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

Lei Wei and Michael K. Reiter

Department of Computer Science, University of North Carolina at Chapel Hill
{lwei,reiter}@cs.unc.edu

Abstract. Motivated by the need to outsource file storage to untrusted clouds while still permitting limited use of that data by third parties, we present practical protocols by which a client (the third-party) can evaluate a deterministic finite automaton (DFA) on an encrypted file stored at a server (the cloud), 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 further present simple techniques to detect client or server misbehavior.

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. 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 partially trusted third parties, however. 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. Moreover, with some exceptions (see §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., [37]). 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 paper

S. Foresti, M. Yung, and F. Martinelli (Eds.): ESORICS 2012, LNCS 7459, pp. 523–540, 2012.

we introduce 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. The type of searches that our protocols enable is motivated by the scenarios above, which in many cases involve pattern matching a file against one or more regular expressions. Multi-pattern string matching is especially common in analysis of content for malware (e.g., [31,24]) and also is commonplace in searches on genome data, for example. In fact, there are now a number of available genome databases (e.g., [1,2]) and accompanying tools for multi-pattern matching against them (e.g., [6]). With the goal of improving privacy in such applications, we develop 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 computation including the specific case of private DFA evaluation on a private file (see §2), 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 to 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 §4, we present a protocol that leverages

additively homomorphic encryption (e.g., [28]) and transmits $O(\kappa\ell nm)$ bits, for κ a security parameter, to evaluate a DFA of n states. In §5, we leverage additively homomorphic encryption that also supports *one* homomorphic multiplication of ciphertexts (e.g., [8]) to construct a protocol that transmits only $O(\kappa\ell(n+m))$ bits. 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. Before describing our protocols, we discuss related work in §2 and clarify our goals in §3.

2 Related Work

The functionality offered by our protocols could be implemented with general "computing on encrypted data" [30] or two-party secure computation [36,18]. These techniques tend to yield less efficient protocols than one designed for a specific purpose, and our case will be no exception. The former achieves computations non-interactively using fully homomorphic encryption, for which existing implementations [14,11,32,33] are much more costly than the techniques we use [15]. 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 [26,5,29], this limitation renders it much more communication-intensive for the problem we consider.

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. Troncoso-Pastoriza et al. [34] presented the first such protocol, which they proved secure in the honest-but-curious setting. Frikken [12] presented a protocol for the same setting that improved on the round complexity and computational costs. Gennaro et al. [13] developed a protocol that they proved secure against arbitrary adversaries, and Mohassel et al. [27] presented a protocol for arbitrary adversaries that significantly reduces the number of asymmetric operations. Our work differs from these in that in our protocols, the file is available to the parties only in ciphertext form. In this respect, the protocol of Blanton and Aliasgari [7] is relevant; they proposed a protocol for an "outsourcing" model, in which the DFA owner and file owner secret-share the DFA and file, respectively, between two other hosts, who then interactively evaluate the DFA on the file without reconstructing either one. While our protocol utilizes secret sharing, as well — in our case, of the file owner's file-decryption key — it shares much less data and does not share the client's DFA (or thus require two parties between which to share it) at all.

By two-party sharing the file-decryption key and using this to compute on encrypted data, our protocols are related to Choi et al.'s [9]. 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.

Two-party pattern-matching and search problems other than DFA evaluation have also been studied, e.g., by Jha et al. [22], Hazay and Lindell [19], Katz and Malka [23], and Hazay and Toft [20]. Again, these works input the plaintext file to one party and so do not directly apply to our setting. Of particular note, though, is a protocol due to Ishai and Paskin [21] to evaluate a branching program (which can be used to encode a DFA) on encrypted data. Translated to our context, their scheme enables a client holding a branching program P and provided the ciphertext c_1 for plaintext data $\sigma \in \Sigma^\ell$ to compute a ciphertext c_2 of $P(\sigma)$. Conceivably if the data owner shared the decryption key between the client and the server who provided c_1 (as in our protocol), the client could then recover $P(\sigma)$ by jointly decrypting c_2 with the server, without involvement from the data owner. However, when this protocol is applied to DFAs, c_1 could be of length quadratic in ℓ and, because c_2 is encrypted in a nested fashion, its joint decryption would seem to require ℓ rounds of interaction, each round with messages of length $O(\ell)$.

Additional related work is discussed in our accompanying technical report [35].

