Statistical Analysis on Large Encrypted Datasets

A thesis submitted in partial fulfillment
of the requirements for the degree of

BACHELOR OF TECHNOLOGY

in

Computer Science & Engineering

by

Nishant Kumar

Entry No. 2012CS10239

K. Trivikrama Reddy

Entry No. 2012CS10229

Under the guidance of

Dr. Shweta Agrawal

Department of Computer Science and Engineering,
Indian Institute of Technology Delhi.

May 2016.

Certificate

This is to certify that the thesis titled **Statistical Analysis on Large Encrypted Datasets** being submitted by **Nishant Kumar** and **K. Trivikrama Reddy** for the award of **Bachelor of Technology** in **Computer Science & Engineering** is a record of bona fide work carried out by them under my guidance and supervision at the **Department of Computer Science & Engineering**. The work presented in this thesis has not been submitted elsewhere either in part or full, for the award of any other degree or diploma.

Dr. Shweta Agrawal
Department of Computer Science and Engineering
Indian Institute of Technology, Delhi

Abstract

The problem of performing statistical analysis on massive encrypted databases has received much attention in the recent years. Formally, consider a data owner $\mathcal{D}$ who wishes to outsource massive sensitive data to an untrusted server $\mathcal{E}$ and multiple clients who wish to compute certain functions on the encrypted database. Existing approaches given by cryptography fall short in either efficiency or functionality – functional encryption (FE) and homomorphic encryption (HE) schemes support general functionalities but take time polynomial in the size of the dataset, multi-client searchable symmetric encryption (MC-SSE) and structured encryption schemes are efficient but do not support desired functionality.

In this work, domain specific knowledge is used to identify functionalities that are relevant to statistical analysis and also design protocols that support computation of these functionalities in time that is sublinear in the database size. To achieve this, techniques from MC-SSE as well as HE/FE are unified and extended to provide schemes that are efficient, compute meaningful (restricted) functionalities and achieve simulation based security with quantified leakage to adversarial server and clients, which is confined to well defined forms of data access and query patterns, as in the case of MC-SSE.

Two main protocols are provided: the first to compute basic statistical functions such as mean and variance of data records that satisfy some filtering criteria specified by a Boolean predicate, including range queries, and the second to compute arbitrary functions on data records that match some keyword. These constructions provide the first efficient multi-client "search and compute" schemes on massive datasets.

Acknowledgments

We would like to express our sincere gratitude to Dr. Shweta Agrawal for giving us this opportunity to work with her. It has been a wonderful experience working under her and has given us a strong flavor of research in this area. Our project has been filled with ups and downs and we are highly grateful to her to not lose faith in us even in the worst of times. Her sheer ability to simplify complex statements and protocols and look at things the way we never even thought, astounded us at times. She was always prompt to give us time to meet whenever we needed it. We feel thankful for the constant motivation, push and especially the amount of time she provided us in the last one month of the project which enabled us to finish this project in time and bring it in the shape it is today.

Finally, we would like to thank our family and friends, who helped us endure through worst circumstances and emotinal distress and provided us support whenever we needed it.

Nishant Kumar
K. Trivikrama Reddy

Contents

1 Introduction **1**

1.1 Motivation 1

1.2 Existing Approaches 2

1.3 Our Approach 3

2 Preliminaries **4**

2.1 Definitions 4

2.1.1 Correctness 5

2.1.2 Security 6

2.2 Building Blocks 7

3 Multi-Client Searchable Symmetric Encryption **10**

4 Computing Filtered Average **12**

4.1 Protocol 12

4.1.1 Correctness 13

4.1.2 Security against adversarial server 14

4.1.3 Security against malicious clients 18

4.2 Support for Range Queries 19

4.3 Support for Variance and other functions 21

5 Computing General Functions **22**

5.1 Protocol 22

5.2 Correctness and Security 23

5.3 Discussion 25

6 Related Work **26**

© *2016, Indian Institute of Technology Delhi*

Bibliography 27

A Protocol supporting Range Queries 31

B Proof of theorem 6: Security against adversarial clients 33

Chapter 1

Introduction

1.1 Motivation

Suppose the government of some country wishes to provide controlled access to census data to enable social scientists and market researchers to perform statistical analyses using restricted computations that respect users' privacy. Most governments already do publish census data in some sanitized form for the public domain as well as provide restricted access data to qualified researchers for approved projects [19, 20, 18, 46]. Indeed the government itself must perform some analysis on census data, and may wish to outsource the storage of and computation on this massive dataset to some untrusted server.

The ability to enable controlled computation on large encrypted databases can offer a solution in many practical scenarios. Take for example the case of Ipsos Mori, a British marketing research firm, which was recently in the news for offering data of 27 million mobile phones to the police [27]. The organization defended itself by insisting they only released "safe" aggregate data that involved 50 or more customers, which their users purportedly acquiesced to [41, 34]. However, the deal was shelved under public pressure. Another example is provided by a raging controversy regarding the "Aadhaar" project in present-day India [1]. The Aadhaar project is the world's largest national identification number project, launched by government of India, which seeks to collect biometric and demographic data of residents and store these in a centralised database [43]. To date, 1009 million users have enrolled in the system, and the government has spent at least 840 million USD on the project [43]. However, the project is encountering serious difficulties due to privacy concerns and risks being shelved [1, 39]. At least part of these concerns could be addressed by enabling controlled computation over a massive encrypted database.

© *2016, Indian Institute of Technology Delhi*

1.2 Existing Approaches

There has been significant research effort in the cryptographic community over the last decade to provide solutions for secure cloud computing, which generalises the above examples. The above scenario is modeled by considering a data owner who owns large scale sensitive data, an untrusted server who must store and compute over this data, and possibly multiple clients who must be provided keys that let them learn outputs of restricted computations.

A variety of cryptographic solutions have been constructed for this problem, offering various tradeoffs in security, efficiency and generality. On the theoretical side, the above scenario appears like a poster person application for the primitive of *functional encryption* (FE) [38, 37], which enables a central authority (possibly the data owner) to provide keys corresponding to functions from some family to malicious clients. Given a key SK_f corresponding to function f and a ciphertext $\mathsf{CT}_{\mathbf{x}}$ corresponding to data $\mathbf{x}$, a client may run the decryption procedure to learn exactly $f(\mathbf{x})$ and nothing else. However, the primitive of FE was designed for arbitrary computations on arbitrary data, and to compute any function on a dataset, even a "basic" function as keyword search, requires time polynomial in the size of the entire dataset [3]. FE does not use any knowledge about the application, and does not permit any leakage to the adversary, favoring generality and security unilaterally over efficiency. This negates applicability of FE to big data applications, as noted by a recent survey [23].

