing n states and an alphabet of m symbols.

We reduce DFA privacy against server attacks to the IND-CPA [10] security of the encryption scheme. IND-CPA security is defined using the experiment in Fig. 3.3, in which an adversary U is provided a public key $\hat{pk}$ and access to an oracle $\mathsf{Enc}^{\hat{b}}_{\hat{pk}}(\cdot,\cdot)$ that consistently encrypts either the first of its two inputs (if $\hat{b} = 0$) or the second of those inputs (if $\hat{b} = 1$). Eventually U outputs a guess $\hat{b}'$ at $\hat{b}$, and $\mathbf{Expt}^{\texttt{ind-cpa}}_{\mathcal{E}}(U) = 1$ only if $\hat{b}' = \hat{b}$. The IND-CPA advantage of U is defined as $\mathbf{Adv}^{\texttt{ind-cpa}}_{\mathcal{E}}(U) = 2 \cdot \mathbb{P}\left(\mathbf{Expt}^{\texttt{ind-cpa}}_{\mathcal{E}}(U) = 1\right) - 1$. Then, $\mathbf{Adv}^{\texttt{ind-cpa}}_{\mathcal{E}}(t, w) = \max_U \mathbf{Adv}^{\texttt{ind-cpa}}_{\mathcal{E}}(U)$ where the maximum is taken over all adversaries U executing in time t and making w queries to $\mathsf{Enc}^{\hat{b}}_{\hat{pk}}(\cdot,\cdot)$.

Our theorem statements throughout this paper omit terms that are negligible as a function of the security parameter κ.

Theorem 1. *For* $t' = t + t_{\mathsf{Gen}} + t_{\mathsf{Share}} + \ell m \cdot t_{\mathsf{Enc}}$,

$$\mathbf{Adv}^{\texttt{s-dfa}}_{\Pi_1(\mathcal{E})}(t, \ell, n, m) \ \leq 2\mathbf{Adv}^{\texttt{ind-cpa}}_{\mathcal{E}}(t', \ell + 1)$$

Proof. Let S be an adversary meeting the parameters t, ℓ, n, and m. Consider a simulation $\mathbf{Sim}^{\texttt{s-dfa}}_{\Pi_1(\mathcal{E})}$ for $\mathbf{Expt}^{\texttt{s-dfa}}_{\Pi_1(\mathcal{E})}$ that differs only by simulating $\mathsf{clientOr}$ so as to substitute all ciphertexts produced with pk' with encryptions of a random injection π independent of π_1 it chose as in

```
Experiment $\mathbf{Expt}^{\texttt{s-file}}_{\Pi_1(\mathcal{E})}(S_1, S_2)$
    $(pk, sk) \leftarrow \mathsf{Gen}(1^\kappa)$
    $(sk_1, sk_2) \leftarrow \mathsf{Share}(sk)$
    $(pk', sk') \leftarrow \mathsf{Gen}(1^\kappa)$
    $(\ell, \langle \sigma_{0k} \rangle_{k \in [\ell]}, \langle \sigma_{1k} \rangle_{k \in [\ell]}, M, \phi) \leftarrow S_1(pk, sk_2)$
    $b \overset{\$}{\leftarrow} \{0,1\}$
    $m \leftarrow |M.\Sigma|$
    for $k \in [\ell], j \in [m]$
        $c_{kj} \leftarrow \mathsf{Enc}_{pk}((\sigma_{bk})^j)$
    $b' \leftarrow S_2^{\mathsf{clientOr}(pk, sk_1, pk', M)}(\phi, \langle c_{kj} \rangle_{k \in [\ell], j \in [m]})$
    if $b' = b$
        then return $1$
        else return $0$
```

Figure 3.4: Experiments for proving file privacy of $\Pi_1(\mathcal{E})$ against server adversaries

c109 (i.e., $\rho \leftarrow \mathsf{Enc}_{pk'}(\pi)$, $\pi \overset{\$}{\leftarrow} \mathsf{Injs}(Q \rightarrow \mathbb{R})$ in c107 and c115). Then b is hidden information-theoretically from S in $\mathbf{Sim}^{\texttt{s-dfa}}_{\Pi_1(\mathcal{E})}$, since γ is a random element of $\mathbb{R}$ in s104 and since γ^* is a random element of $\mathbb{R}$ (see c109). As a result, $\mathbb{P}\left(\mathbf{Sim}^{\texttt{s-dfa}}_{\Pi_1(\mathcal{E})}(S) = 1\right) = \frac{1}{2}$ and for $\mathbf{Adv}^{\texttt{s-dfa}}_{\Pi_1(\mathcal{E})}(S)$ to be nonzero, S must distinguish $\mathbf{Sim}^{\texttt{s-dfa}}_{\Pi_1(\mathcal{E})}$ from $\mathbf{Expt}^{\texttt{s-dfa}}_{\Pi_1(\mathcal{E})}$.

We construct an IND-CPA adversary U that, on input $\hat{pk}$, sets $pk' \leftarrow \hat{pk}$ and uses its own oracle $\mathsf{Enc}^{\hat{b}}_{\hat{pk}}$ to choose between running $\mathbf{Expt}^{\texttt{s-dfa}}_{\Pi_1(\mathcal{E})}$ and $\mathbf{Sim}^{\texttt{s-dfa}}_{\Pi_1(\mathcal{E})}$ for S by setting $\rho \leftarrow \mathsf{Enc}^{\hat{b}}_{\hat{pk}}(0, r)$ in c107 and c115. (Aside from this, U performs $\mathbf{Expt}^{\texttt{s-dfa}}_{\Pi_1(\mathcal{E})}$ faithfully, using $(pk, sk) \leftarrow \mathsf{Gen}(1^\kappa)$ and $(sk_1, sk_2) \leftarrow \mathsf{Share}(sk)$ it generates itself.) U then returns $\hat{b}' = 1$ if S_2 outputs $b' = b$ and $\hat{b}' = 0$, otherwise. Then,

$$\begin{aligned}
&\mathbb{P}\left(\mathbf{Expt}^{\texttt{ind-cpa}}_{\mathcal{E}}(U) = 1\right) \\
&= \frac{1}{2}\mathbb{P}\left(\mathbf{Expt}^{\texttt{s-dfa}}_{\Pi_1(\mathcal{E})}(S) = 1\right) + \frac{1}{2}\mathbb{P}\left(\mathbf{Sim}^{\texttt{s-dfa}}_{\Pi_1(\mathcal{E})}(S) = 0\right) \\
&= \frac{1}{2}\left(\frac{1}{2} + \frac{1}{2}\mathbf{Adv}^{\texttt{s-dfa}}_{\Pi_1(\mathcal{E})}(S)\right) + \frac{1}{4} \\
&= \frac{1}{2} + \frac{1}{4}\mathbf{Adv}^{\texttt{s-dfa}}_{\Pi_1(\mathcal{E})}(S)
\end{aligned}$$

and so $\mathbf{Adv}^{\texttt{ind-cpa}}_{\mathcal{E}}(U) = \frac{1}{2}\mathbf{Adv}^{\texttt{s-dfa}}_{\Pi_1(\mathcal{E})}(S)$.

Note that U makes $\ell + 1$ oracle queries and runs in time $t' = t + t_{\mathsf{Gen}} + t_{\mathsf{Share}} + \ell m \cdot t_{\mathsf{Enc}}$, due to the need to generate (pk, sk), sk_2 and create the file ciphertexts $\langle c_{kj} \rangle_{k \in [\ell], j \in [m]}$. □

The second server adversary $S = (S_1, S_2)$ attacks the file ciphertexts $\langle c_{kj} \rangle_{k \in [\ell], j \in [m]}$ as in experiment $\mathbf{Expt}^{\texttt{s-file}}_{\Pi_1(\mathcal{E})}$ shown in Fig. 3.4. S_1 produces two equal-length plaintext files $\langle \sigma_{0k} \rangle_{k \in [\ell]}$, $\langle \sigma_{1k} \rangle_{k \in [\ell]}$ and a DFA M. S_2 receives the ciphertexts $\langle c_{kj} \rangle_{k \in [\ell], j \in [m]}$ for file $\langle \sigma_{bk} \rangle_{k \in [\ell]}$ where b is chosen randomly. S_2 is also given oracle access to $\mathsf{clientOr}(pk,\ sk_1,\ pk',\ M)$. Eventually S_2 outputs a bit b', and $\mathbf{Expt}^{\texttt{s-file}}_{\Pi_1(\mathcal{E})}(S) = 1$ iff $b' = b$. We say the *advantage* of S is $\mathbf{Adv}^{\texttt{s-file}}_{\Pi_1(\mathcal{E})}(S) = 2 \cdot \mathbb{P}\left(\mathbf{Expt}^{\texttt{s-file}}_{\Pi_1(\mathcal{E})}(S) = 1\right) - 1$ and then $\mathbf{Adv}^{\texttt{s-file}}_{\Pi_1(\mathcal{E})}(t, \ell, n, m) = \max_S \mathbf{Adv}^{\texttt{s-file}}_{\Pi_1(\mathcal{E})}(S)$ where the maximum is taken over all adversaries $S = (S_1, S_2)$ taking time t and producing (from S_1) files of ℓ symbols and a DFA of n states and alphabet of size m. We prove the following theorem:

Theorem 2. *For* $t' = t + t_{\mathsf{Gen}} + t_{\mathsf{Share}} + \ell m \cdot t_{\mathsf{Enc}}$,

$$\mathbf{Adv}^{\texttt{s-file}}_{\Pi_1(\mathsf{Pai})}(t, \ell, n, m) \leq 2\mathbf{Adv}^{\texttt{ind-cpa}}_{\mathsf{Pai}}(t', \ell + 1) + \mathbf{Adv}^{\texttt{ind-cpa}}_{\mathsf{Pai}}(t', \ell m)$$

Proof. Let $\mathbf{Expt}^{\texttt{s-file-}0}_{\Pi_1(\mathsf{Pai})}$ denote experiment $\mathbf{Expt}^{\texttt{s-file}}_{\Pi_1(\mathsf{Pai})}$ with b fixed at $b = 0$, and let $\mathbf{Expt}^{\texttt{s-file-}1}_{\Pi_1(\mathsf{Pai})}$ denote the experiment $\mathbf{Expt}^{\texttt{s-file}}_{\Pi_1(\mathsf{Pai})}$ with b fixed at $b = 1$. Consider a simulation $\mathbf{Sim}^{\texttt{s-file-}0}_{\Pi_1(\mathsf{Pai})}$ for $\mathbf{Expt}^{\texttt{s-file-}0}_{\Pi_1(\mathsf{Pai})}$ that differs only by simulating $\mathsf{clientOr}$ so as to substitute all ciphertexts produced with pk' with encryptions of a random injection π independent of π_1 it chose as in c109 (i.e., $\rho \leftarrow \mathsf{Enc}_{pk'}(\pi)$, $\pi \overset{\$}{\leftarrow} \mathsf{Injs}(Q \to \mathbb{R})$ in c107 and c115). Proceeding as in the proof of Theorem 1, we construct an IND-CPA adversary U_0 that uses its own oracle $\mathsf{Enc}^{\hat{b}}_{\hat{pk}}$ to choose between running $\mathbf{Expt}^{\texttt{s-file-}0}_{\Pi_1(\mathsf{Pai})}$ and $\mathbf{Sim}^{\texttt{s-file-}0}_{\Pi_1(\mathsf{Pai})}$ for S, i.e., by setting $pk' \leftarrow \hat{pk}$ and $\rho \leftarrow \mathsf{Enc}^{\hat{b}}_{\hat{pk}}(\pi_1, \pi)$ in c107 and c115. (Aside from this, U_0 performs $\mathbf{Expt}^{\texttt{s-file-}0}_{\Pi_1(\mathsf{Pai})}$ faithfully, using $(pk, sk) \leftarrow \mathsf{Gen}(1^\kappa)$ and $(sk_1, sk_2) \leftarrow \mathsf{Share}(sk)$ it generates itself.) U_0 returns $\hat{b}' = 0$ if $b' = b$ and $\hat{b}' = 1$, otherwise. Then,

$$\begin{aligned} &1 + \mathbf{Adv}^{\texttt{ind-cpa}}_{\mathsf{Pai}}(U_0) = 2 \cdot \mathbb{P}\left(\mathbf{Expt}^{\texttt{ind-cpa}}_{\mathsf{Pai}}(U_0) = 1\right) = \\ &\mathbb{P}\left(\mathbf{Expt}^{\texttt{s-file-}0}_{\Pi_1(\mathsf{Pai})}(S) = 1\right) + \mathbb{P}\left(\mathbf{Sim}^{\texttt{s-file-}0}_{\Pi_1(\mathsf{Pai})}(S) = 0\right) \end{aligned} \tag{3.2}$$

Now consider a simulation $\mathbf{Sim}^{\texttt{s-file-}1}_{\Pi_1(\mathsf{Pai})}$ for $\mathbf{Expt}^{\texttt{s-file-}1}_{\Pi_1(\mathsf{Pai})}$ that again differs only by simulating $\mathsf{clientOr}$ so as to substitute all ciphertexts produced with pk' with encryptions of a random injection. As above, we construct an IND-CPA adversary U_1 that uses its own oracle $\mathsf{Enc}^{\hat{b}}_{\hat{pk}}$ to choose between running $\mathbf{Expt}^{\texttt{s-file-}1}_{\Pi_1(\mathsf{Pai})}$ and $\mathbf{Sim}^{\texttt{s-file-}1}_{\Pi_1(\mathsf{Pai})}$ for S, i.e., by setting $pk' \leftarrow \hat{pk}$ and $\rho \leftarrow \mathsf{Enc}^{\hat{b}}_{\hat{pk}}(\pi, \pi_1)$, where $\pi \xleftarrow{\$} \mathsf{Injs}(Q \to \mathbb{R})$ in $\mathsf{c107}$ and $\mathsf{c115}$. U_1 returns $\hat{b}' = 1$ if $b' = b$ and $\hat{b}' = 0$, otherwise. Then,

$$1 + \mathbf{Adv}^{\texttt{ind-cpa}}_{\mathsf{Pai}}(U_1) = 2 \cdot \mathbb{P}\left(\mathbf{Expt}^{\texttt{ind-cpa}}_{\mathsf{Pai}}(U_1) = 1\right) =$$
$$\mathbb{P}\left(\mathbf{Sim}^{\texttt{s-file-1}}_{\Pi_1(\mathsf{Pai})}(S) = 0\right) + \mathbb{P}\left(\mathbf{Expt}^{\texttt{s-file-1}}_{\Pi_1(\mathsf{Pai})}(S) = 1\right) \tag{3.3}$$

Finally, consider an adversary U that uses its oracle $\mathsf{Enc}^{\hat{b}}_{\hat{pk}}$ to choose between running $\mathbf{Sim}^{\texttt{s-file-0}}_{\Pi_1(\mathsf{Pai})}$ and $\mathbf{Sim}^{\texttt{s-file-1}}_{\Pi_1(\mathsf{Pai})}$ for S. Specifically, on input $\hat{pk} = \langle N, g\rangle$, U generates $d_2 \xleftarrow{\$} \mathbb{Z}_{N^2}$ and invokes $S_1(\hat{pk}, sk_2)$ where $sk_2 = \langle N, g, d_2\rangle$. Upon receiving $\langle \sigma_{0k}\rangle_{k\in[\ell]}$ and $\langle \sigma_{1k}\rangle_{k\in[\ell]}$ from S_1, U sets $c_{kj} \leftarrow \mathsf{Enc}^{\hat{b}}_{\hat{pk}}((\sigma_{0k})^j, (\sigma_{1k})^j)$. Additionally, in the simulation of $\mathsf{clientOr}$, U selects $r \xleftarrow{\$} \mathbb{R}$ and sets $\alpha \leftarrow \mathsf{Enc}_{\hat{pk}}(r)$ in $\mathsf{c104}$ and $\mathsf{c112}$ and $\beta \leftarrow g^r\alpha^{-d_2} \bmod N^2$ in $\mathsf{c106}$ and $\mathsf{c114}$, so that $\alpha^{d_2}\beta \equiv g^r \bmod N^2$. ($U$ also generates pk' itself and constructs all encryptions for pk' as encryptions of a random injection.) When S_2 outputs b', U outputs b' as $\hat{b}'$. Then,

$$1 + \mathbf{Adv}^{\texttt{ind-cpa}}_{\mathsf{Pai}}(U) =$$
$$2 \cdot \mathbb{P}\left(\mathbf{Expt}^{\texttt{ind-cpa}}_{\mathsf{Pai}}(U) = 1\right) = 2 \cdot \mathbb{P}\left(\mathbf{Sim}^{\texttt{s-file}}_{\Pi_1(\mathsf{Pai})}(S) = 1\right) =$$
$$\mathbb{P}\left(\mathbf{Sim}^{\texttt{s-file-0}}_{\Pi_1(\mathsf{Pai})}(S) = 1\right) + \mathbb{P}\left(\mathbf{Sim}^{\texttt{s-file-1}}_{\Pi_1(\mathsf{Pai})}(S) = 1\right) \tag{3.4}$$

Adding (3.2), (3.3) and (3.4), we get

$$\begin{aligned}
&3 + \mathbf{Adv}^{\texttt{ind-cpa}}_{\mathsf{Pai}}(U_0) + \mathbf{Adv}^{\texttt{ind-cpa}}_{\mathsf{Pai}}(U) + \mathbf{Adv}^{\texttt{ind-cpa}}_{\mathsf{Pai}}(U_1)\\
&= \mathbb{P}\left(\mathbf{Expt}^{\texttt{s-file-0}}_{\Pi_1(\mathsf{Pai})}(S) = 1\right) + \mathbb{P}\left(\mathbf{Sim}^{\texttt{s-file-0}}_{\Pi_1(\mathsf{Pai})}(S) = 0\right)\\
&\ + \mathbb{P}\left(\mathbf{Sim}^{\texttt{s-file-0}}_{\Pi_1(\mathsf{Pai})}(S) = 1\right) + \mathbb{P}\left(\mathbf{Sim}^{\texttt{s-file-1}}_{\Pi_1(\mathsf{Pai})}(S) = 1\right)\\
&\ + \mathbb{P}\left(\mathbf{Sim}^{\texttt{s-file-1}}_{\Pi_1(\mathsf{Pai})}(S) = 0\right) + \mathbb{P}\left(\mathbf{Expt}^{\texttt{s-file-1}}_{\Pi_1(\mathsf{Pai})}(S) = 1\right)\\
&= 2 \cdot \mathbb{P}\left(\mathbf{Expt}^{\texttt{s-file}}_{\Pi_1(\mathsf{Pai})}(S) = 1\right) + 2\\
&= 3 + \mathbf{Adv}^{\texttt{s-file}}_{\Pi_1(\mathsf{Pai})}(S)
\end{aligned}$$

The result then follows because each of U_0 and U_1 makes $\ell + 1$ oracle queries and runs in time $t' = t + t_{\mathsf{Gen}} + t_{\mathsf{Share}} + \ell m \cdot t_{\mathsf{Enc}}$ due to the need to generate (pk, sk) and sk_2, and create the file

```
Experiment $\mathbf{Expt}^{\texttt{c-file}}_{\Pi_1(\mathcal{E})}(C_1, C_2)$
    $(pk, sk) \leftarrow \mathsf{Gen}(1^\kappa)$
    $(sk_1, sk_2) \leftarrow \mathsf{Share}(sk)$
    $(pk', sk') \leftarrow \mathsf{Gen}(1^{\kappa+2})$
    $(\ell, \langle\sigma_{0k}\rangle_{k\in[\ell]}, \langle\sigma_{1k}\rangle_{k\in[\ell]}, M, \phi) \leftarrow C_1(pk, sk_1, pk')$
    if $M(\langle\sigma_{0k}\rangle_{k\in[\ell]}) \neq M(\langle\sigma_{1k}\rangle_{k\in[\ell]})$ then return $0$
    $b \xleftarrow{\$} \{0, 1\}$
    $m \leftarrow |M.\Sigma|$
    for $k \in [\ell], j \in [m]$
        $c_{kj} \leftarrow \mathsf{Enc}_{pk}((\sigma_{bk})^j)$
    $b' \leftarrow C_2^{\mathsf{serverOr}(pk, sk_2, M.\Sigma, \langle c_{kj}\rangle_{k\in[\ell], j\in[m]})}(\phi)$
    if $b' = b$
        then return $1$
        else return $0$
```

Figure 3.5: Experiment for proving file privacy of $\Pi_1(\mathcal{E})$ against client adversaries

ciphertexts $\langle c_{kj}\rangle_{k\in[\ell],j\in[m]}$. U makes ℓm oracle queries and runs in time $t + t_{\mathsf{Gen}} + t_{\mathsf{Share}} + \ell m \cdot t_{\mathsf{Enc}}$ for the same reason. □

3.2.3 Security Against Client Adversaries

In this section we show security of $\Pi_1(\mathcal{E})$ against honest-but-curious client adversaries and heuristically justify its security against malicious ones. Since the client has the DFA in its possession, privacy of the DFA against a client adversary is not a concern. The proof of security against the client therefore is concerned with the privacy of only the file. However, by the nature of what the protocol computes for the client — i.e., the final state of a DFA match on the file — the client can easily distinguish two files of its choosing simply by running the protocol correctly using a DFA that distinguishes between the two files it chose.

For this reason, we adapt the notion of indistinguishability to apply only to files that produce the same final state for the client's DFA. So, in the experiment $\mathbf{Expt}^{\texttt{c-file}}_{\Pi_1(\mathcal{E})}$ (Fig. 3.5) that we use to define file security against client adversaries, the adversary $C = (C_1, C_2)$ succeeds (i.e., $\mathbf{Expt}^{\texttt{c-file}}_{\Pi_1(\mathcal{E})}(C)$ returns 1) only if the two files $\langle\sigma_{0k}\rangle_{k\in[\ell]}$ and $\langle\sigma_{1k}\rangle_{k\in[\ell]}$ output by C_1 both drive the DFA M, also output by C_1, to the same final state (denoted $M(\langle\sigma_{0k}\rangle_{k\in[\ell]}) = M(\langle\sigma_{1k}\rangle_{k\in[\ell]})$).

This caveat aside, the experiment is straightforward: C_1 receives public key pk, private-key share sk_1, and another public key pk', and returns the two ℓ-symbol files (for ℓ of its choosing)

$\langle \sigma_{0k} \rangle_{k \in [\ell]}$ and $\langle \sigma_{1k} \rangle_{k \in [\ell]}$ and a DFA M. Depending on how b is then chosen, one of these files is encrypted using pk and then provided to the server, to which C_2 is given oracle access (denoted $\mathsf{serverOr}(pk, sk_2, M.\Sigma, \langle c_{kj} \rangle_{k \in [\ell], j \in [m]})$).

Adversary C_2 can invoke serverOr first with a message containing an integer n (i.e., with a message of the form m101), to which serverOr returns ℓ (m102). C_2 can then invoke serverOr up to $\ell + 1$ times. The first ℓ such invocations take the form α, β, ρ and correspond to messages of the form m103. Each such invocation elicits a response $\langle \mu_{\sigma i} \rangle_{\sigma \in \Sigma, i \in [n]}$ (i.e., of the form m104). The last client invocation is of the form α, β, ρ and corresponds to m105. This invocation elicits a response γ^* (i.e., m106). Malformed or extra queries are rejected by serverOr.

We show file privacy against *honest-but-curious* client adversaries $C = (C_1, C_2)$, i.e., C_2 invokes serverOr exactly as $\Pi_1(\mathcal{E})$ prescribes, using DFA M output by C_1. We define the advantage of C to be $\mathbf{hbcAdv}^{\texttt{c-file}}_{\Pi_1(\mathcal{E})}(C) = 2 \cdot \mathbb{P}\left(\mathbf{Expt}^{\texttt{c-file}}_{\Pi_1(\mathcal{E})}(C) = 1\right) - 1$ and $\mathbf{hbcAdv}^{\texttt{c-file}}_{\Pi_1(\mathcal{E})}(t, \ell, n, m) = \max_C \mathbf{Adv}^{\texttt{c-file}}_{\Pi_1(\mathcal{E})}(C)$ where the maximum is taken over honest-but-curious client adversaries C running in total time t and producing files of length ℓ and a DFA of n states over an alphabet of m symbols. We prove:

Theorem 3. *For* $t' = t + t_{\mathsf{Gen}} + \ell m \cdot t_{\mathsf{Enc}} + (\ell + 1) \cdot t_{\mathsf{Dec}}$,

$$\mathbf{hbcAdv}^{\texttt{c-file}}_{\Pi_1(\mathsf{Pai})}(t, \ell, n, m) \leq \mathbf{Adv}^{\texttt{ind-cpa}}_{\mathsf{Pai}}(t', \ell m(1 + n))$$

Proof. Given an adversary $C = (C_1, C_2)$ running in time t and selecting files of length ℓ symbols and a DFA of n states over an alphabet of m symbols, we construct an IND-CPA adversary U that demonstrates the theorem as follows. On input $\hat{pk} = \langle N, g \rangle$, U generates $(pk', sk') \leftarrow \mathsf{Gen}(1^{\kappa+2})$ and $d_1 \stackrel{\$}{\leftarrow} \mathbb{Z}_{N^2}$, and invokes $C_1(\hat{pk}, sk_1, pk')$ where $sk_1 = \langle N, g, d_1 \rangle$ to obtain (ℓ, $\langle \sigma_{0k} \rangle_{k \in [\ell]}$, $\langle \sigma_{1k} \rangle_{k \in [\ell]}$, M, ϕ), where $M = \langle Q$, Σ, q_{init}, $\delta \rangle$ is a DFA. Note that d_1 is chosen from a distribution that is statistically indistinguishable from that from which d_1 is chosen in the real system. For $k \in [\ell]$ and $j \in [m]$, U sets $c_{kj} \leftarrow \mathsf{Enc}^{\hat{b}}_{\hat{pk}}((\sigma_{0k})^j, (\sigma_{1k})^j)$.

U then invokes $C_2(\phi)$ and simulates responses to C_2's queries to serverOr as follows (ignoring malformed invocations). In response to the initial query n, the adversary U returns ℓ and, in preparation for the subsequent serverOr invocations by C_2, sets $q_0 \leftarrow q_{\mathsf{init}}$ and $q_1 \leftarrow q_{\mathsf{init}}$. For the k-th query of the form α, β, ρ ($0 \leq k < \ell$), the adversary U sets $\pi \leftarrow \mathsf{Dec}_{sk'}(\rho)$, $\gamma_0 \leftarrow \pi(q_0)$, and

$\gamma_1 \leftarrow \pi(q_1)$, and then sets $\mu_{\sigma i} \leftarrow \mathsf{Enc}^{\hat{b}}_{\hat{pk}}(((\gamma_0)^i \cdot_{\mathbb{R}} \Lambda_\sigma(\sigma_{0k}), ((\gamma_1)^i \cdot_{\mathbb{R}} \Lambda_\sigma(\sigma_{1k}))$ for $\sigma \in \Sigma$ and $i \in [n]$. After this, U updates $q_0 \leftarrow \delta(q_0, \sigma_{0k})$ and $q_1 \leftarrow \delta(q_1, \sigma_{1k})$, and returns $\langle \mu_{\sigma i}, \rangle_{\sigma \in \Sigma, k \in [n]}$ to C_2. For the last query α, β, ρ, adversary U computes $\pi \leftarrow \mathsf{Dec}_{sk'}(\rho)$ and returns $\gamma^* = \pi(q_0)$ $(= \pi(q_1))$ to C_2. When C_2 outputs b', U outputs b', as well.

This simulation is statistically indistinguishable from the real system provided that C is honest-but-curious, and so ignoring terms that are negligible in κ, $\mathbf{hbcAdv}^{\texttt{c-file}}_{\Pi_1(\mathsf{Pai})}(C) = \mathbf{Adv}^{\texttt{ind-cpa}}_{\mathsf{Pai}}(U)$. Note that U runs in $t' = t + t_{\mathsf{Gen}} + \ell m \cdot t_{\mathsf{Enc}} + (\ell + 1) \cdot t_{\mathsf{Dec}}$ due to the need to generate (pk', sk') and sk_1, to create the file ciphertexts $\langle c_{kj} \rangle_{k \in [\ell], j \in [m]}$, and to perform $\ell + 1$ Pai decryption in the simulation. U makes nm oracle queries in order to respond to each of the ℓ oracle queries following the first, plus an additional ℓm queries to create $\langle c_{kj} \rangle_{k \in [\ell], j \in [m]}$. □

We have found extending this result to fully malicious client adversaries to be difficult for two reasons. First, $\mathbf{Expt}^{\texttt{c-file}}_{\Pi_1(\mathcal{E})}$ does not make sense for a malicious client, since C_2 is not bound to use the DFA M output by C_1. As such, C_2 can use a different DFA — in particular, one that enables it to distinguish between the files output by C_1. Second, even ignoring the final state γ^* sent back to the client, we have been unable to reduce the ability of the client adversary to distinguish between two files on the basis of m104 messages to breaking the IND-CPA security of $\mathcal{E}$; intuitively, the difficulty derives from the simulator's inability to decrypt α values provided by C_2. (The ciphertext ρ enables the simulator to "track" the plaintext of α in the honest-but-curious case, but ρ might contain useless information in the malicious case.)

Nevertheless, since *only* ciphertexts for which the client does not hold the decryption key are sent to the client in those messages, we are confident in conjecturing that our protocol leaks no information to even a malicious client about the file, beyond what it gains from the protocol output γ^*, assuming $\mathcal{E}$ is IND-CPA secure. Of course, the above proof difficulties for a malicious client could be ameliorated by introducing zero-knowledge proofs to the protocol to enforce correct behavior, but with considerable added expense to the protocol. Instead, in Section 3.4 we introduce more novel (albeit still heuristic) approaches to detecting client (or server) misbehavior in our setting.

3.3 An Alternative Protocol

The second protocol we present has the same goals as $\Pi_1(\mathcal{E})$ but incurs less communication costs. Specifically, whereas the communication cost of $\Pi_1(\mathcal{E})$ is $O(\kappa\ell nm)$ bits, the protocol we present in this section, called $\Pi_2(\mathcal{E})$, sends only $O(\kappa\ell(n+m))$ bits. $\Pi_2(\mathcal{E})$ accomplishes this in part by exploiting a cryptosystem that is additively homomorphic and that offers the ability to homomorphically "multiply" ciphertexts once. That is, the cryptosystem supports a new operator $\odot_{pk}$ that satisfies $\mathsf{Dec}_{sk}(\mathsf{Enc}_{pk}(m_1) \odot_{pk} \mathsf{Enc}_{pk}(m_2)) = m_1 \cdot_{\mathbb{R}} m_2$, but the result of a $\odot_{pk}$ operation (or any other ciphertext resulting from $+_{pk}$ or $\cdot_{pk}$ operations in which it is used) cannot be used in a $\odot_{pk}$ operation. After we present our protocol, we will discuss various options for instantiating this encryption scheme within it.

Protocol $\Pi_2(\mathcal{E})$ is shown in Fig. 3.6. Note that the input arguments to both the client and the server are identical to those in $\Pi_1(\mathcal{E})$. The structure of the protocol is also very similar to $\Pi_1(\mathcal{E})$, with the only differences being in how the server performs each loop iteration (s204–s212) and how the client forms the new encrypted DFA state α (c212–c216). We now summarize the primary innovations represented by these differences.

After the k-th m203 message, the server constructs an encryption Ψ_σ of $\Lambda_\sigma(\sigma_k)$ (s206). Rather than computing $\mu_{\sigma i} \leftarrow \gamma^i \cdot_{pk} \Psi_\sigma$, however, the server sends $\langle \Psi_\sigma \rangle_{\sigma \in \Sigma}$ to the client in m204. Each $\mu_{\sigma i}$ is then built at the client, instead (c212–c214), which is the main reason we get better communication efficiency.

