Access-Private Outsourcing of Markov Chain and Random Walk based Data Analysis Applications

Ping Lin, K.Selçuk Candan
Department of Computer Science and Engineering
Arizona State University
Tempe, AZ,85287-5406, U.S.A.
{ping.lin, candan}@asu.edu

Abstract

Random walk graph and Markov chain based models are used heavily in many data and system analysis domains, including web, bioinformatics, and queueing. These models enable the description and analysis of various behaviors of stochastic systems. If the system being modelled has certain properties, such as if it is irreducible and aperiodic, close form formulations corresponding to its stationary behavior can be used to analyze its behavior. However, if the system does not have these properties or if the user is not interested in the stationary behavior, then an iterative approach needs to be used to determine potential outcomes based on the initial probability distribution inputs to the model. In this paper, we focus on access-privacy enabled outsourced *Markov chain based data analysis applications, where a non-trusted service provider takes (hidden) user queries that are described in terms of initial state distributions, and evaluates them iteratively in an oblivious manner. We show that this iterative process can leak information regarding the possible values of the hidden input if the server has a priori knowledge about the underlying Markovian process. Hence as opposed to simple obfuscation mechanisms, we develop an algorithm based on* methodical *addition of extra states, which guarantees unbounded feasible regions for the inputs, thus preventing a malicious host from having an informed guess regarding the inputs.*

1. Introduction

Grid computing refers to the system design approach which benefits from resources available in a network to solve complex problems that usually needs a large number of processing cycles or a large amount of data from different origins. Through sharing of computing capabilities and data, grid computing accelerates processing of problems that are expensive for ordinary computers. Hence, it enables computing intensive scientific problems and large data analysis problems to be solved rapidly.

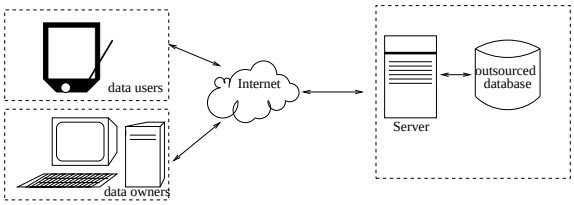


Figure 1. Outsourcing of data and data analysis services to third party data stores in a data grid environment. Note that there are three entities (a) data owners, (b) third party server, and (c) data users which all need to be protected from each others' malicious actions

Outsourcing data-driven applications to remote servers, in data and application grids, reduces the overhead and enhances system availability and scalability. However, the need to share resources and to execute untrusted code from any member of the data grid introduces various privacy concerns. In particular, having to partition the responsibilities of a single task into (a) *data/process owner*, (b) a third-party *data/computation server*, and (c) *data/process users* introduces new security and privacy challenges (Figure 1):

- Although data users may have rights to perform certain operations on the data outsourced by the owner, they may not have rights to have a copy of the data itself. That is, the remote server has to process the users' queries on the data owners data, without leaking data to the users.

- Similarly, users may not want to leak their queries to the server or even the owner. That is, the server has the responsibility of processing the queries on the owner's data, without leaking queries to the data owner.

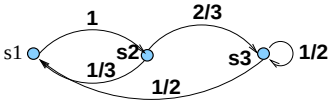
- Furthermore, as a party independent from the data owners and data users, the third party server may itself be malicious (or, at least untrustworthy). Therefore, without proper techniques to protect data and queries from the third party server, outsourcing schemes can not be widely applicable.

Providing end-to-end security in a framework within a shared data management environment is challenging. Coordinated and simultaneous deployment of multiple security provisions and methods such as trust management, policy management, authentication, authorization, confidentiality, and private information retrieval are required for end-to-end security. With the increased popularity of distributed and service oriented systems integration, private data management and mining are also up and coming research areas. Agrawal et. al. [18, 2] introduced the concept of Hippocratic databases where the disclosure of information by database accesses is prevented. Kantarcioglu et. al. [15, 8, 9] studied the problem of mining database without revealing sensitive information about data. We have discussed the current techniques available to application hosting services to tackle some of these challenges in [21]. We also provided private data retrieval techniques for data outsourcing systems in [19, 20]. In [22], building on the *private information retrieval* (PIR) literature, we developed efficient, single server, schemes which protect the data accesses from statistical traffic attacks.

In this paper, we focus on the challenges associated with the outsourcing of Markov chain based data analysis computations, which are used heavily in many scientific application domains, including web, bioinformatics, and queueing.

1.1. Markovian Stochastic Processes

A stochastic process is said to be Markovian if the conditional probability distribution of future states depends only on the present. A Markov chain is a discrete-time stochastic process which is conditionally independent of the past states. A random walk on a graph, $G(V, E)$, on the other hand, is a Markov chain whose state at any time is described by a vertex of G and the transition probability is distributed equally among all outgoing edges. The transition probability distribution for a Markov model can be represented as a matrix. The (i, j)'th element of this matrix, T_{ij}, describes the probability that, given that the current state is i, the process will be in state j in the next time unit; i.e., $T_{ij} = P(S_{now+1} = j|S_{now} = i)$.. Given this 1-step transition probability, the n-step transition probabilities can be computed as the n'th power of the transition matrix (Figure 2).



	s1	s2	s3
s1	0	1	0
s2	1/3	0	2/3
s3	1/2	0	1/2

Figure 2. A Markov chain and its transition matrix

Markovian models are used heavily in many application domains, including web, bioinformatics, and queueing. For instance, web search algorithms, such as HITS [11, 16] or PageRank [23, 5], rely on the analysis of the hyper-link structure of the web and use random walk graphs to identify relevant pages given a user query. Such connectedness analysis is now heavily exploited in other domains, such as bioinformatics. For instance, Gibson et. al. [12] use connectedness analysis and an iterative method for assigning and propagating weights to cluster categorical data in databases.