A different approach to this problem is offered by the multi-client extension of searchable symmetric encryption (MC-SSE) [26], where untrusted clients may be provided keys corresponding to Boolean queries on keywords. These keys let them interact with the server to learn the matching documents in time proportional to the result set. Security is established against an honest but curious server and malicious clients who do not collude with the server [26], additionally, some quantified leakage such as access patterns and query correlations is permitted. These solutions are far more suitable for computing on large encrypted datasets, for the restricted but important search functionality, offering a solution that favors efficiency over generality and security. However these schemes do not achieve the funcitonality we desire. See chapter 6 for more details about related work.

© *2016, Indian Institute of Technology Delhi*

1.3 Our Approach

The starting point of our work is the observation that in many cases, relevant computations in big data applications are concerned with only a small fraction of the entire dataset. In our motivating example of census data, typical computations are of the following form: "What is the average income of college graduates who are of gender A and age in range [B, C] in city D?" [19, 20, 18]. In the context of a genome database, medical practitioners typically desire to analyze connections between DNA and disease, say the connection between the MRE11A gene and breast cancer. The data relevant to these computations (e.g. the number of women with breast cancer) is only a small fraction of the entire database, and additionally, in these and many other examples, relevance of data to a particular computation is dictated by some filtering criteria.

In this work, we design multi-client search and compute schemes (MC-SCE) for computing functions on data that satisfies some filtering critera. We combine and extend techniques from searchable encryption, functional encryption and homomorphic encryption to achieve a new tradeoff between generality, efficiency and security. Our proposed constructions are efficient in that computation depends on the size of the relevant data rather than the entire dataset. Our first construction supports queries of the following form: "compute the aggregate A of field X of users that match filtering criteria Y" . The filtering criteria is expressed as a boolean predicate on a keyword space, which may include range specifications, and the aggregate A represents basic statistical functions such as mean, standard deviation, correlations and such others. Our second construction supports computation of arbitrary functions (represented as circuits) over records that match some keyword X. For more details, see Sections 4 and 5.

© 2016, Indian Institute of Technology Delhi

Chapter 2

Preliminaries

2.1 Definitions

In this section we describe our model formally. Our model extends that of multi-client searchable symmetric encryption [26] to the search and compute setting.

Let λ be the security parameter. A *database* $\mathsf{DB} = \{\mathsf{ind}_i, \mathsf{W}_i\}_{i=1}^{d}$ is a collection of d documents, each consisting of a document identifier $\mathsf{ind}_i \in \{0,1\}^\lambda$, a set of keywords $\mathsf{W}_i \subseteq \{0,1\}^*$ associated with that document, and a set of attributes on which computation may be performed, denoted by $\mathsf{CompAttr} = \{\mathsf{attr}_1, \ldots, \mathsf{attr}_k\}$. Here, for $j \in [k]$, let $\mathsf{attr}_j \in \{0,1\}^\lambda$. We assume for simplicity that all documents have the same set of computable attributes $\{\mathsf{attr}_j\}_{j\in[k]}$ [1]. We denote the value of a given attribute attr_j in a document ind_i as $\mathsf{ind}_i[\mathsf{attr}_j]$ or simply $\mathsf{ind}_i[j]$, and assume that all attribute values have the same support Dom. Let $\mathsf{W} = \cup_{i=1}^{d}\mathsf{W}_i$ and $m = |\mathsf{W}|$. Let $\mathcal{F} = \{f_i : \mathsf{Dom}^{\ell_i} \to \mathsf{Rng}\}_{\mathsf{i}\in\mathbb{N}}$ be a set of functions, where ℓ_i may be arbitrary (i.e. f_i has unbounded arity ℓ_i).

A *query* is a tuple $Q = (\phi(\bar{w}), \mathsf{attr}, f)$ where $\phi(\cdot)$ is a boolean formula on a tuple of keywords $\bar{w}$, $\mathsf{attr} \in \mathsf{CompAttr}$ is a computable attribute in the document and $f \in \mathcal{F}$ is a function. We denote by $\mathsf{DB}(\phi(\bar{w}))$ the set of identitifiers $\{\mathsf{ind}_i\}$ such that the formula $\phi(\bar{w})$ evaluates to true if we replace each word $w_j \in \bar{w}$ with true or false depending on whether $w_j \in \mathsf{W}_i$ or not. Intuitively, $\mathsf{DB}(\phi(\bar{w}))$ is the set of documents that satisfy the Boolean constraint $\phi(\bar{w})$. Let $S = \left\{ \mathsf{ind}_i[\mathsf{attr}] : \mathsf{ind}_i \in \mathsf{DB}(\phi(\bar{w})) \right\}$ be the multiset of values of the queried attribute attr in the matching documents $\mathsf{DB}(\phi(\bar{w}))$. We define the result of the query Q on the database DB as $\mathsf{DB}(Q) = \mathsf{DB}((\phi(\bar{w}), \mathsf{attr}, f)) = f(s_1, \ldots, s_{|\mathsf{DB}(\phi(\bar{w}))|})$, where $s_i \in S$.

Next, we define our notion of multi-client search and compute encryption.

Definition 1. *A Multi-client Search-Compute Encryption (*MC-SCE*) scheme for a functionality $\mathcal{F}$ is a tuple* $\Pi = (\mathit{EDBSetup}, \mathit{GenToken}, \mathit{SCompute})$ *where* *EDBSetup* *is a*

[1] This restriction is easily removable with some additional notation. We choose to impose it since it is a reasonable assumption for relational databases, which is the main setting we are interested in.

© 2016, Indian Institute of Technology Delhi

P.P.T. algorithm run by a data owner $\mathcal{D}$, GenToken is a protocol between the data owner $\mathcal{D}$ and a client $\mathcal{C}$ and SCompute is a protocol between a client $\mathcal{C}$ and a server $\mathcal{E}$ with the following syntax:

- *EDBSetup*$(\mathsf{DB}, \mathcal{F})$ *: The EDBSetup algorithm takes as input a database and a functionality $\mathcal{F}$ and outputs an encrypted database* EDB *and a master key* MSK.
- *GenToken: This is a 2 party protocol between a client $\mathcal{C}$ and data owner $\mathcal{D}$, where the data owner has input* MSK *and no output, whereas the client has input a query Q and output a token* SK_Q. *We will write* $\mathsf{SK}_Q \leftarrow$ *GenToken*(Q, MSK) *to denote the output of the client upon completion of the protocol. For notational convenience, we omit mention of $\mathcal{C}$ and $\mathcal{D}$ from the arguments of GenToken where these are implicit from context.*
- *SCompute : This is a 2 party protocol between a client $\mathcal{C}$ and a server $\mathcal{E}$ where the client $\mathcal{C}$ has input a token* SK_Q *and output a value* $\mathsf{Res} \in \mathsf{Rng} \cup \perp$, *and the server $\mathcal{E}$ has input* EDB *and no output. We will write* Res $\leftarrow$*SCompute*$(\mathsf{EDB}, \mathsf{SK}_Q)$ *to denote the output of the client upon completion of the protocol SCompute. For notational convenience, we omit mention of $\mathcal{C}$ and $\mathcal{E}$ from the arguments of SCompute where these are implicit from context.*

2.1.1 Correctness

An MC-SCE scheme Π = (EDBSetup, GenToken,SCompute) for functionality $\mathcal{F}$ is said to be computationally correct if for all efficient adversaries A, the following game outputs 1 with all but negligible probability.