Since each $\mu_{\sigma i}$ is built at the client, the server must send γ in m204. To hide the current DFA state from the client, the server blinds γ with a random $r \in \mathbb{R}$ (s208–s209) before returning it. So, the client needs to accommodate r without knowing it when performing the DFA state transition. The client cannot perform the polynomial evaluation using the $f(x,y)$ it constructed (c211) on the $\langle \mu_{\sigma i} \rangle_{\sigma \in \Sigma, i \in [n]}$ as in $\Pi_1(\mathcal{E})$ since $f(x,y)$ is designed for an input $q \in \pi_0(Q)$, not $q+r$. To overcome this, the client constructs a shifted polynomial $f'(x,y)$ such that $f'(q+r,\sigma) = f(q,\sigma)$ for all $q \in \pi_0(Q)$, and so $f'(x,y)$ will correctly translate the blinded input to the next DFA state. What is left to describe is how to construct $f'(x,y)$.

client$(pk, sk_1, pk', \langle Q, \Sigma, \delta, q_{\mathsf{init}} \rangle)$		server$(pk, sk_2, \Sigma, \langle c_{kj} \rangle_{k \in [\ell], j \in [m]})$
c201. $n \leftarrow \lvert Q \rvert, m \leftarrow \lvert \Sigma \rvert$		s201. $m \leftarrow \lvert \Sigma \rvert$
c202. $\pi_0 \leftarrow \mathbf{I}$		s202. $\langle \lambda_{\sigma j} \rangle_{\sigma \in \Sigma, j \in [m]}$
c203. $\pi_1 \overset{\$}{\leftarrow}$ Injs$(Q \to \mathbb{R})$		$\quad \leftarrow$ Lagrange(Σ)
c204. $\alpha \leftarrow$ Enc$_{pk}(\pi_1(q_{\mathsf{init}}))$		
	m201. $\xrightarrow{n}$	
	m202. $\xleftarrow{\ell}$	
c205. **for** $k \leftarrow 0 \ldots \ell - 1$		s203. **for** $k \leftarrow 0 \ldots \ell - 1$
c206. $\quad \beta \leftarrow$ Dec$^1_{sk_1}(\alpha)$		
c207. $\quad \rho \leftarrow$ Enc$_{pk'}(\pi_1)$		
	m203. $\xrightarrow{\alpha, \beta, \rho}$	
		s204. $\quad \gamma \leftarrow$ Dec$^2_{sk_2}(\alpha, \beta)$
c208. $\quad \pi_0 \leftarrow \pi_1$		s205. $\quad$ **for** $\sigma \in \Sigma$
c209. $\quad \pi_1 \overset{\$}{\leftarrow}$ Injs$(Q \to \mathbb{R})$		s206. $\quad\quad \Psi_\sigma \leftarrow \sum_{j=0}^{m-1}{}^{pk} \lambda_{\sigma j} \cdot_{pk} c_{kj}$
c210. $\quad \delta' \leftarrow$ Blind(δ, π_0, π_1)		s207. $\quad$ **endfor**
c211. $\quad \langle a_{\sigma i} \rangle_{\sigma \in \Sigma, i \in [n]}$		s208. $\quad r \overset{\$}{\leftarrow} \mathbb{R}$
$\quad\quad \leftarrow$ ToPoly(Q, Σ, δ')		s209. $\quad \gamma \leftarrow \gamma +_{\mathbb{R}} r$
		s210. $\quad$ **for** $i \in [n]$
		s211. $\quad\quad \nu_i \leftarrow$ Enc$_{pk}(r^i)$
		s212. $\quad$ **endfor**
	m204. $\xleftarrow{\gamma, \langle \Psi_\sigma \rangle_{\sigma \in \Sigma}, \langle \nu_i \rangle_{i \in [n]}}$	
c212. $\quad$ **for** $\sigma \in \Sigma, i \in [n]$		
c213. $\quad\quad \mu_{\sigma i} \leftarrow \gamma^i \cdot_{pk} \Psi_\sigma$		
c214. $\quad$ **endfor**		
c215. $\quad \langle \hat{a}'_{\sigma i} \rangle_{\sigma \in \Sigma, i \in [n]}$		
$\quad\quad \leftarrow$ Shift$(\langle \nu_i \rangle_{i \in [n]}, \langle a_{\sigma i} \rangle_{\sigma \in \Sigma, i \in [n]})$		
c216. $\quad \alpha \leftarrow \sum_{\sigma \in \Sigma}{}^{pk} \sum_{i=0}^{n-1}{}^{pk} \hat{a}'_{\sigma i} \odot_{pk} \mu_{\sigma i}$		
c217. **endfor**		s213. **endfor**
c218. $\beta \leftarrow$ Dec$^1_{sk_1}(\alpha)$		
c219. $\rho \leftarrow$ Enc$_{pk'}(\pi_1)$		
	m205. $\xrightarrow{\alpha, \beta, \rho}$	
		s214. $\gamma^* \leftarrow$ Dec$^2_{sk_2}(\alpha, \beta)$
	m206. $\xleftarrow{\gamma^*}$	
c220. **return** $\pi_1^{-1}(\gamma^*)$		

Figure 3.6: Protocol $\Pi_2(\mathcal{E})$, described in Section 3.3

If we set $f'(x,y) = \sideset{}{^{\mathbb{R}}}\sum_{\sigma\in\Sigma} (f'_\sigma(x) \cdot_{\mathbb{R}} \Lambda_\sigma(y))$ where $f'_\sigma(x) = \sideset{}{^{\mathbb{R}}}\sum_{i=0}^{n-1} a'_{\sigma i} \cdot_{\mathbb{R}} x^i$, then it suffices if $f'_\sigma(x +_{\mathbb{R}} r) = f_\sigma(x)$ for all $\sigma \in \Sigma$. Note that

$$\begin{aligned} f_\sigma(x -_{\mathbb{R}} r) &= \sideset{}{^{\mathbb{R}}}\sum_{i=0}^{n-1} a_{\sigma i} \cdot_{\mathbb{R}} (x -_{\mathbb{R}} r)^i = \sideset{}{^{\mathbb{R}}}\sum_{i=0}^{n-1} a_{\sigma i} \cdot_{\mathbb{R}} \sideset{}{^{\mathbb{R}}}\sum_{i'=0}^{i} \binom{i}{i'} \cdot_{\mathbb{R}} x^{i-i'} \cdot_{\mathbb{R}} (-_{\mathbb{R}} r)^{i'} \qquad (3.5) \\ &= \sideset{}{^{\mathbb{R}}}\sum_{i=0}^{n-1} \left(\sideset{}{^{\mathbb{R}}}\sum_{i'=0}^{n-1-i} a_{\sigma(i+i')} \cdot_{\mathbb{R}} \binom{i+i'}{i'} \cdot_{\mathbb{R}} (-_{\mathbb{R}} r)^{i'} \right) \cdot_{\mathbb{R}} x^i \end{aligned}$$

where Eqn. 3.5 follows from the binomial theorem. Therefore, setting

$$a'_{\sigma i} \leftarrow \sideset{}{^{\mathbb{R}}}\sum_{i'=0}^{n-1-i} a_{\sigma(i+i')} \cdot_{\mathbb{R}} \binom{i+i'}{i'} \cdot_{\mathbb{R}} (-_{\mathbb{R}} 1)^{i'} \cdot_{\mathbb{R}} r^{i'} \qquad (3.6)$$

ensures $f'_\sigma(x +_{\mathbb{R}} r) = f_\sigma(x)$ and so $f'(x +_{\mathbb{R}} r, \sigma) = f(x, \sigma)$.

The client knows all the terms in Eqn. 3.6 except $r^{i'}$. That is exactly the reason the server sends in m204 the ciphertext ν_i of r^i, for each $i \in [n]$ (see s211). The client can then calculate a ciphertext $\hat{a}'_{\sigma i}$ of the coefficient of x^i in f'_σ by using the additive homomorphic property of the encryption scheme :

$$\hat{a}'_{\sigma i} \leftarrow \sideset{}{^{pk}}\sum_{i'=0}^{n-1-i} \left(a_{\sigma(i+i')} \cdot_{\mathbb{R}} \binom{i+i'}{i'} \cdot_{\mathbb{R}} (-_{\mathbb{R}} 1)^{i'} \right) \cdot_{pk} \nu_{i'} \qquad (3.7)$$

In our pseudocode, the calculations Eqn. 3.7 are encapsulated within the operation $\langle \hat{a}'_{\sigma i} \rangle_{\sigma\in\Sigma, i\in[n]} \leftarrow \mathsf{Shift}(\langle \nu_i \rangle_{i\in[n]}, \langle a_{\sigma i} \rangle_{\sigma\in\Sigma, i\in[n]})$ on line c215.

After the client obtains $\langle \hat{a}'_{\sigma i} \rangle_{\sigma\in\Sigma, i\in[n]}$ and $\langle \mu_{\sigma i} \rangle_{\sigma\in\Sigma, i\in[n]}$, it performs polynomial evaluation at step c216 to assemble the ciphertext of the next DFA state by taking advantage of the one multiplication homomorphism of the cryptosystem. This is where the additional homomorphism helps to achieve much better communication complexity.

The privacy of the file and DFA from server adversaries and the privacy of the file from client adversaries can be proved for $\Pi_2(\mathcal{E})$ very similarly to how they are proved for $\Pi_1(\mathcal{E})$. In fact, Theorems 1–3 hold for $\Pi_2(\mathcal{E})$ unchanged, once instantiated with a suitable encryption scheme $\mathcal{E}$. That said, certain choices of $\mathcal{E}$ can require that the protocol be adapted, as discussed below.

Instantiating $\mathcal{E}$ Protocol $\Pi_2(\mathcal{E})$ requires an additively homomorphic encryption scheme $\mathcal{E}$ that also supports the "one time" homomorphic multiplication operator $\odot_{pk}$. Perhaps the most well-known such cryptosystem is due to Boneh, Goh and Nissim [17], and moreover, this cryptosystem also supports two-party decryption with a cost comparable to regular decryption [17]. The primary difficulty in instantiating $\mathcal{E}$ with this cryptosystem, however, is that decryption — and specifically in $\Pi_2(\mathcal{E})$, the operation $\mathsf{Dec}^2_{sk_2}$ — requires computing a discrete logarithm in a large group, which is generally intractable. That said, if the ciphertext is known to encode one of a small number of possible plaintexts, then $\mathsf{Dec}^2_{sk_2}$ can be adapted to test the ciphertext for each of these plaintexts efficiently. As such, to adapt $\Pi_2(\mathcal{E})$ to employ this cryptosystem, we can augment messages m203 and m205 with $\pi_1(Q)$ (listed in random order), for the injection π_1 at the time the message is sent. This would permit the server to perform $\mathsf{Dec}^2_{sk_2}(\alpha, \beta)$ in lines s204, s214 by testing for these n possible plaintexts. It does, however, have the unfortunate side effect of enabling our proofs for the analogs of Theorems 1 and 2 for $\Pi_2(\mathcal{E})$ to go through only for honest-but-curious server adversaries. $\Pi_2(\mathcal{E})$ instantiated in this way still appears to be secure even against malicious server adversaries, though at this point we can claim this only heuristically.

Two other possibilities for instantiating $\mathcal{E}$ in $\Pi_2(\mathcal{E})$ are due to Gentry, Halevi and Vaikuntanathan [34][2] and Lauter, Naehrig, and Vaikuntanathan [51]. The primary challenge posed by these cryptosystems is that two-party decryption algorithms for them have not been investigated. Each of these schemes is amenable to sharing its private key securely, after which decryption can be performed using generic two-party computation [74, 7]. These instantiations retain $\Pi_2(\mathcal{E})$'s provable security against malicious server adversaries (i.e., the analogs of Theorems 1 and 2), but $\Pi_2(\mathcal{E})$ instantiated this way may be less cost-efficient than $\Pi_1(\mathsf{Pai})$ for many values of n and m.[3] Of course, customized two-party decryption algorithms for these cryptosystems could restore the efficiency of $\Pi_2(\mathcal{E})$, suggesting a useful open problem for the community.

[2] Because we require the plaintext ring to be commutative, we would restrict the plaintext space of the Gentry et al. cryptosystem to diagonal square matrices, versus the arbitrary square matrices over which it is defined.

[3] For example, for the Gentry et al. scheme, a "garbled" arithmetic circuit [7] for secure two-party decryption using additively shared keys would be of size $O(\kappa^6 \log^5(n+m))$ bits.

3.4 Heuristics to Detect Misbehavior

In this section we describe simple extensions to our protocols to detect client or server misbehavior. The detection ability offered by these techniques is only heuristic, but they provide a practical deterrent to misbehavior and, at least as importantly, highlight possibilities outside standard techniques (zero-knowledge proofs) that might be brought to bear to detect misbehavior in data outsourcing situations.

Detecting server **misbehavior** We showed in Section 3.2.2 that both the file privacy and the client's DFA privacy are protected against an arbitrarily malicious server. That said, a malicious server could cause the protocol to return an incorrect result by undetectably executing the protocol incorrectly. Here we describe a defense that, while offering weak guarantees, gives insight into new opportunities provided in the cloud outsourcing setting studied in this paper.

The central idea is that in addition to the authentic encrypted file, the data owner also stores at the server (i) another "decoy" encrypted file of the same length as the authentic file and (ii) the plaintext of the decoy file, digitally signed by the data owner. However, the server is not told which one of the two encrypted files is the decoy. When a client wants to evaluate a DFA M on the (authentic) file, it executes two instances of the protocol in parallel with the server on each of the two encrypted files, while also retrieving (and authenticating, by its digital signature) the plaintext of the decoy file. If the client's DFA when applied to the plaintext of the decoy file evaluates to state q, then the client checks that at least one of the two protocol executions results in q. If neither outcome is q, then it detects that the server has behaved incorrectly. (Of course, if the client divulges when it has detected the server misbehaving, then this might enable the server to infer which of the encrypted files is the decoy, though the client could nevertheless report the misbehavior to the data owner outside the view of the server.)

A malicious server could try to guess which file is the decoy and execute the protocol faithfully on that file, while misbehaving on the other one to alter the result. Obviously the chance it guesses correctly is $\frac{1}{2}$. A server could also misbehave for both files, hoping that one of the protocol executions results in the correct final state for the decoy file. The probability of succeeding in this attack is a function of the decoy file and of the specific DFA that the client is evaluating. To improve the probability of detecting a misbehaving server, the client could also create more DFA queries to evaluate on both files. Moreover, additional decoy files could be stored at the server to increase the chance that a misbehaving server is detected.

Detecting client **misbehavior** A similar but slightly more involved technique could be used to heuristically detect client misbehavior in our protocols. In this technique, at the beginning of the protocol in which the client will use DFA $\langle Q, \Sigma, \delta, q_{\mathsf{init}}\rangle$, the server creates and sends to the client another DFA $\langle Q, \Sigma', \delta', q_{\mathsf{init}}\rangle$ where $\Sigma' \cap \Sigma = \emptyset$, i.e., another DFA with the same states and the same initial state but a different (and nonoverlapping) alphabet. Note that to create this DFA, the server need only know Q and q_{init}, which in the absence of δ reveal nothing about the pattern for which the client is searching (aside from the number n, which is conveyed to the server in the protocol already). The client then executes the protocol using the combined DFA $\langle Q, \Sigma \cup \Sigma', \delta \cup \delta', q_{\mathsf{init}}\rangle$.[4] As above, the client runs two instances of the protocol in parallel: the server uses the authentic file in one instance; in the other, it creates and uses another file of the same length but consisting of characters in Σ'. After the protocol completes, the client sends the final states back to the server, which checks to be sure that the pair of final states include the result of applying $\langle Q, \Sigma', \delta', q_{\mathsf{init}}\rangle$ to the file it created before telling the client which of the pair of states is the correct result.[5]

This technique for detecting client misbehavior relies on the inability of the client to detect which of the two files consists of elements of Σ and which consists of elements of Σ'— a property that we argued heuristically in Section 3.2.3 holds against a malicious client. It also depends on the file and DFA created by the server; as in the defense against server misbehavior above, this can be strengthened with multiple DFAs and files.

[4] Because doing so requires the server to hold ciphertexts $\langle c_{kj}\rangle_{k\in[\ell], j\in[|\Sigma|+|\Sigma'|]\setminus[|\Sigma|]}$, the data owner must additionally provide these ciphertexts when it stores the file.

[5] Divulging the final states to the server reveals minimal information about the pattern for which the client was searching (assuming the elements of Q are encoded as random elements of $\mathbb{R}$), specifically whether the final state was q_{init}. Even this leakage can be avoided by designing the DFA so it never returns to q_{init}.

CHAPTER 4

Ensuring File Authenticity in Private DFA Evaluation on Encrypted Files in the Cloud

In Section 3.4, we provided a heuristic method to detect server misbehavior in the protocols developed in Chapter 3. Aside from its heuristic nature, this method offers a weak detection probability that can be amplified only at the expense of running more protocol instances in parallel. In this chapter, we instead present a strengthened protocol, with rigorous security proofs (in the random oracle model), that allows the client to detect any misbehavior of the server within a single protocol run; in particular, the client can verify that the result of its DFA evaluation is based on the file stored there by the data owner, and in this sense the file and protocol result are authenticated to the client. Our protocol also protects the privacy of the file and the DFA from the server, and the privacy of the file (except the result of evaluating the DFA on it) from the client. A special case of our protocol solves private DFA evaluation on a private and authenticated file in the traditional two-party model, in which the file contents are known to the server. Our protocol provably achieves these properties for an arbitrarily malicious server and an honest-but-curious client, in the random oracle model. The rest of this chapter is structured as follows. We review our goals in Section 4.1 and detail our protocol and its security proof in Section 4.2. We then discuss extensions in Section 4.4.

4.1 Goals

Recall that a deterministic finite automaton M is a tuple $\langle Q, \Sigma, \delta, q_{\mathsf{init}}\rangle$ where Q is a set of $|Q| = n$ *states*; Σ is a set (*alphabet*) of $|\Sigma| = m$ *symbols*; $\delta : Q \times \Sigma \to Q$ is a transition function; and q_{init} is the initial state. (A DFA can also specify a function Δ indicating whether a state is an accepting state or not, like we defined in Chapter 3, although we ignore it in this chapter.) Our goal is to enable a client holding a DFA M to interact with a server holding a file ciphertext to evaluate M on the file plaintext. More specifically, the client should output the final state to which the file plaintext drives the DFA; i.e., if the plaintext file is a sequence $\langle\sigma_k\rangle_{k\in[\ell]}$ where $[\ell]$ denotes the set $\{0, 1, \ldots, \ell-1\}$ and where each $\sigma_k \in \Sigma$, then the client should output $\delta(\ldots\delta(\delta(q_{\mathsf{init}}, \sigma_0), \sigma_1), \ldots, \sigma_{\ell-1})$. We also permit the client to learn the file length ℓ and the server to learn the number of states n in the client's DFA. (Indeed, because the DFA output leaks $\log n$ bits about the file to the client, the server should know n to measure the leakage to the client and to limit the number of DFA queries the client is allowed, accordingly.) However, the client should learn nothing else about the file; the server should learn nothing else about the client's DFA and nothing about the file plaintext.

An additional goal of our protocols — and their main contribution over prior work — is to ensure that the client detects if the server deviates from the protocol. More specifically, we presume that a data owner stores the file ciphertext at the server, together with accompanying authentication data. We require that the client return the result of evaluating its DFA on the file stored by the data owner or else that the client detect the misbehavior of the server. We do not explicitly concern ourselves with misbehavior of the client, owing to the use cases outlined in Section 1.2 that involve a partially trusted third-party customer or service provider (e.g., antivirus vendor). That said, we believe our protocol to be heuristically secure against an arbitrarily malicious client.

4.2 Private DFA Evaluation on Signed and Encrypted Data

In this section we present a protocol meeting the goals described in Section 4.1: the client learns only the length of the file and the output of his DFA evaluation on the file stored at the server; the server learns only the number of states in the client's DFA and the length of the file; and the client detects any misbehavior by the server that would cause him to return an incorrect result. Again, we do not consider misbehavior of the client here; the client is honest-but-curious only. In this section we consider the file as static. The impact of file updates will be discussed in Section 4.3.

4.2.1 Preliminaries

Let "$\leftarrow$" denote assignment and "$s \overset{\$}{\leftarrow} S$" denote the assignment to s of a randomly chosen element of set S. Let κ be a security parameter. Let $\mathsf{ParamGen}$ be an algorithm that, on input 1^κ, produces $(p,\, \mathbb{G}_1,\, \mathbb{G}_2,\, g,\, e) \leftarrow \mathsf{ParamGen}(1^\kappa)$ where p is a prime; $\mathbb{G}_1$ and $\mathbb{G}_2$ are multiplicative groups of order p; g is a generator of $\mathbb{G}_1$; and $e : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$ is an efficiently computable bilinear map such that $e(P^u, Q^v) = e(P, Q)^{uv}$ for any $P, Q \in \mathbb{G}_1$ and any $u, v \in \mathbb{Z}_p^*$.

BLS Signatures Our protocol makes use of the Boneh-Lynn-Shacham (BLS) signature scheme [18]. Suppose $(p, \mathbb{G}_1, \mathbb{G}_2, g, e) \leftarrow \mathsf{ParamGen}(1^\kappa)$ and let H_1 be a hash function $\mathsf{H}_1 : \{0, 1\}^* \to \mathbb{G}_1$. The BLS scheme consists of a triple of algorithms $(\mathsf{BLSKeyGen}, \mathsf{BLSSign}, \mathsf{BLSVerify})$, defined as follows.

$\underline{\mathsf{BLSKeyGen}(p, \mathbb{G}_1, \mathbb{G}_2, g, e)}$: Select $x \overset{\$}{\leftarrow} \mathbb{Z}_p^*$. Return private signing key $\langle \mathbb{G}_1, x\rangle$ and public verification key $\langle p, \mathbb{G}_1, \mathbb{G}_2, g, e, h\rangle$ where $h \leftarrow g^x$.

$\underline{\mathsf{BLSSign}_{\langle \mathbb{G}_1, x\rangle}(m)}$: Return the signature $\mathsf{H}_1(m)^x$.

$\underline{\mathsf{BLSVerify}_{\langle p, \mathbb{G}_1, \mathbb{G}_2, g, e, h\rangle}(m, s)}$: Return true if $e(\mathsf{H}_1(m), h) = e(s, g)$ and false otherwise.

Paillier encryption Our scheme is built using the additively homomorphic encryption scheme due to Paillier [59]. This cryptosystem has a plaintext space $\mathbb{R}$ where $\langle \mathbb{R}, +_{\mathbb{R}}, \cdot_{\mathbb{R}}\rangle$ denotes a commutative ring. Specifically, this encryption scheme includes algorithms PGen, PEnc, and PDec where: PGen is a randomized algorithm that on input 1^κ outputs a public-key/private-key pair $(pek, pdk) \leftarrow \mathsf{PGen}(1^\kappa)$; PEnc is a randomized algorithm that on input public key pek and plaintext $m \in \mathbb{R}$ (where $\mathbb{R}$ can be determined as a function of pek) produces a ciphertext $c \leftarrow \mathsf{PEnc}_{pek}(m)$, where $c \in C_{pek}$ and C_{pek} is the ciphertext space determined by pek; and PDec is a deterministic algorithm that on input a private key pdk and ciphertext $c \in C_{pek}$ produces a plaintext $m \leftarrow \mathsf{PDec}_{pdk}(c)$ where $m \in \mathbb{R}$. In addition, $\mathcal{E}$ supports an operation $+_{pek}$ on ciphertexts such that for any public-key/private-key pair (pek, pdk), $\mathsf{PDec}_{pdk}(\mathsf{PEnc}_{pek}(m_1) +_{pek} \mathsf{PEnc}_{pek}(m_2)) = m_1 +_{\mathbb{R}} m_2$. Using $+_{pek}$, it is possible to implement $\cdot_{pek}$ for which $\mathsf{PDec}_{pdk}(m_2 \cdot_{pek} \mathsf{PEnc}_{pek}(m_1)) = m_1 \cdot_{\mathbb{R}} m_2$.

In Paillier encryption, the ring $\mathbb{R}$ is $\mathbb{Z}_N$, the ciphertext space $C_{\langle N,g\rangle}$ is $\mathbb{Z}_{N^2}^*$, and the relevant algorithms are as follows.

$\underline{\mathsf{PGen}(1^\kappa)}$: Choose random $\kappa/2$-bit strong primes p, p'; set $N \leftarrow pp'$; choose $g \in \mathbb{Z}_{N^2}^*$ with order a multiple of N; and return the public key $\langle N, g\rangle$ and private key $\langle N, g, \lambda(N)\rangle$ where $\lambda(N)$ is the Carmichael function of N.

$\underline{\mathsf{PEnc}_{\langle N,g\rangle}(m)}$: Select $x \overset{\$}{\leftarrow} \mathbb{Z}_N^*$ and return $g^m x^N \bmod N^2$.

$\underline{\mathsf{PDec}_{\langle N,g,\lambda(N)\rangle}(c)}$: Return $m = \frac{L(c^{\lambda(N)} \bmod N^2)}{L(g^{\lambda(N)} \bmod N^2)} \bmod N$, where L is a function that takes input elements from the set $\{u < N^2 \mid u \equiv 1 \bmod N\}$ and returns $L(u) = \frac{u-1}{N}$.

$\underline{c_1 +_{\langle N,g \rangle} c_2}$: Return $c_1 c_2 \bmod N^2$.

$\underline{m \cdot_{\langle N,g \rangle} c}$: Return $c^m \bmod N^2$.

We use $\sum_{pek}$ to denote summation using $+_{pek}$; $\sum_{\mathbb{R}}$ to denote summation using $+_{\mathbb{R}}$; and $\prod_{\mathbb{R}}$ to denote the product using $\cdot_{\mathbb{R}}$ of a sequence.

4.2.2 Initial Construction Without File Encryption

We denote the file stored at the server as consisting of characters $\sigma_0, \ldots, \sigma_{\ell-1}$, where each $\sigma_k \in \Sigma$. Prior to storing this file at the server, however, the data owner uses its private BLS signing key $\langle \mathbb{G}_1, x \rangle$ to produce $s_k \leftarrow \mathsf{BLSSign}_{\langle \mathbb{G}_1, x \rangle}(\sigma_k || k)$ for each $k \in [\ell]$ — i.e., a per-file-character signature that incorporates the position of the character in the file[1] — and stores these signed characters at the server, instead. (Here, "$||$" denotes concatenation.) Note that since $s_k = \mathsf{H}_1(\sigma_k || k)^x$, anyone knowing the corresponding verification key $\langle p, \mathbb{G}_1, \mathbb{G}_2, g, e, h \rangle$ cannot only verify s_k but can also extract σ_k and k, by simply testing for each $\sigma \in \Sigma$ and $k \in [\ell]$ whether $e(\mathsf{H}_1(\sigma || k), h) = e(s_k, g)$. As such, while in our initial protocol description, the data owner stores $s_0, \ldots, s_{\ell-1}$ at the server, this implicitly conveys $\sigma_0, \ldots, \sigma_{\ell-1,}$ as well.

The basic structure of the protocol, which is similar to $\Pi_1(\mathcal{E})$ in Fig. 3.1, involves the client encoding its DFA transition function δ as a bivariate polynomial $f(x, y)$ over $\mathbb{R}$ where x is the variable representing a DFA state and y is the variable representing an input symbol. In our protocol, the client and server then evaluate this polynomial together, using a single round of interaction per state transition (i.e., per file character), in such a way that the client observes only ciphertexts of states and file characters and the server observes only a randomly blinded state. More specifically, in our protocol, if the current DFA state is q, then the server observes only $\pi(q) +_{\mathbb{R}} \varphi$ for $\varphi \xleftarrow{\$} \mathbb{R}$ chosen by the client and where $\pi : Q \rightarrow \mathbb{R}$ maps DFA states to distinct ring elements. The client, with knowledge of π and φ, can calculate $f(x, y)$ so that $f(\pi(q) +_{\mathbb{R}} \varphi, \sigma) = \pi(\delta(q, \sigma))$ for each $q \in Q$ and $\sigma \in \Sigma$. Then, starting with a ciphertext of $\pi(q)$ for the DFA state q resulting from

[1] The file name or other identifier could be included along with the character position, to detect the exchange of characters between files. Similarly, the length ℓ can be included to detect file truncation. These issues are discussed further in Section 4.3.

processing file characters $\sigma_{0,}$ …, $\sigma_{k-1,}$ the client can interact with the server to obtain a ciphertext of $f(\pi(q) +_{\mathbb{R}} \varphi, \sigma_k)$ [73].

The central innovation in our protocol is a technique by which the client, without knowing s_k, can compute an encoding of the file character σ_k that the server must use in round k of the evaluation. If the server does not, it "throws off" the evaluation in a way that the server cannot predict. As a result, if the server deviates from the protocol, the end result of the evaluation will be an unpredictable element of the ring $\mathbb{R}$, which will not correspond to *any* state of the DFA with overwhelming probability. To accomplish this, the client defines the encoding of character $\sigma \in \Sigma$ and position $k \in [\ell]$ to be $\tau(\sigma, k, \psi_k) = \mathsf{H}_2(e(\mathsf{H}_1(\sigma||k)^{\psi_k}, h))$, where H_2 is a hash function $\mathsf{H}_2 : \mathbb{G}_2 \to \mathbb{R}$ (modeled as a random oracle) and where $\psi_k \overset{\$}{\leftarrow} \mathbb{Z}_p^*$ is selected by the client in the round for the k-th character. If the client sends $\Psi_k \leftarrow g^{\psi_k}$ to the server in the round for the k-th character, then the server can compute $\tau(\sigma_k, k, \psi_k)$ for the file character σ_k as $\tau(\sigma_k, k, \psi_k) = \mathsf{H}_2(e(s_k, \Psi_k))$. However, without ψ_k the server will be unable to compute the encoding $\tau(\sigma, k, \psi_k)$ for any $\sigma \neq \sigma_k$.

The final difficulty to overcome lies in the fact that the client, by altering the encoding of each character $\sigma \in \Sigma$ per round k, must also recompute $f(x, y)$ to account for this new encoding. As such, the client recomputes $f(x, y)$ to satisfy $f(\pi(q) +_{\mathbb{R}} \varphi_k, \tau(\sigma, k, \psi_k)) = \pi(\delta(q, \sigma))$ per round k, for every $q \in Q$ and $\sigma \in \Sigma$. In our algorithm, we encapsulate this calculation as $\langle a_{ij} \rangle_{i \in [n], j \in [m]} \leftarrow$ $\mathsf{ToPoly}(Q, \Sigma, \delta, \pi, k, \varphi_k, \beta_k, \psi_k)$ where $\langle a_{ij} \rangle_{i \in [n], j \in [m]}$ are the coefficients forming f, i.e., so that $f(x, y) = \sum_{i=0}^{\mathbb{R}\,n-1} \sum_{j=0}^{\mathbb{R}\,m-1} a_{ij} \cdot_{\mathbb{R}} x^i \cdot_{\mathbb{R}} y^j$. (The value β_k will become relevant in Section 4.2.3 and can be ignored for now.)

This protocol is shown in Fig. 4.1. The protocol is written with the steps performed by the client listed on the left (lines c301–c320), with those performed by the server on the right (lines s301–s313), and with the messages exchanged between them in the middle (lines m301–m306). The client takes as input the data owner's public verification key $\langle p, \mathbb{G}_1, \mathbb{G}_2, g, e, h \rangle$, a public encryption key ek', and its DFA $\langle Q, \Sigma, \delta, q_{\mathsf{init}} \rangle$. (For the moment, ignore the additional input dk, which will be discussed in Section 4.2.3.) The server takes as input $\langle p, \mathbb{G}_1, \mathbb{G}_2, g, e, h \rangle$, the DFA alphabet Σ, and the signed file characters s_0, …, $s_{\ell-1}$, i.e., signed with the data owner's private key $\langle \mathbb{G}_1, x \rangle$ corresponding to $\langle p, \mathbb{G}_1, \mathbb{G}_2, g, e, h \rangle$. (Again, please ignore the b_k values for now. These will be discussed in Section 4.2.3.) Note that neither the client nor the server receives any information about the private key dk', and so values encrypted under ek' (θ in line c304, and ρ in line c309) are never decrypted or otherwise used in the protocol. These values are included in the protocol only to simplify its proof and need not be included in a real implementation of the protocol.

At the beginning of the protocol, the server generates the public/private key pair (pek, pdk) (line s302) that defines the ring $\mathbb{R}$ for the protocol run. The server conveys pek and the file length ℓ to the client (m301). Upon receiving this message, the client selects an injection $\pi : Q \to \mathbb{R}$ at random from the set of all such injections, denoted $\mathsf{Injs}(Q \to \mathbb{R})$ (c303). The client sends the number n of states in his DFA in message m302. (To simplify our proofs, the client also sends the chosen injection π encrypted under ek' to server, denoted by θ. We will not discuss this further here.)

The heart of the protocol is the loop represented by lines c306–c317 for the client and lines s304–s312 for the server. The client begins each iteration of this loop with a ciphertext α of the current DFA state, which it blinds with the blinding term φ_k (c307) using the additive homomorphic property of Paillier encryption (c308). The client also selects ψ_k (c310) and creates Ψ_k (c311) as described above, and sends the now-blinded ciphertext α and Ψ_k to the server (m303). After decrypting the blinded state γ (s305) and using Ψ_k and s_k to create the encoding $\eta = \tau(\sigma, k, \psi_k)$ for the character σ_k being processed in this loop iteration (s306), the server creates the encryption of $\gamma^i \cdot_{\mathbb{R}} \eta^j$ for each $i \in [n]$ and $j \in [m]$ (s307–s311). After the server sends these values back to the client (m304), the client uses them together with the coefficients of f that it computed as described above (c313) to assemble a ciphertext of the new DFA state (c316).

After this loop iterates ℓ times, the client sends the state ciphertext to the server (m305). The server decrypts the (random) state (s313) and returns it (m306). The client checks to be sure that the result represents a valid state (c318) and, if so, returns the corresponding state as the result (c320).