A common form of questioning in many of these application domains involves checking whether repeated application of a transition matrix from an initial probability distribution ever satisfies a given predicate, as illustrated in the example provided below.

Example 1.1 *The user wants to check if* $(T^{k+1} - T^k)\pi \leq \epsilon$ *for a given small vector* ϵ *for some* k. •

1.2. Iterative Evaluation of User Queries

Given a 1-step transition function, T, and an initial probability distribution vector π, finding k such that $\Theta(T^k\pi)$ for a k, involves the computation of the powers of the transition matrix. For instance, if the user wants to check if $(T^{k+1} - T^k)\pi \leq \epsilon$ for ϵ for some k, then the code needs to evaluate the condition sequence $(T - I)\pi \leq \epsilon$, $(T - I)T\pi \leq \epsilon$, $(T - I)T^2\pi \leq \epsilon$, until a suitable T^k is found.

If the transition matrix T is irreducible (i.e., each state is accessible from all other states) and aperiodic (i.e., for any state s_i, the greatest common divisor of the set $\{n \geq 1|T^n_{ii} > 0\}$ is equal to 1, then in the long run the Markov chain reaches a unique stationary distribution independent of the initial distribution. In such cases, it is generally possible to study this stationary distribution. Note that when the number of states is small it is easy to solve for the stationary distribution. However, when the state space is large, an iterative method (optimized for quick convergence through appropriate decompositions) has to be used [26].

Furthermore, not all transition matrices have those properties that guarantees stationary behavior. Also, users are not always interested in the stationary state behaviors of the system, but in how quickly a system converges to the stationary state [29] or, more generally, whether a given condition is true at any (bounded) future time.

In other words, in general, given an initial probability distribution vector π, users may aim finding if $\Theta(T^k\pi)$ for a k, here Θ denotes some predicate. In these cases, given an initial probability distribution vector π, short of tabulating all possible T, T^2, ... matrices, an iterative approach needs to be used. For example, to identify k where $(T^{k+1}-T^k)\pi \leq \epsilon$ is to execute the following **iterative** program:

```
while (true) {
      if (T-I)π ≤ ε then stop;
      else π=Tπ;
     }
```

Here I denotes identity matrix. Note that this is a costly operation as it requires repeated matrix multiplications and checking of the conditions at each iteration. Therefore, such iterative data analysis operations are perfect candidates for execution in an outsourced manner in data grids.

1.3. Outsourcing Markovian Stochastic Processes

In a computation grid environment, a common way of processing such costly computations is to use a powerful compute server carry out analysis: (1) A compute server hosts the outsourced Markov chain model, i.e., the transition matrix T, which may be outsourced to the server and condition Θ; (2) Clients with queries **access** the compute server with an initial probability distribution π describing the query; (3) The host, then, evaluates $\Theta(T^k\pi)$ and if the iteration stops for some k in a given bound, it sends the result $T^k\pi$ and k back to the requesting party.

On the other hand, such open shared systems introduce many privacy challenges that have to be addressed effectively to convince participants that the use of these services are not detrimental to their privacy. One challenge is to enable the use of data analysis services, while keeping the queries private to protect the users. With their central roles, servers are in positions that can hinder access privacies of the users. Therefore, appropriate methods should be taken to protect sensitive information about the use of data from potential adversaries, including the data store and the information service provider itself. Especially for business data analysis and military applications that do not lend themselves well to the risks involved in outsourcing query processing tasks to third-parties.

1.4. Contributions of this Paper

Unlike the related work on encrypted databases, private information retrieval, and privacy-preserving data mining, in this paper, we focus on access-privacy enabled Markov chain based data analysis computations, where a powerful but non-trusted computation node takes (a) the description of the Markovian process from a data owner and (b) hidden queries from the clients, and executes the queries using its resources in an oblivious manner.

As discussed in the next section, we consider a scenario where the matrices of the Markov chain process is encrypted by the owner before it is delivered to the server, using the encryption scheme such as [3]. Naturally, queries are also encrypted in a complementary manner before they are sent to the server for execution. We show that, even when both the matrices of the Markov chain and queries are encrypted beforehand, if the server has a priori knowledge regarding the underlying Markovian process, the iterative evaluation can leak information regarding the feasible values of the input queries, thus leaking user accesses to the server. Therefore, in Section 3, to hide input queries, we develop an algorithm based on *methodical* addition of extra states to the Markovian process and input queries. We show that, as opposed to simple obfuscation mechanisms, methodical addition of extra states, as proposed in this paper, guarantees unbounded feasible regions for the input vectors, thus preventing the server from learning about the user queries.

2. Access-Private Markov Chain Evaluation

In this paper, access-privacy means hiding the initial probability distributions provided by the clients. Let us consider the following basic iterative Markov chain based data analysis process:

```
while (true){
if Mcπ ≤ ε then stop
else π = Mtπ.
}
```

Here M_c denotes the condition matrix, M_t denotes the transition matrix, and π denotes the input. In Example 1.1, M_c= $(T - I)$ and M_t = T.

If the data source would like to protect its data (i.e., the transition matrix M_t, condition matrix M_c) from the data store, then M_t and M_c need to be encrypted by the owner before it is sent to the data store. Naturally, if M_t, M_c are encrypted before they are sent to the server, then the user queries (i.e, π) should also be encrypted in a compatible manner so that the server can execute the encrypted query on the encrypted data, without having to decrypt the data.

In [3], we introduced an encryption scheme for execution of encrypted linear algebraic programs (on matrices) with conditions and loops. In this framework,

- the data owner would hide its data with its own key, before outsourcing it to the data store (M_{t_H}, M_{c_H} instead of M_t, M_c) and
- each data user would modify its query correspondingly, using the keys provided by the sources, and send the hidden query (π_H) to the server for execution.
- the server would execute the hidden request/query over the encrypted data provided by the data owner and pass the hidden result to the data user, who would then decrypt the output.