- $A(1^\lambda)$ outputs DB and queries $Q_1, \ldots, Q_k$.
- Let $(\mathsf{EDB}, \mathsf{MSK}) \leftarrow \mathsf{EDBSetup}(\mathsf{DB}, \mathcal{F})$.
- For $i \in [k]$, let $\mathsf{SK}_i \leftarrow \mathsf{GenToken}(Q_i, \mathsf{MSK})$.
- For $i \in [k]$, let $\mathsf{Res}_i \leftarrow \mathsf{SCompute}(\mathsf{EDB}, \mathsf{SK}_i)$.
- Output 1 if for all $i \in [k]$, it holds that $\mathsf{Res}_i = \mathsf{DB}(Q_i)$.

© 2016, Indian Institute of Technology Delhi

2.1.2 Security

Security is formulated using a simulation style definition and is parametrised with a function $\mathcal{L}$, which represents the amount of information leaked to the adversary. In our model, the server is assumed to be honest-but-curious and the client is assumed to be malicious. We define the corresponding security games below.

Security against adversarial server We define the security game against an honest but curious server as follows.

Definition 2. *Let* $\Pi = (\mathit{EDBSetup}, \mathit{GenToken}, \mathit{SCompute})$ *be an* MC-SCE *scheme for functionality* $\mathcal{F}$*. Given a stateful algorithm* $\mathcal{L}$ *and algorithms* (A, Sim)*, define experiments* $\mathsf{Real}^{\Pi}_{A}(\lambda)$ *and* $\mathsf{Ideal}^{\Pi}_{A,\mathsf{Sim}}(\lambda)$ *as follows:*

$\mathsf{Real}^{\Pi}_{A}(\lambda)$*:* $A(1^{\lambda})$ *chooses* DB*. We let* $(\mathsf{EDB}, \mathsf{MSK}) \leftarrow \mathit{EDBSetup}(\mathsf{DB}, \mathcal{F})$ *and let* A *get* EDB*. A repeatedly chooses queries, say* Q_i*. For each such query* Q_i*, let* $\mathsf{SK}_i \leftarrow \mathit{GenToken}(Q_i, \mathsf{MSK})$ *and* tr_i *be the transcript of the protocol execution* $\mathit{SCompute}(\mathsf{SK}_i, \mathsf{EDB})$*. The transcript* tr_i *of* $\mathit{SCompute}$ *protocol is provided to* A*. Finally,* A *outputs a bit, which is used as the output of the game.*

$\mathsf{Ideal}^{\Pi}_{A,\mathsf{Sim}}(\lambda)$*:* $A(1^{\lambda})$ *chooses* DB*. Let* $\mathsf{EDB} \leftarrow \mathsf{Sim}(\mathcal{L}(\mathsf{DB}))$ *and let* A *get* EDB*. A repeatedly chooses queries, say* Q_i*. For each such query, let* $\mathsf{tr}_i \leftarrow \mathsf{Sim}(\mathcal{L}(\mathsf{DB}, Q_1, \ldots, Q_i))$ *and give the output* tr_i *to* A*. Finally,* A *outputs a bit, which is used as the output of the game.*

Π *is said to be* ***$\mathcal{L}$-secure against adversarial server*** *with adaptive attacks if there exists an efficient* S *for all efficient adversaries* A *s.t.*

$$|\Pr[\mathsf{Real}^{\Pi}_{A}(\lambda) = 1] - \Pr[\mathsf{Ideal}^{\Pi}_{A,S}(\lambda) = 1]| = \mathrm{negl}(\lambda)$$

Security against adversarial clients Security against adversarial clients captures the fact that even if multiple clients collude together, they should not be able to compute a function which none of them were authorized for. The following definition adapts the one given in [26] to fit our model.

Definition 3. *Let* $\Pi = (\mathit{EDBSetup}, \mathit{GenToken}, \mathit{SCompute})$ *be a* MC-SCE *scheme for functionality* $\mathcal{F}$*. Given algorithms* (A, Sim)*, define 2 experiments* $\mathsf{Real}^{\Pi}_{A}(1^{\lambda})$ *and* $\mathsf{Ideal}^{\Pi}_{A,\mathsf{Sim}}(1^{\lambda})$ *as follows:*

© 2016, Indian Institute of Technology Delhi

$\mathsf{Real}_A^{\Pi}(1^\lambda)$: *$A(1^\lambda)$ chooses* DB. *Let* $(\mathsf{EDB}, \mathsf{MSK}) \leftarrow \mathit{EDBSetup}(\mathsf{DB})$. *$A$ adaptively invokes the* $\mathit{GenToken}$ *protocol with the data owner $\mathcal{D}$, requesting queries $\{Q_i\}_{i\in[k]}$ to obtain* $\mathsf{SK}_i \leftarrow \mathit{GenToken}(Q_i)$ *and then invokes the* $\mathit{SCompute}$ *protocol with the server $\mathcal{E}$ to obtain* $\mathsf{Res}_i \leftarrow \mathit{SCompute}(\mathsf{SK}_i, \mathsf{EDB})$. *Note that A can behave arbitrarily in these interactions. Finally A outputs a bit b, which is set as the output of the game.*
$\mathsf{Ideal}_{A,\mathsf{Sim}}^{\Pi}(1^\lambda)$: *$A(1^\lambda)$ chooses* DB. *The experiment initializes* $(\mathsf{Sim}_0, \mathsf{Sim}_1, \mathsf{Sim}_2) \leftarrow \mathsf{Sim}$ *by running* $\mathsf{st} \leftarrow \mathsf{Sim}_0(1^\lambda)$. *Subsequently A can invoke instances of the protocol* $\mathit{GenToken}(Q_i)$, *in which case it interacts with* $\mathsf{Sim}_1(\mathsf{st})$ *and of the protocol* $\mathit{SCompute}$, *in which case it interacts with* $\mathsf{Sim}_2(\mathsf{st})$. $\mathsf{Sim}_1(\mathsf{st})$ *and* $\mathsf{Sim}_2(\mathsf{st})$ *can update the global state* st *of* Sim. *They can also issue queries Q to an oracle - $\mathcal{O}_{\mathcal{L}}$ which returns* $(\mathsf{DB}(Q), \mathcal{L}(\mathsf{DB}, Q))$, *where* $\mathsf{DB}(Q)$ *is the answer to the query and* $\mathcal{L}(\mathsf{DB}, Q)$ *is the leakage for that particular query. Finally, at some point, A outputs a bit b, which is set as the output of the game.*

Π *is said to be* ***$\mathcal{L}$-semantically secure against adversarial clients*** *if there exists an efficient simulator* Sim *s.t. for every efficient adversary A,*

$$|\Pr[\mathsf{Real}_A^{\Pi}(1^\lambda) = 1] - \Pr[\mathsf{Ideal}_{A,\mathsf{Sim}}^{\Pi}(1^\lambda)]| = \mathrm{negl}(\lambda)$$

A non-adaptive version of these definitions is a straightforward modification of the above games where the adversary provides all of the queries at the start of the game.