4.2.3 Adding File Encryption

As presented so far, our protocol guarantees the integrity of the DFA evaluation against a malicious server. However, the confidentiality of the file content is not protected from the server because the signatures of the file characters are known to the server. With cloud outsourcing becoming increasingly popular, there is need to enable a data owner to outsource her file to the cloud while protecting its privacy, as well, against a potentially untrusted cloud provider. So, in this section, we

```
client($\langle p, \mathbb{G}_1, \mathbb{G}_2, g, e, h \rangle$,                server($\langle p, \mathbb{G}_1, \mathbb{G}_2, g, e, h \rangle$,
       $dk, ek', \langle Q, \Sigma, \delta, q_{\mathsf{init}} \rangle$)                     $\Sigma, \langle s_k, b_k \rangle_{k \in [\ell]}$)

c301. $n \leftarrow |Q|, m \leftarrow |\Sigma|$                        s301. $m \leftarrow |\Sigma|$
                                                    s302. $(pek, pdk) \leftarrow \mathsf{PGen}(1^\kappa)$
                m301. $\xleftarrow{pek,\ell}$
c302. $\langle N, g \rangle \leftarrow pek,\ \mathbb{R} \leftarrow \mathbb{Z}_N$                s303. $\langle N, g \rangle \leftarrow pek,\ \mathbb{R} \leftarrow \mathbb{Z}_N$
c303. $\pi \overset{\$}{\leftarrow} \mathsf{Injs}(Q \rightarrow \mathbb{R})$
c304. $\theta \leftarrow \mathsf{Enc}_{ek'}(\pi)$
                m302. $\xrightarrow{n,\theta}$
c305. $\alpha \leftarrow \mathsf{PEnc}_{pek}(\pi(q_{\mathsf{init}}))$
c306. for $k \leftarrow 0 \ldots \ell - 1$                        s304. for $k \leftarrow 0 \ldots \ell - 1$
c307.   $\varphi_k \overset{\$}{\leftarrow} \mathbb{R}$
c308.   $\alpha \leftarrow \alpha +_{pek} \mathsf{PEnc}_{pek}(\varphi_k)$
c309.   $\rho \leftarrow \mathsf{Enc}_{ek'}(\varphi_k)$
c310.   $\psi_k \overset{\$}{\leftarrow} \mathbb{Z}_p^*$
c311.   $\Psi_k \leftarrow g^{\psi_k}$
                m303. $\xrightarrow{\alpha,\rho,\Psi_k}$
                                                    s305.   $\gamma \leftarrow \mathsf{PDec}_{pdk}(\alpha)$
                                                    s306.   $\eta \leftarrow \mathsf{H}_2(e(s_k, \Psi_k))$
                                                    s307.   for $i \in [n]$
                                                    s308.     for $j \in [m]$
                                                    s309.       $\mu_{ij} \leftarrow \mathsf{PEnc}_{pek}(\gamma^i \cdot_{\mathbb{R}} \eta^j)$
                                                    s310.     endfor
                                                    s311.   endfor
                m304. $\xleftarrow{\langle \mu_{ij} \rangle_{i \in [n], j \in [m]}, b_k}$
c312.   $\beta_k \leftarrow \mathsf{Dec}_{dk}(b_k)$
c313.   $\langle a_{ij} \rangle_{i \in [n], j \in [m]}$
          $\leftarrow \mathsf{ToPoly}(Q, \Sigma, \delta, \pi, k, \varphi_k, \beta_k, \psi_k)$
c314.   if $\exists i, j : a_{ij} \neq 0 \wedge \gcd(a_{ij}, N) > 1$
c315.     then abort
c316.   $\alpha \leftarrow \sum_{pek}{}_{i=0}^{n-1} \sum_{pek}{}_{j=0}^{m-1} a_{ij} \cdot_{pek} \mu_{ij}$
c317. endfor                                        s312. endfor
                m305. $\xrightarrow{\alpha}$
                                                    s313. $\gamma^* \leftarrow \mathsf{PDec}_{pdk}(\alpha)$
                m306. $\xleftarrow{\gamma^*}$
c318. if $\gamma^* \notin \pi(Q)$
c319.   then abort
c320.   else return $\pi^{-1}(\gamma^*)$
```

Figure 4.1: Protocol $\Pi_3(\mathcal{E})$, described in Section 4.2

refine our protocol so that it provides the same guarantees while also protecting the confidentiality of the file content from the server.

As we described our protocol so far, the server holds the BLS signature $s_k = \mathsf{H}_1(\sigma_k||k)^x$, which enables him to learn σ_k by testing for each $\sigma \in \Sigma$ whether $e(\mathsf{H}_1(\sigma||k), h) = e(s_k, g)$. So, to hide σ_k from the server, it is necessary to change the signature s_k to prevent the server from confirming a guess at the value of σ_k.

To do so, in our full protocol the data owner randomizes the signature by raising it to a random power, i.e., $s_k \leftarrow \mathsf{H}_1(\sigma||k)^{x \cdot \beta_k}$ where $\beta_k \overset{\$}{\leftarrow} \mathbb{Z}_p^*$. s_k then does not leak information about σ_k to the server because it is randomly distributed in $\mathbb{G}_1$. However, this randomization also introduces new difficulties for the server and client to perform the DFA evaluation, since both of them need to be able to compute the same encoding for each σ_k despite s_k being randomized in this way.

To facilitate this evaluation, the data owner encrypts β_k under a public key ek of an encryption scheme whose plaintext space includes $\mathbb{Z}_p^*$ and provides its ciphertext, denoted b_k, along with s_k to the server; see the input arguments to server in Fig. 4.1. Of course, the server should not be able to decrypt b_k, since this would again enable him to reconstruct σ_k. As such, the data owner provides the corresponding private decryption key dk only to the client; see the input arguments to the client. Analogous to previous protocols [73], conveying dk can serve as a step by which the data owner authorizes a client to perform DFA queries on its file stored at the server. (In Section 4.4, we summarize an alternative approach that does not disclose dk or $\langle \beta_k \rangle_{k \in [\ell]}$ to the client.)

Given this setup, the full protocol $\Pi_3(\mathcal{E})$ thus executes the following additional steps. First, the client defines the encoding of character $\sigma \in \Sigma$ and position $k \in [\ell]$ to be $\tau(\sigma, k, \beta_k, \psi_k) = \mathsf{H}_2(e(\mathsf{H}_1(\sigma||k)^{\beta_k \psi_k}, h))$, where again H_2 is a hash function $\mathsf{H}_2 : \mathbb{G}_2 \rightarrow \mathbb{R}$ (modeled as a random oracle) and where $\psi_k \overset{\$}{\leftarrow} \mathbb{Z}_p^*$ is selected by the client in the round for character k. Note that the client needs to know β_k to compute $\tau(\sigma, k, \beta_k, \psi_k)$, and recall that the client needs to know $\tau(\sigma, k, \beta_k, \psi_k)$ for each $\sigma \in \Sigma$ in order to compute $f(x, y)$ to satisfy $f(\pi(q) +_{\mathbb{R}} \varphi_k, \tau(\sigma, k, \beta_k, \psi_k)) = \pi(\delta(q, \sigma))$ for every $q \in Q$ and $\sigma \in \Sigma$. Therefore, it is necessary for the client to include β_k as an argument to the ToPoly call (i.e., $\mathsf{ToPoly}(Q, \Sigma, \delta, \pi, k, \varphi_k, \beta_k, \psi_k)$ in c313) and to delay that call until after receiving b_k in m304 and using it to obtain β_k (c312).

4.2.4 Complexity

Protocol $\Pi_3(\mathcal{E})$ has a communication complexity of $O(\ell m n \kappa^2)$ bits, dominated by the message $\mathsf{m304}$ sent by the server in each round. The storage cost on the server depends on the number of BLS signatures of each character of the file. Assuming a BLS signature of length roughly 170 bits [18], it results in total 170ℓ bits for a file with ℓ characters. When using byte as character unit of a file, that amounts to about 21 times blow up in terms of storage compared against the original file.

4.2.5 Security Against Server Adversaries

In this section we prove the security of the protocol against server adversaries. We separately consider the DFA privacy, file privacy and the result authenticity against server adversaries.

DFA privacy against malicious server **adversaries.** Following the security definitions defined in Chapter 3, we formalize our claims against server compromise by defining server adversary $S = (S_1, S_2)$ who attacks the DFA $M = \langle Q,\, \Sigma,\, \delta,\, q_{\mathsf{init}}\rangle$ held by the client, as described in experiment $\mathbf{Expt}^{\mathtt{s\text{-}dfa}}_{\Pi_3(\mathcal{E})}$ in Fig. 4.2a. S_1 is given the BLS signature verification key $vk = \langle p, \mathbb{G}_1, \mathbb{G}_2, g, e, h\rangle$ and a public key ek of an IND-CPA secure encryption scheme, and generates a file $\langle \sigma_k\rangle_{k\in[\ell]}$ and two DFAs M_0 and M_1. S_2 then receives vk, the ciphertexts $\langle s_k, b_k\rangle_{k\in[\ell]}$ of its file, information ϕ created for it by S_1, and oracle access to $\mathsf{clientOr}(vk,\, dk,\, ek',\, M_b)$ for b chosen randomly.

$\mathsf{clientOr}$ responds to queries from S_2 as follows, ignoring malformed queries. S_2 initiates by sending a paillier encryption public key pek and an integer ℓ (as in $\mathsf{m301}$). $\mathsf{clientOr}$ responds with a message containing an integer n and a ciphertext θ (i.e., of the form of $\mathsf{m302}$). In addition, $\mathsf{clientOr}$ sends a message of the form α, ρ, Ψ_k, where $\alpha \in C_{pek}$, $\rho \in C_{ek'}$ and $\Psi_k \in \mathbb{G}_1$, i.e., three values as in $\mathsf{m303}$. The next ℓ queries by S_2 must contain nm elements of C_{pek} and an element of C_{ek}, i.e., $\langle \mu_{ij}\rangle_{i\in[n],j\in[m]}$ and b_k as in $\mathsf{m304}$, to which $\mathsf{clientOr}$ responds with three values as in message $\mathsf{m303}$. After that, $\mathsf{clientOr}$ sends another ciphertext α of C_{pek} as in $\mathsf{m305}$. The next (and last) query by S_2 can consist simply of a value in $\mathbb{R}$, as in message $\mathsf{m306}$.

Eventually S_2 outputs a bit b', and $\mathbf{Expt}^{\mathtt{s\text{-}dfa}}_{\Pi_3(\mathcal{E})}(S) = 1$ only if $b' = b$. We say the *advantage* of S is $\mathbf{Adv}^{\mathtt{s\text{-}dfa}}_{\Pi_3(\mathcal{E})}(S) = 2 \cdot \mathbb{P}\left(\mathbf{Expt}^{\mathtt{s\text{-}dfa}}_{\Pi_3(\mathcal{E})}(S) = 1\right) - 1$ and define $\mathbf{Adv}^{\mathtt{s\text{-}dfa}}_{\Pi_3(\mathcal{E})}(t, \ell, n, m) = \max_S \mathbf{Adv}^{\mathtt{s\text{-}dfa}}_{\Pi_3(\mathcal{E})}(S)$ where the maximum is taken over all adversaries S taking time t, selecting a file of length ℓ and DFAs containing n states and an alphabet of m symbols.

```
Experiment $\mathbf{Expt}^{\texttt{s-dfa}}_{\Pi_3(\mathcal{E})}(S_1, S_2)$
  $(p, \mathbb{G}_1, \mathbb{G}_2, g, e) \leftarrow \mathsf{ParamGen}(1^\kappa)$
  $(\langle p, \mathbb{G}_1, \mathbb{G}_2, g, e, h\rangle, \langle \mathbb{G}_1, x\rangle) \leftarrow \mathsf{BLSKeyGen}(p, \mathbb{G}_1, \mathbb{G}_2, g, e)$
  $vk \leftarrow \langle p, \mathbb{G}_1, \mathbb{G}_2, g, e, h\rangle$
  $(ek, dk) \leftarrow \mathsf{Gen}(1^\kappa)$
  $(ek', dk') \leftarrow \mathsf{Gen}(1^{\kappa+2})$
  $(\ell, \langle \sigma_k\rangle_{k\in[\ell]}, M_0, M_1, \phi) \leftarrow S_1(vk, ek)$
  if $|M_0.Q| \neq |M_1.Q|$ or $M_0.\Sigma \neq M_1.\Sigma$ then return $0$
  $b \overset{\$}{\leftarrow} \{0,1\}$
  for $k \in [\ell]$
    $s_k \leftarrow \mathsf{BLSSign}_{\langle \mathbb{G}_1, x\rangle}(\sigma_k || k)$
    $\beta_k \overset{\$}{\leftarrow} \mathbb{Z}_p^*$
    $s_k \leftarrow s_k^{\beta_k}$
    $b_k \leftarrow \mathsf{Enc}_{ek}(\beta_k)$
  $b' \leftarrow S_2^{\mathsf{clientOr}(vk, dk, ek', M_b)}(\phi, vk, M_b.\Sigma, \langle s_k, b_k\rangle_{k\in[\ell]})$
  if $b' = b$
    then return $1$
    else return $0$
```

(a) Experiment $\mathbf{Expt}^{\texttt{s-dfa}}_{\Pi_3(\mathcal{E})}$

```
Experiment $\mathbf{Expt}^{\texttt{s-file}}_{\Pi_3(\mathcal{E})}(S_1, S_2)$
  $(p, \mathbb{G}_1, \mathbb{G}_2, g, e) \leftarrow \mathsf{ParamGen}(1^\kappa)$
  $(\langle p, \mathbb{G}_1, \mathbb{G}_2, g, e, h\rangle, \langle \mathbb{G}_1, x\rangle) \leftarrow \mathsf{BLSKeyGen}(p, \mathbb{G}_1, \mathbb{G}_2, g, e)$
  $vk \leftarrow \langle p, \mathbb{G}_1, \mathbb{G}_2, g, e, h\rangle$
  $(ek, dk) \leftarrow \mathsf{Gen}(1^\kappa)$
  $(ek', dk') \leftarrow \mathsf{Gen}(1^{\kappa+2})$
  $(\ell, \langle \sigma_{0k}\rangle_{k\in[\ell]}, \langle \sigma_{1k}\rangle_{k\in[\ell]}, M, \phi) \leftarrow S_1(vk, ek)$
  $b \overset{\$}{\leftarrow} \{0,1\}$
  for $k \in [\ell]$
    $s_k \leftarrow \mathsf{BLSSign}_{\langle \mathbb{G}_1, x\rangle}(\sigma_{bk} || k)$
    $\beta_k \overset{\$}{\leftarrow} \mathbb{Z}_p^*$
    $s_k \leftarrow s_k^{\beta_k}$
    $b_k \leftarrow \mathsf{Enc}_{ek}(\beta_k)$
  $b' \leftarrow S_2^{\mathsf{clientOr}(vk, dk, ek', M), \mathsf{H}_1(\cdot)}(\phi, vk, M.\Sigma, \langle s_k, b_k\rangle_{k\in[\ell]})$
  if $b' = b$
    then return $1$
    else return $0$
```

(b) Experiment $\mathbf{Expt}^{\texttt{s-file}}_{\Pi_3(\mathcal{E})}$

Figure 4.2: Experiments for proving DFA and file privacy of $\Pi_3(\mathcal{E})$ against server adversaries

We reduce DFA privacy against server attacks to the IND-CPA [10] security of the encryption scheme. IND-CPA security is defined using the experiment in Fig. 3.3 in Chapter 3, in which an adversary U is provided a public key $\hat{pk}$ and access to an oracle $\mathsf{Enc}^{\hat{b}}_{\hat{pk}}(\cdot,\cdot)$ that consistently encrypts either the first of its two inputs (if $\hat{b} = 0$) or the second of those inputs (if $\hat{b} = 1$). Eventually U outputs a guess $\hat{b}'$ at $\hat{b}$, and $\mathbf{Expt}^{\texttt{ind-cpa}}_{\mathcal{E}}(U) = 1$ only if $\hat{b}' = \hat{b}$. The IND-CPA advantage of U is defined as $\mathbf{Adv}^{\texttt{ind-cpa}}_{\mathcal{E}}(U) = 2 \cdot \mathbb{P}\left(\mathbf{Expt}^{\texttt{ind-cpa}}_{\mathcal{E}}(U) = 1\right) - 1$. Then, $\mathbf{Adv}^{\texttt{ind-cpa}}_{\mathcal{E}}(t, w) = \max_U \mathbf{Adv}^{\texttt{ind-cpa}}_{\mathcal{E}}(U)$ where the maximum is taken over all adversaries U executing in time t and making w queries to $\mathsf{Enc}^{\hat{b}}_{\hat{pk}}(\cdot,\cdot)$.

We now prove the DFA privacy of the protocol.

Theorem 4. *For* $t' = t + t_{\mathsf{ParamGen}} + t_{\mathsf{BLSKeyGen}} + t_{\mathsf{Gen}} + \ell \cdot (t_{\mathsf{BLSSign}} + t_{\mathsf{Enc}})$*,*

$$\mathbf{Adv}^{\texttt{s-dfa}}_{\Pi_3(\mathcal{E})}(t, \ell, n, m) \leq 2\mathbf{Adv}^{\texttt{ind-cpa}}_{\mathcal{E}}(t', \ell + 1)$$

Proof. Let S be an adversary meeting the parameters t, ℓ, n, m. Consider a simulation $\mathbf{Sim}^{\texttt{s-dfa}}_{\Pi_3(\mathcal{E})}$ for $\mathbf{Expt}^{\texttt{s-dfa}}_{\Pi_3(\mathcal{E})}$ that differs only by simulating $\mathsf{clientOr}$ so as to substitute the ciphertext produced with ek' in c304 with encryptions of a random injection π' independent of π it chose as in c303 (i.e., $\rho \leftarrow \mathsf{Enc}_{pk'}(\pi')$, $\pi' \overset{\$}{\leftarrow} \mathsf{Injs}(Q \rightarrow \mathbb{R})$) and to substitute all ciphertexts created in c309 with encryptions of zero. Then b is hidden information-theoretically from S in $\mathbf{Sim}^{\texttt{s-dfa}}_{\Pi_3(\mathcal{E})}$, since γ is a random element of $\mathbb{R}$ in s305 (see c308) and Ψ_k is a random element in $\mathbb{G}_1$ (see c310), and since γ^* is a random element of $\mathbb{R}$ (see c303). As a result, $\mathbb{P}\left(\mathbf{Sim}^{\texttt{s-dfa}}_{\Pi_3(\mathcal{E})}(S) = 1\right) = \frac{1}{2}$ and for $\mathbf{Adv}^{\texttt{s-dfa}}_{\Pi_3(\mathcal{E})}(S)$ to be nonzero, S must distinguish $\mathbf{Sim}^{\texttt{s-dfa}}_{\Pi_3(\mathcal{E})}$ from $\mathbf{Expt}^{\texttt{s-dfa}}_{\Pi_3(\mathcal{E})}$.

We construct an IND-CPA adversary U that, on input $\hat{pk}$, sets $ek' \leftarrow \hat{pk}$ and uses its own oracle $\mathsf{Enc}^{\hat{b}}_{\hat{pk}}$ to choose between running $\mathbf{Expt}^{\texttt{s-dfa}}_{\Pi_3(\mathcal{E})}$ and $\mathbf{Sim}^{\texttt{s-dfa}}_{\Pi_3(\mathcal{E})}$ for S by setting $\theta \leftarrow \mathsf{Enc}^{\hat{b}}_{\hat{pk}}(\pi, \pi')$ in c304 and $\rho \leftarrow \mathsf{Enc}^{\hat{b}}_{\hat{pk}}(r, 0)$ in c307. (Aside from this, U performs $\mathbf{Expt}^{\texttt{s-dfa}}_{\Pi_3(\mathcal{E})}$ faithfully, generating the BLS signature signing key $\langle \mathbb{G}_1, x\rangle$ and using $(ek, dk) \leftarrow \mathsf{Gen}(1^\kappa)$ it generates itself.) U then returns $\hat{b}' = 0$ if S_2 outputs $b' = b$ and $\hat{b}' = 1$, otherwise. Then,

$$\begin{aligned}
&\mathbb{P}\left(\mathbf{Expt}^{\texttt{ind-cpa}}_{\mathcal{E}}(U) = 1\right) \\
&= \frac{1}{2}\mathbb{P}\left(\mathbf{Expt}^{\texttt{s-dfa}}_{\Pi_3(\mathcal{E})}(S) = 1\right) + \frac{1}{2}\mathbb{P}\left(\mathbf{Sim}^{\texttt{s-dfa}}_{\Pi_3(\mathcal{E})}(S) = 0\right) \\
&= \frac{1}{2}\left(\frac{1}{2} + \frac{1}{2}\mathbf{Adv}^{\texttt{s-dfa}}_{\Pi_3(\mathcal{E})}(S)\right) + \frac{1}{4} \\
&= \frac{1}{2} + \frac{1}{4}\mathbf{Adv}^{\texttt{s-dfa}}_{\Pi_3(\mathcal{E})}(S)
\end{aligned}$$

and so $\mathbf{Adv}^{\texttt{ind-cpa}}_{\mathcal{E}}(U) = \frac{1}{2}\mathbf{Adv}^{\texttt{s-dfa}}_{\Pi_3(\mathcal{E})}(S)$.

Note that U makes $\ell+1$ oracle queries and runs in time $t' = t + t_{\mathsf{ParamGen}} + t_{\mathsf{BLSKeyGen}} + t_{\mathsf{Gen}} + \ell \cdot (t_{\mathsf{BLSSign}} + t_{\mathsf{Enc}})$, due to the need to generate BLS signature signing key, (ek, dk) and encrypt ℓ file characters. □

File privacy against malicious server **adversary.** Next, we prove that the protocol protects the file privacy against an arbitrarily malicious server adversary. We define the server adversary $S = (S_1, S_2)$ attacking the file ciphertexts $\langle s_k, b_k\rangle_{k\in[\ell]}$ as in experiment $\mathbf{Expt}^{\texttt{s-file}}_{\Pi_3(\mathcal{E})}$ shown in Fig. 4.2b. S_1 produces two equal-length plaintext files $\langle \sigma_{0k}\rangle_{k\in[\ell]}$, $\langle \sigma_{1k}\rangle_{k\in[\ell]}$ and a DFA M. S_2 receives the ciphertexts $\langle s_k, b_k\rangle_{k\in[\ell]}$ for file $\langle \sigma_{bk}\rangle_{k\in[\ell]}$ where b is chosen randomly. S_2 is also given oracle access to $\mathsf{clientOr}(vk, dk, ek', M)$ and hash oracle access to $\mathsf{H}_1(\cdot)$. Eventually S_2 outputs a bit b', and $\mathbf{Expt}^{\texttt{s-file}}_{\Pi_3(\mathcal{E})}(S) = 1$ iff $b' = b$. We say the *advantage* of S is $\mathbf{Adv}^{\texttt{s-file}}_{\Pi_3(\mathcal{E})}(S) = 2 \cdot \mathbb{P}\left(\mathbf{Expt}^{\texttt{s-file}}_{\Pi_3(\mathcal{E})}(S) = 1\right) - 1$ and then $\mathbf{Adv}^{\texttt{s-file}}_{\Pi_3(\mathcal{E})}(t, \ell, n, m, h_1) = \max_S \mathbf{Adv}^{\texttt{s-file}}_{\Pi_3(\mathcal{E})}(S)$ where the maximum is taken over all adversaries $S = (S_1, S_2)$ taking time t, producing (from S_1) files of ℓ symbols and a DFA of n states and alphabet of size m and making h_1 queries to $\mathsf{H}_1(\cdot)$.

We now prove the following theorem:

Theorem 5. *Let* $\mathsf{H}_1(\cdot)$ *be a random oracle. For* $t' = t + t_{\mathsf{ParamGen}} + t_{\mathsf{BLSKeyGen}} + t_{\mathsf{Gen}} + \ell \cdot (t_{\mathsf{BLSSign}} + t_{\mathsf{Enc}})$,

$$\mathbf{Adv}^{\texttt{s-file}}_{\Pi_3(\mathcal{E})}(t, \ell, n, m, h_1) \le 2\mathbf{Adv}^{\texttt{ind-cpa}}_{\mathcal{E}}(t', \ell+1) + \mathbf{Adv}^{\texttt{ind-cpa}}_{\mathcal{E}}(t', \ell)$$

Proof. Let $\mathbf{Expt}^{\texttt{s-file-0}}_{\Pi_3(\mathcal{E})}$ denote experiment $\mathbf{Expt}^{\texttt{s-file}}_{\Pi_3(\mathcal{E})}$ with b fixed at $b = 0$, and let $\mathbf{Expt}^{\texttt{s-file-1}}_{\Pi_3(\mathcal{E})}$ denote the experiment $\mathbf{Expt}^{\texttt{s-file}}_{\Pi_3(\mathcal{E})}$ with b fixed at $b = 1$. Consider a simulation $\mathbf{Sim}^{\texttt{s-file-0}}_{\Pi_3(\mathcal{E})}$ for $\mathbf{Expt}^{\texttt{s-file-0}}_{\Pi_3(\mathcal{E})}$ that differs only by simulating $\mathsf{clientOr}$ so as to substitute the ciphertext produced with ek' in $\mathsf{c304}$ with encryptions of a random injection π' independent of π it chose as in $\mathsf{c303}$ (i.e., $\rho \leftarrow \mathsf{Enc}_{pk'}(\pi')$, $\pi' \xleftarrow{\$} \mathsf{Injs}(Q \to \mathbb{R})$) and to substitute all ciphertexts created in $\mathsf{c309}$ with encryptions of zero. Proceeding as in the proof of Theorem 4, we construct an IND-CPA adversary U_0 that, on input $\hat{pk}$ of an encryption scheme $\mathcal{E}'$, sets $ek' \leftarrow \hat{pk}$ and uses its own oracle $\mathsf{Enc}^{\hat{b}}_{\hat{pk}}$ to choose between running $\mathbf{Expt}^{\texttt{s-file-0}}_{\Pi_3(\mathcal{E})}$ and $\mathbf{Sim}^{\texttt{s-file-0}}_{\Pi_3(\mathcal{E})}$ for S, i.e., by setting $\theta \leftarrow \mathsf{Enc}^{\hat{b}}_{\hat{pk}}(\pi, \pi')$ in $\mathsf{c304}$ and $\rho \leftarrow \mathsf{Enc}^{\hat{b}}_{\hat{pk}}(r, 0)$ in $\mathsf{c307}$. (Aside from this, U performs $\mathbf{Expt}^{\texttt{s-dfa}}_{\Pi_3(\mathcal{E})}$ faithfully, using $(ek, dk) \leftarrow \mathsf{Gen}(1^\kappa)$ it generates itself). Finally, U_0 returns $\hat{b}' = 0$ if $b' = b$ and $\hat{b}' = 1$, otherwise. Then,

$$\begin{aligned} &1 + \mathbf{Adv}^{\texttt{ind-cpa}}_{\mathcal{E}}(U_0) = 2 \cdot \mathbb{P}\left(\mathbf{Expt}^{\texttt{ind-cpa}}_{\mathcal{E}}(U_0) = 1\right) = \\ &\mathbb{P}\left(\mathbf{Expt}^{\texttt{s-file-0}}_{\Pi_3(\mathcal{E})}(S) = 1\right) + \mathbb{P}\left(\mathbf{Sim}^{\texttt{s-file-0}}_{\Pi_3(\mathcal{E})}(S) = 0\right) \end{aligned} \tag{4.1}$$

Now consider a simulation $\mathbf{Sim}^{\texttt{s-file-1}}_{\Pi_3(\mathcal{E})}$ for $\mathbf{Expt}^{\texttt{s-file-1}}_{\Pi_3(\mathcal{E})}$ that again differs only by simulating $\mathsf{clientOr}$ so as to substitute all ciphertexts produced with ek' with encryptions of a random injection π' independent of π it chose as in $\mathsf{c303}$ (i.e., $\rho \leftarrow \mathsf{Enc}_{pk'}(\pi')$, $\pi' \xleftarrow{\$} \mathsf{Injs}(Q \to \mathbb{R})$) and to substitute all ciphertexts created in $\mathsf{c309}$ with encryptions of zero. As above, we construct an IND-CPA adversary U_1 that, on input $\hat{pk}$ of an encryption scheme $\mathcal{E}'$, sets $ek' \leftarrow \hat{pk}$ and uses its own oracle $\mathsf{Enc}^{\hat{b}}_{\hat{pk}}$ to choose between running $\mathbf{Expt}^{\texttt{s-file-1}}_{\Pi_3(\mathcal{E})}$ and $\mathbf{Sim}^{\texttt{s-file-1}}_{\Pi_3(\mathcal{E})}$ for S, i.e., by setting $\theta \leftarrow \mathsf{Enc}^{\hat{b}}_{\hat{pk}}(\pi', \pi)$ in $\mathsf{c304}$ and $\rho \leftarrow \mathsf{Enc}^{\hat{b}}_{\hat{pk}}(0, r)$ in $\mathsf{c307}$. (Aside from this, U performs $\mathbf{Expt}^{\texttt{s-file-1}}_{\Pi_3(\mathcal{E})}$ faithfully, using $(ek, dk) \leftarrow \mathsf{Gen}(1^\kappa)$ it generates itself). Finally, U_1 returns $\hat{b}' = 1$ if $b' = b$ and $\hat{b}' = 0$, otherwise. Then,

$$\begin{aligned} &1 + \mathbf{Adv}^{\texttt{ind-cpa}}_{\mathcal{E}}(U_1) = 2 \cdot \mathbb{P}\left(\mathbf{Expt}^{\texttt{ind-cpa}}_{\mathcal{E}}(U_1) = 1\right) = \\ &\mathbb{P}\left(\mathbf{Sim}^{\texttt{s-file-1}}_{\Pi_3(\mathcal{E})}(S) = 0\right) + \mathbb{P}\left(\mathbf{Expt}^{\texttt{s-file-1}}_{\Pi_3(\mathcal{E})}(S) = 1\right) \end{aligned} \tag{4.2}$$

Finally, consider an adversary U that uses its oracle $\mathsf{Enc}^{\hat{b}}_{\hat{pk}}$ to choose between running $\mathbf{Sim}^{\texttt{s-file-0}}_{\Pi_3(\mathcal{E})}$ and $\mathbf{Sim}^{\texttt{s-file-1}}_{\Pi_3(\mathcal{E})}$ for S. Specifically, on input $\hat{pk}$ of an encryption scheme $\mathcal{E}$, U generates $(p, \mathbb{G}_1, \mathbb{G}_2, g, e) \leftarrow \mathsf{ParamGen}(1^\kappa)$, $(vk = \langle p, \mathbb{G}_1, \mathbb{G}_2, g, e, h\rangle, \langle \mathbb{G}_1, x\rangle) \leftarrow \mathsf{BLSKeyGen}(p, \mathbb{G}_1, \mathbb{G}_2, g, e)$ and invokes $S_1(vk, \hat{pk})$. Upon receiving $\langle \sigma_{0k}\rangle_{k\in[\ell]}$ and $\langle \sigma_{1k}\rangle_{k\in[\ell]}$ from S_1, for each $k \in [\ell]$, U sets $\mathsf{H}_1(\sigma_{0k}||k) \leftarrow g^{u_k}$ for $u_k \xleftarrow{\$} \mathbb{Z}_p^*$ and $\mathsf{H}_1(\sigma_{1k}||k) \leftarrow g^{v_k}$ for $v_k \xleftarrow{\$} \mathbb{Z}_p^*$. U computes $s_k \leftarrow g^{x\cdot\beta_k}$ for

$\beta_k \xleftarrow{\$} \mathbb{Z}_p^*$ and sets $b_k \leftarrow \mathsf{Enc}_{\hat{pk}}^{\hat{b}}(u_k \cdot \beta_k^{-1} \bmod p, v_k \cdot \beta_k^{-1} \bmod p)$. Note that the way that b_k is computed determined whether σ_{0k} or σ_{1k} is encrypted. U then invokes $S_2(\phi, vk, M.\Sigma, \langle s_k, b_k \rangle_{k \in [\ell]})$. In the simulation of $\mathsf{clientOr}$, U selects $r \xleftarrow{\$} \mathbb{R}$ and sets $\alpha \leftarrow \mathsf{Enc}_{pek}(r)$ in c305 and c316 using the Paillier public key pek it received from S_2 in the first query (as in m301). (U also generates ek' itself and constructs an encryption of a random injection as in c304 and encryptions of zero as in c309). For S_2's other queries to $\mathsf{H}_1(\cdot)$, for any query that was previously posed to H_1, U returns the value returned to that previous query, and for new queries, U generates a random element from $\mathbb{G}_1$. Finally when S_2 outputs b', U outputs b' as $\hat{b}'$. Then,

$$\begin{aligned}
&1 + \mathbf{Adv}_{\mathcal{E}}^{\texttt{ind-cpa}}(U) = \\
&2 \cdot \mathbb{P}\left(\mathbf{Expt}_{\mathcal{E}}^{\texttt{ind-cpa}}(U) = 1\right) = 2 \cdot \mathbb{P}\left(\mathbf{Sim}_{\Pi_3(\mathcal{E})}^{\texttt{s-file}}(S) = 1\right) = \\
&\mathbb{P}\left(\mathbf{Sim}_{\Pi_3(\mathcal{E})}^{\texttt{s-file-0}}(S) = 1\right) + \mathbb{P}\left(\mathbf{Sim}_{\Pi_3(\mathcal{E})}^{\texttt{s-file-1}}(S) = 1\right)
\end{aligned} \tag{4.3}$$