We omit the details of the encryption and encrypted domain processing techniques presented in [3]. **On the other hand**, in this paper, we note that even if we can encrypt the above Markov chain process (using [3] or any other scheme) into

```
while (true) {
if (M_{c_H} π_H ≤ ε) then stop
else π_H= M_{t_H} π_H;
}
```

where M_{c_H}, M_{t_H}, and π_H are all encrypted, a fundamental crib (or weakness) is that for correct execution of the program, if-statements need to be evaluated to *true* or *false* with results of the condition evaluations being in the **open**.

> Thus, even when the input vectors, transition matrices, and conditions themselves are encrypted, the host will see the result of condition evaluations and this can be used to attack at the input values.

This, along with the fact that existing encryption schemes for M_c and M_t are weak for iteration (they leak eigenvalues of the matrices [3]) especially for sparse matrices, enables the following attack by the server.

2.1. Attack Model

Let us consider the following iterative process of the if-statement:

```
while (true) {
if (M_c π ≤ ε) then stop
 else π = M_t π;
 }
```

and its encrypt version

```
while (true) {
if (M_{c_H} π_H ≤ ε) then stop
else π_H= M_{t_H} π_H;
}
```

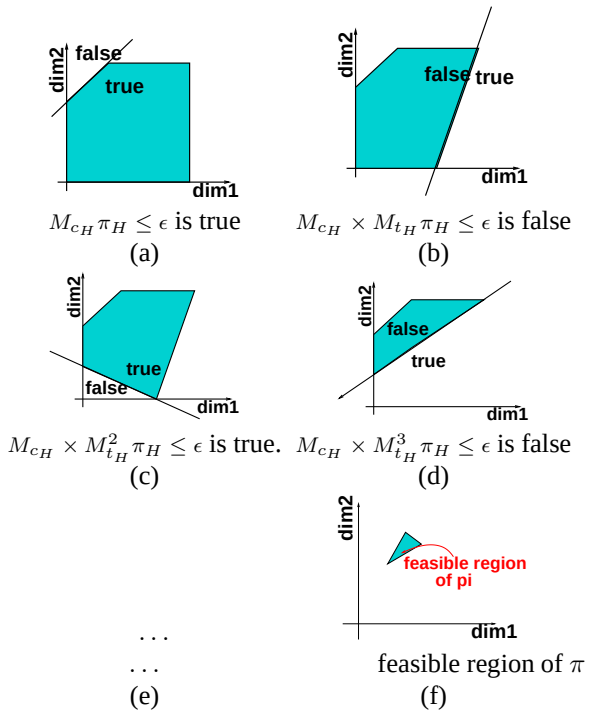


Figure 3. Attack example: derivation of feasible region of π through repeated evaluation of the condition

Whatever hiding scheme is used to hide M_c, M_t and π into M_{c_h}, M_{t_h} and π_h, for the host to create a correct execution flow, the execution of the hidden condition $M_{c_h}\pi_h \leq \epsilon$ has to match the evaluation of the original condition $M_c\pi \leq \epsilon$.

This means that, if the host knows has a priori knowledge regarding the actual condition matrix, M_c, and the transition matrix M_t, then by repeating this if-statement multiple times, it can gather information about the region where all possible values of input vector π lie (we call this the *feasible region* of π). In particular, the attacker can evaluate the following sequence of conditions:

$$\{M_{c_H}(\pi_H) \leq \epsilon,\ \ M_{c_H}(M_{t_H}\pi_H) \leq \epsilon, \ldots,\ \ M_{c_H}(M^j_{t_H}\pi_H) \leq \epsilon, \ldots\}$$

which is equivalent to:

$$\{M_c(\pi) \leq \epsilon,\ \ M_c(M_t\pi) \leq \epsilon, \ldots,\ \ M_{c_H}(M^j_t\pi) \leq \epsilon, \ldots\}$$

and observe the outcomes.

Let us assume that this sequence of conditions evaluates to

$$\{true,\ \ false,\ \ true,\ \ false,\ \ false,\ \ ...\}.$$

By observing these truth/falsehood values, the host may derive a region where all possible values of π may lie. Figure 3 depicts the process that the attacker can use for this purpose.

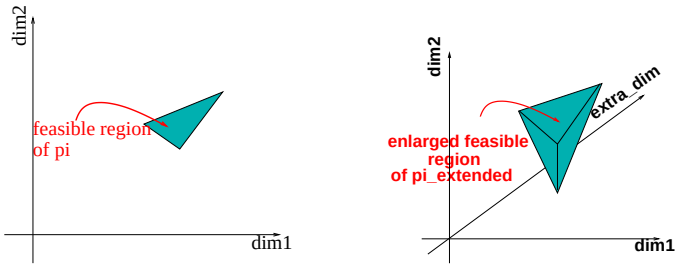


Figure 4. Expansion of feasible region of π

In this example, without a loss of generality, π is assumed to be a two dimensional vector and M_c is a 1×2 condition matrix, i.e., M_c denotes a single linear condition with elements of π. Figure 3(a) shows the line $M_c\pi = \epsilon$ which divides this plane into two regions: in the region above the line, the evaluation of $M_c\pi \leq \epsilon$ is false, in the region below, the evaluation is true. Since the host knows that the evaluation result is true, the host knows that the feasible region of π is below this line. Figure 3(b), then, shows that by repeating this process, the host can confine the set of feasible values of π to a narrower region: using the second result and fact that in the region above the line $M_c \times M_t\pi = \epsilon$, the evaluation of $M_c \times M_t\pi \leq \epsilon$ is false, and in the area below the line, the evaluation is true, the host learns that the feasible region of π should be above this line. Similarly, Figure 3 (c) through Figure 3(f) show that the host can progressively constrain the feasible region into increasingly smaller ones, gaining more knowledge of the input vector. Since the derivation of feasible region is through repeated evaluation of the conditions, we call this attack *condition evaluation attack*.