2.2 Building Blocks

In this section we define the building blocks required for our constructions.

Data Structures To begin, we define the data structures used by the OXT protocol [8] and its multi-client extension [26]. The OXT protocol uses two main data structures – TSet and XSet. For each keyword $w \in \mathsf{W}$, $\mathsf{TSet}(w)$ is an inverted index which points to the ind values of all the documents in the database that contain keyword w. Each $\mathsf{TSet}(w)$ is associated with a unique identifier stag, which is used to retrieve $\mathsf{TSet}(w)$. We define $\mathsf{TSet} = \{\mathsf{TSet}(w)\}_{w\in\mathsf{W}}$.

The TSet data structure is abstracted by the following API: the $\mathsf{TSetSetup}$ operation receives a collection T of lists $\mathsf{T}(w)$ for each $w \in \mathsf{W}$, builds a TSet data structure out of these lists and returns TSet and a key K_T. The $\mathsf{TSetGetTag}$ function is a deterministic

© 2016, Indian Institute of Technology Delhi

function that takes as input the key K_T and a keyword w and returns $\mathsf{stag}(w)$, while the $\mathsf{TSetRetrieve}$ operation takes as input TSet and $\mathsf{stag}(w)$ and returns the corresponding list $\mathsf{T}(w)$. Correctness requires that $\mathsf{TSetRetrieve}\big(\mathsf{TSet}, \mathsf{TSetGetTag}(K_T, w)\big)$ return $\mathsf{T}(w)$ with all but negligible probability while security posits that the TSet data structure does not reveal anything other than some quantified leakage.

Throughout our paper, we assume TSet to be instantiated with the specific implementation - Σ from [8]. We note that this only eases the way the leakage is presented in later parts of our proof and is not necessarily required for our protocol. The corresponding leakage incurred by Σ is just $\sum_w |\mathsf{T}[w]|$, which for our case will turn out to be $N = \sum_w |\mathsf{DB}(w)|$.

The data structure XSet stores encrypted version of the set $X = \{f(w, \mathsf{ind}) \ : w \in \mathsf{W}, \mathsf{ind} \in \mathsf{DB}(w)\}$, namely all the keyword, document-index pairs such that the document contains the keyword. Thus, checking whether a keyword occurs in a given document can now by rephrased as checking containment of $f(w, \mathsf{ind}) \in X$, which is accomplished efficiently and securely using XSet.

Yao's Garbled Circuits Our protocol for computing general functions on filtered data makes use of Yao's garbled circuits. Yao's garbled circuits have been studied extensively from both the theoretical [44, 2] and practical [31, 24] perspectives, and a series of recent works have optimized its performance [29, 28, 30, 45, 16, 25].

Adversary Model We prove security against an honest but curious server and malicious clients. The server can be strengthened to malicious using techniques in [14] but we do not explore this in the present work. We crucially assume that the server and clients do not collude with each other. We note that this assumption is standard in the setting of SSE [8, 26].

Homomorphic Encryption Our protocols make use of homomorphic schemes supporting simple operations. We make use of additive homomorphic schemes [15] for computing the mean, and schemes for computing 2-DNF formulae [5] for computing second order statistics such as variance, correlation and such others.

© *2016, Indian Institute of Technology Delhi*

DDH Assumption Let $G = G_\tau$ be a prime order cyclic group of order $p = p(\tau)$, generated by g. We say that the decision Diffie-Hellman (DDH) assumption holds in G if $\mathsf{Adv}^{\mathsf{DDH}}_{G,A}(\tau)$ is negligible for all efficient adversaries A, where

$$\mathsf{Adv}^{\mathsf{DDH}}_{G,A}(\tau) = |\Pr\left(A(g, g^a, g^b, g^{ab}) = 1\right) - \Pr\left(A(g, g^a, g^b, g^c) = 1\right)|$$

where probability is over the randomness of A, and uniformly chosen $a, b, c \in \mathbb{Z}_p^*$.

© *2016, Indian Institute of Technology Delhi*

Chapter 3

Multi-Client Searchable Symmetric Encryption

Our first construction builds upon the multi-client SSE scheme of Jarecki et al. [26], which we recap here. In multi-client SSE, we have a data owner $\mathcal{D}$ who processes a large database DB to produce an encrypted database EDB and a master key MSK, of which it sends EDB to the server $\mathcal{E}$. The model allows for multiple clients $\mathcal{C}_i$, who may request $\mathcal{D}$ for keys corresponding to certain queries Q_j, which $\mathcal{D}$ constructs using its MSK. It provides the client with a key corresponding to Q_j, along with a signature so that $\mathcal{E}$ may verify that $\mathcal{C}$ is authorized to make this query. The client transforms the received key, say SK_j, into search tokens required by the OXT protocol [8]. By choosing a (singly) homomorphic signature in the previous step, the client may also transform the received signature on the key into one for the tokens. It sends the tokens and corresponding signatures to the server, who verifies the signature, and executes the OXT search protocol. It obtains encrypted ind values, which it returns to the client, who is in possession of the decryption key.

Next, we provide a (simplified) overview of the protocol, which builds on OXT. The protocol is designed to support arbitrary Boolean queries but for ease of exposition here, we will consider the special case where the client makes conjunctive queries of the form $w_1 \wedge w_2 \wedge \ldots \wedge w_n$. The least fequent word, w_1 (say) is known as s-word, while the remaining words $w_2, \ldots, w_n$ are called x-words. It is assumed that the data owner maintains information that lets it compute the s-word for any query of a client.

The setup phase of the $\mathsf{MC\text{-}SSE}$ protocol, as in OXT, constructs the TSet and XSet as follows. First, it picks PRFs F_τ and F_p with the appropriate ranges, and their keys K_S and $\{K_X, K_I\}$ respectively. Intuitively, the keys K_S and K_X are used to generate pseudorandom handles of keywords $w \in \mathsf{W}$, denoted by $\mathsf{strap} = F_\tau(K_S, w)$ and $\mathsf{xtrap} = g^{F_p(K_X,w)}$ respectively. The key K_I is used to generate a pseudorandom handle of document indices $\mathsf{ind} \in \mathsf{DB}$ as $\mathsf{xind} = F_p(K_I, \mathsf{ind})$. K_T is a key used to compute stag values as required by the TSet implementation. Next, the protocol computes $\mathsf{xtag} = g^{F_p(K_X,w)\cdot \mathsf{xind}}$

© 2016, Indian Institute of Technology Delhi

for all "valid" (w, ind) pairs, that is pairs such that the document ind contains the keyword w, and adds xtag to XSet.