Adding (4.1), (4.2) and (4.3), we get

$$\begin{aligned}
&3 + \mathbf{Adv}_{\mathcal{E}}^{\texttt{ind-cpa}}(U_0) + \mathbf{Adv}_{\mathcal{E}}^{\texttt{ind-cpa}}(U) + \mathbf{Adv}_{\mathcal{E}}^{\texttt{ind-cpa}}(U_1) \\
&= \mathbb{P}\left(\mathbf{Expt}_{\Pi_3(\mathcal{E})}^{\texttt{s-file-0}}(S) = 1\right) + \mathbb{P}\left(\mathbf{Sim}_{\Pi_3(\mathcal{E})}^{\texttt{s-file-0}}(S) = 0\right) \\
&\quad + \mathbb{P}\left(\mathbf{Sim}_{\Pi_3(\mathcal{E})}^{\texttt{s-file-0}}(S) = 1\right) + \mathbb{P}\left(\mathbf{Sim}_{\Pi_3(\mathcal{E})}^{\texttt{s-file-1}}(S) = 1\right) \\
&\quad + \mathbb{P}\left(\mathbf{Sim}_{\Pi_3(\mathcal{E})}^{\texttt{s-file-1}}(S) = 0\right) + \mathbb{P}\left(\mathbf{Expt}_{\Pi_3(\mathcal{E})}^{\texttt{s-file-1}}(S) = 1\right) \\
&= 2 \cdot \mathbb{P}\left(\mathbf{Expt}_{\Pi_3(\mathcal{E})}^{\texttt{s-file}}(S) = 1\right) + 2 \\
&= 3 + \mathbf{Adv}_{\Pi_3(\mathcal{E})}^{\texttt{s-file}}(S)
\end{aligned}$$

The result then follows because each of U_0 and U_1 makes $\ell + 1$ oracle queries and runs in time $t' = t + t_{\mathsf{ParamGen}} + t_{\mathsf{BLSKeyGen}} + t_{\mathsf{Gen}} + \ell \cdot (t_{\mathsf{BLSSign}} + t_{\mathsf{Enc}})$ due to the need to generate the BLS signature signing key, (ek, dk) and encrypt ℓ file characters. U makes ℓ oracle queries in order to create the file ciphertexts and runs in time t' for similar reasons. □

Detection of server **misbehavior** We first formally define what it means for a client to be able to detect any server misbehavior. Such an experiment is shown in Fig. 4.3. In this experiment, S_2

```
Experiment $\mathbf{Expt}^{\mathtt{s\text{-}auth}}_{\Pi_3(\mathcal{E})}(S)$
    $(p, \mathbb{G}_1, \mathbb{G}_2, g, e) \leftarrow \mathsf{ParamGen}(1^\kappa)$
    $(\langle p, \mathbb{G}_1, \mathbb{G}_2, g, e, h\rangle, \langle \mathbb{G}_1, x\rangle) \leftarrow \mathsf{BLSKeyGen}(p, \mathbb{G}_1, \mathbb{G}_2, g, e)$
    $vk \leftarrow \langle p, \mathbb{G}_1, \mathbb{G}_2, g, e, h\rangle$
    $(ek, dk) \leftarrow \mathsf{Gen}(1^\kappa)$
    $(ek', dk') \leftarrow \mathsf{Gen}(1^{\kappa+2})$
    $(\ell, \langle \sigma_k\rangle_{k\in[\ell]}, M, \phi) \leftarrow S_1(\langle p, \mathbb{G}_1, \mathbb{G}_2, g, e, h\rangle, ek)$
    for $k \in [\ell]$
        $s_k \leftarrow \mathsf{BLSSign}_{\langle \mathbb{G}_1, x\rangle}(\sigma_k || k)$
        $\beta_k \xleftarrow{\$} \mathbb{Z}_p^*$
        $s_k \leftarrow s_k^{\beta_k}$
        $b_k \leftarrow \mathsf{Enc}_{ek}(\beta_k)$
    $\pi \xleftarrow{\$} \mathsf{Injs}(Q \rightarrow \mathbb{R})$
    $\gamma^* \leftarrow S_2^{\mathsf{clientOr}(vk, dk, M, \pi), \mathsf{H}_1(\cdot), \mathsf{H}_2(\cdot)}(vk, M.\Sigma, \langle s_k, b_k\rangle_{k\in[\ell]})$
    if $\gamma^* \in \pi(Q) \wedge \gamma^* \neq \pi(M(\langle \sigma_k\rangle_{k\in[\ell]}))$
        then return $1$
        else return $0$
```

Figure 4.3: Experiment for proving result authenticity against server adversaries

is invoked with the public verification key $\langle p, \mathbb{G}_1, \mathbb{G}_2, g, e, h\rangle$ of the BLS signature and and file ciphertexts $\langle s_k, b_k\rangle_{k\in[\ell]}$. S_2 can then invoke $\mathsf{clientOr}$ first with a Paillier public key pek and an integer ℓ (as in m301) and receives n and θ in response (as in m302). S_2 can then invoke $\mathsf{clientOr}$ ℓ times, each time with ciphertexts α, ρ and Ψ_k (as in m303), and receives ciphertexts $\langle \mu_{ij}\rangle_{i\in[n], j\in[m]}$, and b_k in response (as in m304). Finally, S_2 outputs a ring element γ^* as in m306. The experiment outputs 1 if and only if $\gamma^* \in \pi(Q)$ and $\gamma^* \neq \pi(M(\langle \sigma_k\rangle_{k\in[\ell]}))$, where π is a random injection that was given to $\mathsf{clientOr}$ as input so that it will not need to select one by itself as in c303. This means that the protocol outputs an erroneous final state to the client and goes undetected by the client. For an arbitrarily malicious server adversary S, we define its advantage as:

$$\mathbf{Adv}^{\mathtt{s\text{-}auth}}_{\Pi_3(\mathcal{E})}(S) = \mathbb{P}\left(\mathbf{Expt}^{\mathtt{s\text{-}auth}}_{\Pi_3(\mathcal{E})}(S) = 1\right)$$

and define $\mathbf{Adv}^{\mathtt{s\text{-}auth}}_{\Pi_3(\mathcal{E})}(t, \ell, n, m, h_1, h_2) = \max_S \mathbf{Adv}^{\mathtt{s\text{-}auth}}_{\Pi_3(\mathcal{E})}(S)$ where the maximum is taken over all adversaries S executing in time t and selecting a file of length ℓ and a DFA of n states and alphabet of size m and making h_1 hash queries to $\mathsf{H}_1(\cdot)$ and h_2 queries to $\mathsf{H}_2(\cdot)$.

We reduce the result authenticity against a server adversary to the *bilinear computational Diffie-Hellman problem* (BCDH) [18]. The BCDH problem is defined using the experiment in Fig. 4.4, in which an adversary A is given two bilinear groups $\mathbb{G}_1$ and $\mathbb{G}_2$ both of order p, a random generator g of $\mathbb{G}_1$ and $g^{z_1}, g^{z_2}, g^{z_3}$ where $z_1, z_2, z_3 \overset{\$}{\leftarrow} \mathbb{Z}_p^*$. The experiment outputs 1 if and only if A is able to compute $e(g,g)^{z_1 z_2 z_3}$. The BCDH advantage of A is defined as

$$\mathbf{Adv}^{\mathsf{bcdh}}(A) = \mathbb{P}\left(\mathbf{Expt}^{\mathsf{bcdh}}(A) = 1\right)$$

and then $\mathbf{Adv}^{\mathsf{bcdh}}(t) = \max_A \mathbf{Adv}^{\mathsf{bcdh}}(A)$ where the maximum is taken over all adversaries A executing in time t.

```
Experiment $\mathbf{Expt}^{\mathsf{bcdh}}(A)$
  $(p, \mathbb{G}_1, \mathbb{G}_2, g, e) \leftarrow \mathsf{ParamGen}(1^\kappa)$
  $z_1, z_2, z_3 \overset{\$}{\leftarrow} \mathbb{Z}_p^*$
  $v \leftarrow A(p, \mathbb{G}_1, \mathbb{G}_2, g, e, g^{z_1}, g^{z_2}, g^{z_3})$
  if $v = e(g,g)^{z_1 z_2 z_3}$
    then return $1$
    else return $0$
```

Figure 4.4: Experiment for defining BCDH problem

We now prove that the protocol guarantees the authenticity of the evaluation result against an arbitrarily malicious server adversary.

Theorem 6. *Let* $\mathsf{H}_1(\cdot)$ *and* $\mathsf{H}_2(\cdot)$ *be random oracles. For* $t_1 = t + t_{\mathsf{ParamGen}} + t_{\mathsf{BLSKeyGen}} + t_{\mathsf{Gen}} + \ell \cdot (t_{\mathsf{BLSSign}} + t_{\mathsf{Enc}})$ *and* $t_2 = t + 2 \cdot t_{\mathsf{Gen}} + \ell \cdot (t_{\mathsf{BLSSign}} + t_{\mathsf{Enc}})$

$$\mathbf{Adv}^{\texttt{s-auth}}_{\Pi_3(\mathcal{E})}(t, \ell, n, m, h_1, h_2) \le \mathbf{Adv}^{\texttt{ind-cpa}}_{\mathcal{E}}(t_1, \ell+1) + (m-1) \cdot \ell \cdot h_2 \cdot \mathbf{Adv}^{\mathsf{bcdh}}(t_2)$$

Proof. Given a server adversary S, there are essentially two avenues by which a S might attempt to misbehave while escaping detection. The first is to create $\tau(\sigma, k, \beta_k, \psi_k) = \mathsf{H}_2(e(\mathsf{H}_1(\sigma||k)^{\beta_k \psi_k}, h))$ for some $\sigma \neq \sigma_k$, and to use $\tau(\sigma, k, \beta_k, \psi_k)$ as η in the protocol. The second is to cause the client to execute a state transition into an erroneous state in Q without computing $\tau(\sigma, k, \beta_k, \psi_k)$ for some $\sigma \neq \sigma_k$. Let $\mathsf{event1}$ denote the fact that the former event happens, and $\neg\mathsf{event1}$ denote that the latter case happens. We prove in Lemma 1 that $\neg\mathsf{event1}$ can only happen in probability negligible with respect to the security parameter. Here we show that the occurrence of $\mathsf{event1}$ implies the ability to solve the BCDH problem.

Given an adversary $S = (S_1, S_2)$ for which $\mathsf{event1}$ happens, and that runs in time t, produces a file of length ℓ, and produces a DFA of n states over an alphabet of m symbols, while making h_1 and h_2 hash queries to $\mathsf{H}_1(\cdot)$ and $\mathsf{H}_2(\cdot)$ respectively, we construct a BCDH attacker A to attack the BCDH assumption. Consider the following simulation $\mathbf{Sim}^{\texttt{s-auth}}_{\Pi_3(\mathcal{E})}(S)$ for $\mathbf{Expt}^{\texttt{s-auth}}_{\Pi_3(\mathcal{E})}(S)$. On input two bilinear groups $\mathbb{G}_1$ and $\mathbb{G}_2$ both of order p, a random generator g of $\mathbb{G}_1$ and $Z_1 = g^{z_1}, Z_2 = g^{z_2}, Z_3 = g^{z_3}$ where $z_1, z_2, z_3 \stackrel{\$}{\leftarrow} \mathbb{Z}_p^*$, A generates two public/private key pairs (ek, dk) and (ek', dk') for an IND-CPA encryption scheme and then invokes $S_1(\langle p, \mathbb{G}_1, \mathbb{G}_2, g, e, Z_1\rangle, ek)$ to obtain $(\ell, \langle\sigma_k\rangle_{k\in[\ell]}, M, \phi)$. Let $|M.Q| = n$ and $|M.\Sigma| = m$. A then sets $\mathsf{H}_1(\sigma_k||k) \leftarrow g^{u_k}$ where $u_k \stackrel{\$}{\leftarrow} \mathbb{Z}_p^*$ and then computes the encrypted file sequence $\langle s_k, b_k\rangle_{k\in[\ell]}$ such that $s_k \leftarrow Z_1^{u_k\beta_k}$ for $\beta_k \stackrel{\$}{\leftarrow} \mathbb{Z}_p^*$ and $b_k \leftarrow \mathsf{Enc}_{ek}(\beta_k)$. Note that the file ciphertext $\langle s_k, b_k\rangle_{k\in[\ell]}$ is well formed because $e(s_k, g) = e(Z_1^{u_k\beta_k}, g) = e(g^{z_1u_k\beta_k}, g) = e(g, g)^{z_1u_k\beta_k} = e(g^{u_k}, g^{z_1})^{\beta_k} = e(\mathsf{H}_1(\sigma_k||k), Z_1)^{\beta_k}$, as in the real protocol. A then chooses $k^* \stackrel{\$}{\leftarrow} [\ell]$ and $\sigma^* \stackrel{\$}{\leftarrow} \Sigma \setminus \{\sigma_{k^*}\}$ as its guesses on the round k^* and the input symbol σ^* that S_2 will attempt forgery. Finally, A invokes $S_2(\langle p, \mathbb{G}_1, \mathbb{G}_2, g, e, Z_1\rangle, M.\Sigma, \langle s_k, b_k\rangle_{k\in[\ell]})$ and simulates responses to S_2's queries to $\mathsf{clientOr}$ as follows.

After receiving pek and ℓ from S_2 (m301), A sets $\theta \leftarrow \mathsf{Enc}_{ek'}(\pi'), \pi' \stackrel{\$}{\leftarrow} \mathsf{Injs}(Q \rightarrow \mathbb{R})$ and sends n and θ to S_2 (as in m302). In round $k \in [\ell]$, A sets $\alpha \leftarrow \mathsf{Enc}_{pek}(r), r \stackrel{\$}{\leftarrow} \mathbb{Z}_N$ and sets $\rho \leftarrow \mathsf{Enc}_{ek}(0)$. If $k \neq k^*$, then A generates the random challenge Ψ_k exactly as specified in c310–c311. If $k = k^*$, then A sets $\Psi_k \leftarrow Z_3$. In either case, A then sends α, ρ and Ψ_k to S_2 (m303). After ℓ such rounds, A computes α to be the ciphertext of a random element of $\mathbb{R}$, and sends it to $\mathcal{S}$ (m305).

Meanwhile, A answers S_2's queries to the random oracle $\mathsf{H}_1(\cdot)$ as follows. For any query that was previously posed to H_1, A returns the value returned to that previous query, and for new queries, A generates a return value as follows. If the query is $\sigma^*||k^*$, then A returns Z_2. For all other queries, A picks $u \stackrel{\$}{\leftarrow} \mathbb{Z}_p^*$ and returns g^u. For S_2's queries to $\mathsf{H}_2(\cdot)$, for any query that was previously posed to H_2, A returns the value returned to that previous query. For new queries, A picks $r \stackrel{\$}{\leftarrow} \mathbb{Z}_N$ and returns r to S_2.

If S_2 computes

$$\begin{aligned}\tau(\sigma^*, k^*, \beta_{k^*}, \psi_{k^*}) &= \mathsf{H}_2(e(\mathsf{H}_1(\sigma^*||k^*)^{\beta_{k^*}\psi_{k^*}}, Z_1)) \\ &= \mathsf{H}_2(e(Z_2^{\beta_{k^*}z_3}, Z_1)) \\ &= \mathsf{H}_2(e(g,g)^{z_1 z_2 z_3 \beta_{k^*}})\end{aligned}$$

then A can output $e(g,g)^{z_1 z_2 z_3}$ by selecting a random query χ that S_2 made of H_2 and returning $\chi^{\beta_{k^*}^{-1} \bmod p}$. The probability that A outputs $e(g,g)^{z_1 z_2 z_3}$ is then $\frac{1}{(m-1)\cdot\ell\cdot h_2}$ times the probability that $\mathcal{S}$ produces $\tau(\sigma,\, k,\, \beta_k,\, \psi_k)$ for some $\sigma \neq \sigma_k$, where h_2 is the number of queries that S_2 poses to H_2.

So we have:

$$\begin{aligned}\mathbb{P}\left(\mathbf{Expt}^{\mathtt{bcdh}}(A) = 1\right) = \mathbb{P}\left(\mathbf{Sim}^{\mathtt{s\text{-}auth}}_{\Pi_3(\mathcal{E})}(S) = 1 \mid \mathsf{event1}\right) \cdot \mathbb{P}\left(\mathsf{event1}\right) + \\ \mathbb{P}\left(\mathbf{Sim}^{\mathtt{s\text{-}auth}}_{\Pi_3(\mathcal{E})}(S) = 1 \mid \neg\mathsf{event1}\right) \cdot \mathbb{P}\left(\neg\mathsf{event1}\right)\end{aligned}$$

We prove in Lemma 1 that $\mathbb{P}\left(\neg\mathsf{event1}\right)$ is negligible as a function of the security parameter. So, ignoring terms that are negligible as a function of the security parameter, we have:

$$\mathbb{P}\left(\mathbf{Expt}^{\mathtt{bcdh}}(A) = 1\right) \geq \frac{1}{(m-1)\cdot\ell\cdot h_2} \cdot \mathbb{P}\left(\mathbf{Sim}^{\mathtt{s\text{-}auth}}_{\Pi_3(\mathcal{E})}(S) = 1\right) \tag{4.4}$$

The only difference between the above described $\mathbf{Sim}^{\mathtt{s\text{-}auth}}_{\Pi_3(\mathcal{E})}(S)$ and $\mathbf{Expt}^{\mathtt{s\text{-}auth}}_{\Pi_3(\mathcal{E})}(S)$ is the way that θ and ρ are created. To show that this difference does not affect S's ability to succeed in its attack, we reduce its ability to distinguish $\mathbf{Sim}^{\mathtt{s\text{-}auth}}_{\Pi_3(\mathcal{E})}(S)$ from $\mathbf{Expt}^{\mathtt{s\text{-}auth}}_{\Pi_3(\mathcal{E})}(S)$ to the IND-CPA security of the encryption scheme from which ek' is generated. We create an IND-CPA adversary U that, on input $\hat{pk}$ of an encryption scheme $\mathcal{E}'$, uses his encryption oracle to select between $\mathbf{Sim}^{\mathtt{s\text{-}auth}}_{\Pi_3(\mathcal{E})}(S)$ and $\mathbf{Expt}^{\mathtt{s\text{-}auth}}_{\Pi_3(\mathcal{E})}(S)$ by setting $\theta \leftarrow \mathsf{Enc}^{\hat{b}}_{\hat{pk}}(\pi', \pi)$ as in c304 and $\rho \leftarrow \mathsf{Enc}^{\hat{b}}_{\hat{pk}}(0, r)$ as in c307. Aside from this, U performs the simulation exactly the same as described above for A. U returns $\hat{b}' = 1$ if

S_2 succeeds in its attack and returns $\hat{b}' = 0$ otherwise. So we have,

$$\begin{aligned}
\mathbf{Adv}_{\mathcal{E}}^{\texttt{ind-cpa}}(U) &= 2 \cdot \mathbb{P}\left(\mathbf{Expt}_{\mathcal{E}}^{\texttt{ind-cpa}}(U) = 1\right) - 1 \\
&= \mathbb{P}\left(\mathbf{Expt}_{\mathcal{E}}^{\texttt{ind-cpa-0}}(U) = 1\right) + \mathbb{P}\left(\mathbf{Expt}_{\mathcal{E}}^{\texttt{ind-cpa-1}}(U) = 1\right) - 1 \\
&= \mathbb{P}\left(\mathbf{Sim}_{\Pi_3(\mathcal{E})}^{\texttt{s-auth}}(S) = 0\right) + \mathbb{P}\left(\mathbf{Expt}_{\Pi_3(\mathcal{E})}^{\texttt{s-auth}}(S) = 1\right) - 1 \\
&= 1 - \mathbb{P}\left(\mathbf{Sim}_{\Pi_3(\mathcal{E})}^{\texttt{s-auth}}(S) = 1\right) + \mathbb{P}\left(\mathbf{Expt}_{\Pi_3(\mathcal{E})}^{\texttt{s-auth}}(S) = 1\right) - 1
\end{aligned} \tag{4.5}$$

Substituting in Eqn. 4.4, we have:

$$\mathbf{Adv}_{\mathcal{E}}^{\texttt{ind-cpa}}(U) \geq \mathbf{Adv}_{\Pi_3(\mathcal{E})}^{\texttt{s-auth}}(S) - (m-1) \cdot \ell \cdot h_2 \cdot \mathbf{Adv}^{\texttt{bcdh}}(A)$$

thus completing the proof. U takes time $t_1 = t + t_{\mathsf{ParamGen}} + t_{\mathsf{BLSKeyGen}} + t_{\mathsf{Gen}} + \ell \cdot (t_{\mathsf{BLSSign}} + t_{\mathsf{Enc}})$ due to the need to generate a BLS signing key, the encryption key (ek, dk) and to encrypt ℓ file characters. U also needs to make $\ell + 1$ encryption oracle queries. A takes time $t_2 = t + 2 \cdot t_{\mathsf{Gen}} + \ell \cdot (t_{\mathsf{BLSSign}} + t_{\mathsf{Enc}})$ due to the need to generate both (ek, dk) and (ek', dk') and to encrypt ℓ file characters. □

Lemma 1. *Let* H_2 *be a random oracle, and let* S_2 *be a* server*-compromising adversary. If in no round* k *does* S_2 *compute* $\tau(\sigma, k, \beta_k, \psi_k)$ *for some* $\sigma \neq \sigma_k$*, then the* client *outputs an incorrect state* $q \in Q$ *with probability at most negligibly more than* $\frac{n-1}{N}$*.*

Proof. In round k, the client transitions to the next DFA state by encoding the DFA transition function using a polynomial f satisfying $f(\pi(q) +_{\mathbb{R}} \varphi_k, \tau(\sigma, k, \beta_k, \psi_k)) = \pi(\delta(q, \sigma))$ for every $q \in Q$ and $\sigma \in \Sigma$; let $f(x, y) = \sum_{i=0}^{\mathbb{R}\, n-1} \sum_{j=0}^{\mathbb{R}\, m-1} a_{ij} \cdot_{\mathbb{R}} x^i \cdot_{\mathbb{R}} y^j$. To cause a state transition to an erroneous state $q' \in Q$, a server adversary must therefore produce ciphertexts $\langle \mu_{ij} \rangle_{i \in [n], j \in [m]}$ with corresponding plaintexts $\langle \nu_{ij} \rangle_{i \in [n], j \in [m]}$ so that

$$\pi(q') = \sum_{i=0}^{n-1}{}^{\mathbb{R}} \sum_{j=0}^{m-1}{}^{\mathbb{R}} a_{ij} \cdot_{\mathbb{R}} \nu_{ij} \tag{4.6}$$

without having any information about the injection π or $\tau(\sigma, k, \beta_k, \psi_k)$ for any $\sigma \neq \sigma_k$ (since H_2 is a random oracle). Note that the distribution of $\langle a_{ij} \rangle_{i \in [n], j \in [m]}$ is *not* independent of the DFA transition function δ and the injection π. That is, once π is fixed, only certain values for $\langle a_{ij} \rangle_{i \in [n], j \in [m]}$ are possible.

We argue the result under the conservative assumption that δ and π *uniquely determine* $\langle a_{ij} \rangle_{i \in [n], j \in [m]}$ (which in general they do not). Even then, for any $i' \in [n]$ and $j' \in [m]$ such that $a_{i'j'} \neq 0$ and $\gcd(a_{i'j'}, N) = 1$ (lines c314–c315 abort the protocol if $\gcd(a_{ij}, N) > 1$ for some $a_{ij} \neq 0$), and for any choices of $\langle \nu_{ij} \rangle_{i \in [n], j \in [m]}$ excepting $\nu_{i'j'}$, there is exactly one value for $\nu_{i'j'}$ in $\mathbb{Z}_N$ that satisfies Eqn. 4.6. Moreover, prior to the last message sent by the client (m305), the random injection π is hidden information theoretically from S_2, and so $\pi(q)$ for each $q \in Q$ is uniformly distributed from S_2's perspective. So, the probability S_2 succeeds in selecting $\langle \nu_{ij} \rangle_{i \in [n], j \in [m]}$ to satisfy Eqn. 4.6 is $\frac{1}{N}$, and since there are $n - 1$ possible erroneous states q', the probability $\mathcal{S}$ succeeds in causing an erroneous state transition to any $q' \in Q$ is at most $\frac{n-1}{N}$.

Finally, while the server learns $\pi(q)$ for one $q \in Q$ in the last client-to-server message (m305) — if it behaved thus far — it does so only for the correct state q at this point. Again, it can then guess $\pi(q')$ for an incorrect $q' \in Q$ to return as γ^* with probability only $\frac{n-1}{N}$. □

4.2.6 Security Against Client Adversaries

In this section we show security of $\Pi_3(\mathcal{E})$ against honest-but-curious client adversaries. Since the client has the DFA in its possession, privacy of the DFA against a client adversary is not a concern. Therefore we only focus on the privacy of the file. However, by the nature of what the protocol computes for the client — i.e., the final state of a DFA match on the file — the client can easily distinguish two files of its choosing simply by running the protocol correctly using a DFA that distinguishes between the two files it chose. For this reason, we adapt the notion of indistinguishability to apply only to files that produce the same final state for the client's DFA. So, in the experiment $\mathbf{Expt}^{\texttt{c-file}}_{\Pi_3(\mathcal{E})}$ (Fig. 4.5) that we use to define file security against client adversaries, the adversary $C = (C_1, C_2)$ succeeds (i.e., $\mathbf{Expt}^{\texttt{c-file}}_{\Pi_3(\mathcal{E})}(C)$ returns 1) only if the two files $\langle \sigma_{0k} \rangle_{k \in [\ell]}$ and $\langle \sigma_{1k} \rangle_{k \in [\ell]}$ output by C_1 both drive the DFA M, also output by C_1, to the same final state (denoted $M(\langle \sigma_{0k} \rangle_{k \in [\ell]}) = M(\langle \sigma_{1k} \rangle_{k \in [\ell]})$). Otherwise, the experiment is straightforward: C_1 receives the BLS signature verification key vk, a private decryption key dk and a public key ek', and returns

```
Experiment $\mathbf{Expt}^{\texttt{c-file}}_{\Pi_3(\mathcal{E})}(C_1, C_2)$
    $(p, \mathbb{G}_1, \mathbb{G}_2, g, e) \leftarrow \mathsf{ParamGen}(1^\kappa)$
    $(\langle p, \mathbb{G}_1, \mathbb{G}_2, g, e, h\rangle, \langle \mathbb{G}_1, x\rangle) \leftarrow \mathsf{BLSKeyGen}(p, \mathbb{G}_1, \mathbb{G}_2, g, e)$
    $vk \leftarrow \langle p, \mathbb{G}_1, \mathbb{G}_2, g, e, h\rangle$
    $(ek, dk) \leftarrow \mathsf{Gen}(1^\kappa)$
    $(ek', dk') \leftarrow \mathsf{Gen}(1^{\kappa+2})$
    $(\ell, \langle \sigma_{0k}\rangle_{k\in[\ell]}, \langle \sigma_{1k}\rangle_{k\in[\ell]}, M, \phi) \leftarrow C_1(vk, dk, ek')$
    if $M(\langle \sigma_{0k}\rangle_{k\in[\ell]}) \neq M(\langle \sigma_{1k}\rangle_{k\in[\ell]})$ then return $0$
    $b \xleftarrow{\$} \{0, 1\}$
    for $k \in [\ell]$
        $s_k \leftarrow \mathsf{BLSSign}_{\langle \mathbb{G}_1, x\rangle}(\sigma_{bk} || k)$
        $\beta_k \xleftarrow{\$} \mathbb{Z}_p^*$
        $s_k \leftarrow s_k^{\beta_k}$
        $b_k \leftarrow \mathsf{Enc}_{ek}(\beta_k)$
    $b' \leftarrow C_2^{\mathsf{serverOr}(vk, M.\Sigma, \langle s_k, b_k\rangle_{k\in[\ell]})}(\phi, vk, dk, ek', M)$
    if $b' = b$
        then return $1$
        else return $0$
```

Figure 4.5: Experiment for proving file privacy against client adversaries

the two ℓ-symbol files (for ℓ of its choosing) $\langle \sigma_{0k}\rangle_{k\in[\ell]}$ and $\langle \sigma_{1k}\rangle_{k\in[\ell]}$ and a DFA M. Depending on how b is then chosen, one of these files is encrypted and then provided to the server. C_2 is then invoked with vk, dk, ek' and M, and oracle access to serverOr$(vk, M.\Sigma, \langle s_k, b_k\rangle_{k\in[\ell]})$.

serverOr then responds to C_2's queries as follows, ignoring malformed queries. The first query (could be simply "start") initiates serverOr to begin the protocol. serverOr responds first with a Paillier public key pek and an integer ℓ (as in m301), to which serverOr returns n and θ (as in m302). C_2 can then invoke serverOr up to $\ell + 1$ times. The first ℓ such invocations take the form α, ρ, Ψ_k and correspond to messages of the form m303. For each such invocation, serverOr responds with $\langle \mu_{\sigma i}\rangle_{\sigma\in\Sigma, i\in[n]}$ and b_k (i.e., of the form m304). The last C_2's invocation contains one ciphertext α corresponding to m305. This invocation elicits a response γ^* (i.e., m306). Malformed or extra queries are rejected by serverOr.

We prove file privacy against *honest-but-curious* client adversaries $C = (C_1, C_2)$, i.e., C_2 invokes serverOr exactly as $\Pi_3(\mathcal{E})$ prescribes, using DFA M output by C_1. We define the advantage of C to be $\mathbf{hbcAdv}^{\texttt{c-file}}_{\Pi_3(\mathcal{E})}(C) = 2 \cdot \mathbb{P}\left(\mathbf{Expt}^{\texttt{c-file}}_{\Pi_3(\mathcal{E})}(C) = 1\right) - 1$ and $\mathbf{hbcAdv}^{\texttt{c-file}}_{\Pi_3(\mathcal{E})}(t, \ell, n, m)$ $= \max_C \mathbf{Adv}^{\texttt{c-file}}_{\Pi_3(\mathcal{E})}(C)$ where the maximum is taken over honest-but-curious client adversaries C

running in total time t and producing files of length ℓ and a DFA of n states over an alphabet of m symbols. We now prove:

Theorem 7. *For* $t' = t + t_{\mathsf{ParamGen}} + t_{\mathsf{BLSKeyGen}} + 2 \cdot t_{\mathsf{Gen}} + \ell * (t_{\mathsf{BLSSign}} + t_{\mathsf{Enc}}) + (\ell+1) \cdot t_{\mathsf{Dec}}$,

$$\mathbf{hbcAdv}^{\texttt{c-file}}_{\Pi_3(\mathcal{E})}(t, \ell, n, m) \leq \mathbf{Adv}^{\texttt{ind-cpa}}_{\mathsf{Pai}}(t', \ell mn)$$

Proof. Given an adversary $C = (C_1, C_2)$ running in time t and selecting files of length ℓ and a DFA of n states over an alphabet of m symbols, we construct an IND-CPA adversary U that demonstrates the theorem as follows. On input a Paillier public key $\hat{pk} = \langle N, g \rangle$, U sets $pek \leftarrow \hat{pk}$ and generates $(p, \mathbb{G}_1, \mathbb{G}_2, g, e) \leftarrow \mathsf{ParamGen}(1^\kappa)$ and $(vk \leftarrow \langle p, \mathbb{G}_1, \mathbb{G}_2, g, e, h \rangle, \langle \mathbb{G}_1, x \rangle) \leftarrow \mathsf{BLSKeyGen}(p, \mathbb{G}_1, \mathbb{G}_2, g, e)$. U also sets $(ek, dk) \leftarrow \mathsf{Gen}(1^\kappa)$ and $(ek', dk') \leftarrow \mathsf{Gen}(1^{\kappa+2})$, and then invokes $C_1(vk, dk, ek')$ to obtain $(\ell, \langle \sigma_{0k} \rangle_{k \in [\ell]}, \langle \sigma_{1k} \rangle_{k \in [\ell]}, M, \phi)$, where $M = \langle Q, \Sigma, q_{\mathsf{init}}, \delta \rangle$ is a DFA. For each $k \in [\ell]$, U computes $s_{0k} \leftarrow \mathsf{H}_1(\sigma_{0k} || k)^{x \cdot \beta_k}$ for $\beta_k \overset{\$}{\leftarrow} \mathbb{Z}_p^*$, and sets $b_{0k} \leftarrow \mathsf{Enc}_{ek}(\beta_k)$. Similarly, U sets $s_{1k} \leftarrow \mathsf{H}_1(\sigma_{1k} || k)^{x \cdot \beta_k}$ and sets $b_{1k} \leftarrow \mathsf{Enc}_{ek}(\beta_k)$. Note that the same β_k is assigned to both s_{0k} and s_{1k}, thus resulting in b_{0k} and b_{1k} encrypting the same plaintext value.