2.2. Measuring the Degree of Access Privacy

Since the feasible region of a hidden input vector includes the possible values the input can take, the larger the feasible region of the hidden input vector is, the harder it is for the malicious server to discover the input. Hence the degree of access privacy can be measured by the volume of the feasible region. On the other hand, when the feasible region is unbounded (i.e., it is of infinite volume), the number of free (unbounded) dimensions of the feasible region measures the degree of access privacy of the system.

2.3. Proposed Solution: Systematic Enlargement of the Feasible Regions of the Input Vectors

One way to prevent the host from learning the feasible region of the input vectors, is to add extra dimensions into π and correspondingly enlarge transition matrices and conditions. Figure 4 informally shows how the expansion of feasible region of π might help in hiding π. In this figure, π is the original input vector. Figure 4 (a) shows the situation where the host used the *condition evaluation attack* to bound the possible values of π into a small triangular region. Figure 4 (b), on the other hand, shows the situation after the addition of an extra variable (represented by extra_dim) to expand π into a 3 dimensional vector, $\pi_{expanded}$. As shown by Figure 4(b), with the extra dimension, the owner increases the size (volume) of the set of all possible values of π that the server can derive from repeated evaluation of if-statement, hence guarantees that the feasible region of $\pi_{expanded}$ is larger than the feasible region of π, which makes it harder for the host to guess the original input. In fact, if we could ensure that the feasible region of $\pi_{expanded}$ is not only larger but is *unbounded*, then the feasible region will become infinite.

Adding dummy dimensions (states) to enlarge feasible region involves expanding the transition and condition matrices correspondingly. This increases the storage cost for the matrices and introduces a performance penalty for the server. For instance, if the original number of states of the Markovian process is N, since the rows and columns of the transition matrix correspond to the states of the Markov chain, introducing one dummy state will increase the storage cost for the transition matrix by $(N+1)^2 - N^2 = 2N+1 = O(N)$. Since both condition evaluation and state transition calculation involve multiplying a N^2 matrix with a vector of size N, the computation cost for condition evaluation or state transition will increase by $(N+1)^2 - N^2 = O(N)$. Thus, since the expansion process involves both storage and computation overheads, it is essential to choose the new states and expand the transition and condition matrices carefully so as to maximize the feasible region with minimal cost.

3. Enlargement of the Feasible Regions of the Input Vectors

Increasing the number of dimensions of the input vector (and the transition matrices) such that the feasible region is larger is not trivial. Intuitively, each new dimension of the input vector corresponds to a new state in the Markov chain. However, the expanded Markov chain has to have the same semantics as the original one. Therefore, if a dummy state

is added without properly connecting it with the remainder of the original states, then the only value (initial probability) it can take will be 0. Therefore, such an extra state will not increase the feasible region of the inputs. On the other hand, once a new state is connected to the rest of the original states, then transition probabilities has to be modified to maintain the original semantics of the Markov chain.

Furthermore, it should not be trivial for the attacker to identify which are the new dimensions and which are the old ones. Therefore, dimensions can not be added blindly. In this section, we present a method to systematically add new dimensions (or states) to prevent the host from using the condition evaluation attack to learn the feasible region of the inputs. In effect, given

```
while (true) {
if (M_c π ≤ ε) then stop
 else π = M_t π;
 }
```

we expand the original matrix, the condition matrix, and the input vector such that

```
while (true) {
if (M_c^exp π^exp ≤ ε) then stop
else π^exp= M_t^exp π^exp;
}
```

These expanded matrices will then be hidden using an encryption scheme such as [3] before sending to the outsourcing server.

3.1. Methodically Increasing the Degrees of Freedom of the Input Vector

Let us consider the iterative Markov chain evaluation process:

```
while (true) {
if (M_c π ≤ ε) then stop
 else π = M_t π;
}
```

where π is an l dimension vector, M_t is an $l \times l$ matrix, and M_c is an $n \times l$ matrix. Let us further assume that the matrix M_t is diagonalizable[1] and M_t's eigenvalues are $\lambda_1, \lambda_2, \ldots \lambda_l$. We can then rewrite M_t as $P\Lambda P^{-1}$. Here, P is an invertible matrix and Λ are diagonal matrices with their diagonal elements being eigenvalues (λ_1,λ_2,...λ_l) of M_t. In other words:

$$\Lambda = diag(\lambda_1, \lambda_2, ..., \lambda_l)$$

[1]This is a limiting assumption. However, Markov chains with certain important properties, such as irreducible, aperiodic, and reversible Markov chains [29] and certain random walks on the set of chambers of a real hyperplane arrangement that are often used as shuffling schemes in computer science, biology, or card games, fall within this category [7]

From this the following theorem follows:

Theorem 3.1 (Dimensions of the Feasible Region) If M_t has s $(s \leq l)$ distinct eigenvalues, then it follows that if the host knows the transition matrix, M_t and condition matrix, M_c, then the feasible region of π that the host can learn has at least $l - s \times n + 1$ dimensions; in other words, π has $l - s \times n + 1$ degrees of freedom. •

Note that the greater number of distinct eigenvalues M_t has, the smaller number of dimensions the feasible region of π has. Hence, the *easier* it will be for a malicious host to find the value of the input data. In the extreme case, if $l-s\times n+1$ is less than or equal to 0, the feasible region may be bounded at all dimensions. This suggests a systematic way to add extra dimensions.

Insigt 1 (Unbounding the Feasible Region) If we can transform matrix M_t into M_t^{exp}, such that the corresponding $l^{exp} > l$, while s and n stay the same. This will increase the degrees of freedom, rendering possibly unbounded dimensions. Thus the feasible region host can observe is not only enlarged, but made unbounded. •

Before we present how we can leverage this insight to design an algorithm to hide the input from an attacker, we first prove the key theorem (Theorem 3.1).