3 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 set $F \subseteq Q$ of accepting states. 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

[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.

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.

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., [17]). Moreover, some of our motivating scenarios in §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.

4 A Secure DFA Evaluation Protocol

In this section we present a protocol that meets the goals described in §3. We give the construction in §4.1, and then we define and prove security against server and client adversaries in §4.2 and §4.3, respectively.

4.1 Construction

Let "$\leftarrow$" denote assignment and "$s \xleftarrow{\$} S$" 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 [28] with modifications by Damgård and Jurik [10]; 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

$$f(x, y) = \sum_{\sigma \in \Sigma}^{\mathbb{R}} (f_\sigma(x) \cdot_{\mathbb{R}} \Lambda_\sigma(y)) \quad \text{where} \quad \Lambda_\sigma(y) = \prod_{\substack{\sigma' \in \Sigma \\ \sigma' \neq \sigma}}^{\mathbb{R}} \frac{y -_{\mathbb{R}} \sigma'}{\sigma -_{\mathbb{R}} \sigma'} \tag{1}$$

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 (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 (1), we can calculate coefficients $\langle \lambda_{\sigma j} \rangle_{j \in [m]}$ so that $\Lambda_\sigma(y) = \sum_{j=0}^{m-1}{}^{\mathbb{R}} \lambda_{\sigma j} \cdot_{\mathbb{R}} y^j$. 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 $f_\sigma(x) = \sum_{i=0}^{n-1}{}^{\mathbb{R}} a_{\sigma i} \cdot_{\mathbb{R}} x^i$. 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. 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 σ_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 (§4.3) 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 $\pi_1 : Q \to \mathbb{R}$ it chose in the $(k-1)$-th loop at line c109 (or, if $k = 0$, then in line c103), where $\mathsf{Injs}(Q \to \mathbb{R})$ 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).

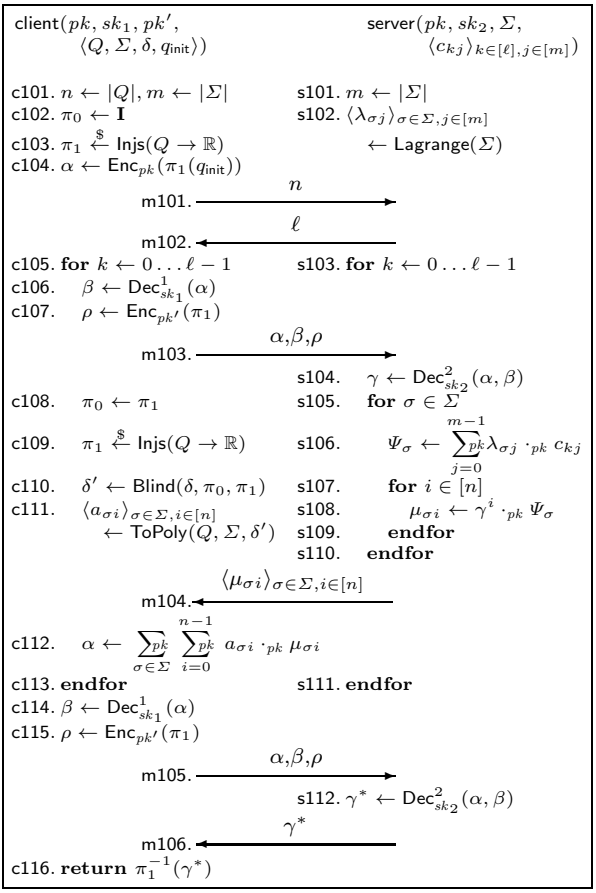


Fig. 1. Protocol $\Pi_1(\mathcal{E})$, described in §4

Meanwhile, the client selects a new random injection $\pi_1 \xleftarrow{\$} \mathsf{Injs}(Q \to \mathbb{R})$ (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 $\langle \mu_{\sigma i} \rangle_{\sigma\in\Sigma, i\in[n]}$ 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).