U then invokes $C_2(\phi, vk, dk, ek', M)$ and simulates responses to C_2's queries to $\mathsf{serverOr}$ as follows (ignoring malformed invocations). In response to the initial "start" query from C_1, U returns $\hat{pk}$ and ℓ and gets n and θ in return. U decrypts the ciphertext θ and sets $\pi \leftarrow \mathsf{Dec}_{dk'}(\theta)$. U sets $m \leftarrow |M.\Sigma|$ and in preparation for the subsequent $\mathsf{serverOr}$ invocations by C_2, U sets $q_0 \leftarrow M.q_{\mathsf{init}}$ and $q_1 \leftarrow M.q_{\mathsf{init}}$. For the k-th query of the form α, ρ, Ψ_k ($0 \leq k < \ell$), the adversary U sets $\varphi_k \leftarrow \mathsf{Dec}_{dk'}(\rho)$, $\gamma_0 \leftarrow \pi(q_0) +_{\mathbb{R}} \varphi_k$, and $\gamma_1 \leftarrow \pi(q_1) +_{\mathbb{R}} \varphi_k$. U computes $\eta_0 \leftarrow \mathsf{H}_2(e(s_{0k}, \Psi_k))$, and $\eta_1 \leftarrow \mathsf{H}_2(e(s_{1k}, \Psi_k))$ exactly as done in s306 and then sets $\mu_{ij} \leftarrow \mathsf{Enc}^{\hat{b}}_{\hat{pk}}((\gamma_0)^i \cdot_{\mathbb{R}} (\eta_0)^j, (\gamma_1)^i \cdot_{\mathbb{R}} (\eta_1)^j)$ for $i \in [n]$, $j \in [m]$. After this, U updates $q_0 \leftarrow \delta(q_0, \sigma_{0k})$ and $q_1 \leftarrow \delta(q_1, \sigma_{1k})$, and returns $\langle \mu_{ij} \rangle_{i \in [n], j \in [m]}$ and b_{0k} $(= b_{1k})$ to C_2. For the last query α, adversary U returns $\gamma^* = \pi(q_0)$ to C_2. When C_2 outputs b', U outputs b', as well.

This simulation is distributed identically with the real system provided that C is honest-but-curious, and so ignoring terms that are negligible in κ, $\mathbf{hbcAdv}^{\texttt{c-file}}_{\Pi_3(\mathsf{Pai})}(C) = \mathbf{Adv}^{\texttt{ind-cpa}}_{\mathsf{Pai}}(U)$. Note that U runs in $t' = t + t_{\mathsf{ParamGen}} + t_{\mathsf{BLSKeyGen}} + 2 \cdot t_{\mathsf{Gen}} + \ell * (t_{\mathsf{BLSSign}} + t_{\mathsf{Enc}}) + (\ell+1) \cdot t_{\mathsf{Dec}}$ where $2 \cdot t_{\mathsf{Gen}}$, due to the need to generate a BLS signing key, to generate (ek, dk) and (ek', dk'), and to encrypt ℓ file characters, and to perform one decryption of a ciphertext produced by ek' in each round. U makes nm oracle queries in order to respond to each of the ℓ oracle queries following the first. $\square$

4.3 On File Updates

Protocol $\Pi_3(\mathcal{E})$ is presented for a static file, and so in this section we consider the impact of file updates. As we discuss below, these impacts are nontrivial, and so our protocol is arguably most useful for static files.

To enable protocol $\Pi_3(\mathcal{E})$, the data owner signs the file position k along with σ_k when producing s_k to detect the server reordering file characters, i.e., $s_k \leftarrow \mathsf{H}_1(\sigma||k)^{x\cdot\beta_k}$ where $\beta_k \overset{\$}{\leftarrow} \mathbb{Z}_p^*$. Such a representation would require any character insertion or deletion at position k to further require updating the signature $s_{k'}$ for all $k' > k$. If the total file length ℓ is also included as an input to H_1 to detect file truncation, then insertions and deletions may require updating the signatures $s_{k'}$ for all $k' < k$, as well. This latter cost can be eliminated by not including ℓ as an input to H_1 but rather to have the data owner sign ℓ and the server to forward this signature along with ℓ to the client in message m301. The former cost can be mitigated somewhat by breaking each file into blocks (essentially smaller files) so that insertions and deletions require only the affected blocks to be rewritten. In this case, the block index within the file should presumably also be included as an input to H_1 to detect block reorderings by the server.

Even with these modifications, there remain other complexities in handling file updates, in that a server could simply use a stale version of the file when performing protocol $\Pi_3(\mathcal{E})$ with the client, ignoring any earlier updates to the file by the data owner. Detecting a server that selectively suppresses updates seems to require additional interaction between the data owner and the client and has been the subject of much study (for file stores subject to reads and updates only) under the banner of *fork consistency* [55]. We leave as future work the integration of our DFA evaluation techniques with these ideas, i.e., so that DFA evaluations performed against stale files are efficiently detected when the client subsequently interacts with the data owner.

4.4 Extensions

The protocol $\Pi_3(\mathcal{E})$ can be extended in various ways that may be of interest and that we will discuss here. The first "extension" is simply the removal of the file encryption step described in Section 4.2.3, which is suitable for the standard two-party model where the server's input need not be kept secret from the server himself. This simplification eliminates the dk, β_k and b_k values from the protocol, implicitly setting $\beta_k = 1$.

A more interesting variant of the protocol addresses the concern that the protocol as stated in Fig. 4.1 discloses the decryption key dk and the values $\langle \beta_k \rangle_{k\in[\ell]}$ to the client, either of which can be used to decrypt the file from its ciphertext $\langle s_k, b_k \rangle_{k\in[\ell]}$. While this file ciphertext is not disclosed to the client during the protocol, it seems unnecessarily permissive to disclose its decryption key to every client that performs a DFA evaluation on the file: if the file ciphertext were ever unintentionally disclosed, then any such client could decrypt the file if it retained the key. In the rest of this section we discuss an extension to the protocol in Fig. 4.1 to avoid disclosing dk and the values $\langle \beta_k \rangle_{k\in[\ell]}$ to the client.

In order to avoid disclosing dk to the client, one alternative is for the data owner to provide shares of dk to both the client and the server, so as to enable a two-party decryption of each b_k. Then, rather than sending only b_k to the client in message m304, the server can also send its contribution to the decryption of b_k, enabling the client to complete the decryption of b_k without learning dk itself.

Still, however, this alternative would disclose β_k to the client, which would enable it to determine σ_k if s_k were ever disclosed. To avoid disclosing β_k, one strategy is for the server to first blind β_k with another random value t_k, i.e., to execute the protocol with $\beta_k t_k$ in place of just β_k. Of course, this factor t_k would also then need to be reflected in k-th file character used in the protocol, i.e., so the server would use $s_k^{t_k} = \mathsf{H}_1(\sigma_k || k)^{x\beta_k t_k}$ in place of s_k in the protocol. Because the server does not have access to β_k but rather has access only to its ciphertext b_k, it is necessary that the encryption scheme used to construct b_k enable the computation of a ciphertext $\hat{b}_k$ from b_k and t_k such that $\mathsf{Dec}_{dk}(\hat{b}_k) = \beta_k t_k \bmod N'$ for some value N' such that $p \mid N'$. In this case, selecting $t_k \xleftarrow{\$} \mathbb{Z}_{N'}$ suffices to ensure that $\beta_k t_k \bmod N'$ is distributed independently of β_k and so hides β_k from the client when it learns $\beta_k t_k \bmod N'$.

An encryption scheme meeting our requirements (supporting two-party decryption and homomorphism on ciphertexts) is ElGamal encryption [30] in a subgroup of $\mathbb{Z}_{N'}^*$. However, note that setting $N' = p$ is inefficient: the security parameter κ and so the size of p required for security is an order of magnitude less for BLS signing than it would be for ElGamal encryption in a subgroup of $\mathbb{Z}_p^*$ [52], and so setting $N' = p$ would add considerable expense to the protocol. As such, a more efficient construction would be to choose $N' = pp'$ for another prime p'. ElGamal encryption is believed to be secure with a composite modulus even if its factorization is known [14].

CHAPTER 5

Toward Practical Encrypted Email That Supports Private, Regular-Expression Searches

In this chapter, we prototype a system to perform private regular expression searches on encrypted emails and evaluate its performance. Toward this goal, we develop new protocols to enable private regular expression searches on encrypted data stored at a server. The novelty of the protocol lies in allowing a user to securely delegate an encrypted regular-expression search query to a proxy, which interacts with the server where user's data is stored encrypted to produce the search result for the user. The privacy of the query and the data are both provably protected against an arbitrarily malicious server and a partially trusted proxy under rigorous security definitions. We then develop a working implementation of this protocol together with several optimizations to make it perform well and then evaluate the performance of this implementation on a real-world email data set. We describe our initial protocol design in Section 5.1 and detail a series of optimizations we employed in Section 5.2, and finally present the implementation and its performance evaluation in Section 5.4.

5.1 Protocol Design

In this chapter, we define a deterministic finite automaton M to be a tuple $\langle Q, \Sigma, \delta, q_{\mathsf{init}}, \Delta \rangle$ where Q is a set of $|Q| = n$ *states*; Σ is a set (*alphabet*) of $|\Sigma| = m$ *symbols*; $\delta : Q \times \Sigma \rightarrow Q$ is a transition function; and q_{init} is the initial state; and $\Delta : Q \rightarrow \{0, 1\}$ is a function for which $\Delta(q) = 1$ indicates that q is an accepting state.

```
client($pk, sk_1, \langle Q, \Sigma, \delta, q_{\mathsf{init}}, \Delta \rangle$)        server($pk, sk_2, \Sigma, \langle c_{kj} \rangle_{k \in [\ell], j \in [m]}$)

c401. $n \leftarrow |Q|, m \leftarrow |\Sigma|$                       s401. $m \leftarrow |\Sigma|$
c402. $\pi_0 \leftarrow \mathbf{I}$                                    s402. $\langle \lambda_{\sigma j} \rangle_{\sigma \in \Sigma, j \in [m]}$
c403. $\pi_1 \overset{\$}{\leftarrow} \mathsf{Injs}(Q \to \mathbb{R})$                    $\leftarrow \mathsf{Lagrange}(\Sigma)$
c404. $\alpha \leftarrow \mathsf{Enc}_{pk}(\pi_1(q_{\mathsf{init}}))$
                   m401. $\xrightarrow{n}$
                   m402. $\xleftarrow{\ell}$
c405. for $k \leftarrow 0 \ldots \ell - 1$                            s403. for $k \leftarrow 0 \ldots \ell - 1$
c406.     $\beta \leftarrow \mathsf{Dec}^1_{sk_1}(\alpha)$
                   m403. $\xrightarrow{\alpha,\beta}$
                                                            s404.     $\gamma \leftarrow \mathsf{Dec}^2_{sk_2}(\alpha, \beta)$
c407.     $\pi_0 \leftarrow \pi_1$                                  s405.     for $\sigma \in \Sigma$
c408.     $\pi_1 \overset{\$}{\leftarrow} \mathsf{Injs}(Q \to \mathbb{R})$                  s406.         $\Psi_\sigma \leftarrow \sum^{pk}{}_{j=0}^{m-1} \lambda_{\sigma j} \cdot_{pk} c_{kj}$
c409.     $\delta' \leftarrow \mathsf{Blind}(\delta, \pi_0, \pi_1)$                 s407.         for $i \in [n]$
c410.     $\langle a_{\sigma i} \rangle_{\sigma \in \Sigma, i \in [n]}$                       s408.             $\mu_{\sigma i} \leftarrow \gamma^i \cdot_{pk} \Psi_\sigma$
              $\leftarrow \mathsf{ToPoly}(Q, \Sigma, \delta')$                 s409.         endfor
                                                            s410.     endfor
                   m404. $\xleftarrow{\langle \mu_{\sigma i} \rangle_{\sigma \in \Sigma, i \in [n]}}$
c411.     $\alpha \leftarrow \sum^{pk}{}_{\sigma \in \Sigma} \sum^{pk}{}_{i=0}^{n-1} a_{\sigma i} \cdot_{pk} \mu_{\sigma i}$
c412. endfor                                                s411. endfor
c413. $\beta \leftarrow \mathsf{Dec}^1_{sk_1}(\alpha)$
                   m405. $\xrightarrow{\alpha,\beta}$
c414. $\pi_0 \leftarrow \pi_1$                                       s412. $\gamma \leftarrow \mathsf{Dec}^2_{sk_2}(\alpha, \beta)$
c415. $\pi_1 \overset{\$}{\leftarrow} \mathsf{Injs}(\{0,1\} \to \{0,1\})$               s413. for $i \in [n]$
c416. $\Delta' \leftarrow \mathsf{Blind}(\Delta, \pi_0, \pi_1)$                   s414.     $\mu_i \leftarrow \mathsf{Enc}_{pk}(\gamma^i)$
c417. $\langle z_i \rangle_{i \in [n]}$                                  s415. endfor
      $\leftarrow \mathsf{ToPoly}(Q, \Sigma, \Delta')$
                   m406. $\xleftarrow{\langle \mu_i \rangle_{i \in [n]}}$
c418. $\Theta \leftarrow \sum^{pk}{}_{i=0}^{n-1} z_i \cdot_{pk} \mu_i$
c419. $\beta \leftarrow \mathsf{Dec}^1_{sk_1}(\Theta)$
                   m407. $\xrightarrow{\Theta,\beta}$
                                                            s416. $\theta \leftarrow \mathsf{Dec}^2_{sk_2}(\Theta, \beta)$
                   m408. $\xleftarrow{\theta}$
c420. return $\pi_1^{-1}(\theta)$
```

Figure 5.1: Protocol $\Pi_1^{'}(\mathcal{E})$, described in Section 5.1.1

5.1.1 Our Starting Point

Our starting point is the protocol $\Pi_1(\mathcal{E})$ that we built in Chapter 3. The goal of the protocol is to enable a client having a DFA M as input to interact with a server storing the ciphertext and a file to obtain the result as though M was evaluated on the file plaintext. We first modify the protocol $\Pi_1(\mathcal{E})$ so that at the end only a binary answer is returned to the client, as opposed to the final state of the evaluation as originally designed. More precisely, the client should output a bit indicating whether the final state to which the file plaintext drives the DFA is accepting or not; i.e., if the plaintext of the file is a sequence $\langle \sigma_k \rangle_{k \in [\ell]}$ where $[\ell]$ denotes the set $\{0, 1, \ldots, \ell - 1\}$ and where each $\sigma_k \in \Sigma$, then the client should output $\Delta(\delta(\ldots \delta(\delta(q_{\mathsf{init}}, \sigma_0), \sigma_1), \ldots, \sigma_{\ell-1}))$. We also permit the client to learn the file length ℓ and the server to learn both ℓ and the number of states n in the client's DFA. The client should learn nothing else about the file, however, and the server should learn nothing else about the file or the client's DFA.

We show the modified protocol $\Pi_1^{'}(\mathcal{E})$ in Fig. 5.1. It follows $\Pi_1(\mathcal{E})$ in Fig. 3.1 except for the additional two rounds of interaction (m405-m408) at the end in order to obtain a binary answer of whether the final state is an accepting state or not. For that purpose, the client creates another polynomial $F(x) = \sum_{i=0}^{\mathbb{R}\, n-1} z_i \cdot_{\mathbb{R}} x^i$ such that $F(q) = 1$ if and only if $\Delta(q) = 1$ and $F(q) = 0$ otherwise. That is, $F(x)$ "converges" all accepting states to 1 and all non-accepting states to 0. Since the client needs help from the server to decrypt the final result (s416), it applies another random injection $\pi_1 \overset{\$}{\leftarrow} \mathsf{Injs}(\{0, 1\} \rightarrow \{0, 1\})$ on the output of the function Δ to hide the results from the server. In the protocol, we use $\Delta' \leftarrow \mathsf{Blind}(\Delta, \pi_0, \pi_1)$ (c416) to denote the step to generate the blinded function that maps the accepting state to a random number between 0 and 1. The client then uses the polynomial interpolation procedure to obtain the coefficients of $F(x)$ in c417. After "evaluating" $F(x)$ in c418 and obtaining the encrypted binary output Θ, the client interacts with the server one last time to decrypt it and returns the final result in c420.

5.1.2 Our Initial Construction

Starting from the protocol of the previous section, we develop a protocol in this section that replaces the client with two parties: a user that holds the the DFA $\langle Q, \Sigma, \delta, q_{\mathsf{init}}, \Delta \rangle$ and a proxy that the user invokes to conduct a protocol to evaluate this DFA on a file stored at the server. Notably, the protocol we develop here protects the secrecy of the DFA $\langle Q, \Sigma, \delta, q_{\mathsf{init}}, \Delta\rangle$ and the evaluation result from the proxy, and so this modification enables the proxy to execute the protocol on behalf of others who do not trust it with knowledge of the DFA. One scenario in which this protection is desirable is if the user does not have the bandwidth or processing available for performing the evaluation herself.

The protocol, denoted $\Pi_4(\mathcal{E})$, protects the DFA privacy by giving to the proxy the encryptions of the coefficients of the DFA polynomial f, denoted $\langle \hat{a}_{\sigma i}\rangle_{\sigma\in\Sigma, i\in[n]}$ where $\hat{a}_{\sigma i} \leftarrow \mathsf{Enc}_{pk}(a_{\sigma i})$ and $\langle a_{\sigma i}\rangle_{\sigma\in\Sigma, i\in[n]} \leftarrow \mathsf{ToPoly}(Q, \Sigma, \delta)$, and the encryptions of the coefficients of the converging polynomial F, i.e., $\langle \hat{z}_i\rangle_{i\in[n]}$ where $\hat{z}_i \leftarrow \mathsf{Enc}_{pk}(z_i)$ and $\langle \hat{z}_i\rangle_{i\in[n]} \leftarrow \mathsf{ToPoly}(Q, \Sigma, \Delta)$. The implications of this change to the protocol are far-reaching, due to the operations that the proxy needs to perform using these now-encrypted coefficients.

In the original protocol, in order to hide the current state transition from the server, the client blinds the current transition state by choosing a random injection π_1 of the state encodings in each round so that the server obtains a random ring element γ in s404 every time. A new DFA polynomial is then interpolated to accommodate the injections chosen in the last and current round (c407–c410) to continue state transitions consistently. When the coefficients are encrypted, however, the proxy will not be able to interpolate new polynomials because it does not have access to δ. We thus need another strategy to achieve these “blinding” and “unblinding” effects. Rather than blinding with a random injection, the new protocol does so by additively adding in a random ring element r to the ciphertext representing the current state (c503–c504). The consequence of this additive blinding operation is that the proxy needs a way to “shift” f (and its encrypted coefficients) to produce a polynomial $f'(x, y)$ satisfying $f'(q +_{\mathbb{R}} r, \sigma) = \delta(q, \sigma)$ for each $q \in Q$ and $\sigma \in \Sigma$, for a specified $r \in \mathbb{R}$. We observe that if we set

$$f'(x, y) = \sum_{\sigma\in\Sigma}^{\mathbb{R}} \left(f'_\sigma(x) \cdot_{\mathbb{R}} \Lambda_\sigma(y)\right)$$

where $f'_\sigma(x) = \mathop{\sum\nolimits^{\mathbb{R}}}_{i=0}^{n-1} a'_{\sigma i} \cdot_{\mathbb{R}} x^i$, then it suffices if $f'_\sigma(x +_{\mathbb{R}} r) = f_\sigma(x)$ for all $\sigma \in \Sigma$. Note that

$$\begin{aligned} f_\sigma(x -_{\mathbb{R}} r) &= \mathop{\sum\nolimits^{\mathbb{R}}}_{i=0}^{n-1} a_{\sigma i} \cdot_{\mathbb{R}} (x -_{\mathbb{R}} r)^i \\ &= \mathop{\sum\nolimits^{\mathbb{R}}}_{i=0}^{n-1} a_{\sigma i} \cdot_{\mathbb{R}} \mathop{\sum\nolimits^{\mathbb{R}}}_{i'=0}^{i} \binom{i}{i'} \cdot_{\mathbb{R}} x^{i-i'} \cdot_{\mathbb{R}} (-_{\mathbb{R}} r)^{i'} \qquad (5.1) \\ &= \mathop{\sum\nolimits^{\mathbb{R}}}_{i=0}^{n-1} \left(\mathop{\sum\nolimits^{\mathbb{R}}}_{i'=0}^{n-1-i} a_{\sigma(i+i')} \cdot_{\mathbb{R}} \binom{i+i'}{i'} \cdot_{\mathbb{R}} (-_{\mathbb{R}} r)^{i'} \right) \cdot_{\mathbb{R}} x^i \end{aligned}$$

where (5.1) follows from the binomial theorem. As such, setting

$$a'_{\sigma i} \leftarrow \mathop{\sum\nolimits^{\mathbb{R}}}_{i'=0}^{n-1-i} a_{\sigma(i+i')} \cdot_{\mathbb{R}} \binom{i+i'}{i'} \cdot_{\mathbb{R}} (-_{\mathbb{R}} r)^{i'} \tag{5.2}$$

ensures $f'_\sigma(x +_{\mathbb{R}} r) = f_\sigma(x)$ and, therefore, $f(x +_{\mathbb{R}} r, \sigma) = f'(x, \sigma)$. When the proxy has access to only the encrypted coefficients, represented by $\langle \hat{a}_{\sigma i} \rangle_{\sigma \in \Sigma, i \in [n]}$, the operation in (5.2) needs to be changed to

$$\hat{a}'_{\sigma i} \leftarrow \mathop{\sum\nolimits^{pk}}_{i'=0}^{n-1-i} \left(\binom{i+i'}{i'} \cdot_{\mathbb{R}} (-_{\mathbb{R}} r)^{i'} \right) \cdot_{pk} \hat{a}_{\sigma(i+i')} \tag{5.3}$$

In our pseudocode, we encapsulate calculations (Eqn. 5.3) in the invocation $\langle \hat{a}'_{\sigma i} \rangle_{\sigma \in \Sigma, i \in [n]} \leftarrow \mathsf{Shift}(r, \langle \hat{a}_{\sigma i} \rangle_{\sigma \in \Sigma, i \in [n]})$.

Now that the coefficients are encrypted, the operation by the client to combine coefficients with ciphertexts as was done in line c411 in Fig. 5.1 no longer works for the proxy. For this reason, we need to expand the properties we require of the encryption system we use, to include the ability to homomorphically “multiply” ciphertexts *once*. We emphasize that we do *not* require fully homomorphic encryption. Our construction can be instantiated with any additively homomorphic encryption scheme that allows a single homomorphic multiplication of two ciphertexts (e.g., [17, 34]), provided that it also supports two-party decryption. Here we build from the more well-studied scheme of Boneh, Goh and Nissim [17], which we denote by BGN.

Encryption scheme Specifically, BGN uses an algorithm BGNInit that, on input 1^κ, outputs $(p, p', \mathbb{G}, \mathbb{G}', e)$ where p, p' are random $\kappa/2$-bit primes, $\mathbb{G}$ and $\mathbb{G}'$ are cyclic groups of order $N = pp'$, and

$e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}'$ is a bilinear map. In this encryption scheme, the ring $\mathbb{R}$ is $\mathbb{Z}_N$, the ciphertext space $C_{\langle N,\mathbb{G},\mathbb{G}',e,g,h,\hat{g}\rangle}$ is $\mathbb{G} \cup \mathbb{G}'$, and the relevant algorithms are defined as follows. Note that we assume that elements of $\mathbb{G}$ and $\mathbb{G}'$ are encoded distinctly.

$\underline{\mathsf{Gen}(1^\kappa)}$: Generate $(p, p', \mathbb{G}, \mathbb{G}', e) \leftarrow \mathsf{BGNInit}(1^\kappa)$; select random generators $g, u \overset{\$}{\leftarrow} \mathbb{G}$; set $N \leftarrow pp'$, $h \leftarrow u^{p'}$, and $\hat{g} \leftarrow e(g,g)^p$; and return public key $\langle N, \mathbb{G}, \mathbb{G}', e, g, h, \hat{g}\rangle$ and private key $\langle N, \mathbb{G}, \mathbb{G}', e, g, \hat{g}, p\rangle$.

$\underline{\mathsf{Enc}_{\langle N,\mathbb{G},\mathbb{G}',e,g,h,\hat{g}\rangle}(m)}$: Select $x \overset{\$}{\leftarrow} \mathbb{Z}_N$ and return $g^m h^x$.

$\underline{\mathsf{Dec}_{\langle N,\mathbb{G},\mathbb{G}',e,g,\hat{g},p\rangle}(c)}$: If $c \in \mathbb{G}$, then return the discrete logarithm of $e(c,g)^p$ with respect to base $\hat{g}$. If $c \in \mathbb{G}'$, then return the discrete logarithm of c^p with respect to base $\hat{g}$.

$\underline{c_1 +_{\langle N,\mathbb{G},\mathbb{G}',e,g,h,\hat{g}\rangle} c_2}$: If c_1 and c_2 are in the same group (i.e., both are in $\mathbb{G}$ or both are in $\mathbb{G}'$), then return c_1c_2. Otherwise, if $c_1 \in \mathbb{G}$ and $c_2 \in \mathbb{G}'$, then return $e(c_1, g)c_2$.

$\underline{m \cdot_{\langle N,\mathbb{G},\mathbb{G}',e,g,h,\hat{g}\rangle} c}$: Return c^m.

$\underline{c_1 \odot_{\langle N,\mathbb{G},\mathbb{G}',e,g,h,\hat{g}\rangle} c_2}$: If $c_1, c_2 \in \mathbb{G}$, then return $e(c_1, c_2)$. Otherwise, return $\perp$.

$\underline{\mathsf{Share}(\langle N, \mathbb{G}, \mathbb{G}', e, g, \hat{g}, p\rangle)}$: Return $sk_1 = \langle \mathbb{G}, \mathbb{G}', d_1\rangle$ and $sk_2 = \langle \mathbb{G}, \mathbb{G}', e, g, \hat{g}, d_2\rangle$ where $d_1 \overset{\$}{\leftarrow} \mathbb{Z}_N$ and $d_2 \leftarrow p - d_1 \bmod N$.

$\underline{\mathsf{Dec}^1_{\langle \mathbb{G},\mathbb{G}',d_1\rangle}(c)}$: Return c^{d_1}.

$\underline{\mathsf{Dec}^2_{\langle \mathbb{G},\mathbb{G}',e,g,\hat{g},d_2\rangle}(c_1, c_2)}$: If $c_1, c_2 \in \mathbb{G}$, then return the discrete logarithm of $e(c_2c_1^{d_2}, g)$ with respect to base $\hat{g}$. If $c_1, c_2 \in \mathbb{G}'$, then return the discrete logarithm of $c_2c_1^{d_2}$ with respect to base $\hat{g}$.

Note the new operator $\odot_{pk}$ that homomorphically multiplies two ciphertexts in $\mathbb{G}$. Since the result is in $\mathbb{G}'$, it is not possible to use the result as an argument to $\odot_{pk}$. This is the sense in which this scheme permits homomorphic multiplication "once". Also note that though the basic scheme of Boneh et al. did not include $\hat{g} = e(g,g)^p$ in the public key, Boneh et al. proposed an extension supporting multiparty threshold decryption [17, Section 5] that did so[1]; it is this extension that we adopt here.

A complication of using BGN is the need to compute a discrete logarithm to decrypt in $\mathsf{Dec}_{\langle N,\mathbb{G},\mathbb{G}',e,g,\hat{g},p\rangle}$ and $\mathsf{Dec}^2_{\langle \mathbb{G},\mathbb{G}',e,g,\hat{g},d_2\rangle}$. We thus need to design our protocol so that any ciphertext that a party attempts to decrypt should hold a plaintext from a small range $0 \ldots L$. Then, Pollard's lambda method [56, p.

[1] The exact construction supporting threshold decryption was left implicit by Boneh et al. [17], but we have confirmed that including $\hat{g} = e(g,g)^p$ in the public key is what they intended [15].

128] enables recovery of the plaintext in $O(\sqrt{L})$ time. Alternatively, a precomputed table that maps $\hat{g}^m$ to the plaintext $m \in \{0 \ldots L\}$ enables decryption to be performed by table lookup.

```
proxy($pk, sk_1, \Sigma, n, \alpha_{\mathsf{init}}$                         server($pk, sk_2, \Sigma,$
      $\langle \hat{a}_{\sigma i} \rangle_{\sigma \in \Sigma, i \in [n]}, \langle \hat{z}_i \rangle_{i \in [n]}$)                     $\langle c_{kj} \rangle_{k \in [\ell], j \in [m]}$)

c501. $\alpha \leftarrow \alpha_{\mathsf{init}}$                                s501. $m \leftarrow |\Sigma|$
                                                       s502. $\langle \lambda_{\sigma j} \rangle_{\sigma \in \Sigma, j \in [m]}$
                                                               $\leftarrow \mathsf{Lagrange}(\Sigma)$
                          m501. $\xrightarrow{n}$
                          m502. $\xleftarrow{\ell}$
c502. for $k \leftarrow 0 \ldots \ell - 1$                     s503. for $k \leftarrow 0 \ldots \ell - 1$
c503.     $r \overset{\$}{\leftarrow} \{0,1\}^{\kappa'}$
c504.     $\alpha \leftarrow \alpha +_{pk} \mathsf{Enc}_{pk}(r)$
c505.     $\beta \leftarrow \mathsf{Dec}^1_{sk_1}(\alpha)$
                          m503. $\xrightarrow{\alpha, \beta}$
                                                       s504.     $\gamma \leftarrow \mathsf{Dec}^2_{sk_2}(\alpha, \beta)$
                                                       s505.     for $\sigma \in \Sigma$
                                                       s506.         $\Psi_\sigma \leftarrow \sum_{j=0}^{m-1}{}_{pk} \lambda_{\sigma j} \cdot_{pk} c_{kj}$
c506.     $\langle \hat{a}'_{\sigma i} \rangle_{\sigma \in \Sigma, i \in [n]}$                    s507.         for $i \in [n]$
            $\leftarrow \mathsf{Shift}(r, \langle \hat{a}_{\sigma i} \rangle_{\sigma \in \Sigma, i \in [n]})$   s508.             $\mu_{\sigma i} \leftarrow \gamma^i \cdot_{pk} \Psi_\sigma$
                                                       s509.         endfor
                                                       s510.     endfor
                          m504. $\xleftarrow{\langle \mu_{\sigma i} \rangle_{\sigma \in \Sigma, i \in [n]}}$
c507.     $\alpha \leftarrow \sum_{\sigma \in \Sigma}{}_{pk} \sum_{i=0}^{n-1}{}_{pk} \hat{a}'_{\sigma i} \odot_{pk} \mu_{\sigma i}$
c508. endfor                                           s511. endfor
c509. $r \overset{\$}{\leftarrow} \{0,1\}^{\kappa'}$
c510. $\alpha \leftarrow \alpha +_{pk} \mathsf{Enc}_{pk}(r)$
c511. $\beta \leftarrow \mathsf{Dec}^1_{sk_1}(\alpha)$
                          m505. $\xrightarrow{\alpha, \beta}$
                                                       s512. $\gamma \leftarrow \mathsf{Dec}^2_{sk_2}(\alpha, \beta)$
c512. $\langle \hat{z}_i{}' \rangle_{i \in [n]}$                               s513. for $i \in [n]$
      $\leftarrow \mathsf{Shift}(r, \langle \hat{z}_i \rangle_{i \in [n]})$                   s514.     $\mu_i \leftarrow \mathsf{Enc}_{pk}(\gamma^i)$
                                                       s515. endfor
                          m506. $\xleftarrow{\langle \mu_i \rangle_{i \in [n]}}$
c513. $\Theta \leftarrow \sum_{i=0}^{n-1}{}_{pk} \hat{z}_i{}' \odot_{pk} \mu_i$
c514. return $\Theta$
```

Figure 5.2: Protocol $\Pi_4(\mathcal{E})$, described in Section 5.1.2

Protocol steps Protocol $\Pi_4(\mathcal{E})$ is shown in Fig. 5.2. It has a similar structure to $\Pi_1(\mathcal{E})$, but differs in many respects.

- Rather than taking $\langle Q,\ \Sigma,\ \delta,\ q_{\mathsf{init}},\ \Delta\rangle$ as input, the proxy takes $\alpha_{\mathsf{init}} \leftarrow \mathsf{Enc}_{pk}(q_{\mathsf{init}})$ and encrypted coefficients $\langle \hat{a}_{\sigma i}\rangle_{\sigma\in\Sigma, i\in[n]}$ and $\langle \hat{z}_i\rangle_{i\in[n]}$ as input. Fig. 5.2 presumes that these coefficients are created by performing $\langle a_{\sigma i}\rangle_{\sigma\in\Sigma, i\in[n]} \leftarrow \mathsf{ToPoly}(Q, \Sigma, \delta)$ and $\langle z_i\rangle_{i\in[n]} \leftarrow \mathsf{ToPoly}(Q, \Sigma, \Delta)$ and then encrypting each coefficient using pk, i.e., $\hat{a}_{\sigma i} \leftarrow \mathsf{Enc}_{pk}(a_{\sigma i})$ for each $\sigma \in \Sigma$ and $i \in [n]$, and $\hat{z_i} \leftarrow \mathsf{Enc}_{pk}(z_i)$ for each $i \in [n]$.