To populate $\mathsf{TSet}(w)$ for $w \in \mathsf{W}$ so as to enable efficient response to queries such as $w_1 \wedge w_2 \wedge \ldots \wedge w_n$, the protocol does the following. If w_i has T_{w_i} matching documents, then $\mathsf{TSet}(w_i)$ contains T_{w_i} entries, each containing the relevant document index (encrypted) as well as "blinded" xind values to enable the server to check whether the remaining keywords also match the given document. More formally, for each list $\mathsf{TSet}(w)$ for $w \in \mathsf{W}$, the data owner generates a fresh pair of keys (K_z, K_e) using $\mathsf{strap}(w)$ and encrypts the T_w document indexes as $e = \mathsf{Enc}(K_e, \mathsf{ind})$. Next, for $c \in [T_w]$, it computes a blinding of the T_w index representatives $\mathsf{xind}_1, \ldots, \mathsf{xind}_{T_w}$ as follows: set $z_c = F_p(K_z, c)$ and $y = \mathsf{xind} \cdot {z_c}^{-1}$. Store (e, y) in $\mathsf{TSet}(w)$. Note that (e, y) are constructed using keys that are specific to s-word w. It chooses a key K_M for $\mathsf{AuthEnc}$ - an authenticated encryption scheme, and this key is shared with the server. The keys $(K_S, K_X, K_I, K_T, K_M)$ are retained by $\mathcal{D}$ as the master secret.

When the client requests a key for the query $w_1 \wedge w_2 \wedge \ldots \wedge w_n$, the data owner computes the s-word, computes stag using K_T, strap using K_S as well as $\mathsf{xtrap}_2, \ldots, \mathsf{xtrap}_n$ corresponding to $w_2, \ldots, w_n$ using K_X. Next, it blinds the xtrap values as $\mathsf{bxtrap}_i = g^{F_p(K_X, w_i) \cdot \rho_i}$ where ρ_i are chosen randomly from $\mathbb{Z}_p^*$. Then it creates an authenticated encryption $\mathsf{env} \leftarrow \mathsf{AuthEnc}(K_M, (\mathsf{stag}, \rho_2, \ldots, \rho_n))$ and returns $\mathsf{SK} = (\mathsf{env}, \mathsf{strap}, \mathsf{bxtrap}_2, \ldots, \mathsf{bxtoken}_n)$ to the client.

The client uses SK to construct search tokens as follows. Use strap to compute K_z and for $c \in [T_w]$, compute $z_c = F_p(K_z, c)$. Now, it sets $\mathsf{bxtoken}[c, i] = \mathsf{bxtrap}_i^{z_c}$ and sends env along with all the $\mathsf{bxtoken}[c, i]$ to the server. The server verifies the authenticity of env using K_M, and decrypts it to find stag and $\rho_2, \ldots, \rho_n$. It uses stag to obtain $\mathsf{TSet}(w_1)$ and for $i \in \{2, \ldots, n\}$, it checks if $\mathsf{bxtoken}[c, i]^{y/\rho_i} \in \mathsf{XSet}$. If so, it retrieves the corresponding encrypted document index e and sends it to the client.

© 2016, Indian Institute of Technology Delhi

Chapter 4

Computing Filtered Average

4.1 Protocol

In this section, we describe a multi-client search and compute scheme, denoted by FltAvg, for the functionality of filtered average defined below. Recall that a client makes a query of the form $Q = (\phi(\bar{w}), \mathsf{attr}, f)$ where $\phi(\cdot)$ is a boolean formula on a tuple of keywords $\bar{w}$ [1], $\mathsf{attr} \in \mathsf{CompAttr}$ is a computable attribute in the document and f is a function. Let S be the set of attribute values obtained from the documents matching the search, namely $S = \{ \mathsf{ind}_i[\mathsf{attr}] : \mathsf{ind}_i \in \mathsf{DB}(\phi(\bar{w})) \}$. We denote the elements of S as s_i for $i \in |S|$. Then filtered average is defined as $f(S) = \frac{1}{|S|} \sum_{i \in |S|} s_i$.

Our construction begins with the $\mathsf{MC\text{-}SSE}$ protocol of Jarecki et al. and augments it with a symmetric key additively homomorphic encryption scheme AHE to enable computation of the average of some computable attribute attr_j for $j \in [k]$. For $w \in \mathsf{W}$, the setup algorithm now precomputes the sum of all the values $\mathsf{ind}_c[\mathsf{attr}_j]$ for $c \in [T_w]$ and $j \in [k]$, and encrypts this sum using AHE under a key K_h which is derived from strap. This tuple of encrypted sums - $\mathsf{sct}_1, \ldots, \mathsf{sct}_k$ [2] are added to the head of the list. For $c \in [T_w]$, i.e. each tuple in the list, the blinded xind value y is computed exactly as before, and additionally, AHE encryptions of $-\mathsf{ind}_c[\mathsf{attr}_j]$ are computed under a fresh key $K_h^j = F_\tau(K_h, j)$. The protocol computes documents matching the filtering criteria $\bar{w}$ as before by iterating over the list and checking if each document matches the remaining keywords. However now, if a given document does *not* match one of the x-words, then we add the corresponding encryption of $-\mathsf{ind}_c[\mathsf{attr}_j]$ to the encryption of $\sum_{c \in [T_w]} \mathsf{ind}_c[\mathsf{attr}_j]$ so that the value of the attribute in the non-matching document is subtracted from the sum. The final encryption is returned to $\mathcal{C}$ who is in possession of the decryption key K_h and can recover the value.

However, while this modification achieves the functionality we are after, it also introduces vulnerabilities. Note that in the $\mathsf{MC\text{-}SSE}$ protocol, if a malicious client produces certain

[1] For simplicity, we defer handling of range queries to later part

[2] Along with an extra element 0 to make the size of all tuples equal, as required by the TSet implementation of [8].

© 2016, Indian Institute of Technology Delhi

corrupt values for $\mathsf{bxtoken}[c, i]$, for example by not computing z_c honestly, then this does not offer it any advantage. To see this, say that some token $\mathsf{bxtoken}^*$ is generated maliciously. Then, the corresponding test $(\mathsf{bxtoken}^*)^{y/\rho_i} \in \mathsf{XSet}$ returns false, and the client does not receive the encrypted index corresponding to that particular document. Since maliciously generated tokens produce a result that is a subset of the honest result, the protocol need not safeguard against such attacks. However, in the case of the filtered average functionality, this is no longer true. If the client produces a certain token maliciously, then the XSet test will fail, even when actually it shouldn't, and the corresponding attribute value will get subtracted from the sum. By mixing and matching honest and corrupt tokens, the client can learn individual attribute values for documents, which is leakage far in excess of functionality. For instance, by using honest tokens $\mathsf{bxtoken}_1, \ldots, \mathsf{bxtoken}_{T_w}$, the client learns some value, say val. Now, if $\mathsf{bxtoken}_1^{y/\rho}$ results in an XSet hit, then by executing a query with tokens $\mathsf{bxtoken}_1^*, \mathsf{bxtoken}_2 \ldots, \mathsf{bxtoken}_{T_w}$ where $\mathsf{bxtoken}_1^*$ is corrupt, the client may learn the exact value of the attribute in the first document in the list.