Proof 3.1 (Dimensions of the Feasible Region) *The feasible region of π the host gets from the evaluation of the following condition sequence*

$\{M_c\pi \leq \epsilon, M_c \times M_t\pi \leq \epsilon,\ M_c \times M_t^2\pi \leq \epsilon, M_c \times M_t^3\pi \leq \epsilon, \ldots, M_c \times M_t^j\pi \leq \epsilon, \ldots\}$*, is at least one dimension larger than the dimension of the feasible region of the following equations:*

$$\begin{aligned} M_c\pi &= \epsilon \\ M_c \times M_t\pi &= \epsilon \\ M_c \times M_t^2\pi &= \epsilon \\ \vdots & \quad \vdots \\ M_c \times M_t^j\pi &= \epsilon \\ \vdots & \quad \vdots \end{aligned}$$

Consider that $M_t = P \times \Lambda \times P^{-1}$. They are equivalent to:

$$\begin{aligned} M_c \times P \times P^{-1}\pi &= \epsilon \\ M_c \times P \times \Lambda \times P^{-1}\pi &= \epsilon \\ M_c \times P \times \Lambda^2 \times P^{-1}\pi &= \epsilon \\ \vdots & \quad \vdots \\ M_c \times P \times \Lambda^j \times P^{-1}\pi &= \epsilon \\ \vdots & \quad \vdots \end{aligned}$$

$$\begin{bmatrix} 1 & 1 & 1 & . & 1 \\ \lambda_1 & \lambda_2 & \lambda_3 & . & \lambda_l \\ \lambda_1^2 & \lambda_2^2 & \lambda_3^2 & . & \lambda_l^2 \\ . & . & . & . & . \\ \lambda_1^i & \lambda_2^i & \lambda_3^i & . & \lambda_l^i \\ . & . & . & . & . \end{bmatrix}$$

(a) Matrix V

$$\begin{bmatrix} V & 0 & 0 & . & 0 \\ 0 & V & 0 & . & 0 \\ . & . & . & . & . \\ 0 & 0 & 0 & . & V \end{bmatrix},$$

(b) Matrix Y_1

$$\begin{bmatrix} a_{1,1} & 0 & . & 0 & a_{2,1} & . & 0 & ... & a_{n,1} & 0 & . & 0 \\ 0 & a_{1,2} & . & 0 & 0 & . & 0 & ... & 0 & a_{n,2} & . & 0 \\ 0 & 0 & . & 0 & 0 & . & . & ... & 0 & 0 & . & 0 \\ . & . & . & . & . & . & . & ... & . & . & . & . \\ 0 & 0 & . & a_{1,l} & 0 & . & a_{n,l} & ... & 0 & 0 & . & a_{n,l} \end{bmatrix}$$

(c) Matrix Y_2^T

Figure 5. Definitions of V, Y_1, and Y_2^T for Proof 3.1

Suppose $M_c = \begin{bmatrix} m_1^T & m_2^T & m_3^T & . & . & . & m_n^T \end{bmatrix}^T$ *and* $m_i^T = \begin{bmatrix} m_{i,1} & m_{i,2} & . & . & . & m_{i,l} \end{bmatrix}$, $i \in \{1, 2, ...n\}$ *are row vectors of matrix* M_c. *Let* a_i^T=$m_i^T \times P$ *and* $a_i^T = \begin{bmatrix} a_{i,1} & a_{i,2} & . & . & . & a_{i,l} \end{bmatrix}$. *Furthermore, let* $y = P^{-1}\pi$. *Since* P *is an invertible matrix, the dimension of feasible region of* y *is the same as the dimension of feasible region of* π. *We can transform the above equations into equations of* y*:*

$$\begin{bmatrix} a_1^T \\ a_2^T \\ \dots \\ a_n^T \end{bmatrix} \times \Lambda^j \times y = \epsilon,$$

where $j \geq 0$.

If we denote the corresponding matrix of linear equations for y *as* Y *and define matrices* V, Y_1, *and* Y_2^T, *as in Figure 5, then, we get the equality* $Y = Y_1 \times Y_2$.

Therefore, the number of the dimensions of the feasible region of y *(if not empty) is* $l - rank(Y)$. *We also know that* $rank(Y) \leq min(rank(Y_1), rank(Y_2))$, $rank(Y_1) = n \times rank(V)$, *and for matrix* V, *if* M_t *has* $s(s \leq l)$ *distinct eigenvalues, then* $rank(V) = s$, *thus we get that* $rank(Y) \leq n \times s$.

So, the number of the dimensions of the feasible region of y *we get from linear equations is at least* $l - n \times s$. *Consequently, we can conclude that the dimension of feasible region of* π *host can get from evaluation of the condition sequence is at least* $l - n \times s + 1$. •

3.2. Feasible Region Enlargement Algorithm

The algorithm to add extra states into program to enlarge the feasible region which a malicious host may observe through condition evaluation attack is as follows:

1. Diagonalize the input transition matrix and find the dimension of the feasible region of original program;
2. Choose a target number of dimensions such that the feasible region of the input becomes unbounded. Let d be the number of extra dimensions to be added;
3. Choose an invertible random transformation to map the input vector to a vector in the target space with higher number of dimensions (note that this transformation does not need to be linear);
4. Based on this transformation, expand the condition and transition matrices, in such a way that all elements of the expanded input vector properly participate in the condition evaluation and state transition. This is achieved as follows:
   (a) Select d eigenvalues randomly from original eigenvalues
   (b) Expand transition matrix according to the mapping between enlarged dimensions and original dimensions so that new eigenvalues introduced in the expanded transition matrix are those d eigenvalues;
   (c) Expand the condition matrix similarly.