- Because server decrypts the (blinded) DFA state in line s504, the plaintext should be adequately small so that decryption — which as discussed above, involves computing (or looking up) a discrete logarithm if BGN encryption is in use — is not too costly. For this reason, and assuming $\mathbb{R} = \mathbb{Z}_N$ (as it is in BGN) and $Q = [n]$, the blinding term r is drawn from $\{0,1\}^{\kappa'}$ instead of $\mathbb{R}$, where $\kappa' \ll \kappa$ is another security parameter. Then, the statistical distance between the distribution of γ seen by server in line s504 when the blinded state is q (i.e., when $\gamma = q +_{\mathbb{R}} r$) and uniformly random choices from $\{0,1\}^{\kappa'}$ is

$$\begin{aligned} &\sum_x \left| \begin{array}{r} \mathbb{P}(q + r = x \mid r \overset{\$}{\leftarrow} \{0,1\}^{\kappa'}) \\ -\mathbb{P}(r = x \mid r \overset{\$}{\leftarrow} \{0,1\}^{\kappa'}) \end{array} \right| \\ &= \sum_{0 \le x < q} \frac{1}{2^{\kappa'}} + \sum_{2^{\kappa'} \le x < q + 2^{\kappa'}} \frac{1}{2^{\kappa'}} \\ &= \frac{q}{2^{\kappa'-1}} \end{aligned}$$

  Since $q \in [n]$, we anticipate setting $\kappa' \approx \log_2 n + 15$ to achieve a reasonable balance between decryption cost and security for moderately sized n. It is important to note, however, that generally κ' will need to grow with n (though only logarithmically so).

- The fact that each $\hat{a}_{\sigma i}$ is a ciphertext necessitates using the "one-time multiplication" operator $\odot_{pk}$ in line c507 to produce the ciphertext of the new state, versus $\cdot_{pk}$ as in line c411. The same is true for each $\hat{z_i}$ in c513.

- The protocol returns an encrypted evaluation result Θ to the proxy (c514), and so the original round to decrypt the result (m407–m408) is omitted.

Protocol security We are able to prove $\Pi_4(\mathcal{E})$ protects DFA privacy and file content privacy against arbitrarily malicious server adversaries, and DFA privacy and file privacy against *honest-but-curious* proxy adversaries. We do not present the proofs here, but in the next section we develop an optimized protocol that has better efficiency and achieves similar security properties. We will formally define the security notions and prove that protocol secure in Section 5.3.

5.2 Optimizations

In this section, we detail a series of optimizations that we developed for our protocol that, in our implementation, achieved an order of magnitude improvement in performance.

5.2.1 File Representation

We first observe that, in protocol $\Pi_4(\mathcal{E})$, the computation done by the server in s506 using the Lagrange coefficients it computed in s502 is effectively evaluating the ciphertext of $\Lambda_\sigma(\sigma_k)$ for each $\sigma \in \Sigma$, using the values $\langle c_{kj} \rangle_{j\in[m]}$ provided as input to the server where $c_{kj} \leftarrow \mathsf{Enc}_{pk}((\sigma_k)^j)$. Recall that only for $\sigma = \sigma_k$ does $\Lambda_\sigma(\sigma_k) = 1$; otherwise, $\Lambda_\sigma(\sigma_k) = 0$. Since this calculation only depends on the ciphertexts of the current file character, the result of it, i.e., $\langle \mathsf{Enc}_{pk}(\Lambda_\sigma(\sigma_k)) \rangle_{\sigma\in\Sigma}$, could have been provided by the data owner as *the* ciphertext of file character σ_k so that the server would not need to compute it itself.

With this observation, our first optimization is to eliminate the use of the Lagrange polynomial $\Lambda_\sigma(y)$ completely and decompose the original bivariate polynomial $f(x, y)$ to m univariate polynomials $f_\sigma(x)$ for each $\sigma \in \Sigma$. The encryption of a file character σ_k now becomes a vector of encryptions of 0's and one 1. Specifically, σ_k is provided to the server as ciphertexts $\langle c_{k\sigma} \rangle_{\sigma\in\Sigma}$ where $c_{k\sigma} \leftarrow \mathsf{Enc}_{pk}(1)$ if $\sigma = \sigma_k$ and $c_{k\sigma} \leftarrow \mathsf{Enc}_{pk}(0)$ otherwise. This representation has the same storage costs per file character as the original protocol, i.e., m ciphertexts per encrypted file character.

5.2.2 Pairing Operations

During the implementation of our protocol, we noticed that the pairing operations performed by the proxy in c507 are very costly and became the bottleneck of the overall performance. Accelerating pairing operations is a research area of substantial interest and any progress made would be beneficial to protocols such as ours that utilize pairing. Our focus here, however, is twofold. One is to adapt the protocol to reduce the number of pairing operations. In particular, mn pairing operations are needed in c507 in each round. In this section, we redesign the protocol to reduce the number of pairing operation down to m. The other focus is to make the protocol design amenable to pairing preprocessing [53].

Informally, given a bilinear map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}'$, if it is known in advance that a particular value $c \in \mathbb{G}$ will be paired with other elements multiple times, then preprocessing on c can be performed in advance to achieve a significant reduction in pairing time. For example, for the class of machines used in our experiments in Section 5.4, a pairing operation for a 1024-bit BGN scheme costs around 35ms without preprocessing but only 10ms after preprocessing. In c507, the pairing operation performed is $e(\hat{a}'_{\sigma i}, \mu_{\sigma i})$. Unfortunately, both $\hat{a}'_{\sigma i}$ and $\mu_{\sigma i}$ change in each round, which prohibits preprocessing. This suggests that performing pairing operations on the proxy side may not be the best choice in terms of the potential for optimization.

We therefore redesigned the protocol with the goals of reducing the number of pairing operations and making pairing preprocessing possible. Fortunately, we were able to achieve both goals by shifting the pairing operations to the server side. The resulting protocol $\Pi_5(\mathcal{E})$ is shown in Fig. 5.3. The new protocol essentially switches the roles of the proxy and server (though not entirely, since each still receives the same inputs). Note that the directions of the messages m603 and m604 are reversed from those in Fig. 5.2. The values α and β, which used to be produced by the proxy in c504 and c505, are now produced by the server in s604 and s605. This role reversal imposes some significant changes in the computations done by the proxy and server.

We now describe the changes made in the protocol. We ask the readers to ignore the operations c603 and s606 for the time being; these will be discussed in Section 5.2.3. The proxy now obtains γ in c602, which is equal to $q + r$, where q is the current DFA state and r was chosen by the server in s603. It now uses γ as input (as opposed to r in $\Pi_4(\mathcal{E})$) into the Shift procedure first described in Section 5.1.2, i.e., computing new coefficients as:

$$\hat{a}'_{\sigma i} \leftarrow \sideset{}{^{pk}}\sum_{i'=0}^{n-1-i} \hat{a}_{\sigma(i+i')} \cdot_{pk} \binom{i+i'}{i'} \cdot_{pk} \gamma^{i'} \tag{5.4}$$

As a consequence of this "shift", the plaintexts of the coefficients $\langle \hat{a}'_{\sigma i} \rangle_{\sigma \in \Sigma, i \in [n]}$ define a new polynomial $f'_\sigma(x)$ such that $f'_\sigma(x) = f_\sigma(x +_{\mathbb{R}} (q +_{\mathbb{R}} r))$. The proxy then sends $\langle \hat{a}'_{\sigma i} \rangle_{\sigma \in \Sigma, i \in [n]}$ to the server in m604. The server, knowing r, blindly "evaluates" the polynomial $f'_\sigma(x)$ on value $(-_{\mathbb{R}}\ r)$ for each $\sigma \in \Sigma$, in lines s607–s609. Specifically, it computes a ciphertext of $f'_\sigma(-_{\mathbb{R}} r) = f_\sigma(-_{\mathbb{R}} r +_{\mathbb{R}} (q +_{\mathbb{R}} r)) = f_\sigma(q)$ as:

$$\omega_\sigma \leftarrow \sideset{}{^{pk}}\sum_{i=0}^{n-1} \hat{a}'_{\sigma i} \cdot_{pk} (-_{\mathbb{R}} r)^i \tag{5.5}$$

A naive way to compute Eqn. 5.5 requires $O(n^2)$ exponentiations, but by leveraging Horner's rule it can be reduced to $O(n)$ exponentiations. Once the server obtains $\{\omega_\sigma\}_{\sigma \in \Sigma}$, it calculates a ciphertext α of the correct next DFA state in line s610, i.e., by homomorphically summing $\omega_\sigma \odot_{pk} c_{k\sigma}$ over all $\sigma \in \Sigma$. (Recall from Section 5.2.1 that, for fixed k, exactly one of $\langle c_{k\sigma} \rangle_{\sigma \in \Sigma}$ is a ciphertext of 1 and the rest are ciphertexts of 0.) The key point to notice here is that, by rearranging the protocol messages and letting the proxy send over the shifted coefficients, the number of pairing operations are chopped down to only m from nm, a major improvement.

We have already alluded the potential benefit of pairing preprocessing to reduce the online cost of pairing operations. The only question left is how to adapt the protocol so that it is amenable to using this technique. Fortunately, the changes we have just made to the protocol also makes pairing preprocessing possible. The pairing operation that the server needs to perform in s610 is $e(\omega_\sigma, c_{k\sigma})$, for $\sigma \in \Sigma$ and $k \in [\ell]$. The ciphertext $c_{k\sigma}$ is fixed and known even before the protocol starts. This allows the server to perform pairing preprocessing using these ciphertexts offline and store them to stable storage for future use. During the protocol run, the preprocessing information can be retrieved and used to greatly reduce the online costs of the pairing operations.

5.2.3 Shifting

After the above optimizations, a remaining computation in the protocol that is especially expensive is the Shift procedure, i.e., Eqn. 5.4, which is performed as part of c604 (and c608). Computing each $\hat{a}'_{\sigma i}$ requires $O(n)$ exponentiations with exponents being powers of γ. Since γ is κ' bits, this exponentiation is increasingly expensive as κ' grows, and is one of the performance bottlenecks of our implementation for the κ' values we employ. (As discussed in Section 5.1.2, we take $\kappa' \approx \log_2 n + 15$ in our present implementation, though this setting is an artifact of using BGN encryption and could be larger with another encryption scheme.) Our next target is thus to find ways to optimize this operation.

One possibility is to use a smaller κ' to speed up the exponentiations. However, κ' cannot be arbitrarily reduced without compromising the security of the protocol. Instead, here we propose letting the proxy reduce γ modulo n before feeding it into the Shift procedure, i.e., to compute:

$$\hat{a}'_{\sigma i} \leftarrow \sum_{i'=0}^{n-1-i}{}_{pk}\ \hat{a}_{\sigma(i+i')} \cdot_{pk} \binom{i+i'}{i'} \cdot_{pk} (\gamma \bmod n)^{i'} \tag{5.6}$$

in Shift, instead. (See c603 and c607.) By reducing γ, the exponents that used to be $O(n\kappa')$ bits long are now reduced to $O(n \log n)$ bits, after taking into account the exponentiations on γ itself. However, this change does have implications for the correctness of the computation. Referring to the derivation in Eqn. 5.5, now that the proxy shifted the polynomial by $\gamma \bmod n$, the server needs to adapt to this change accordingly. Intuitively, it should evaluate the new polynomial on $(-r \bmod n)$ as opposed to on $-_{\mathbb{R}} r$ in Eqn. 5.5, in which case it computes a ciphertext ω_σ of

$$\begin{aligned} f'_\sigma(-r \bmod n) &= f_\sigma((-r \bmod n) +_{\mathbb{R}} ((q +_{\mathbb{R}} r) \bmod n)) \\ &= \begin{cases} f_\sigma(q) & \text{if } n \mid r \text{ or } q + (r \bmod n) \geq n \\ f_\sigma(q+n) & \text{otherwise} \end{cases} \end{aligned}$$

assuming that $\kappa' + 1 < \kappa$. (See lines s606 and s615.) However, as indicated, there are two possible outcomes from this calculation. One is exactly what we want, i.e., a ciphertext of $f_\sigma(q)$. The other possibility is a ciphertext of $f_\sigma(q+n)$, which is problematic because $f_\sigma(q+n)$ is arbitrary. The server unfortunately cannot tell which case happened because everything it operates on is encrypted.

Our solution to this problem is to add constraints when constructing $f_\sigma(x)$ so that $f_\sigma(q+n) = f_\sigma(q)$ for all $q \in Q$ and $\sigma \in \Sigma$. These additional constraints guarantee the correct state transition regardless of which case happens. However, the price we pay is that the degree of $f_\sigma(x)$ increases to $2n-1$ since additional n constraints need to be added to define the polynomial. But the performance gains we achieve outweigh this loss.

Another key insight to draw from this technique is the fact that $\gamma \bmod n$ can only take on n different values and so there can be at most n different sets of coefficients $\langle \hat{a}'_{\sigma i} \rangle_{\sigma \in \Sigma, i \in [2n]}$ from the calculation of the Shift procedure in Eqn. 5.6. This allows the proxy to precompute all possible sets of $\langle \hat{a}'_{\sigma i} \rangle_{\sigma \in \Sigma, i \in [2n]}$ for each $\gamma \bmod n \in [n]$ and store them in a table before the start of the protocol. It can then simply perform table lookups depending on which value of $\gamma \bmod n$ it obtains in c603 (or c607). This way, the proxy does not need to perform the computations in Eqn. 5.6 during the protocol except for randomizing the ciphertexts before sending them back to the server. This offers tremendous performance gains for the protocol: Without applying this optimization, the cost for the proxy to calculate Eqn. 5.6 for all $\langle \hat{a}'_{\sigma i} \rangle_{\sigma \in \Sigma, i \in [2n]}$ involves $O(mn^2)$ exponentiations in each round. After applying this optimization, it is reduced to only $O(mn)$ exponentiations, due to the need for ciphertext randomizations.

5.2.4 Packing the Result Ciphertexts

When using $\Pi_5(\mathcal{E})$ to evaluate a DFA on k files, k encrypted evaluation results $\Theta_0, \ldots, \Theta_{k-1}$ — each the ciphertext of a 0 or 1 — need to be communicated back to the user. Sending these k ciphertexts individually to the user introduces an undesirably high communication cost between the proxy and the user. A better approach is for the proxy to aggregate multiple such results into a single ciphertext before sending these results back to the user. Specifically, the proxy can aggregate these k ciphertexts into ciphertexts $\vec{\Theta}_0, \ldots, \vec{\Theta}_{\lceil k/z \rceil - 1}$ where

$$\vec{\Theta}_i \leftarrow \sum_{j=0}^{z-1}{}_{pk}\, 2^j \cdot_{pk} \Theta_{iz+j}$$

for each $i \in [\lceil k/z \rceil]$. This aggregation is omitted from Fig. 5.3. The user can then decrypt each $\vec{\Theta}_i$ to recover all the evaluation results. The value of z is upper bounded by the bit length of the

```
proxy$(pk, sk_1, \Sigma, n, \alpha_{\mathsf{init}},$                    server$(pk, sk_2, \Sigma, \langle c_{k\sigma} \rangle_{k \in [\ell], \sigma \in \Sigma})$
    $\langle \hat{a}_{\sigma i} \rangle_{\sigma \in \Sigma, i \in [2n]}, \langle \hat{z}_i \rangle_{i \in [2n]})$

                  m601.  $\xrightarrow{n, \alpha_{\mathsf{init}}}$
                                                  s601.  $\alpha \leftarrow \alpha_{\mathsf{init}}$
                  m602.  $\xleftarrow{\ell}$
c601.  for $k \leftarrow 0 \ldots \ell - 1$                   s602.  for $k \leftarrow 0 \ldots \ell - 1$
                                                  s603.      $r \overset{\$}{\leftarrow} \{0,1\}^{\kappa'}$
                                                  s604.      $\alpha \leftarrow \alpha +_{pk} \mathsf{Enc}_{pk}(r)$
                                                  s605.      $\beta \leftarrow \mathsf{Dec}^1_{sk_2}(\alpha)$
                  m603.  $\xleftarrow{\alpha, \beta}$
c602.      $\gamma \leftarrow \mathsf{Dec}^2_{sk_1}(\alpha, \beta)$
c603.      $\gamma \leftarrow \gamma \bmod n$                     s606.      $r \leftarrow -r \bmod n$
c604.      $\langle \hat{a}'_{\sigma i} \rangle_{\sigma \in \Sigma, i \in [2n]}$
               $\leftarrow \mathsf{Shift}(\gamma, \langle \hat{a}_{\sigma i} \rangle_{\sigma \in \Sigma, i \in [2n]})$
                  m604.  $\xrightarrow{\langle \hat{a}'_{\sigma i} \rangle_{\sigma \in \Sigma, i \in [2n]}}$
                                                  s607.      for $\sigma \in \Sigma$
                                                  s608.          $\omega_\sigma \leftarrow \sum_{i=0}^{2n-1}{}_{pk}\ \hat{a}'_{\sigma i} \cdot_{pk} r^i$
                                                  s609.      endfor
                                                  s610.      $\alpha \leftarrow \sum_{\sigma \in \Sigma}{}_{pk}\ \omega_\sigma \odot_{pk} c_{k\sigma}$
c605.  endfor                                     s611.  endfor
                                                  s612.  $r \overset{\$}{\leftarrow} \{0,1\}^{\kappa'}$
                                                  s613.  $\alpha \leftarrow \alpha +_{pk} \mathsf{Enc}_{pk}(r)$
                                                  s614.  $\beta \leftarrow \mathsf{Dec}^1_{sk_2}(\alpha)$
                  m605.  $\xleftarrow{\alpha, \beta}$
c606.  $\gamma \leftarrow \mathsf{Dec}^2_{sk_1}(\alpha, \beta)$
c607.  $\gamma \leftarrow \gamma \bmod n$                         s615.  $r \leftarrow -r \bmod n$
c608.  $\langle \hat{z}_i{}' \rangle_{i \in [2n]}$
           $\leftarrow \mathsf{Shift}(\gamma, \langle \hat{z}_i \rangle_{i \in [2n]})$
                  m606.  $\xrightarrow{\langle \hat{z}_i{}' \rangle_{i \in [2n]}}$
                                                  s616.  $\Theta \leftarrow \sum_{i=0}^{2n-1}{}_{pk}\ \hat{z}_i{}' \cdot_{pk} r^i$
                  m607.  $\xleftarrow{\Theta}$
c609.  return $\Theta$
```

Figure 5.3: Optimized protocol $\Pi_5(\mathcal{E})$, described in Section 5.2

plaintext space of the cryptosystem $\mathcal{E}$ in general, and in the case of BGN, z must be restricted to a small value such as κ' to enable efficient decryption by the user.

This packing technique generalizes nicely to support evaluating conjunctions or disjunctions of d DFAs on k files. That is, after the proxy interacts with the server to evaluate DFAs M_0, ..., M_d on each of k files, yielding encrypted results $\Theta_{0,0}$, ..., $\Theta_{k-1,d-1}$, the proxy can aggregate these kd ciphertexts into ciphertexts $\vec{\Theta}_0, \ldots, \vec{\Theta}_{\lceil k/z' \rceil - 1}$ where $z' = \lfloor z / \lceil \log_2(d+1) \rceil \rfloor$ and

$$\vec{\Theta}_i \leftarrow \sum_{j=0}^{z'-1}{}_{pk} \left(2^{j \lceil \log_2(d+1) \rceil} \cdot_{pk} \sum_{d'=0}^{d-1}{}_{pk} \Theta_{iz'+j,d'} \right)$$

for each $i \in [\lceil k/z' \rceil]$. Upon decrypting each such aggregate ciphertext, each $\lceil \log_2(d+1) \rceil$-length sequence of bits represents the number of DFAs M_0, ... M_{d-1} that the corresponding file matched. That is, the file satisfies the disjunction of these DFAs if that count is nonzero, and it satisfies the conjunction of these DFAs if that count is d.

To evaluate other Boolean combinations of d DFAs on files, it suffices for the proxy and server to evaluate each DFA individually on each file and communicate the results per DFA to the user, and the user can herself determine which files match the Boolean combination she is interested in. While less communication-efficient than the above approach for conjunctions and disjunctions, this approach is more computationally efficient for the proxy and server than combining all d DFAs into a single large DFA that represents the Boolean combination of interest.

5.3 Protocol Security

In this section, we prove the security of $\Pi_5(\mathcal{E})$. We show that the protocol provably protects the privacy of both the DFA and file contents from either arbitrarily malicious proxy or arbitrarily malicious server adversaries.

5.3.1 Security Against Server Adversaries

In this section we bound the advantage that an arbitrarily malicious server gains by executing this protocol, in terms of its ability to determine either the DFA that the proxy is evaluating or the

```
Experiment $\mathbf{Expt}^{\mathtt{s\text{-}dfa}}_{\Pi_5(\mathcal{E})}(S_1, S_2)$
  $(pk, sk) \leftarrow \mathsf{Gen}(1^\kappa)$
  $(sk_1, sk_2) \leftarrow \mathsf{Share}(sk)$
  $(\ell, \langle \sigma_k \rangle_{k \in [\ell]}, M_0, M_1, \phi) \leftarrow S_1(pk, sk_2)$
  $\mathbf{if}\ |M_0.Q| \neq |M_1.Q|\ \mathbf{or}\ M_0.\Sigma \neq M_1.\Sigma\ \mathbf{then\ return}\ 0$
  $b \xleftarrow{\$} \{0, 1\}$
  $\langle Q, \Sigma, \delta, q_{\mathsf{init}}, \Delta \rangle \leftarrow M_b$
  $n \leftarrow |Q|$, $m \leftarrow |\Sigma|$
  $\mathbf{for}\ k \in [\ell], \sigma \in \Sigma$
    $\mathbf{if}\ \sigma = \sigma_k\ \mathbf{then}\ c_{k\sigma} \leftarrow \mathsf{Enc}_{pk}(1)$
                $\mathbf{else}\ c_{k\sigma} \leftarrow \mathsf{Enc}_{pk}(0)$
  $\langle a_{\sigma i} \rangle_{\sigma \in \Sigma, i \in [2n]} \leftarrow \mathsf{ToPoly}(Q, \Sigma, \delta)$
  $\langle z_i \rangle_{i \in [2n]} \leftarrow \mathsf{ToPoly}(Q, \Sigma, \Delta)$
  $\mathbf{for}\ \sigma \in \Sigma, i \in [2n]$
    $\hat{a}_{\sigma i} \leftarrow \mathsf{Enc}_{pk}(a_{\sigma i})$
  $\mathbf{for}\ i \in [2n]$
    $\hat{z}_i \leftarrow \mathsf{Enc}_{pk}(z_i)$
  $\alpha_{\mathsf{init}} \leftarrow \mathsf{Enc}_{pk}(q_{\mathsf{init}})$
  $b' \leftarrow S_2^{\mathsf{proxyOr}\left(\substack{pk,\, sk_1,\, \Sigma,\, n,\, \alpha_{\mathsf{init}},\\ \langle \hat{a}_{\sigma i} \rangle_{\sigma \in \Sigma, i \in [2n]},\, \langle \hat{z}_i \rangle_{i \in [2n]}}\right)}(\phi, \langle c_{k\sigma} \rangle_{k \in [\ell], \sigma \in \Sigma})$
  $\mathbf{if}\ b' = b\ \mathbf{then\ return}\ 1$
          $\mathbf{else\ return}\ 0$
```

Figure 5.4: Experiments for proving DFA privacy of $\Pi_5(\mathcal{E})$ against server adversaries

plaintext of the file in its possession. That is, we prove the *privacy* of the file and DFA inputs against server adversaries.

Following the security definitions in Chapter 3, we formalize our security claims against server compromise by defining two separate server adversaries. The first server adversary $S = (S_1, S_2)$ attacks the encrypted DFA $M = \langle Q,\ \Sigma,\ \delta,\ q_{\mathsf{init}},\ \Delta \rangle$, i.e., $\langle \hat{a}_{\sigma i} \rangle_{\sigma \in \Sigma, i \in [n]}$ held by the proxy, as described in experiment $\mathbf{Expt}^{\mathtt{s\text{-}dfa}}_{\Pi_5(\mathcal{E})}$ in Fig. 5.4. S_1 first generates a file $\langle \sigma_k \rangle_{k \in [\ell]}$ and two DFAs M_0, M_1. (Note that we use, e.g., "$M_0.Q$" and "$M_1.Q$" to disambiguate their state sets.) S_2 is then invoked with the ciphertexts $\langle c_{k\sigma} \rangle_{k \in [\ell], \sigma \in \Sigma}$ of its file and information ϕ created for it by S_1, and is given oracle access to proxyOr. proxyOr is given input arguments pertaining to one of the two DFAs output by S_1, selected at random (as indicated by b).

proxyOr responds to queries from S_2 as follows, ignoring malformed queries. The first query (say, consisting of simply "start") causes proxyOr to begin the protocol; proxyOr responds with a message of the form n, α_{init} (i.e., of the form of message m601). The second invocation by S_2 must include a single integer ℓ (i.e., of the form of message m602). The next ℓ queries by S_2 must be the form α and β, i.e., two values as in message m603, to which proxyOr responds by sending $2nm$ elements of C_{pk}, i.e., $\langle \hat{a}'_{\sigma i} \rangle_{\sigma \in \Sigma, i \in [2n]}$ as in m604. S_2's next query to proxyOr again must contain two values of the form α and β (as in m605), to which proxyOr responds with $2n$ ciphertexts $\langle \hat{z_i}' \rangle_{i \in [2n]}$ as in m606. The next (and last) query by S_2 can consist one element of C_{pk} as in m607.

Eventually S_2 outputs a bit b', and $\mathbf{Expt}^{\texttt{s-dfa}}_{\Pi_5(\mathcal{E})}(S) = 1$ only if $b' = b$. We say the *advantage* of an arbitrarily malicious S is

$$\mathbf{Adv}^{\texttt{s-dfa}}_{\Pi_5(\mathcal{E})}(S) = 2 \cdot \mathbb{P}\left(\mathbf{Expt}^{\texttt{s-dfa}}_{\Pi_5(\mathcal{E})}(S) = 1\right) - 1$$

and define $\mathbf{Adv}^{\texttt{s-dfa}}_{\Pi_5(\mathcal{E})}(t, \ell, n, m) = \max_S \mathbf{Adv}^{\texttt{s-dfa}}_{\Pi_5(\mathcal{E})}(S)$ where the maximum is taken over all adversaries S taking time t and selecting a file of length ℓ and DFAs containing n states and an alphabet of m symbols.

We reduce DFA privacy against server attacks to the IND-CPA [10] security of the encryption scheme, which was defined in the experiment in Fig. 3.3 in Chapter 3, in which an adversary U is provided a public key $\hat{pk}$ and access to an oracle $\mathsf{Enc}^{\hat{b}}_{\hat{pk}}(\cdot, \cdot)$ that consistently encrypts either the first of its two inputs (if $\hat{b} = 0$) or the second of those inputs (if $\hat{b} = 1$). Eventually U outputs a guess $\hat{b}'$ at $\hat{b}$, and $\mathbf{Expt}^{\texttt{ind-cpa}}_{\mathcal{E}}(U) = 1$ only if $\hat{b}' = \hat{b}$. The IND-CPA advantage of U is defined as

$$\mathbf{Adv}^{\texttt{ind-cpa}}_{\mathcal{E}}(U) = 2 \cdot \mathbb{P}\left(\mathbf{Expt}^{\texttt{ind-cpa}}_{\mathcal{E}}(U) = 1\right) - 1$$

and then $\mathbf{Adv}^{\texttt{ind-cpa}}_{\mathcal{E}}(t, w) = \max_U \mathbf{Adv}^{\texttt{ind-cpa}}_{\mathcal{E}}(U)$ where the maximum is taken over all adversaries U executing in time t and making w queries to $\mathsf{Enc}^{\hat{b}}_{\hat{pk}}(\cdot, \cdot)$.

In our theorem statements, we omit terms that are negligible as a function of the security parameters κ and κ'. For any $\mathcal{E}$ operation op, we use t_{op} to denote the time required to perform op; e.g., t_{Dec} is the time to perform a Dec operation.

Theorem 8. *For* $t' = t + t_{\mathsf{Share}} + (\ell m + 2nm + 2n + 1) \cdot t_{\mathsf{Enc}}$,

$$\mathbf{Adv}^{\texttt{s-dfa}}_{\Pi_5(\mathsf{BGN})}(t, \ell, n, m) \le n^{\ell+1} \mathbf{Adv}^{\texttt{ind-cpa}}_{\mathsf{BGN}}(t', 2nm + 2n + 1)$$

Proof. Given an adversary $S = (S_1, S_2)$ for $\Pi_5(\mathsf{BGN})$ that runs in time t, produces a file of length ℓ, and produces DFAs of n states over an alphabet of m symbols, we construct an IND-CPA attacker U for BGN to demonstrate the theorem as follows. On input a BGN public key $\hat{pk} = \langle N, \mathbb{G}, \mathbb{G}', e, g, h, \hat{g}\rangle$, U sets $d_2 \xleftarrow{\$} \mathbb{Z}_N$, and invokes $S_1(\hat{pk}, sk_2)$ where $sk_2 = \langle \mathbb{G}, \mathbb{G}', d_2\rangle$ to obtain $(\ell, \langle \sigma_k\rangle_{k\in[\ell]}, M_0, M_1, \phi)$. Note that d_2 is chosen from a distribution that is perfectly indistinguishable from that from which d_2 is chosen in the real system. If $|M_0.Q| \neq |M_1.Q|$ or $M_0.\Sigma \neq M_1.\Sigma$, then U aborts the simulation. Otherwise, letting $\Sigma = M_0.\Sigma$, $m = |\Sigma|$ and $n = |M_0.Q|$, U computes $\langle a_{0\sigma i}\rangle_{\sigma\in\Sigma, i\in[2n]} \leftarrow \mathsf{ToPoly}(M_0.Q, \Sigma, M_0.\delta)$ and $\langle a_{1\sigma i}\rangle_{\sigma\in\Sigma, i\in[2n]} \leftarrow \mathsf{ToPoly}(M_1.Q, \Sigma, M_1.\delta)$, and it sets $\hat{a}_{\sigma i} \leftarrow \mathsf{Enc}^{\hat{b}}_{\hat{pk}}(a_{0\sigma i}, a_{1\sigma i})$ for $\sigma \in \Sigma$ and $i \in [n]$. It then computes $\langle z_{0i}\rangle_{i\in[2n]} \leftarrow \mathsf{ToPoly}(M_0.Q, \Sigma, M_0.\Delta)$ and $\langle z_{1i}\rangle_{i\in[2n]} \leftarrow \mathsf{ToPoly}(M_1.Q, \Sigma, M_1.\Delta)$, and it sets $\hat{z}_i \leftarrow \mathsf{Enc}^{\hat{b}}_{\hat{pk}}(z_{0i}, z_{1i})$ for $i \in [n]$. U finally sets $\alpha_{\mathsf{init}} \leftarrow \mathsf{Enc}^{\hat{b}}_{\hat{pk}}(M_0.q_{\mathsf{init}}, M_1.q_{\mathsf{init}})$, and then for all $k \in [\ell], \sigma \in \Sigma$, it sets $c_{k\sigma} \leftarrow \mathsf{Enc}_{pk}(1)$ if $\sigma = \sigma_k$ and $c_{k\sigma} \leftarrow \mathsf{Enc}_{pk}(0)$ otherwise.

U then invokes $S_2(\phi, \langle c_{k\sigma}\rangle_{k\in[\ell],\sigma\in\Sigma})$ and simulates responses to S_2's queries to $\mathsf{proxyOr}$ as follows (ignoring malformed invocations). Upon initializing S_2, U sends $n, \alpha_{\mathsf{init}}$ to S_2 and gets ℓ in return. For the k-th query of the form α, β ($0 \leq k < \ell$), U selects $\gamma \xleftarrow{\$} [n]$, as opposed to decrypting it as in a real execution (see c602 and c603). It computes $\langle \hat{a}'_{\sigma i}\rangle_{\sigma\in\Sigma, i\in[2n]} \leftarrow \mathsf{Shift}(\gamma, \langle \hat{a}_{\sigma i}\rangle_{\sigma\in\Sigma, i\in[2n]})$ as in c604 and returns $\langle \hat{a}'_{\sigma i}\rangle_{\sigma\in\Sigma, i\in[2n]}$ to S_2. For the $(\ell+1)$-th query of the form α, β, U again randomly sets $\gamma \xleftarrow{\$} [n]$ and $\langle \hat{z}_i{}'\rangle_{i\in[2n]} \leftarrow \mathsf{Shift}(\gamma, \langle \hat{z}_i\rangle_{i\in[2n]})$ and then sends $\langle \hat{z}_i{}'\rangle_{i\in[2n]}$ to S_2. Finally, when S_2 outputs b', U outputs b', as well.