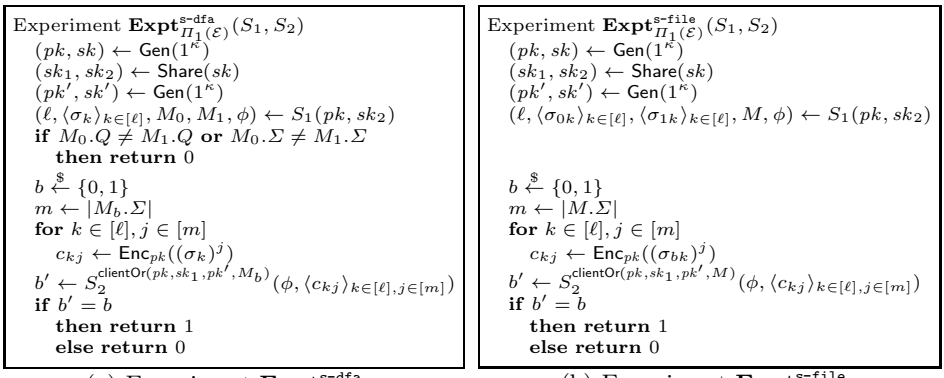
Protocol $\Pi_1(\mathcal{E})$ can be modified to return only a binary indication of whether the DFA's final state is an accepting one, if the DFA specifies a set F of accepting states. Specifically, the client can construct a polynomial $\hat{f}(x)$ that evaluates to 1 on states in F and 0 on other states. Then, rather than interacting with the server to decrypt the final state, the client can interact with the server once to evaluate $\hat{f}(x)$ on the (unknown) final state and again to decrypt this result. We omit details here due to space limitations.

For brevity, Fig. 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 [14,11] 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.

4.2 Security against Server Attacks

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) Experiment $\mathbf{Expt}^{\mathtt{s\text{-}dfa}}_{\Pi_1(\mathcal{E})}$ (b) Experiment $\mathbf{Expt}^{\mathtt{s\text{-}file}}_{\Pi_1(\mathcal{E})}$

Fig. 2. Experiments for proving security of $\Pi_1(\mathcal{E})$ against server adversaries

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 §6 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 $\mathbf{Expt}^{\mathtt{s\text{-}dfa}}_{\Pi_1(\mathcal{E})}$ in Fig. 2(a). 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.

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

Eventually S_2 outputs a bit b', and $\mathbf{Expt}^{\mathtt{s\text{-}dfa}}_{\Pi_1(\mathcal{E})}(S) = 1$ only if $b' = b$. We say the *advantage* of S is $\mathbf{Adv}^{\mathtt{s\text{-}dfa}}_{\Pi_1(\mathcal{E})}(S) = 2 \cdot \mathbb{P}\left(\mathbf{Expt}^{\mathtt{s\text{-}dfa}}_{\Pi_1(\mathcal{E})}(S) = 1\right) - 1$ and define $\mathbf{Adv}^{\mathtt{s\text{-}dfa}}_{\Pi_1(\mathcal{E})}(t, \ell, n, m) = \max_S \mathbf{Adv}^{\mathtt{s\text{-}dfa}}_{\Pi_1(\mathcal{E})}(S)$ 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 [4] security of the encryption scheme. IND-CPA security is defined using the experiment

in Fig. 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 $\mathbf{Expt}^{\mathtt{ind\text{-}cpa}}_{\mathcal{E}}(U)=1$ only if $\hat{b}'=\hat{b}$. The IND-CPA advantage of U is defined as $\mathbf{Adv}^{\mathtt{ind\text{-}cpa}}_{\mathcal{E}}(U) = 2\cdot\mathbb{P}\left(\mathbf{Expt}^{\mathtt{ind\text{-}cpa}}_{\mathcal{E}}(U)=1\right)-1$. Then, $\mathbf{Adv}^{\mathtt{ind\text{-}cpa}}_{\mathcal{E}}(t,w) = \max_U \mathbf{Adv}^{\mathtt{ind\text{-}cpa}}_{\mathcal{E}}(U)$ 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. The following theorem is proved in our accompanying technical report [35].

Theorem 1. $\mathbf{Adv}^{\mathtt{s\text{-}dfa}}_{\Pi_1(\mathcal{E})}(t,\ell,n,m) \le 2\mathbf{Adv}^{\mathtt{ind\text{-}cpa}}_{\mathcal{E}}(t',\ell+1)$ *for* $t' = t + t_{\mathsf{Gen}} + t_{\mathsf{Share}}$.