In Section 3.3, we provide a detailed example showing how these steps are carried out. It is important to highlight that, to ensure that the transition behavior equivalent after expansion (i.e., to keep the mapping between original space and expanded space after state transitions) as well as to maximize the feasible region of the input, the algorithm chooses the eigenvalues of the expanded transition matrix from the original eigenvalues.

Note that, the expanded transition matrix is technically not a Markov chain. In particular, the transition probabilities as well as state probabilities can take values outside of the $[0, 1]$ range, including negative values. However, behaviorally, the expanded matrix is the same as the original transition matrix.

3.3. A Detailed Example

In this section, we provide a detailed example showing the expansion process.

Let us be given the following program:

```
while (true) {
      if ( −2/3 x + 2/3 y ≤ ε1 and 2/3 x - 2/3 y ≤ ε2) then
          stop
      else
          [ x ]   [ 1/3  2/3 ] [ x ]
          [ y ] = [ 2/3  1/3 ] [ y ] ;
}
```

In this program

- the initial distribution is $\pi = \left[\begin{array}{cc} x & y \end{array}\right]^T$,
- the transition matrix is $M_t = \left[\begin{array}{cc} 1/3 & 2/3 \\ 2/3 & 1/3 \end{array}\right]$,
- the condition matrix is
$$M_c = M_t - I = \left[\begin{array}{cc} -2/3 & 2/3 \\ 2/3 & -2/3 \end{array}\right],$$
and
- the bound is $\epsilon = \left[\begin{array}{cc} \epsilon_1 & \epsilon_2 \end{array}\right]^T$.

The transition matrix in the program is diagonalizable and its two eigenvalues are $-1/3$ and 1. Also, in this example we have $l = 2$, $s = 2$, and $n = 2$.

Therefore, in this example, $l - s \times n + 1$ is equal to -1. Since this value is less than 0, the feasible region of the input might be bounded. To render the feasible region unbounded, we need to add at least 2 new states. However, to simplify the example, here we will only show how to expand the matrix by the addition of (only) 1 new state, z. The expansion by 2 or more states is similar. Let us assume that in Step 3 of the algorithm, the following transformation is used for mapping the states of the original matrix to the expanded states:

$$\begin{array}{rcl} x' & = & a_1x + b_1y + c_1; \\ y' & = & a_2x + b_2y + c_2; \\ z' & = & a_3x + b_3y + c_3; \end{array}$$

The transformation needs to be invertible, i.e., from the expanded vector and the transformation, we can recover the original vector. This transformation between original input vector and expanded vector should also be always true before and after each iteration step. If we use M'_t (M_t) to denote the expanded(original) transition matrix, t'_{ij} (t_{ij}) to denote elements in M'_t (M_t), variable with subscript $+$ $(-)$ to denote the corresponding variable after (before) iteration, then for this example, we get

$$\begin{array}{rcl} x'^{+} & = & t'_{11}x'^{-} + t'_{12}y'^{-} + t'_{13}z'^{-} \\ & = & t'_{11}(a_1x^- + b_1y^- + c_1) + \\ & & t'_{12}(a_2x^- + b_2y^- + c_2) + \\ & & t'_{13}(a_3x^- + b_3y^- + c_3) \\ & = & (a_1t'_{11} + a_2t'_{12} + a_3t'_{13})x^- + \\ & & (b_1t'_{11} + b_2t'_{12} + b_3t'_{13})y^- + \\ & & (c_1t'_{11} + c_2t'_{12} + c_3t'_{13}); \\ y'^{+} & = & (a_1t'_{21} + a_2t'_{22} + a_3t'_{23})x^- + \\ & & (b_1t'_{21} + b_2t'_{22} + b_3t'_{23})y^- + \\ & & (c_1t'_{21} + c_2t'_{22} + c_3t'_{23}); \\ z'^{+} & = & (a_1t'_{31} + a_2t'_{32} + a_3t'_{33})x^- + \\ & & (b_1t'_{31} + b_2t'_{32} + b_3t'_{33})y^- + \\ & & (c_1t'_{31} + c_2t'_{32} + c_3t'_{33}); \end{array}$$

Also considering that we have

$$\begin{array}{rcl} x'^{+} & = & a_1x^+ + b_1y^+ + c_1 \\ & = & a_1(t_{11}x^- + t_{12}y^-) + b_1(t_{21}x^- + t_{22}y^-) + c_1 \\ & = & (a_1t_{11} + b_1t_{21})x^- + (a_1t_{12} + b_1t_{22})y^- + c_1 \\ y'^{+} & = & (a_2t_{11} + b_2t_{21})x^- + (a_2t_{12} + b_2t_{22})y^- + c_2; \\ z'^{+} & = & (a_3t_{11} + b_3t_{21})x^- + (a_3t_{12} + b_3t_{22})y^- + c_3; \end{array}$$

we get the following equations to solve:

$$\begin{array}{rcl} a_1t'_{11} + a_2t'_{12} + a_3t'_{13} & = & a_1t_{11} + b_1t_{21} = \frac{1}{3}a_1 + \frac{2}{3}b_1; \\ b_1t'_{11} + b_2t'_{12} + b_3t'_{13} & = & a_1t_{12} + b_1t_{22} = \frac{2}{3}a_1 + \frac{1}{3}b_1; \\ c_1t'_{11} + c_2t'_{12} + c_3t'_{13} & = & c_1; \\ a_1t'_{21} + a_2t'_{22} + a_3t'_{23} & = & a_2t_{11} + b_2t_{21} = \frac{1}{3}a_2 + \frac{2}{3}b_2; \\ b_1t'_{21} + b_2t'_{22} + b_3t'_{23} & = & a_2t_{12} + b_2t_{22} = \frac{2}{3}a_2 + \frac{1}{3}b_2; \\ c_1t'_{21} + c_2t'_{22} + c_3t'_{23} & = & c_2; \\ a_1t'_{31} + a_2t'_{32} + a_3t'_{33} & = & a_3t_{11} + b_3t_{21} = \frac{1}{3}a_3 + \frac{2}{3}b_3; \\ b_1t'_{31} + b_2t'_{32} + b_3t'_{33} & = & a_3t_{12} + b_3t_{22} = \frac{2}{3}a_3 + \frac{1}{3}b_3; \\ c_1t'_{31} + c_2t'_{32} + c_3t'_{33} & = & c_3; \end{array} \quad (1)$$