U's simulation is perfectly indistinguishable from the real system to an arbitrarily malicious server adversary S if and only if U made the correct guesses on γ in each round. When that happens, the advantage of U winning his game is the same with that of S. So, $\mathbf{Adv}^{\texttt{ind-cpa}}_{\mathsf{BGN}}(U) \geq (\frac{1}{n})^{\ell+1}\mathbf{Adv}^{\texttt{s-dfa}}_{\Pi_5(\mathsf{BGN})}(S)$. Note that U runs in time $t' = t + t_{\mathsf{Share}} + \ell m \cdot t_{\mathsf{Enc}} + (2nm+2n+1)\cdot t_{\mathsf{Enc}}$ due to the need to generate a secret key share for S, to generate $\langle c_{kj}\rangle_{k\in[\ell], j\in[m]}$, and to make $2nm+2n+1$ encryption oracle queries to create $\langle \hat{a}_{\sigma i}\rangle_{\sigma\in\Sigma, i\in[2n]}$, $\langle \hat{z}_i\rangle_{i\in[2n]}$ and α_{init}. □

The second server adversary $S = (S_1, S_2)$ attacks the file for which it holds the per-symbol ciphertexts $\langle c_{k\sigma}\rangle_{k\in[\ell],\sigma\in\Sigma}$ as in experiment $\mathbf{Expt}^{\texttt{s-file}}_{\Pi_5(\mathcal{E})}$ shown in Fig. 5.5. Here, S_1 produces two separate, equal-length plaintext files $\langle \sigma_{0k}\rangle_{k\in[\ell]}$, $\langle \sigma_{1k}\rangle_{k\in[\ell]}$ and a DFA M. S_2 then receives the ciphertexts $\langle c_{k\sigma}\rangle_{k\in[\ell],\sigma\in\Sigma}$ for file $\langle \sigma_{bk}\rangle_{k\in[\ell]}$ where b is chosen randomly. S_2 is also given oracle

```
Experiment $\mathbf{Expt}^{\texttt{s-file}}_{\Pi_5(\mathcal{E})}(S_1, S_2)$
    $(pk, sk) \leftarrow \mathsf{Gen}(1^\kappa)$
    $(sk_1, sk_2) \leftarrow \mathsf{Share}(sk)$
    $(\ell, \langle \sigma_{0k} \rangle_{k \in [\ell]}, \langle \sigma_{1k} \rangle_{k \in [\ell]}, M, \phi) \leftarrow S_1(pk, sk_2)$
    $b \xleftarrow{\$} \{0, 1\}$
    $\langle Q, \Sigma, \delta, q_{\mathsf{init}}, \Delta \rangle \leftarrow M$
    $n \leftarrow |Q|, m \leftarrow |\Sigma|$
    for $k \in [\ell], \sigma \in \Sigma$
        if $\sigma = \sigma_{bk}$ then $c_{k\sigma} \leftarrow \mathsf{Enc}_{pk}(1)$
                    else $c_{k\sigma} \leftarrow \mathsf{Enc}_{pk}(0)$
    $\langle a_{\sigma i} \rangle_{\sigma \in \Sigma, i \in [2n]} \leftarrow \mathsf{ToPoly}(Q, \Sigma, \delta)$
    $\langle z_i \rangle_{i \in [2n]} \leftarrow \mathsf{ToPoly}(Q, \Sigma, \Delta)$
    for $\sigma \in \Sigma, i \in [2n]$
        $\hat{a}_{\sigma i} \leftarrow \mathsf{Enc}_{pk}(a_{\sigma i})$
    for $i \in [2n]$
        $\hat{z}_i \leftarrow \mathsf{Enc}_{pk}(z_i)$
    $\alpha_{\mathsf{init}} \leftarrow \mathsf{Enc}_{pk}(q_{\mathsf{init}})$
    $b' \leftarrow S_2^{\mathsf{proxyOr}\left(\substack{pk, sk_1, \Sigma, n, \alpha_{\mathsf{init}}, \\ \langle \hat{a}_{\sigma i} \rangle_{\sigma \in \Sigma, i \in [2n]}, \langle \hat{z}_i \rangle_{i \in [2n]}}\right)}(\phi, \langle c_{k\sigma} \rangle_{k \in [\ell], \sigma \in \Sigma})$
    if $b' = b$ then return $1$
            else return $0$
```

Figure 5.5: Experiments for proving file privacy of $\Pi_5(\mathcal{E})$ against server adversaries

access to $\mathsf{proxyOr}(pk, sk_1, \Sigma, n, \alpha_{\mathsf{init}}, \langle \hat{a}_{\sigma i} \rangle_{\sigma \in \Sigma, i \in [2n]}, \langle \hat{z}_i{}' \rangle_{i \in [2n]})$. The interaction between S_2 and proxyOr is similar to what was described for the server DFA adversary. Eventually S_2 outputs a bit b', and $\mathbf{Expt}^{\texttt{s-file}}_{\Pi_5(\mathcal{E})}(S) = 1$ iff $b' = b$. The *advantage* of S is

$$\mathbf{Adv}^{\texttt{s-file}}_{\Pi_5(\mathcal{E})}(S) = 2 \cdot \mathbb{P}\left(\mathbf{Expt}^{\texttt{s-file}}_{\Pi_5(\mathcal{E})}(S) = 1\right) - 1$$

and then $\mathbf{Adv}^{\texttt{s-file}}_{\Pi_5(\mathcal{E})}(t, \ell, n, m) = \max_S \mathbf{Adv}^{\texttt{s-file}}_{\Pi_5(\mathcal{E})}(S)$ where the maximum is taken over all adversaries $S = (S_1, S_2)$ taking time t and producing (from S_1) files of ℓ symbols and a DFA of n states and alphabet of size m.

Theorem 9. *For* $t' = t + t_{\mathsf{Share}} + (\ell m + 2nm + 2n + 1) \cdot t_{\mathsf{Enc}}$,

$$\mathbf{Adv}^{\texttt{s-file}}_{\Pi_5(\mathsf{BGN})}(t, \ell, n, m) \leq n^{\ell+1} \mathbf{Adv}^{\texttt{ind-cpa}}_{\mathsf{BGN}}(t', \ell m)$$

Proof. Given an adversary $S = (S_1, S_2)$ running in time t and selecting files of length ℓ symbols and a DFA of n states over an alphabet of m symbols, we construct an IND-CPA adversary U. On input a BGN public key $\hat{pk} = \langle N, \mathbb{G}, \mathbb{G}', e, g, h, \hat{g}\rangle$, U sets $d_2 \xleftarrow{\$} \mathbb{Z}_N$, and invokes $S_1(\hat{pk}, sk_2)$ where $sk_2 = \langle \mathbb{G}, \mathbb{G}', d_2\rangle$ to obtain $(\ell, \langle\sigma_{0k}\rangle_{k\in[\ell]}, \langle\sigma_{1k}\rangle_{k\in[\ell]}, M, \phi)$, where $M = \langle Q, \Sigma, q_{\mathsf{init}}, \delta, \Delta\rangle$ is a DFA. Note that d_2 is chosen from a distribution that is perfectly indistinguishable from that from which d_2 is chosen in the real system. For $k \in [\ell]$ and $\sigma \in \Sigma$, U sets $c_{kj} \leftarrow \mathsf{Enc}^{\hat{b}}_{\hat{pk}}(I_0, I_1)$ where

$$I_0 \leftarrow \begin{cases} 1 & \text{if } \sigma = \sigma_{0k} \\ 0 & \text{otherwise} \end{cases} \qquad I_1 \leftarrow \begin{cases} 1 & \text{if } \sigma = \sigma_{1k} \\ 0 & \text{otherwise} \end{cases}$$

U also sets $\alpha_{\mathsf{init}} \leftarrow \mathsf{Enc}_{\hat{pk}}(q_{\mathsf{init}})$, $\langle a_{\sigma i}\rangle_{\sigma\in\Sigma, i\in[2n]} \leftarrow \mathsf{ToPoly}(Q, \Sigma, \delta)$, and $\langle z_i\rangle_{i\in[2n]} \leftarrow \mathsf{ToPoly}(Q, \Sigma, \Delta)$. U then computes $\hat{a}_{\sigma i} \leftarrow \mathsf{Enc}_{\hat{pk}}(a_{\sigma i})$ and $\hat{z}_i \leftarrow \mathsf{Enc}_{\hat{pk}}(z_i)$ for all $\sigma \in \Sigma$ and $i \in [2n]$.

U then invokes $S_2(\phi, \langle c_{k\sigma}\rangle_{k\in[\ell],\sigma\in\Sigma})$ and simulates responses to S_2's queries to proxyOr as follows (ignoring malformed invocations). Upon initializing S_2, U sends n, α_{init} to S_2 and gets ℓ in return. For the k-th query of the form α, β ($0 \le k < \ell$), U selects $\gamma \xleftarrow{\$} [n]$, as opposed to decrypting it as in a real execution c602 and c603. U then sets $\langle \hat{a}'_{\sigma i}\rangle_{\sigma\in\Sigma,i\in[2n]} \leftarrow \mathsf{Shift}(\gamma, \langle\hat{a}_{\sigma i}\rangle_{\sigma\in\Sigma,i\in[2n]})$ as done in c604 and returns $\langle \hat{a}'_{\sigma i}\rangle_{\sigma\in\Sigma,i\in[2n]}$ to S_2. For the $(\ell+1)$-th query of the form α, β, U again randomly sets $\gamma \xleftarrow{\$} [n]$ and then computes $\langle \hat{z}_i{}'\rangle_{i\in[2n]} \leftarrow \mathsf{Shift}(\gamma, \langle\hat{z}_i\rangle_{i\in[2n]})$ and sends $\langle\hat{z}_i{}'\rangle_{i\in[2n]}$ to S_2. Finally when S_2 outputs b', U outputs b', as well.

This simulation is perfectly indistinguishable from the real system provided that U made correct guesses for γ on each round of the simulation. When that happens, U wins his game if and only if S wins his. So we have $\mathbf{Adv}^{\texttt{ind-cpa}}_{\mathsf{BGN}}(U) \ge (\frac{1}{n})^{\ell+1}\mathbf{Adv}^{\texttt{s-file}}_{\Pi_5(\mathsf{BGN})}(S)$. U runs in time $t' = t + t_{\mathsf{Share}} + (2nm + 2n + 1)\cdot t_{\mathsf{Enc}} + \ell m \cdot t_{\mathsf{Enc}}$ due to the need to generate a secret key share for S, to generate α_{init}, $\langle\hat{a}_{\sigma i}\rangle_{\sigma\in\Sigma,i\in[2n]}$ and $\langle\hat{z}_i\rangle_{i\in[2n]}$, and to make ℓm queries to its encryption oracle to create $\langle c_{k\sigma}\rangle_{k\in[\ell],\sigma\in\Sigma}$. □

The multiplicative factor of $n^{\ell+1}$ that appears in Thm. 8 and Thm. 9, while independent of the security parameters κ and κ', nevertheless renders these theorems of limited practical use. That said, we have no reason to believe that the actual security of $\Pi_5(\mathsf{BGN})$ against server adversaries decays so dramatically as a function of ℓ. Rather, this factor is simply an artifact of our proof method, and since the server in $\Pi_5(\mathsf{BGN})$ receives (aside from the value n) only ciphertexts from the proxy

```
Experiment $\mathbf{Expt}^{\texttt{p-dfa}}_{\Pi_5(\mathcal{E})}(P_1, P_2)$
    $(pk, sk) \leftarrow \mathsf{Gen}(1^\kappa)$
    $(sk_1, sk_2) \leftarrow \mathsf{Share}(sk)$
    $(\ell, \langle \sigma_k \rangle_{k \in [\ell]}, M_0, M_1, \phi) \leftarrow P_1(pk, sk_1)$
    $\mathbf{if}\ |M_0.Q| \neq |M_1.Q|\ \mathbf{or}\ M_0.\Sigma \neq M_1.\Sigma,\ \mathbf{then\ return}\ 0$
    $b \xleftarrow{\$} \{0, 1\}$
    $\langle Q, \Sigma, \delta, q_{\mathsf{init}}, \Delta \rangle \leftarrow M_b$
    $n \leftarrow |Q|, m \leftarrow |\Sigma|$
    $\mathbf{for}\ k \in [\ell], \sigma \in \Sigma$
        $\mathbf{if}\ \sigma = \sigma_k\ \mathbf{then}\ c_{k\sigma} \leftarrow \mathsf{Enc}_{pk}(1)$
                 $\mathbf{else}\ c_{k\sigma} \leftarrow \mathsf{Enc}_{pk}(0)$
    $\langle a_{\sigma i} \rangle_{\sigma \in \Sigma, i \in [2n]} \leftarrow \mathsf{ToPoly}(Q, \Sigma, \delta)$
    $\langle z_i \rangle_{i \in [2n]} \leftarrow \mathsf{ToPoly}(Q, \Sigma, \Delta)$
    $\mathbf{for}\ \sigma \in \Sigma, i \in [2n]$
        $\hat{a}_{\sigma i} \leftarrow \mathsf{Enc}_{pk}(a_{\sigma i})$
    $\mathbf{for}\ i \in [2n]$
        $\hat{z}_i \leftarrow \mathsf{Enc}_{pk}(z_i)$
    $\alpha_{\mathsf{init}} \leftarrow \mathsf{Enc}_{pk}(q_{\mathsf{init}})$
    $b' \leftarrow P_2^{\mathsf{serverOr}\left(\substack{pk, sk_2, \Sigma, \\ \langle c_{k\sigma} \rangle_{k \in [\ell], \sigma \in \Sigma}}\right)}(\phi, \alpha_{\mathsf{init}}, \langle \hat{a}_{\sigma i} \rangle_{\sigma \in \Sigma, i \in [2n]}, \langle \hat{z}_i \rangle_{i \in [2n]})$
    $\mathbf{if}\ b' = b\ \mathbf{then\ return}\ 1$
              $\mathbf{else\ return}\ 0$
```

Figure 5.6: Experiments for proving DFA privacy of $\Pi_5(\mathcal{E})$ against proxy adversaries

created using a public key for which it does not hold the private key, we believe these theorems to be overwhelmingly conservative.

5.3.2 Security Against Proxy Adversaries

In this section we analyze the privacy of the DFA and file from proxy adversaries, specifically *honest-but-curious* ones. Our protocol's security is limited to honest-but-curious proxies as an artifact of using BGN encryption, specifically because this forces us to employ $\kappa' \ll \kappa$. Advances in additively homomorphic encryption that also supports "one-time" homomorphic multiplication, and that also permits us to employ $\kappa' \approx \kappa$, would permit us to prove security against malicious proxy adversaries, as well.

The case of proxy adversaries in $\Pi_5(\mathcal{E})$ differs more substantially from that in Chapter 3. For one, we need to formalize and prove a result about the degree to which the DFA is protected from the proxy. Such an experiment for defining this type of security is shown in Fig. 5.6. In this experiment,

P_2 is invoked with encrypted coefficients $\langle \hat{a}_{\sigma i} \rangle_{\sigma \in \Sigma, i \in [2n]}$, $\langle \hat{z}_i \rangle_{i \in [2n]}$ and the encrypted initial state α_{init} for one of two DFAs output by P_1 (determined by random selection of b). P_2 can invoke serverOr first with an integer n and a ciphertext (as in m601), in response to which serverOr returns ℓ (as in m602). In the next ℓ rounds, each time serverOr sends ciphertexts α and β (as in m603), to P_2. P_2 then responds with ciphertexts $\langle \hat{a}'_{\sigma i} \rangle_{\sigma \in \Sigma, i \in [2n]}$ in response (as in m604). The next round, serverOr again sends α and β (as in m605) to S_2, who responds with ciphertexts $\langle \hat{z_i}' \rangle_{i \in [2n]}$ as in m606. serverOr sends one last message consisting one element in C_{pk} to S_2 as in m607. Finally, P_2 outputs a bit b', and $\mathbf{Expt}^{\texttt{p-dfa}}_{\Pi_5(\mathcal{E})}(P) = 1$ only if $b' = b$.

We prove DFA privacy against *honest-but-curious* proxy adversaries. A proxy adversary (P_1, P_2) is honest-but-curious if P_2 invokes serverOr exactly as $\Pi_5(\mathcal{E})$ prescribes. The honest-but-curious advantage $\mathbf{hbcAdv}^{\texttt{p-dfa}}_{\Pi_5(\mathcal{E})}(P)$ of $P = (P_1, P_2)$ is

$$\mathbf{hbcAdv}^{\texttt{p-dfa}}_{\Pi_5(\mathcal{E})}(P) = 2 \cdot \mathbb{P}\left(\mathbf{Expt}^{\texttt{p-file}}_{\Pi_5(\mathcal{E})}(P) = 1\right) - 1$$

and $\mathbf{hbcAdv}^{\texttt{p-dfa}}_{\Pi_5(\mathcal{E})}(t, \ell, n, m) = \max_P \mathbf{Adv}^{\texttt{p-dfa}}_{\Pi_5(\mathcal{E})}(P)$ where the maximum is taken over all honest-but-curious client adversaries P running in total time t and producing files of length ℓ and a DFA of n over an alphabet of m symbols.

We now prove the DFA privacy against an honest-but-curious proxy adversary.

Theorem 10. *For* $t' = t + t_{\mathsf{Share}} + (\ell m + 2nm + 2n + 2) \cdot t_{\mathsf{Enc}}$,

$$\mathbf{hbcAdv}^{\texttt{p-dfa}}_{\Pi_5(\mathsf{BGN})}(t, \ell, n, m) \leq \mathbf{Adv}^{\texttt{ind-cpa}}_{\mathsf{BGN}}(t', 2(nm + n + 1))$$

Proof. Given an adversary $P = (P_1, P_2)$ running in time t and selecting files of length ℓ and a DFA of n states over an alphabet of m symbols, we construct an IND-CPA adversary U. On input a BGN public key $\hat{pk} = \langle N, \mathbb{G}, \mathbb{G}', e, g, h, \hat{g} \rangle$, U sets $d_1 \xleftarrow{\$} \mathbb{Z}_N$, and invokes $P_1(\hat{pk}, sk_1)$ where $sk_1 = \langle \mathbb{G}, \mathbb{G}', d_1 \rangle$ to obtain $(\ell, \langle \sigma_k \rangle_{k \in [\ell]}, M_0, M_1, \phi)$. Note that d_1 is chosen from a distribution that is perfectly indistinguishable from that from which d_1 is chosen in the real system. If $|M_0.Q| \neq |M_1.Q|$ or $M_0.\Sigma \neq M_1.\Sigma$, then U aborts the simulation. Letting $\Sigma = M_0.\Sigma$, $m = |\Sigma|$ and $n = |M_0.Q|$, U computes $\langle a_{0\sigma i} \rangle_{\sigma \in \Sigma, i \in [2n]} \leftarrow \mathsf{ToPoly}(M_0.Q,\ \Sigma,\ M_0.\delta)$ and $\langle a_{1\sigma i} \rangle_{\sigma \in \Sigma, i \in [2n]} \leftarrow \mathsf{ToPoly}(M_1.Q, \Sigma, M_1.\delta)$, and then sets $\hat{a}_{\sigma i} \leftarrow \mathsf{Enc}^{\hat{b}}_{\hat{pk}}(a_{0\sigma i},\ a_{1\sigma i})$ for $\sigma \in \Sigma$ and $i \in [2n]$. It also computes $\langle z_{0i} \rangle_{i \in [2n]} \leftarrow \mathsf{ToPoly}(M_0.Q, \Sigma, M_0.\Delta)$ and $\langle z_{1i} \rangle_{i \in [2n]} \leftarrow \mathsf{ToPoly}(M_1.Q, \Sigma, M_1.\Delta)$, and then sets $\hat{z}_i \leftarrow \mathsf{Enc}^{\hat{b}}_{\hat{pk}}(z_{0i}, z_{1i})$ for $i \in [2n]$. U then sets $\alpha_{\mathsf{init}} \leftarrow \mathsf{Enc}^{\hat{b}}_{\hat{pk}}(M_0.q_{\mathsf{init}}, M_1.q_{\mathsf{init}})$. For all $k \in [\ell], \sigma \in \Sigma$, U also sets $c_{k\sigma} \leftarrow \mathsf{Enc}_{pk}(1)$ if $\sigma = \sigma_k$ and $c_{k\sigma} \leftarrow \mathsf{Enc}_{pk}(0)$ otherwise.

U invokes $P_2(\phi, \alpha_{\mathsf{init}}, \langle \hat{a}_{\sigma i} \rangle_{\sigma \in \Sigma, i \in [2n]}, \langle \hat{z}_i \rangle_{i \in [2n]})$ and simulates responses to P_2's queries to $\mathsf{serverOr}$ as follows. Upon receiving the first message from P_2, U sends back ℓ. In each round, U sets $r \overset{\$}{\leftarrow} \{0,1\}^{\kappa'}$, $\alpha \leftarrow \mathsf{Enc}_{\hat{pk}}(r)$ and $\beta \leftarrow g^r \alpha^{-d_1}$ so that $\alpha^{d_1}\beta = g^r$. It then sends α and β to P_2. In the last round, upon receiving $\langle \hat{z}_i \rangle_{i \in [n]}$ as in $\mathsf{m606}$, U sets $\Theta \leftarrow \mathsf{Enc}^{\hat{b}}_{\hat{pk}}(M_0(\langle \sigma_k \rangle_{k \in [\ell]}), M_1(\langle \sigma_k \rangle_{k \in [\ell]}))$ where $M_0(\langle \sigma_k \rangle_{k \in [\ell]})$ denotes the evaluation result of M_0 on the file and similarly for $M_1(\langle \sigma_k \rangle_{k \in [\ell]})$. U then sends Θ to P_2. Finally, when P_2 outputs b', U outputs b' as well.

U's simulation is statistically indistinguishable (as a function of κ') from a real protocol execution as long as P_2 is honest-but-curious. So $\mathbf{Adv}^{\texttt{ind-cpa}}_{\mathsf{BGN}}(U) \geq \mathbf{hbcAdv}^{\texttt{p-dfa}}_{\Pi_5(\mathsf{BGN})}(P)$. U runs in time $t' = t + t_{\mathsf{Share}} + \ell m \cdot t_{\mathsf{Enc}} + (2nm + 2n + 2) \cdot t_{\mathsf{Enc}}$ due to the need to generate a secret key share for P, to create $\langle c_{k\sigma} \rangle_{k \in [\ell], \sigma \in \Sigma}$, and to make $2nm + 2n + 2$ queries to its encryption oracle in order to generate $\langle \hat{a}_{\sigma i} \rangle_{\sigma \in \Sigma, i \in [2n]}$, $\langle \hat{z}_i \rangle_{i \in [2n]}$, α_{init} and Θ in the final round. □

Next, we consider security against attacks on the encrypted files from a proxy adversary. Since the proxy no longer learns the final state of the DFA evaluation, we do not require the proxy adversary to choose two files *that produce the same final result* for the user's DFA, compared to the definition defined in Fig. 3.5 in Chapter 3. The experiment that we use to define file security against proxy adversaries $\mathbf{Expt}^{\texttt{p-file}}_{\Pi_5(\mathcal{E})}$ is shown in Fig. 5.7. There, P_1 produces two separate, equal-length plaintext files $\langle \sigma_{0k} \rangle_{k \in [\ell]}$, $\langle \sigma_{1k} \rangle_{k \in [\ell]}$ and a DFA M. P_2 then receives the ciphertexts $\langle c_{k\sigma} \rangle_{k \in [\ell], \sigma \in \Sigma}$ for file $\langle \sigma_{bk} \rangle_{k \in [\ell]}$ where b is chosen randomly. P_2 is also given oracle access to $\mathsf{serverOr}$ and finally P_2 outputs a bit b', and $\mathbf{Expt}^{\texttt{p-file}}_{\Pi_5(\mathcal{E})}(P) = 1$ iff $b' = b$. The advantage of an honest-but-curious adversary $P = (P_1, P_2)$ is defined as:

$$\mathbf{hbcAdv}^{\texttt{p-file}}_{\Pi_5(\mathcal{E})}(P) = 2 \cdot \mathbb{P}\left(\mathbf{Expt}^{\texttt{p-file}}_{\Pi_5(\mathcal{E})}(P) = 1\right) - 1$$

and $\mathbf{hbcAdv}^{\texttt{p-file}}_{\Pi_5(\mathcal{E})}(t, \ell, n, m) = \max_P \mathbf{hbcAdv}^{\texttt{p-file}}_{\Pi_5(\mathcal{E})}(P)$ where the maximum is taken over all honest-but-curious proxy adversaries P running in total time t and producing files of length ℓ and a DFA of n over an alphabet of m symbols. We now prove:

```
Experiment $\mathbf{Expt}^{\texttt{p-file}}_{\Pi_5(\mathcal{E})}(P_1, P_2)$
    $(pk, sk) \leftarrow \mathsf{Gen}(1^\kappa)$
    $(sk_1, sk_2) \leftarrow \mathsf{Share}(sk)$
    $(\ell, \langle \sigma_{0k} \rangle_{k \in [\ell]}, \langle \sigma_{1k} \rangle_{k \in [\ell]}, M, \phi) \leftarrow P_1(pk, sk_1)$
    $b \xleftarrow{\$} \{0, 1\}$
    $\langle Q, \Sigma, \delta, q_{\mathsf{init}}, \Delta \rangle \leftarrow M$
    $n \leftarrow |Q|, m \leftarrow |\Sigma|$
    for $k \in [\ell], \sigma \in \Sigma$
        if $\sigma = \sigma_{bk}$ then $c_{k\sigma} \leftarrow \mathsf{Enc}_{pk}(1)$
                  else $c_{k\sigma} \leftarrow \mathsf{Enc}_{pk}(0)$
    $\langle a_{\sigma i} \rangle_{\sigma \in \Sigma, i \in [2n]} \leftarrow \mathsf{ToPoly}(Q, \Sigma, \delta)$
    $\langle z_i \rangle_{i \in [2n]} \leftarrow \mathsf{ToPoly}(Q, \Sigma, \delta)$
    for $\sigma \in \Sigma, i \in [2n]$
        $\hat{a}_{\sigma i} \leftarrow \mathsf{Enc}_{pk}(a_{\sigma i})$
    for $i \in [2n]$
        $\hat{z}_i \leftarrow \mathsf{Enc}_{pk}(z_i)$
    $\alpha_{\mathsf{init}} \leftarrow \mathsf{Enc}_{pk}(q_{\mathsf{init}})$
    $b' \leftarrow P_2^{\mathsf{serverOr}\left(\substack{pk, sk_2, \Sigma, \\ \langle c_{k\sigma} \rangle_{k \in [\ell], \sigma \in \Sigma}}\right)}(\phi, \alpha_{\mathsf{init}}, \langle \hat{a}_{\sigma i} \rangle_{\sigma \in \Sigma, i \in [2n]}, \langle \hat{z}_i \rangle_{i \in [2n]})$
    if $b' = b$ then return $1$
               else return $0$
```

Figure 5.7: Experiments for proving file privacy of $\Pi_5(\mathcal{E})$ against proxy adversaries

Theorem 11. *For* $t' = t + t_{\mathsf{Share}} + (2nm + 2n + \ell m + 2) \cdot t_{\mathsf{Enc}}$,

$$\mathbf{hbcAdv}^{\texttt{p-file}}_{\Pi_5(\mathsf{BGN})}(t, \ell, n, m) \leq \mathbf{Adv}^{\texttt{ind-cpa}}_{\mathsf{BGN}}(t', \ell m + 1)$$

Proof. Given an adversary $P = (P_1, P_2)$ running in time t and selecting files of length ℓ and a DFA of n states over an alphabet of m symbols, we construct an IND-CPA adversary U. On input a BGN public key $\hat{pk} = \langle N, \mathbb{G}, \mathbb{G}', e, g, h, \hat{g} \rangle$, U sets $d_1 \xleftarrow{\$} \mathbb{Z}_N$, and invokes $P_1(\hat{pk}, sk_1)$ where $sk_1 = \langle \mathbb{G}, \mathbb{G}', d_1 \rangle$ to obtain $(\ell, \langle \sigma_{0k} \rangle_{k \in [\ell]}, \langle \sigma_{1k} \rangle_{k \in [\ell]}, M, \phi)$. Note that d_1 is chosen from a distribution that is perfectly indistinguishable from that from which d_1 is chosen in the real system. Let $\Sigma = M.\Sigma$, $Q = M.Q$, $\Delta = M.\Delta$, $m = |\Sigma|$ and $n = |Q|$. For $k \in [\ell]$ and $\sigma \in \Sigma$, U sets $c_{kj} \leftarrow \mathsf{Enc}^{\hat{b}}_{\hat{pk}}(I_0, I_1)$ where

$$I_0 \leftarrow \begin{cases} 1 & \text{if } \sigma = \sigma_{0k} \\ 0 & \text{otherwise} \end{cases} \qquad I_1 \leftarrow \begin{cases} 1 & \text{if } \sigma = \sigma_{1k} \\ 0 & \text{otherwise} \end{cases}$$

U also sets $\alpha_{\mathsf{init}} \leftarrow \mathsf{Enc}_{\hat{pk}}(q_{\mathsf{init}})$, $\langle a_{\sigma i}\rangle_{\sigma\in\Sigma, i\in[2n]} \leftarrow \mathsf{ToPoly}(Q, \Sigma, \delta)$, and $\langle z_i\rangle_{i\in[2n]} \leftarrow \mathsf{ToPoly}(Q, \Sigma, \Delta)$. U then computes $\hat{a}_{\sigma i} \leftarrow \mathsf{Enc}_{\hat{pk}}(a_{\sigma i})$ and $\hat{z}_i \leftarrow \mathsf{Enc}_{\hat{pk}}(z_i)$ for all $\sigma \in \Sigma$ and $i \in [2n]$.

U invokes $P_2(\phi, \alpha_{\mathsf{init}}, \langle \hat{a}_{\sigma i}\rangle_{\sigma\in\Sigma, i\in[2n]}, \langle \hat{z}_i\rangle_{i\in[2n]})$ and simulates responses to P_2's queries to serverOr as follows. Upon receiving the first message from P_2, U sends back ℓ. In each round, U sets $r \overset{\$}{\leftarrow} \{0,1\}^{\kappa'}$, $\alpha \leftarrow \mathsf{Enc}_{\hat{pk}}(r)$ and $\beta \leftarrow g^r \alpha^{-d_1}$ so that $\alpha^{d_1}\beta = g^r$. It then sends α and β to P_2. In the last round, upon receiving $\langle \hat{z}_i\rangle_{i\in[n]}$ as in m606, U sets $\Theta \leftarrow \mathsf{Enc}^{\hat{b}}_{\hat{pk}}(M(\langle \sigma_{0k}\rangle_{k\in[\ell]}), M(\langle \sigma_{1k}\rangle_{k\in[\ell]}))$. U then sends Θ to P_2. Finally, when P_2 outputs b', U outputs b' as well.

U's simulation is statistically indistinguishable (as a function of κ') from a real protocol execution as long as P_2 is honest-but-curious. So $\mathbf{Adv}^{\texttt{ind-cpa}}_{\mathsf{BGN}}(U) \geq \mathbf{hbcAdv}^{\texttt{p-file}}_{\Pi_5(\mathsf{BGN})}(P)$. U runs in time $t' = t + t_{\mathsf{Share}} + (2nm + 2n + 1)\cdot t_{\mathsf{Enc}} + (\ell m + 1)\cdot t_{\mathsf{Enc}}$ due to the need to generate a secret key share for P, to generate $\langle \hat{a}_{\sigma i}\rangle_{\sigma\in\Sigma, i\in[2n]}$, $\langle \hat{z}_i\rangle_{i\in[2n]}$ and α_{init}, and to make $\ell m + 1$ queries to its encryption oracle in order to create $\langle c_{k\sigma}\rangle_{k\in[\ell],\sigma\in\Sigma}$ and Θ in the last round. □

5.4 Performance Evaluation

5.4.1 Implementation

We implemented our optimized protocol $\Pi_5(\mathcal{E})$ in Java using an open source Java pairing based cryptography library jPBC [21], which is built on the original C pairing library PBC [53]. Our regular-expression-to-DFA conversion engine is built around the Java dk.brics.automaton library [57]. The complete implementation contains about 5000 physical source lines of code.

For the evaluation we report here, we chose a BGN public key of size $\kappa = 1024$ and a secondary security parameter of $\kappa' = 22$. To further improve performance, we utilized a fixed-base windowing exponentiation technique [56] to accelerate exponentiation operations on the proxy side. We also take advantage of pairing preprocessing at the server (see Section 5.2.2) and compare the performance with and without this optimization. In particular, pairing preprocessing produces approximately 600KB of information per BGN ciphertext and so increases the required storage dramatically. As such, it may not be appropriate for use in some environments.

To exploit parallelisms available in the protocol computation and the physical hardware, we implemented two levels of parallelization for the server and proxy programs. The first level is a thread pool of *workers* each running a single server or proxy instance. Each server worker grabs an encrypted email from a shared queue of all the encrypted emails being searched and runs a protocol instance with its paired proxy worker independently. Each server or proxy worker can further spawn up to m threads to assist in its computation. This level of parallelization is designed to take advantage of the computational independence found in many calculations in the protocol. For example, for server workers, the calculation in line s608 can be split among up to m threads before combining the results to obtain α. Similarly for client workers, the Shift procedure in line c604 can also be dispatched to up to m threads.