A natural solution is to force the client to provide a proof that he performed the computation correctly. However, despite substantial improvements in the efficiency of proofs (see [22] and references therein), computing a proof may be too expensive for a resource constrained client. A different way to handle this issue relies on the fact that the server can verify whether the tokens were generated honestly at the cost of increasing its storage. To see this, note that $\mathsf{bxtoken}^{y/\rho} \notin \mathsf{XSet}$, implies that the keyword corresponding to the token does not appear in the document corresponding to y. However, even in this case, an honestly generated bxtoken must always yield $\mathsf{bxtoken}^{y/\rho} = g^{F_p(K_X, w)\cdot \mathsf{xind}}$ for some (w, ind) pair. The server can store the set $\mathsf{XSetAll} = \{g^{F_p(K_X,w)\cdot \mathsf{xind}}\}$ for all possible (w, ind) pairs and check whether $\mathsf{bxtoken}^{y/\rho} \in \mathsf{XSetAll}$ for the received token and if not, it returns $\perp$. Our protocol is described in the following figures.

4.1.1 Correctness

We argue that our protocol is correct according to definition discussed previously.

Theorem 4. *The* MC-SCE *scheme described above is computationally correct, assuming that* TSet *has a computationally correct implementation, F_τ and F_p are secure PRFs and that* AHE *is a correct symmetric additively homomorphic encryption scheme.*

© 2016, Indian Institute of Technology Delhi

$\underline{\mathsf{EDBSetup}(\mathsf{DB}, \mathcal{F})}$: Let G be a cyclic group of prime order p, generated by an element g. Let F_τ and F_p be 2 PRFs with range in $\{0,1\}^\tau$ and $\mathbb{Z}_p^*$ respectively, where τ is the security parameter. Pick keys K_S for PRF F_τ and K_X, K_I for PRF F_p.

- Initialize $\mathsf{XSet}, \mathsf{XSetAll}$ to an empty set and T to an empty array indexed by $w \in \mathsf{W}$; Parse $(\mathsf{ind}_i, \mathsf{W}_i)_{i=1}^d \leftarrow \mathsf{DB}$.
- For $w \in \mathsf{W}$, build $\mathsf{XSet}, \mathsf{XSetAll}$ and $\mathsf{T}[w]$ as follows:
  - Initialize t to an empty list.
  - $\mathsf{strap} \leftarrow F_\tau(K_S, w)$ and $(K_z,\ K_h) \leftarrow (F_\tau(\mathsf{strap}, 1),\ F_\tau(\mathsf{strap}, 2))$.
  - For $1 \le i \le d$, add $g^{F_p(K_X, w) \cdot F_p(K_I, \mathsf{ind}_i)}$ to $\mathsf{XSetAll}$.
  - Initialize $(\mathsf{sum}_1, \mathsf{sum}_2 \ldots \mathsf{sum}_k)$ with zeros. Initialize $c \leftarrow 0$ and for all $\mathsf{ind} \in \mathsf{DB}(w)$ in random order, do the following:
    - Let $\mathsf{xind} \leftarrow F_p(K_I, \mathsf{ind})$. Let $c = c + 1$, $z_c \leftarrow F_p(K_z, c)$, and $y \leftarrow \mathsf{xind} \cdot z_c^{-1}$.
    - Set $\mathsf{xtag} \leftarrow g^{F_p(K_X, w) \cdot \mathsf{xind}}$ and add xtag to XSet.
    - For $j \in [k]$, let $\mathsf{val} \leftarrow \mathsf{ind}[\mathsf{attr}_j]$, $K_h^j \leftarrow F_\tau(K_h, j)$, $\mathsf{ct}_j \leftarrow \mathsf{AHE.Enc}(K_h^j, -\mathsf{val})$ and $\mathsf{sum}_j = \mathsf{sum}_j + \mathsf{val}$.
    - Let $\mathsf{ct} \leftarrow (\mathsf{ct}_1, \mathsf{ct}_2 \ldots \mathsf{ct}_k)$ and append (y, ct) to t.
  - for $j \in [k]$, let $K_h^j \leftarrow F_\tau(K_h, j)$, $\mathsf{sct}_j \leftarrow \mathsf{AHE.Enc}(K_h^j, \mathsf{sum}_j)$.
  - Append $(0, \mathsf{sct}_1, \mathsf{sct}_2 \ldots \mathsf{sct}_k)$ to head of t.
  - $\mathsf{T}[w] = t$.
- Let $(\mathsf{TSet}, K_T) \leftarrow \mathsf{TSetSetup}(\mathsf{T})$ and $K_M \leftarrow \mathsf{AuthEnc.KeyGen}(1^\tau)$.
- Output $\mathsf{EDB} = (\mathsf{TSet}, \mathsf{XSet}, K_M)$ and $\mathsf{MSK} = (K_S, K_X, K_I, K_T, K_M)$.

Correctness of our protocol follows from the correctness of the MC-SSE scheme of Jarecki et al. and correctness of our additively homomorphic encryption scheme. Also, since the TSet implementation is computationally correct, the only additional errors that can occur in the correctnesss game, can be due to either collisions in the PRF outputs or two keyword-identifier pairs mapping to the same xtag using $F_p(K_I, \cdot)$ and $F_p(K_X, \cdot)$. But since the PRF outputs are indistinguishable from random, the probability of this happening is negligible.

Next, we provide a proof that our protocol FltAvg is secure. To begin, we show security against honest but curious servers.

4.1.2 Security against adversarial server

Security follows the same high level idea as [8]. However, there are some important differences. While the leakage pattern captured in our protocol is still the same as in [8], the server's leakage profile in [8] is defined in terms of the indices of the relevant

© 2016, Indian Institute of Technology Delhi

$\underline{\mathsf{GenToken}(Q, \mathsf{EDB})}$: Parse Q as $(\phi(\bar{w}), \mathsf{attr}, f)$ and MSK as $(K_S, K_X, K_I, K_T, K_M)$. Assume $\phi(\bar{w}) = w_1 \wedge w_2 \ldots \wedge w_n$. WLOG, let w_1 be the s-term, i.e. the term with the least frequency, chosen by $\mathcal{D}$. Also, let attr be the j_0^{th} computable attribute.

- $\mathcal{D}$ computes $\mathsf{stag} \leftarrow \mathsf{TSetGetTag}(K_T, w_1)$, $\mathsf{strap} \leftarrow F_\tau(K_S, w_1)$.
- For $i = 2$ to n, do : $\rho_i \xleftarrow{\$} \mathbb{Z}_p^*$ and set $\mathsf{bxtrap}_i \leftarrow g^{F_p(K_X, w_i)\cdot\rho_i}$.
- $\mathsf{env} \leftarrow \mathsf{AuthEnc.Enc}(K_M, (j_0, \mathsf{stag}, \rho_2, \ldots, \rho_n))$.
- Output $\mathsf{SK}_Q = (j_0, \mathsf{env}, \mathsf{strap}, \mathsf{bxtrap}_2, \ldots, \mathsf{bxtrap}_n)$.