From the above equations, we can get two constraints $c_2 = \frac{a_2+b_2}{a_1+b_1}c_1$ and $c_3 = \frac{a_3+b_3}{a_1+b_1}c_1$. Hence for this example, we need to select random invertible transformations, such that c_1, c_2, c_3 satisfy these two constraints. Suppose that we choose the following transformation to map input vector from 2 dimensions to 3 dimensions:

$$\begin{array}{rcl} x' & = & x + y + 2; \\ y' & = & 2x + y + 3; \\ z' & = & 3x + 2y + 5; \end{array}$$

In order to enlarge the feasible regions of the variables, we also choose redundant eigenvalues from the original ones; that is, in this example we choose either -1/3 or 1. Suppose we choose $-1/3$. Then, one possible expansion of Λ is

$$\left[\begin{array}{ccc} -1/3 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1/3 \end{array}\right].$$

Therefore, using this Λ, the equation set 1, and the fact that $M'_t = P\Lambda P^{-1}$, we can compute possible values for M'_t. For this example, one such possible

$$M'_t \text{ is } \left[\begin{array}{ccc} 3/11 & -8/11 & 8/11 \\ 10/11 & -47/33 & 12/11 \\ 50/33 & -20/11 & 49/33 \end{array}\right].$$

Similarly, by utilizing the chosen transformation and the equivalence between expanded condition matrix and the original one, we can compute can a corresponding expanded condition matrix M'_c as

$$\left[\begin{array}{ccc} 18/11 & -56/33 & 4/11 \\ -26/11 & 188/33 & 4/11 \end{array}\right].$$

As shown by Figure 6, the original Markov chain (depicted in Figure 6(a)) is transformed into a larger stochastic process (depicted in Figure 6(b)). Note that, after the transformation, the resulting process is technically not a Markov chain, due to the negative values (e.g., $\frac{-8}{11}$) or larger than

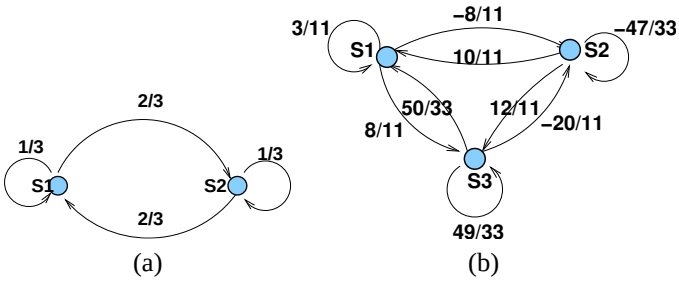


Figure 6. (a) The original Markov chain and (b) the state transition graph looks after expansion; note that after expansion the stochastic process is not technically a Markov chain, due to the negative values (e.g, the value $\frac{-8}{11}$ that is associated with edge (s_1,s_2)) or larger than 1 values (e.g., the value $\frac{50}{33}$ that is associated with edge (s_3, s_1)) that are not within [0,1] range of transition probabilities; however the graph is suitable for processing the query

1 values (e.g., $\frac{50}{33}$) in the expanded transition matrix. However, the two processes' transition behaviors are equivalent for the purposes of the evaluation of the given program; the first one operates on the original input, while the second one operates on the hidden inputs and prevents leakages by if-statement attacks.

This finishes the example and the description of the proposed scheme for ensuring access privacy even when outsourced model is broken by the malicious service provider.

3.4. Related Work: Hiding Code and Applications from Execution Environments

Protecting outsourced application/data from a malicious execution environment, is also challenging. To prove that the application is executed properly, the owner can benefit from replicated execution or execution traces [28] and trusted third parties [27]. Sander and Tschudin [24] proposed that mobile code can be executed by hosts in an encrypted form so that function is hidden from adversary and nobody except code owner can modify the program in goal-oriented way. The scheme they proposed is suitable for evaluating polynomial and relational functions. Lee et. al. [17] proposed a hybrid approach to implement this protocol. This approach mixes a function composition technique and a homomorphic encryption scheme. In [1], a technique for letting a host compute its own function with client's encrypted data so that no information about input data is leaked to the hostile host was proposed. Sander, and Tschudin [25] considered computing with encrypted input and stated that algebraic homomorphic one-way trapdoor functions allow not only non-interactive computing with encrypted data, but also non-interactive computing with encrypted function for evaluation of polynomials. In [3], we introduced an encryption scheme for preserving access privacy and content privacy of outsourced linear algebraic programs with conditions and loops.

To encrypt data before data outsourcing is a common practice to protect outsourced data from untrusted host server. This leads to encrypted data access research [14, 13, 4, 6, 10]. Hacigümüs et. al. [13] discussed executing SQL queries on encrypted data. Bouganim and Pucheral [4] proposed to use assistant hardware such as smart cards to help with data encryption/decryption when accessing encrypted data. Hore et. al. [14] suggested to cluster attribute values into buckets and use bucket identities as index for range queries over encrypted data. Inspired by the idea of secret sharing, Brinkman et. al. [6] split the XML documents into client part and server part so that client can recover query result from the partial result returned from the server. Elovici et. al. [10] studied how to encrypt database without altering data structure and transform index into a secure one that would not reveal data content.