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 $\mathbf{Expt}^{\mathtt{s\text{-}file}}_{\Pi_1(\mathcal{E})}$ shown in Fig. 2(b). 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 $\mathbf{Expt}^{\mathtt{s\text{-}file}}_{\Pi_1(\mathcal{E})}(S)=1$ iff $b'=b$. We say the *advantage* of S is $\mathbf{Adv}^{\mathtt{s\text{-}file}}_{\Pi_1(\mathcal{E})}(S) = 2\cdot\mathbb{P}\left(\mathbf{Expt}^{\mathtt{s\text{-}file}}_{\Pi_1(\mathcal{E})}(S)=1\right)-1$ and then $\mathbf{Adv}^{\mathtt{s\text{-}file}}_{\Pi_1(\mathcal{E})}(t,\ell,n,m) = \max_S \mathbf{Adv}^{\mathtt{s\text{-}file}}_{\Pi_1(\mathcal{E})}(S)$ 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. The following theorem is proved in our technical report [35].

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

Fig. 3. $\mathbf{Expt}^{\mathtt{ind\text{-}cpa}}_{\mathcal{E}}(U)$

Theorem 2. $\mathbf{Adv}^{\mathtt{s\text{-}file}}_{\Pi_1(\mathsf{Pai})}(t,\ell,n,m) \le 2\mathbf{Adv}^{\mathtt{ind\text{-}cpa}}_{\mathsf{Pai}}(t',\ell{+}1) + \mathbf{Adv}^{\mathtt{ind\text{-}cpa}}_{\mathsf{Pai}}(t',\ell m)$ *for* $t' = t + t_{\mathsf{Gen}} + t_{\mathsf{Share}}$.

4.3 Security against Client Attacks

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. (We also introduce novel extensions to detect a misbehaving client in §6.) 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

$\mathbf{Expt}^{\mathsf{c\text{-}file}}_{\Pi_1(\mathcal{E})}$ (Fig. 4) that we use to define file security against client adversaries, the adversary $C = (C_1, C_2)$ succeeds (i.e., $\mathbf{Expt}^{\mathsf{c\text{-}file}}_{\Pi_1(\mathcal{E})}(C)$ 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 $\mathsf{serverOr}(pk, sk_2, M.\Sigma, \langle c_{kj} \rangle_{k \in [\ell], j \in [m]})$).

Experiment $\mathbf{Expt}^{\mathsf{c\text{-}file}}_{\Pi_1(\mathcal{E})}(C_1, C_2)$
- $(pk, sk) \leftarrow \mathsf{Gen}(1^\kappa)$
- $(sk_1, sk_2) \leftarrow \mathsf{Share}(sk)$
- $(pk', sk') \leftarrow \mathsf{Gen}(1^\kappa)$
- $(\ell, \langle \sigma_{0k} \rangle_{k \in [\ell]}, \langle \sigma_{1k} \rangle_{k \in [\ell]}, M, \phi) \leftarrow C_1(pk, sk_1, pk')$
- **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

Fig. 4. Experiment $\mathbf{Expt}^{\mathsf{c\text{-}file}}_{\Pi_1(\mathcal{E})}$

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.

As discussed in §1, 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 $\mathbf{hbcAdv}^{\mathsf{c\text{-}file}}_{\Pi_1(\mathcal{E})}(C) = 2 \cdot \mathbb{P}\left[\mathbf{Expt}^{\mathsf{c\text{-}file}}_{\Pi_1(\mathcal{E})}(C) = 1\right] - 1$ and $\mathbf{hbcAdv}^{\mathsf{c\text{-}file}}_{\Pi_1(\mathcal{E})}(t, \ell, n, m) = \max_C \mathbf{Adv}^{\mathsf{c\text{-}file}}_{\Pi_1(\mathcal{E})}(C)$ 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. Our technical report [35] proves:

Theorem 3. $\mathbf{hbcAdv}^{\mathsf{c\text{-}file}}_{\Pi_1(\mathsf{Pai})}(t, \ell, n, m) \leq \mathbf{Adv}^{\mathsf{ind\text{-}cpa}}_{\mathsf{Pai}}(t', \ell m(1+n))$ *for* $t' = t + t_{\mathsf{Gen}} + (\ell + 1) \cdot t_{\mathsf{Dec}}$.