$\underline{\mathsf{SCompute}(\mathsf{EDB}, \mathsf{SK}_Q)}$: Client $\mathcal{C}$ on input the token $\mathsf{SK}_Q = (j_0, \mathsf{env}, \mathsf{strap}, \mathsf{bxtrap}_2, \ldots, \mathsf{bxtrap}_n)$, does the following:

- $(K_z, K_h) \leftarrow (F_\tau(\mathsf{strap}, 1), F_\tau(\mathsf{strap}, 2))$.
- Send to the server $\mathcal{E}$ the message $(\mathsf{env}, \mathsf{bxtoken}[1], \mathsf{bxtoken}[2] \ldots)$, where $\mathsf{bxtoken}[.]$ is computed as follows :
  - For $c = 1, 2 \ldots$, until $\mathcal{E}$ sends stop, do:
    - $z_c \leftarrow F_p(K_z, c)$ and $\mathsf{bxtoken}[c, i] \leftarrow \mathsf{bxtrap}_i^{z_c} \;\; \forall\, 2 \le i \le n$.
    - Set $\mathsf{bxtoken}[c] \leftarrow (\mathsf{bxtoken}[c, 2] \ldots \mathsf{bxtoken}[c, n])$.
- $K_h^{j_0} \leftarrow F_\tau(K_h, j_0)$.
- When the server sends stop, along with it, it sends (temp, n). Output $\mathsf{AHE.Dec}(K_h^{j_0}, \mathsf{temp})/n$.

Server $\mathcal{E}$ with input $\mathsf{EDB} = (\mathsf{TSet}, \mathsf{XSet}, K_M)$ responds as follows:

- $(j_0, \mathsf{stag}, \rho_2, \ldots \rho_n) \leftarrow \mathsf{AuthEnc.Dec}(K_M, \mathsf{env})$.
- Get $t \leftarrow \mathsf{TSetRetrieve}(\mathsf{TSet}, \mathsf{stag})$.
- Let $(0, \mathsf{sct}_1, \mathsf{sct}_2 \ldots \mathsf{sct}_k)$ be the first tuple of t. $\mathsf{temp} \leftarrow \mathsf{sct}_{j_0}$ and $\mathsf{cnt} \leftarrow 0$.
- For $c = 1, 2, \ldots, |t| - 1$, do:
  - Let the $c + 1^{th}$ tuple of t be $(y, \mathsf{ct}_1, \ldots, \mathsf{ct}_k)$ and set $flag$ to $False$
  - For $\mathsf{bxtoken}[c]$ as received from $\mathcal{C}$, do for $i \in [2 \ldots n]$:
    - If $\mathsf{bxtoken}[c, i]^{y/\rho_i} \notin \mathsf{XSetAll}$, then return $\perp$.
    - If $\mathsf{bxtoken}[c, i]^{y/\rho_i} \notin \mathsf{XSet}$, then set $flag$ to $True$.
  - If $flag == True$, then $\mathsf{temp} = \mathsf{temp} + \mathsf{ct}_{j_0}$ and $\mathsf{cnt} = \mathsf{cnt} + 1$.
- When the last tuple in t is reached, send $(\mathsf{stop}, (\ \mathsf{temp}, |t| - 1 - \mathsf{cnt}\ ))$ to $\mathcal{C}$.

documents, whereas, this cannot be the case with our protocol, as the functionality now does not allow the server to learn the indices of the relevant documents.

For ease of exposition, the following proof outline restricts to non-adaptive security, and to 2 keyword conjunctive queries of the form $w_1 \wedge w_2$. Additionally, we assume there is only one attribute in $\mathsf{CompAttr}$, which is attr. The proof can be readily adapted to the general case.

The sequence of queries - $\mathbf{q}$, consists of Q queries with $\mathbf{q}[i] = ((\mathbf{s}[i] \wedge \mathbf{x}[i]), \mathsf{attr}, f)$, where $\mathsf{attr} \in \mathsf{CompAttr}$ and $f \in \mathsf{FltAvg}$. To ease notation, we henceforth represent each query $\mathbf{q}[i]$ as a pair of the s-word and x-word: $(\mathbf{s}[i], \mathbf{x}[i])$ since the other 2 components of

© *2016, Indian Institute of Technology Delhi*

the query are fixed. For simplicity, we also assume, as in [8], that there are no repeating queries. The leakage from the protocol is quantified as $\mathcal{L}(\mathsf{DB}, \mathbf{q}) = (\mathsf{N}, \bar{\mathsf{s}}, \mathsf{SP}, \mathsf{RP}, \mathsf{IP})$ where $\mathsf{N}, \bar{\mathsf{s}}, \mathsf{SP}$ are as defined in [8], whereas RP, IP, albeit capturing the same pattern as in [8], are defined differently. For completeness, we describe the full leakage here.

Let $T_w = |\mathsf{DB}(w)|$ and let $I(w)$ denote the list of indices corresponding to keyword w permuted in random order. We denote by $I(w)[c]$ the c^{th} entry of $I(w)$. $\mathcal{L}(\mathsf{DB}, \mathbf{q})$ is then defined as consisting of the following:

- N is equal to $\sum_{i=1}^{d} |\mathsf{W}_i|$, the total number of times keywords appear in documents, which is equal to the size of XSet leaked to adversary.
- $\bar{\mathsf{s}}$ is the equality pattern, which represents the repetition pattern of s-words. For example, if there were 5 queries with s-words being $(s_1, s_2, s_1, s_3, s_2)$, then $\bar{\mathsf{s}} = (1, 2, 1, 3, 2)$. Formally, to define $\bar{\mathsf{s}}[i]$, find the least $1 \leq j \leq i$ s.t. $\mathbf{s}[j] = \mathbf{s}[i]$ and set $\bar{\mathsf{s}}[j] = |\{\mathbf{s}[1], \mathbf{s}[2] \ldots \mathbf{s}[j]\}|$, i.e. the number of unique s-words occuring till query j.
- SP is the size pattern, which represents the number of documents matching s-word of each query. Formally, define $\mathsf{SP}[\mathsf{i}] = |\mathsf{DB}(\mathbf{s}[\mathsf{i}])|$.
- RP is the results pattern, which represents the positions of the documents in $I(\mathbf{s}[i])$ which also contain $\mathbf{x}[i]$. Formally, it is equal to $\{R_1, R_2 \ldots R_Q\}$, where R_i is defined as $\{c : I(\mathbf{s}[i])[c] \in \mathsf{DB}(\mathbf{x}[i])\}$. Thus, for query i, if $\mathbf{s}[i] = w$, then R_i contains the positions in the permutation of list $\mathsf{T}[w]$ such that the associated documents contain $\mathbf{x}[i]$ in addition to $\mathbf{s}[i] = w$.
- IP is the conditional intersection pattern and is defined as $\{S_1, S_2 \ldots S_Q\}$, where $S_i = \{S_i^1, S_i^2 \ldots S_i^{T_{\mathbf{s}[i]}}\}$ and $S_i^j = \{(k, c) \mid (1 \leq k < i) \wedge (\mathbf{x}[k] = \mathbf{x}[i]) \wedge (I(\mathbf{s}[k])[c] = I(\mathbf{s}[i])[j])\}$. The intersection pattern captures the information leaked when distinct queries with the same x-words and a common document index in the corresponding s-word document list, result in the same xtag element being checked for containment in XSet. For instance, query 5 may access a list whose 4^{th} tuple contains a pointer to document ind^*, and query 8, with same x-word, may access a different list whose 6^{th} tuple contains a pointer to the same document ind^*. Then, S_8^6 contains $(5, 4)$.