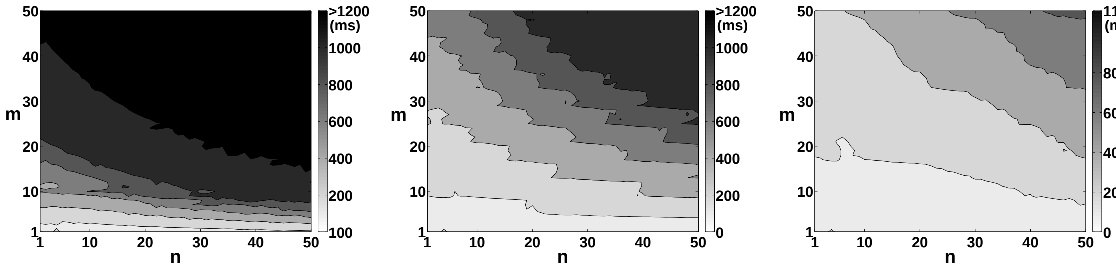
5.4.2 Microbenchmarks

We first report microbenchmarks for our implementation. The experiments reported below were conducted using two machines, each equipped with 2 quad-core Intel(R) Xeon(R) 2.67GHz CPUs with simultaneous multithreading enabled. All proxy workers ran on one of these machines, and all server workers ran on the other.

To understand the performance cost of the protocol and the impact of its two parameters, i.e., the number of DFA states n and the alphabet size m, we conducted experiments measuring the average time spent by the server and proxy for processing one character (or one round of protocol execution) for various combinations of n and m. For this purpose, we generated encrypted files each consisting of 20 characters for $m = 1$ to $m = 50$. We then created random DFAs with number of states n ranging from 1 to 50 and ran them against the files. We computed the average time spent per character by dividing the total time spent processing a file by 20. The results, with pairing preprocessing disabled, are presented in contour graphs in Fig. 5.8 where the times are binned into ranges, each shown as a band representing the range indicated in the sidebar legend.

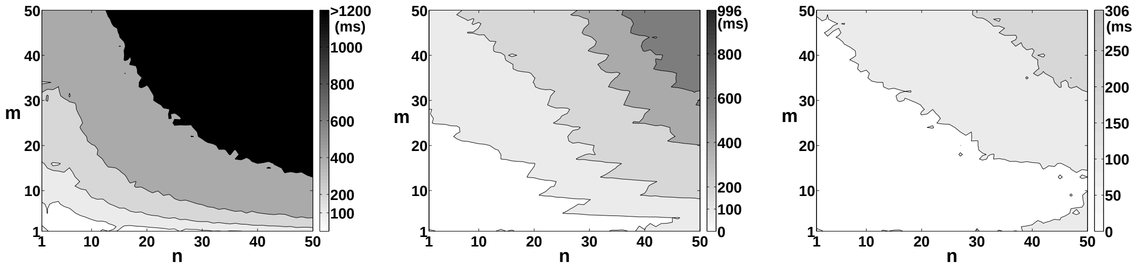
To demonstrate that the computation of the protocol is highly parallelizable, in this experiment we launched a single worker on both server and proxy machines and tested its performance when 1, 4 and 16 threads are spawned by each worker to assist in their computations, shown in Fig. 5.8a, Fig. 5.8b, and Fig. 5.8c, respectively. It is clear from all three graphs that the protocol performance scales much better with the increase of n than with m. This phenomenon is due to the fact that the number of expensive pairing operations performed by the server in each round is equal to m, and the cost resulted from the increase of m significantly outweighs that resulting from the increase of

n. These results also show that the protocol is highly amenable to parallelization, with dramatically decreased processing time as the number of threads increases.



(a) 1 thread per worker (b) 4 threads per worker (c) 16 threads per worker

Figure 5.8: Time spent per file character in milliseconds, with pairing preprocessing disabled

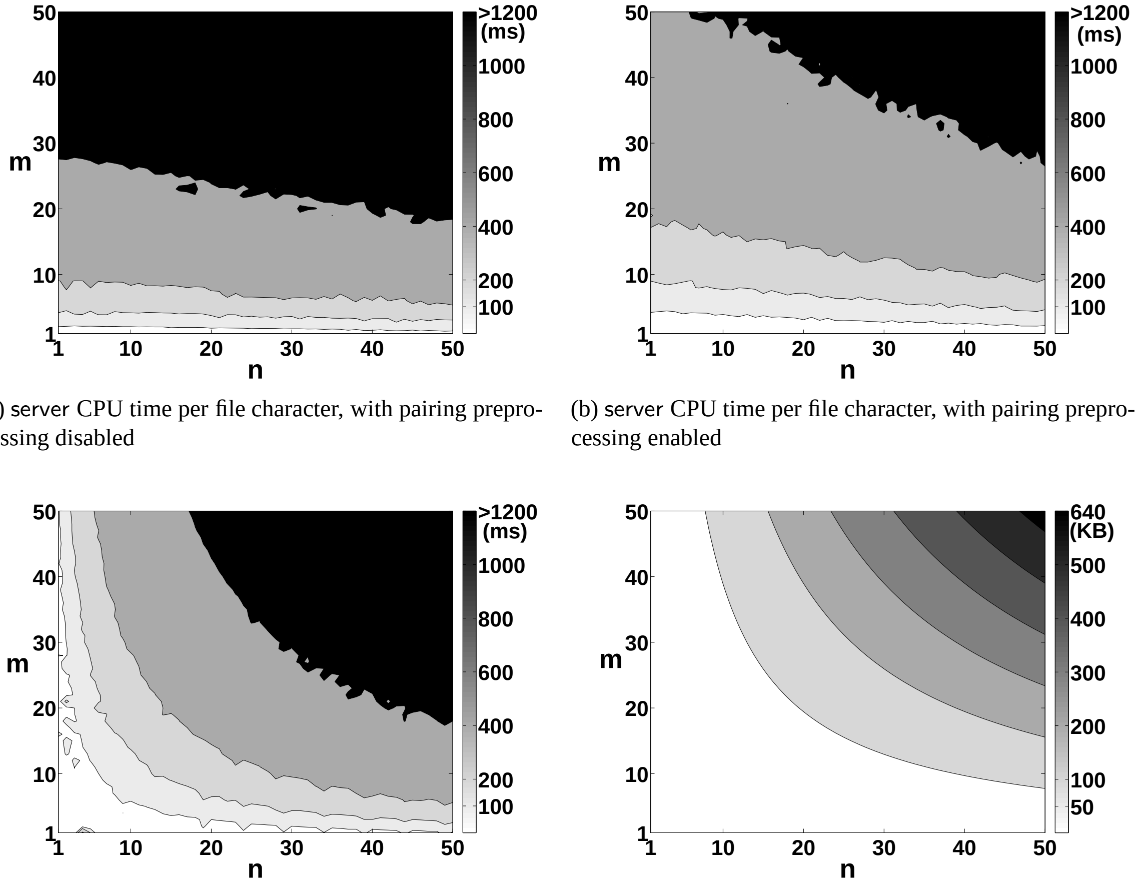


(a) 1 thread per worker (b) 4 threads per worker (c) 16 threads per worker

Figure 5.9: Time spent per file character in milliseconds, with pairing preprocessing enabled

Since Fig. 5.8 clearly shows the impact of the pairing operations on the overall performance of the protocol, we went on to evaluate how much improvement pairing preprocessing can provide. In these experiments, we applied pairing processing on the file ciphertexts before conducting the same experiments as described above. The results are shown in Fig. 5.9. As expected, the overall protocol performance improves significantly in each of the multi-threading cases, with darker bands reduced dramatically in size. More importantly, the protocol performance now scales much better with the increase of m because of the significantly reduced cost of pairing operations on the server side.

In order to better understand the relative computational burden imposed on the server and proxy by the protocol, we also conducted experiments measuring the average CPU time spent processing one character for the server and proxy processes for each combination of n and m. To perform these tests, we instantiated one server worker and one proxy worker, each with a single thread. The CPU time takes into account the amount of time spent in both user and kernel modes. The results



(a) server CPU time per file character, with pairing preprocessing disabled

(b) server CPU time per file character, with pairing preprocessing enabled

(c) proxy CPU time per file character

(d) Network bandwidth per file character

Figure 5.10: CPU time and network bandwidth measurements

for the server and proxy are plotted in Fig. 5.10a and Fig. 5.10c respectively. For the server side, it generally takes below or around 100ms for one round of computation when m is less than 10. The proxy side enjoys a slightly lighter computational cost and spends around or below 100ms even for m as high as 50 with n less than 15. The results reveal that the server side takes more hit when m increases due to the need to perform the pairing operations, while the proxy achieves a more balanced degradation with the increasing of n or m.

We also conducted the same experiments when pairing preprocessing is used on the file ciphertexts in advance. Since it does not affect the proxy side processing time, we only show the server side CPU time in Fig. 5.10b. Compared with Fig. 5.10a, the CPU time spent is reduced significantly and the performance scales better as m increases.

Since the protocol is interactive, we also measured the aggregate network bandwidth consumption between the server and proxy in one round of protocol execution. As shown in Fig. 5.10d, the bandwidth usage ranges from about 15KB per round (i.e., per file character) for moderate n and m to as much as 640KB per round when n and m are 50.

5.4.3 Case Study: Regular Expression Search on Encrypted Emails

To further provide insight into the expected performance when using our protocol in real-world application, we conducted a case study for performing regular expression search on public-key encrypted emails. We envision an email system in which the sender encrypts the email body using a traditional hybrid encryption scheme, in which the email body is encrypted using a symmetric encryption key which itself is encrypted by the receiver's public key. To enable search operations, however, the sender also attaches an encrypted searchable "header" to the encrypted email body that consists of all the information from the email that allows searching. We now detail the design of this header.

5.4.3.1 Header Information

Our current design allows searching on selected header fields of the email that are most commonly searched: (1) date; (2) sender email address; (3) sender name; and (4) subject line. The character-by-character encryption of the four headers are attached to the encrypted email body to enable searches. Since characters in each header are usually drawn from different distributions, we define the *dictionary* of a header as the set containing all possible characters and field-specific words that can be used in that header. Each header-field text is encoded using the dictionary before encryption, including sanitizing any characters not present in the dictionary (e.g., converting uppercase letters to lowercase, if only lowercase are included in the dictionary). We stress that this sanitization is only applied to the encrypted header that facilitates the search operations. The original field value in the email body is left intact.

The benefit of defining different dictionaries for different header fields is that adding field-specific words provides an opportunity for compressing the header fields, which is reminiscent of dictionary-based compression schemes. In addition, we envision dictionaries to be receiver-specific, e.g., distributed within the public-key certificate for the receiver. Below we describe how each dictionary is defined for each header in our evaluation.

Date: Date is converted into YYMMDD, where year, month and day each consists of two digits of numerical values. For years, we expect to store emails dated from 1990 to 2050. So we included “90” to “50”, as words, in the dictionary to encode the year. Similarly for months and days, we also added “01” to “31” into the dictionary. So the dictionary is defined as $\{00, 01, 02, \ldots, 49, 50\} \cup \{90, 91, \ldots, 99\}$.

Sender address: The sender email address is represented in the usual format, e.g., “alice@abc.com”, where the dictionary consists of “a” through “z”, “0” through “9”, “.”, “@”, “_” and “−”. We also added into the dictionary several common email service names like “gmail”, domain names like “com”, and “enron” because the email dataset used in our evaluation (see Section 5.4.3.3) was from Enron. (Again, dictionaries can be receiver-specific). In total, the dictionary for this field is {a, . . ., z } ∪ {0, . . ., 9 } ∪ {@, ., _, − } ∪ {gmail, yahoo, aol, hotmail, enron, com, edu, net, org }.

Sender name: The sender name field represents the sender’s name with first name followed by the last name, separated by a space. The name dictionary consists of “a” through “z” and the space character.

Subject line: The subject line is allowed to include arbitrary characters that can be typed from a keyboard. However, in practice, users rarely create search queries including special characters [40]. So in our design we restrict the dictionary to include “a” through “z”, “0” through “9”, and selected special characters including “@”, “!”, “%”, “.” and the space character.

5.4.3.2 Encoding

To enable DFA evaluations, we need to define the input alphabet Σ that drives the DFA state transitions. The simplest way is to define it as the union of all the dictionaries defined for all header fields, which would result in an m well above 50. However, the experiment results in Section 5.4.2 suggested that the protocol performance is very sensitive to a large m. So we make the DFA alphabet Σ and its size m a user defined parameter and designed a method to encode each word in the dictionary into a representation using the input symbols in Σ. Since the exact representation of the input symbols in Σ is not important, for simplicity we use numerical values to represent each symbol. For example, a size m alphabet will consists of $\Sigma = \{0, 1, \ldots, m-1\}$. Then, each word in a dictionary is represented using a distinct sequence of symbols from Σ. Each of the header fields is first encoded using this method and then encrypted symbol by symbol. The regular expression query is encoded in the same way before converting it into a DFA.

5.4.3.3 Evaluations

In order to shed light on the expected performance when using our protocol to perform search operations in real-world email systems, we implemented a prototype search system and evaluated its performance based on the Enron email dataset [1], which is a publicly available real-world email corpus that contains roughly 0.6 million messages from about 150 then-employees of Enron. For privacy reasons, the attachments on the original emails were excluded from the data set. Since our implementation does not support searches on email bodies or attachments, this has no effect on our evaluation except to exaggerate the average multiplicative increase in email size resulting from the encryption needed to support our search (in comparison to email encryption using standard tools). We randomly sampled 1000 emails from the inboxes of all the users in the dataset and performed evaluations using selected representative search queries. In the experiments, we fixed a DFA alphabet of size $m = 4$.

Motivated by the email search features found in ThunderBird [4], we selected four different queries to evaluate the protocol efficiency. For the date field, we selected a range query to search for all emails with date stamps between 2001/09/10 and 2002/04/20. The corresponding regular expression is

$$\begin{gathered}(\texttt{0109}(\texttt{10}|\texttt{11}|\ldots|\texttt{31}))\,|\,(\texttt{01}(\texttt{10}|\texttt{11}|\texttt{12})(\texttt{01}|\ldots|\texttt{31}))\,|\\ (\texttt{02}|(\texttt{01}|\texttt{02}|\texttt{03})(\texttt{01}|\ldots|\texttt{31}))\,|\,(\texttt{0204}(\texttt{01}|\texttt{02}|\ldots|\texttt{20}))\end{gathered}$$

which results in a DFA of 23 states using our conversion engine. For the sender address field, we selected a query to search for emails with sender address ending with the string “enron.com”. The resulting regular expression is $*\texttt{enron.com}$ where $*$ denotes zero or more occurrences of dictionary words, which converts into a DFA with 9 states. For the name field, we selected the query to search for sender name containing the word “John”, which translates into the regular expression $*\texttt{John}*$

,with a corresponding DFA containing 17 states. Lastly for the subject line field, we chose to search for emails with subject lines containing the word “meet” followed by “Jan” followed by a space and two arbitrary characters. This translates into a regular expression of $*$meet Jan $??*$ where $?$ denotes exactly one occurrence of a dictionary word, which results in a DFA of 36 states.

We encrypted the bodies of 1000 randomly selected emails using GnuPG [3], which results in an average size of 1.5KB per email. We wrote our own tool to generate the encrypted searchable headers, which take up about 185KB per email. To understand the performance impact when using the two parallelization techniques described in Section 5.4.1, we report performance numbers for various combinations of the number of workers and the number of threads that each worker spawns. The average time spent processing each email is shown in Table 5.1, which was calculated by dividing the total time to finish processing all 1000 emails by 1000. In order to demonstrate the performance improvement when pairing preprocessing is applied on email ciphertexts, we also precomputed pairing-preprocessing information of the email ciphertexts and stored them on disk. The numbers are shown in the same table inside braces. The performance gain is very compelling, as it offers an approximately 30% improvement over the version without preprocessing. However, the downside is that it needs significantly more storage space to store the pairing-preprocessing information.

	Date query $(2001/09/10 - 2002/04/20)$			Sender address query $(.*$ enron.com$)$		
Threads Workers	1	2	4	1	2	4
1	3.55 (2.38)	2.03 (1.39)	1.23 (0.99)	13.09 (6.95)	8.00 (4.69)	6.75 (5.26)
2	1.68 (1.17)	0.96 (0.71)	0.63 (0.50)	6.68 (3.58)	3.97 (2.72)	3.45 (2.68)
4	0.84 (0.57)	0.49 (0.36)	0.42 (0.28)	3.31 (1.81)	2.01 (1.51)	2.01 (1.44)
8	0.43 (0.30)	0.30 (0.20)	0.29 (0.19)	1.70 (0.98)	1.40 (0.93)	1.34 (**0.89**)
16	0.27 (0.18)	0.26 (0.18)	**0.25 (0.17)**	1.27 (0.94)	**1.25** (0.94)	1.26 (0.95)

	Sender name query (“$.*$ John$.*$”)			Subject line query $(.*$ meet Jan $??.*)$		
Threads Workers	1	2	4	1	2	4
1	12.34 (7.93)	7.55 (4.84)	4.84 (3.50)	35.97 (27.17)	20.17 (15.02)	11.38 (9.52)
2	6.47 (3.99)	3.56 (2.40)	2.40 (1.96)	17.68 (12.85)	9.49 (7.41)	5.75 (4.80)
4	3.13 (1.98)	1.82 (1.28)	1.56 (1.16)	8.59 (6.30)	4.81 (3.71)	4.20 (2.97)
8	1.62 (1.05)	1.25 (0.80)	1.15 (0.77)	4.41 (3.21)	2.81 (2.10)	3.04 (1.91)
16	1.09 (**0.79**)	**1.06** (0.81)	1.06 (0.81)	2.49 (1.93)	2.55 (**1.75**)	**2.39** (1.77)

Table 5.1: Average time spent per email in seconds (numbers in braces are when pairing preprocessing is applied on email ciphertexts in advance)

The experiment results also demonstrate the benefit of concurrently processing multiple emails by instantiating multiple workers. In most cases, doubling the number of workers results in a decrease of the timing results by a factor of two. Meanwhile, spawning multiple threads for each worker has similar effect, although to a lesser extent. This can be seen by reading the entries horizontally, where the timing results are typically reduced by about 40% as the number of threads per worker doubles. The date query records the fastest time to finish, averaging only a quarter of a second to process one email and 0.17 second when pairing preprocessing is used. This is due to the fact that the date field is very short for all emails. The sender name query came at second with 1.06 second per email and 0.76 second with pairing preprocessing. This is followed by the sender address query, which achieves a 1.25 second and 0.89 second respectively. The subject line query is the slowest, mainly due to the fact that subject lines in the email corpus are usually much longer in length than the other fields.

CHAPTER 6

Conclusion

With the growth of cloud storage due to the cost savings it offers, it is imperative that we develop efficient techniques for enabling the same sorts of third-party access to cloud-resident files that is commonplace today for privately stored files — e.g., malware scans or searches by authorized partners. The fact that cloud-resident files are generally at greater risk of exposure, however, mandates their encryption, hindering these sorts of third-party access.

In this dissertation, we have developed a family of protocols for enabling regular expression evaluation on encrypted files by third parties authorized by the file owner. Our protocols developed in Chapter 3 provably protect the privacy of the DFA from an arbitrarily malicious server holding the ciphertext file, as well as the privacy of the file from the server and from an honest-but-curious client performing the DFA evaluation. We further developed a strengthened protocol in Chapter 4 that allows the client to detect any malicious behavior of the server. In addition, the client is also able to tell if the server used the real encrypted file from the data owner as the input of the protocol. In that sense, the evaluation result is authenticated to the client. The design of the protocol deviates from the traditional paradigm of using zero-knowledge proof techniques to enforce the correct behavior of the participants. Instead, we leveraged novel algebraic techniques that make the evaluation result verifiable so that any misbehavior by the server would be easily detected by the client, without resorting to zero-knowledge techniques. This results in the first published protocol that we are aware to perform secure DFA evaluation on both encrypted and authenticated data.

Motivated by the growing trend of outsourcing *computation* from resource-constrained devices (e.g., smartphones) to more powerful proxy servers to assist in its computation, in Chapter 5 we presented a protocol that allows a user to outsource her DFA evaluations to a proxy server by sending an "encrypted" DFA to the proxy, which then interacts with the server hosting the encrypted file to obtain an encrypted evaluation result for the user. This protocol differs from the one developed in Chapter 3 in that it additionally protects the privacy of the DFA from an honest-but-curious proxy (or client), thus allowing a user to outsource the computation without divulging the query to the proxy. We then went on to develop an optimized protocol that offers an order-of-magnitude of improvement on the protocol efficiency with similar security properties. We detailed a series of optimization techniques we employed and proved the protocol protects the privacy of the DFA and file contents from arbitrarily malicious server and honest-but-curious proxy adversaries. To provide insight on the performance of our protocol in a real world application, we implemented our optimized protocol and a prototype for an encrypted email system. We evaluated its performance using a real-world email datasets and demonstrated the practicality of the protocol.

BIBLIOGRAPHY

[1] Enron email dataset. http://www.cs.cmu.edu/~enron/.

[2] GenBank. http://www.ncbi.nlm.nih.gov/genbank/.

[3] The gnu privacy guard. http://www.gnupg.org/.

[4] Mozilla thunderbird. https://www.mozilla.org/en-US/thunderbird/.

[5] United Kingdom National DNA Database. http://www.npia.police.uk/en/8934.htm.

[6] M. Abdalla, M. Bellare, D. Catalano, E. Kiltz, T. Kohno, T. Lange, J. Malone-Lee, G. Neven, P. Paillier, and H. Shi. Searchable encryption revisited: Consistency properties, relation to anonymous IBE, and extensions. *Journal of Cryptology*, 21(3):350–391, July 2008.

[7] B. Applebaum, Y. Ishai, and E. Kushilevitz. How to garble arithmetic circuits. In *52nd IEEE Symposium on Foundations of Computer Science*, pages 120–129, 2011.

[8] G. Ateniese, E. De Cristofaro, and G. Tsudik. (If) size matters: Size-hiding private set intersection. In *14th International Conference on Practice and Theory of Public Key Cryptography*, pages 156–173, 2011.

[9] J. Baek, R. Safavi-Naini, and W. Susilo. Public key encryption with keyword search revisited. In *Computational Science and Its Applications – ICCSA 2008*, volume 5072 of *Lecture Notes in Computer Science*, pages 1249–1259, 2008.

[10] M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway. Relations among notions of security for public-key encryption schemes. In *Advances in Cryptology – Crypto ’98*, pages 26–45, August 1998.

[11] A. Ben-David, N. Nisan, and B. Pinkas. FairplayMP: A system for secure multi-party computation. In *15th ACM Conference on Computer and Communications Security*, pages 257–266, 2008.

[12] D. Betel and C. Hogue. Kangaroo – a pattern-matching program for biological sequences. *BMC Bioinformatics*, 3, 2002.

[13] M. Blanton and M. Aliasgari. Secure outsourcing of DNA searching via finite automata. In *Data and Applications Security and Privacy XXIV*, pages 49–64, June 2010.

[14] D. Boneh. The decision Diffie-Hellman problem. In *3rd International Symposium on Number Theory*, pages 48–63, June 1998.

[15] D. Boneh. Personal communication, July 2011.

[16] D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano. Public key encryption with keyword search. In *Advances in Cryptology – Eurocrypt ’04*, volume 3027 of *Lecture Notes in Computer Science*, pages 506–522, 2004.

[17] D. Boneh, E.-J. Goh, and K. Nissim. Evaluating 2-DNF formulas on ciphertexts. In *2nd Theory of Cryptography Conference*, pages 325–342, 2005.

[18] D. Boneh, B. Lynn, and H. Shacham. Short signatures from the weil pairing. In *Advances in Cryptology – Asiacrypt '01*, pages 514–532, 2001.

[19] D. Boneh and B. Waters. Conjunctive, subset, and range queries on encrypted data. In *4th Theory of Cryptography Conference*, pages 535–554, February 2007.

[20] J. Camenisch and G. M. Zaverucha. Private intersection of certified sets. In *13th International Conference on Financial Cryptography and Data Security*, pages 108–127, 2009.

[21] A. De Caro. Java pairing based cryptography library. `http://gas.dia.unisa.it/projects/jpbc/`.

[22] Y.-C. Chang and M. Mitzenmacher. Privacy preserving keyword searches on remote encrypted data. In *Applied Cryptography and Network Security, 3rd International Conference*, pages 442–455, 2005.

[23] K. Chen, R. Kavuluru, and S. Guo. RASP: Efficient multidimensional range query on attack-resilient encrypted databases. In *1st ACM Conference on Data and Application Security and Privacy*, February 2011.

[24] S. G. Choi, A. Elbaz, A. Juels, T. Malkin, and M. Yung. Two-party computing with encrypted data. In *Advances in Crypotology – Asiacrypt 2007*, pages 298–314, 2007.

[25] V. Ciriani, S. De Capitani Di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. Combining fragmentation and encryption to protect privacy in data storage. *ACM Transactions on Information and System Security*, 13(3), July 2010.

[26] E. De Cristofaro, J. Kim, and G. Tsudik. Linear-complexity private set intersection protocols secure in malicious model. In *Advances in Cryptology – Asiacrypt '10*, pages 213–231, 2010.

[27] E. De Cristofaro, J. Kim, and G. Tsudik. Practical private set intersection protocols with linear complexity. In *14th International Conference on Financial Cryptography and Data Security*, pages 143–159, 2010.

[28] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky. Searchable symmetric encryption: Improved definitions and efficient constructions. In *13th ACM Conference on Computer and Communications Security*, pages 79–88, 2006.

[29] I. Damgård and M. Jurik. A generalisation, a simplification and some applications of Paillier's probabilistic public-key system. In *Public Key Cryptography, 4th International Workshop on Practice and Theory in Public Key Cryptosystems*, pages 119–136, February 2001.

[30] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.

[31] K. B. Frikken. Practical private DNA string searching and matching through efficient oblivious automata evaluation. In *Data and Applications Security XXIII*, pages 81–94, July 2009.

[32] R. Gennaro, C. Hazay, and J. S. Sorensen. Text search protocols with simulation based security. In *13th International Conference on Practice and Theory in Public Key Cryptography*, pages 332–350, 2010.

[33] C. Gentry. Fully homomorphic encryption using ideal lattices. In *41st ACM Symposium on Theory of Computing*, pages 169–178, 2009.

[34] C. Gentry, S. Halevi, and V. Vaikuntanathan. A simple BGN-type cryptosystem from LWE. In *Advances in Cryptology – Eurocrypt '10*, pages 506–522, May 2010.

[35] C. Gentry and Z. Ramzan. Single-database private information retrieval with constant communication rate. In *Automata, Languages and Programming, 32nd International Colloquium*, pages 803–815, July 2005.

[36] E.-J. Goh. Secure indexes. Cryptology ePrint Archive, Report 2003/216, 2003. `http://eprint.iacr.org/`.

[37] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *19th ACM Symposium on Theory of Computing*, pages 218–229, 1987.

[38] J. Goyvaerts and S. Levithan. *Regular Expressions Cookbook*. O'Reilly Media, Inc., second edition, 2012.

[39] H. Hacigümüş, B. Iyer, C. Li, and S. Mehrotra. Executing SQL over encrypted data in the database-service-provider model. In *2002 ACM SIGMOD International Conference on Management of Data*, pages 216–227, June 2002.

[40] M. Harvey and D. Elsweiler. Exploring query patterns in email search. In *34th European Conference on Advances in Information Retrieval*, pages 25–36, 2012.

[41] C. Hazay and Y. Lindell. Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries. *Journal of Cryptology*, 23(3):422–456, 2010.

[42] C. Hazay and T. Toft. Computationally secure pattern matching in the presence of malicious adversaries. In *Advances in Cryptology – Asiacrypt 2010*, pages 195–212, 2010.

[43] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.

[44] B. Hore, S. Mehrotra, and G. Tsudik. A privacy-preserving index for range queries. In *30th International Conference on Very Large Data Bases*, pages 720–731, August 2004.

[45] S. Jha, L. Kruger, and V. Shmatikov. Towards practical privacy for genomic computation. In *29th IEEE Symposium on Security and Privacy*, pages 216–230, 2008.

[46] S. Kamara, C. Papamanthou, and T. Roeder. Cs2: A searchable cryptographic cloud storage system. Technical Report MSR-TR-2011-58, Microsoft, 2011.

[47] S. Kamara, C. Papamanthou, and T. Roeder. Dynamic searchable symmetric encryption. In *19th ACM Conference on Computer and Communications Security*, pages 965–976, October 2012.

[48] J. Katz and L. Malka. Secure text processing with applications to private DNA matching. In *17th ACM Conference on Computer and Communications Security*, pages 485–492, 2010.

[49] J. Katz, A. Sahai, and B. Waters. Predicate encryption supporting disjunctions, polynomial equations, and inner products. In *Advances in Cryptology – Eurocrypt '08*, pages 146–162, April 2008.

[50] T. Kojm. *ClamAV*. `http://www.clamav.net`.

[51] K. Lauter, M. Naehrig, and V. Vaikuntanathan. Can homomorphic encryption be practical? In *3rd ACM Workshop on Cloud Computing Security*, October 2011.

[52] A. K. Lenstra and E. R. Verheul. Selecting cryptographic key sizes. *Journal of Cryptology*, 14(4):255–293, 2001.

[53] B. Lynn. The pairing based cryptography library. `http://crypto.stanford.edu/pbc/`.

[54] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella. Fairplay – a secure two-party computation system. In *13th USENIX Security Symposium*, pages 287–302, August 2004.

[55] D. Mazières and D. Shasha. Building secure file systems out of Byzantine storage. In *21st Symposium on Principles of Distributed Computing*, pages 108–117, July 2002.

[56] A. J. Menezes, P. C. Van Oorschot, and S. A Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997.

[57] A. Moller. dk.brics.automaton. `http://www.brics.dk/automaton/`.

[58] D. Needle. Cloud storage poised to save enterprises money: Report. `http://itmanagement.earthweb.com/datbus/article.php/3896116/Cloud-Storage-Poised-to-Save-Enterprises-Money-Report.htm`, July 30, 2010.

[59] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology – Eurocrypt '99*, pages 223–238, May 1999.

[60] B. Pinkas, T. Schneider, N. Smart, and S. Williams. Secure two-party computation is practical. In *Advances in Cryptology – Asiacrypt '09*, pages 250–267, December 2009.

[61] R. A. Popa, C. M. S. Redfield, N. Zeldovich, and H. Balakrishnan. Cryptdb: protecting confidentiality with encrypted query processing. In *23rd ACM Symposium on Operating Systems Principles*, pages 85–100, 2011.

[62] H. S. Rhee, J. H. Park, W. Susilo, and D. H. Lee. Improved searchable public key encryption with designated tester. In *4th ACM Conference on Information, Computer, and Communications Security*, pages 376–379, March 2009.

[63] R. Rivest, L. Adleman, and M. Dertouzos. On data banks and privacy homomorphisms. *Foundations of Secure Computation*, pages 169–177, 1978.

[64] M. Roesch. Snort – lightweight intrusion detection for networks. In *13th USENIX Conference on System Administration*, pages 229–238, 1999.

[65] E. Shi, J. Bethencourt, T-H. Chan, D. Song, and A. Perrig. Multi-dimensional range query over encrypted data. In *28th IEEE Symposium on Security and Privacy*, pages 350–364, May 2007.

[66] N. P. Smart and F. Vercauteren. Fully homomorphic encryption with relatively small key and ciphertext sizes. In *13th International Conference on Practice and Theory in Public Key Cryptography*, pages 420–443, May 2010.

[67] D. X. Song, D. Wagner, and A. Perrig. Practical techniques for searches on encrypted data. In *2000 IEEE Symposium on Security and Privacy*, 2000.

[68] E. Stefanov, E. Shi, and D. Song. Policy-enhanced private set intersection: sharing information while enforcing privacy policies. In *15th International Conference on Practice and Theory in Public Key Cryptography*, pages 413–430, 2012.

[69] D. Stehlé and R. Steinfeld. Faster fully homomorphic encryption. In *Advances in Cryptology – Asiacrypt '10*, pages 377–394, December 2010.

[70] J. R. Troncoso-Pastoriza, S. Katzenbeisser, and M. Celik. Privacy preserving error resilient DNA searching through oblivious automata. In *14th ACM Conference on Computer and Communications Security*, pages 519–528, 2007.

[71] M. van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan. Fully homomorphic encryption over the integers. In *Advances in Cryptology – EUROCRYPT 2010*, pages 24–43, 2010.

[72] B. Waters, D. Balfanz, G. Durfee, and D. K. Smetters. Building an encrypted and searchable audit log. In *11th Annual Network and Distributed System Security Symposium*, 2004.

[73] L. Wei and M. K. Reiter. Third-party private DFA evaluation on encrypted files in the cloud. In *Computer Security – ESORICS 2012: 17th European Symposium on Research in Computer Security*, 2012.

[74] A. C. Yao. Protocols for secure computations. In *23rd IEEE Symposium on Foundations of Computer Science*, pages 160–164, 1982.

[75] J. Zhuge, T. Holz, C. Song, J. Guo, X. Han, and W. Zou. Studying malicious websites and the underground economy on the Chinese web. In *Workshop on the Economics of Information Security*, June 2008.