We have found extending this result to fully malicious client adversaries to be difficult for two reasons. First, $\mathbf{Expt}^{\mathsf{c\text{-}file}}_{\Pi_1(\mathcal{E})}$ 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 — see the

proof of Theorem 3 in our technical report [35] — 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 §6 we introduce more novel (albeit still heuristic) approaches to detecting client (or server) misbehavior in our setting.

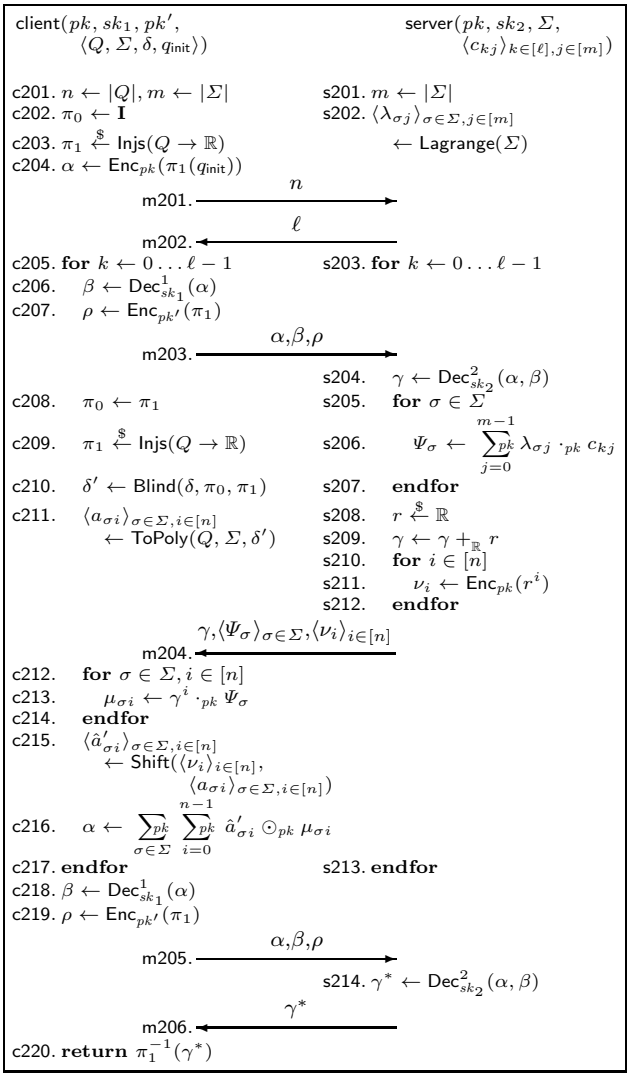


Fig. 5. Protocol $\Pi_2(\mathcal{E})$, described in §5

5 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. 5. 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 $f'(x, y)$.

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

$$
\begin{aligned}
f_\sigma(x -_{\mathbb{R}} r) &= \sum_{i=0}^{n-1}{}_{\mathbb{R}} a_{\sigma i} \cdot_{\mathbb{R}} (x -_{\mathbb{R}} r)^i = \sum_{i=0}^{n-1}{}_{\mathbb{R}} a_{\sigma i} \cdot_{\mathbb{R}} \sum_{i'=0}^{i}{}_{\mathbb{R}} \binom{i}{i'} \cdot_{\mathbb{R}} x^{i-i'} \cdot_{\mathbb{R}} (-_{\mathbb{R}} r)^{i'} \quad (2) \\
&= \sum_{i=0}^{n-1}{}_{\mathbb{R}} \left(\sum_{i'=0}^{n-1-i}{}_{\mathbb{R}} a_{\sigma(i+i')} \cdot_{\mathbb{R}} \binom{i+i'}{i'} \cdot_{\mathbb{R}} (-_{\mathbb{R}} r)^{i'} \right) \cdot_{\mathbb{R}} x^i
\end{aligned}
$$

where (2) follows from the binomial theorem. Therefore, setting

$$
a'_{\sigma i} \leftarrow \sum_{i'=0}^{n-1-i}{}_{\mathbb{R}} a_{\sigma(i+i')} \cdot_{\mathbb{R}} \binom{i+i'}{i'} \cdot_{\mathbb{R}} (-_{\mathbb{R}} 1)^{i'} \cdot_{\mathbb{R}} r^{i'} \quad (3)
$$