4. Conclusion and Future Work

This paper focuses on access-privacy enabled Markov chain evaluation where a non-trusted service provider takes hidden user queries, described in terms of initial state distributions, and evaluates them with the hosted Markov chain model in an oblivious manner. We find that due to the iteration of state transition, any possible encryption scheme for such purpose requires self-composability. Hence if the Markov chain model is known a priori by the non-trusted compute environment, the host can deduce the feasible region of initial state distributions. To protect user queries from such attack, we develop an algorithm based on methodical addition of extra dimensions. We show that if the data as well as the Markov chain model are properly encrypted, then user queries will be protected from compute environments even when the model itself is publicly known. In the presented work, we assume that transition matrix of the Markov chain is diagonalizable. Although this captures an important class of Markov chains, we admit that it is a restricting assumption. Hence in the future, we will extend our work to a more general class of Markov chains with transition matrixes in Jordan normal form.

5. Acknowledgement

We acknowledge Dr. Rida Bazzi and Zhichao Liu for their participation in the discussion of the work.

References

[1] M. Abadi, J. Feigenbaum, and J. Kilian, "On Hiding Information from an Oracle", JCSS, 39(1) 1989. pp. 21-50.

[2] R. Agrawal et. al. "Hippocratic databases", VLDB 2002, pp. 143-154.

[3] R. Badin, R.A. Bazzi, K.S. Candan, and A. Fajri, "Provably Secure Data Hiding and Tamper Resistance for a Simple Loop Program", SPIE Volume: 5071 Sensors, and Command, Control, Communications, and Intelligence (C3I) Technologies for Homeland Defense and Law Enforcement II, 2003.

[4] L. Bouganim, P. Pucheral, "Chip-secured Data Access: Confidencial Data on Untrusted Servers", VLDB 2002, pp. 131-142.

[5] S. Brin and L. Page "The anatomy of a large-scale hypertextual Web search engine". Computer Networks and ISDN Systems, 30(1-7), 1998. pp.107-117.

[6] R. Brinkman, J. Douman, and W. Jonker, "Using Secret Sharing for Searching in Encrypted Data", SDM 2004, pp. 18-27.

[7] K. S. Brown, and P. Diaconis, "Random walks and hyperplane arrangements", Annuals of Probability. 26 (1998), no. 4, 1998, pp.1813-1854.

[8] C. Clifton et. al. Sahuguet, "Privacy-Enhanced Data Management for Next-Generation e-Commerce", VLDB 2003, pp.1147.

[9] C. Clifton et. al. "Tools for Privacy Preserving Data Mining", SIGKDD Explorations 4(2), 2002, pp. 28-34.

[10] Y. Elovici et. al. "A structure Preserving Database Encryption Scheme". SDM 2004, pp. 28-40.

[11] D. Gibson, J. Kleinberg, and P. Raghavan, "Inferring Web Communities from Link Topology", the 9th ACM Conference on Hypertext and Hypermedia: Links, Objects, Time and Space - Structure in Hypermedia Systems, 1998, pp. 225-234.

[12] D. Gibson, J. Kleinberg, and P. Raghavan, "Clustering Categorical Data: An Approach Based on Dynamical Systems", VLDB 1998, pp. 311-322.

[13] H. Hacigümüs et. al. Data in the Database Service Provider Model", ACM SIGMOD International Conference on Management of Data, 2002, pp. 216-227.

[14] B. Hore , S. Mehrotra and G. Tsudik, "A Privacy-Preserving Index for Range Queries", VLDB 2004, pp. 720-731.

[15] M. Kantarcioglu and C. Clifton, "Privacy-Preserving Distributed Mining of Association Rules on Horizontally Partitioned Data", IEEE Trans. Knowl. Data Eng. 16(9), 2004, pp. 1026-1037.

[16] J. Kleinberg, "Authoritative sources in a hyperlinked environment", Journal of the ACM, 46(5), September 1999, pp.604-632.

[17] H. Lee, J. Alves-Foss, and S. Harrison, "The Use of Encrypted Functions for Mobile Agent Security", the 37th Hawaii International Conference on System Sciences, 2004.

[18] K. LeFevre et. al. "Limiting Disclosure in Hippocratic Databases", VLDB 2004, pp. 108-119.

[19] P. Lin, and K. S. Candan, "Hiding Tree Structured Data and Queries from Untrusted Data Stores", Information Systems Security Journal, June 2005.

[20] P. Lin, and K. S. Candan, "Secure and Privacy Preserving Outsourcing of Tree Structured Data", SDM 2004, pp. 1-17.

[21] P. Lin and K. S. Candan, "Data and Application Security for Distributed Application Hosting Services", Information Security Policies and Actions in Modern Integrated Systems, Idea Group Publishing, 2004, pp. 273-316.

[22] P. Lin and K. S. Candan, "Communication and Computation Efficient Schemes for Protecting the Privacy of Outsourced Data against Traffic Analysis Attacks", Submitted to VLDB Journal, 2005.

[23] L. Page et. al. "The pagerank citation ranking: Bringing order to the web". Technical report, Stanford SIDL-WP-1999-0120, 1999.

[24] T. Sander and C. F. Tschudin, "Towards Mobile Cryptography", Proceedings of the IEEE Symposium on Security and Privacy, 1998.

[25] T. Sander, and C. F. Tschudin, "Protecting Mobile Agents Against Malicious Host"., Mobile Agents Security, 1998, pp. 44-60.

[26] W. J. Stewart and W. Wu. Numerical experiments with iteration and aggregation for Markov chains. ORSA J. Comput., 4, 1992, pp.336-350.

[27] H. K. Tan and L. Moreau, "Certificates for mobile code security", SAC, ACM, 2002, pp. 76-81.

[28] G. Vigna, "Cryptographic Traces for Mobile Agents", Mobile Agents and Security, 1998, pp.137-153.

[29] E. L. Wilmer, "Exact Rates of Convergence for Some Simple Non-Reversible Markov Chains", Ph.D. thesis presented to the Univerity of Harvard.