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 (3) 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:

$$
\hat{a}'_{\sigma i} \leftarrow \sum_{i'=0}^{n-1-i}{}_{pk} \left(a_{\sigma(i+i')} \cdot_{\mathbb{R}} \binom{i+i'}{i'} \cdot_{\mathbb{R}} (-_{\mathbb{R}} 1)^{i'} \right) \cdot_{pk} \nu_{i'} \quad (4)
$$

In our pseudocode, the calculations (4) 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 [8], and moreover, this cryptosystem also supports two-party decryption with a cost comparable to regular decryption [8]. 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 [16][2] and Lauter, Naehrig, and Vaikuntanathan [25]. 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 [36,3]. 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 algo-

[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 [3] for secure two-party decryption using additively shared keys would be of size $O(\kappa^6 \log^5(n+m))$ bits.

rithms for these cryptosystems could restore the efficiency of $\Pi_2(\mathcal{E})$, suggesting a useful open problem for the community.

6 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 §4.2 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 $\langle Q, \Sigma',$

δ', $q_{\mathsf{init}}\rangle$ 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 §4.3 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.

7 Conclusion

With the growth of cloud storage, it is imperative to 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. Encryption of cloud-resident files, however, hinders these sorts of third-party access.

In this paper, we have developed protocols for enabling DFA evaluation on encrypted files by third parties authorized by the file owner. Our protocols 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 (and even from an arbitrarily malicious client, heuristically). Our protocols employ additively homomorphic cryptosystems or small extensions thereof, for which practical implementations exist. The costs of our protocols in terms of storage, communication and computation suggest that they are practical for many domains, particularly ones where files consist of symbols from a limited alphabet, and are more practical than protocols that would result from applying general private two-party computation or fully homomorphic encryption to this problem.

[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}.

Acknowledgments. We are grateful to Dan Boneh for helpful clarifications about the Boneh-Goh-Nissim cryptosystem [8]. This research was supported in part by NSF award 0910483 and by a gift from NEC.

References

1. GenBank, `http://www.ncbi.nlm.nih.gov/genbank/`
2. United Kingdom National DNA Database, `http://www.npia.police.uk/en/8934.htm`
3. Applebaum, B., Ishai, Y., Kushilevitz, E.: How to garble arithmetic circuits. In: 52nd IEEE Symposium on Foundations of Computer Science (2011)
4. Bellare, M., Desai, A., Pointcheval, D., Rogaway, P.: Relations among Notions of Security for Public-Key Encryption Schemes. In: Krawczyk, H. (ed.) CRYPTO 1998. LNCS, vol. 1462, p. 26. Springer, Heidelberg (1998)
5. Ben-David, A., Nisan, N., Pinkas, B.: FairplayMP: A system for secure multi-party computation. In: 15th ACM Conference on Computer and Communications Security (2008)
6. Betel, D., Hogue, C.: Kangaroo – a pattern-matching program for biological sequences. BMC Bioinformatics 3 (2002)
7. Blanton, M., Aliasgari, M.: Secure Outsourcing of DNA Searching via Finite Automata. In: Foresti, S., Jajodia, S. (eds.) Data and Applications Security and Privacy XXIV. LNCS, vol. 6166, pp. 49–64. Springer, Heidelberg (2010)
8. Boneh, D., Goh, E.J., Nissim, K.: Evaluating 2-DNF formulas on ciphertexts. In: Kilian, J. (ed.) TCC 2005. LNCS, vol. 3378, pp. 325–341. Springer, Heidelberg (2005)
9. Choi, S.G., Elbaz, A., Juels, A., Malkin, T., Yung, M.: Two-Party Computing with Encrypted Data. In: Kurosawa, K. (ed.) ASIACRYPT 2007. LNCS, vol. 4833, pp. 298–314. Springer, Heidelberg (2007)
10. Damgård, I., Jurik, M.: A Generalisation, a Simplification and Some Applications of Paillier's Probabilistic Public-Key System. In: Kim, K.-c. (ed.) PKC 2001. LNCS, vol. 1992. Springer, Heidelberg (2001)
11. van Dijk, M., Gentry, C., Halevi, S., Vaikuntanathan, V.: Fully Homomorphic Encryption over the Integers. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 24–43. Springer, Heidelberg (2010)
12. Frikken, K.B.: Practical Private DNA String Searching and Matching through Efficient Oblivious Automata Evaluation. In: Gudes, E., Vaidya, J. (eds.) Data and Applications Security XXIII. LNCS, vol. 5645, pp. 81–94. Springer, Heidelberg (2009)
13. Gennaro, R., Hazay, C., Sorensen, J.S.: Text Search Protocols with Simulation Based Security. In: Nguyen, P.Q., Pointcheval, D. (eds.) PKC 2010. LNCS, vol. 6056, pp. 332–350. Springer, Heidelberg (2010)
14. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: 41st ACM Symposium on Theory of Computing (2009)
15. Gentry, C., Halevi, S.: Implementing Gentry's Fully-Homomorphic Encryption Scheme. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 129–148. Springer, Heidelberg (2011)
16. Gentry, C., Halevi, S., Vaikuntanathan, V.: A Simple BGN-Type Cryptosystem from LWE. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 506–522. Springer, Heidelberg (2010)
17. Gentry, C., Ramzan, Z.: Single-Database Private Information Retrieval with Constant Communication Rate. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 803–815. Springer, Heidelberg (2005)

18. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game. In: 19th ACM Symposium on Theory of Computing (1987)
19. Hazay, C., Lindell, Y.: Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries. Journal of Cryptology 23(3) (2010)
20. Hazay, C., Toft, T.: Computationally Secure Pattern Matching in the Presence of Malicious Adversaries. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 195–212. Springer, Heidelberg (2010)
21. Ishai, Y., Paskin, A.: Evaluating Branching Programs on Encrypted Data. In: Vadhan, S.P. (ed.) TCC 2007. LNCS, vol. 4392, pp. 575–594. Springer, Heidelberg (2007)
22. Jha, S., Kruger, L., Shmatikov, V.: Towards practical privacy for genomic computation. In: 29th IEEE Symposium on Security and Privacy (2008)
23. Katz, J., Malka, L.: Secure text processing with applications to private DNA matching. In: 17th ACM Conference on Computer and Communications Security (2010)
24. Kojm, T.: ClamAV, `http://www.clamav.net`
25. Lauter, K., Naehrig, M., Vaikuntanathan, V.: Can homomorphic encryption be practical? In: 3rd ACM Workshop on Cloud Computing Security (October 2011)
26. Malkhi, D., Nisan, N., Pinkas, B., Sella, Y.: Fairplay – a secure two-party computation system. In: 13th USENIX Security Symposium (August 2004)
27. Mohassel, P., Niksefat, S., Sadeghian, S., Sadeghiyan, B.: An Efficient Protocol for Oblivious DFA Evaluation and Applications. In: Dunkelman, O. (ed.) CT-RSA 2012. LNCS, vol. 7178, pp. 398–415. Springer, Heidelberg (2012)
28. Paillier, P.: Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, p. 223. Springer, Heidelberg (1999)
29. Pinkas, B., Schneider, T., Smart, N., Williams, S.: Secure Two-Party Computation Is Practical. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 250–267. Springer, Heidelberg (2009)
30. Rivest, R., Adleman, L., Dertouzos, M.: On data banks and privacy homomorphisms. Foundations of Secure Computation (1978)
31. Roesch, M.: Snort – lightweight intrusion detection for networks. In: 13th USENIX Conference on System Administration (1999)
32. Smart, N.P., Vercauteren, F.: Fully Homomorphic Encryption with Relatively Small Key and Ciphertext Sizes. In: Nguyen, P.Q., Pointcheval, D. (eds.) PKC 2010. LNCS, vol. 6056, pp. 420–443. Springer, Heidelberg (2010)
33. Stehlé, D., Steinfeld, R.: Faster Fully Homomorphic Encryption. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 377–394. Springer, Heidelberg (2010)
34. Troncoso-Pastoriza, J.R., Katzenbeisser, S., Celik, M.: Privacy preserving error resilient DNA searching through oblivious automata. In: 14th ACM Conference on Computer and Communications Security (2007)
35. Wei, L., Reiter, M.K.: Third-party DFA evaluation on encrypted files. Tech. Rep. TR11-005, Department of Computer Science, University of North Carolina at Chapel Hill (2011)
36. Yao, A.C.: Protocols for secure computations. In: 23rd IEEE Symposium on Foundations of Computer Science (1982)
37. Zhuge, J., Holz, T., Song, C., Guo, J., Han, X., Zou, W.: Studying malicious websites and the underground economy on the Chinese web. In: Workshop on the Economics of Information Security (June 2008)