Security is captured in the following theorem.

© *2016, Indian Institute of Technology Delhi*

Theorem 5. *Let $\mathcal{L}$ be the leakage function defined as above and suppose that* TSet *uses the implementation Σ from [8]. Then the scheme* FltAvg *is an $\mathcal{L}$-semantically secure* MC-SCE *scheme, against non-adaptive attacks where all the queries are 2-conjunctions, assuming that the* DDH *assumption holds in G, F_τ and F_p are secure* PRF*s,* AHE *is a symmetric additively homomorphic IND-CPA secure encryption scheme and that the conditions corresponding to Σ, as stated in [8] hold.*

The proof follows the high level strategy of [8]. To construct $\mathsf{EDB} = (\mathsf{TSet}, \mathsf{XSet})$, the simulator does the following. To construct $\mathsf{T}[w]$, for each queried and unique s-word w (found using $\bar{\mathsf{s}}$), it chooses ct as encryptions of 0^λ and samples y randomly for each document in the list. Then, it invokes the TSet simulator to obtain TSet and also stag for all queries. For constructing XSet, it picks random elements in G. Recall that $\mathsf{xtag} = g^{F_p(K_X,w)\cdot \mathsf{xind}}$ and $y = \mathsf{xind} \cdot z_c^{-1}$, where xind, z_c are computed by PRFs. A natural proof strategy is to replace z_c, $F_p(K_X, w)$, xind by random values and then invoke the DDH assumption to choose xtag randomly. However, as noted by [8, Proof of Theorem 5], there is a subtlety here since xind is also used in computing y, and z_c is additionally used in token generation. [8] deals with this issue by a careful rearrangement of computation, which allows them to argue indistinguishability – the same strategy can be applied in our case.

Additionally, the simulator must simulate the transcipts of the SCompute protocol. To do this, it must generate tokens that are indistinguishable from an honest client's tokens using the leakage, namely, the result pattern RP and intersection pattern IP. Recall that for a 2 conjunction, the client's query contains the encryption env and tokens $\mathsf{bxtoken}_1, \ldots, \mathsf{bxtoken}_{T_w}$. Here, env contains attribute index j_0, the tag stag for the query's s-word, and a randomly chosen element ρ. The tag stag may be generated using the TSet simulator and the remaining values can be computed honestly. It remains to generate indistinguishable tokens $\{\mathsf{bxtoken}_c\}_{c\in[T_w]}$.

To do this, the simulator constructs a table XSetCheck indexed by query number k and a counter c over T_w. The entry (k, c) contains the value xtag that is checked for XSet containment by that (query, tuple) pair. Now, if the token bxtoken corresponding to (query k, tuple c) results in an XSet hit, then the simulator learns this via the result pattern RP. Via the intersection pattern IP, the simulator may also learn whether this entry in XSet has been accessed before. If the entry has not been accessed before but results in a XSet hit, the simulator randomly chooses a new element xtag of XSet, marks

© *2016, Indian Institute of Technology Delhi*

it "used" and stores this value in the table XSetCheck at index (k, c). If the entry has not been accessed before, but results in a XSet miss, the simulator just chooses a random element from $\mathbb{Z}_p^*$. It computes the corresponding token as $\mathsf{bxtoken} = \mathsf{xtag}^{\rho/y}$ to mimic the real world. Finally, if the entry has been accessed before, then the simulator uses IP to compute the previous (query, tuple) pair that accessed it, and retrieves the corresponding xtag from XSetCheck. It generates the new $\mathsf{bxtoken} = \mathsf{xtag}^{\rho'/y'}$ with a fresh ρ' and y' which are specified in env and TSet respectively. This enables the simulator to generate tokens to mimic the real world.

4.1.3 Security against malicious clients

Next, we proceed to show that our protocol is secure against adversarial clients. The leakage function is $\mathcal{L}(\mathsf{DB}, \bar{w}) = (|\mathsf{DB}(w_1)|, \mathsf{cnt})$, where $\bar{w} = (w_1, w_2, \ldots, w_n)$, w_1 WLOG is the assumed s-word and cnt is the number of documents matching the query, which is given to client in the clear for average computation. Recall that each query, $Q = (\phi(\bar{w}), \mathsf{attr}, f)$ may be represented by $\bar{w}$, since we assume for ease of exposition that attr, f are fixed and the boolean function is a conjunction.

Security against adversarial clients is captured in the following theorem.

Theorem 6. *Let $\mathcal{L}$ be the leakage function as defined above. The scheme* FltAvg *is an $\mathcal{L}$-semantically secure* MC-SCE *scheme, against malicious clients with adaptive queries assuming that the* DDH *assumption holds in G, F_τ, F_p are secure* PRF*s,* AHE *is a symmetric additively homomorphic IND-CPA secure encryption scheme,* TSet *has a computationally correct implementation and* AuthEnc *is an IND-CPA and Strongly-UF-CMA authenticated encryption scheme.*

The proof is similar to the proof of security against malicious clients described in [26]. The main difference is the attack described in Section 4, which is that by constructing some bxtoken maliciously, the client can arrange that the value $\mathsf{bxtoken}^{y/\rho}$ does not belong to XSet, even when it should. This ensures that the server does not include the corresponding attribute value in the overall answer. By making successive queries that contain honest and corrupt tokens respectively in the same position, the client may learn exact attribute values in individual documents, instead of aggregate values.

© 2016, Indian Institute of Technology Delhi

To get around this, the protocol includes the check $\mathsf{bxtoken}^{y/\rho} \stackrel{?}{\in} \mathsf{XSetAll}$, which the simulator mimics as follows. It checks the validity of received tokens by checking if $\mathsf{bxtoken}[c,i] = g^{F_p(K_X,w_i)\cdot\rho_i\cdot z_c}$, where the terms on the right hand side are generated honestly. If this does not hold, the simulator returns $\perp$. This test is equivalent to the real world check of whether $\mathsf{bxtoken}[c,i]^{y/\rho_i} \in \mathsf{XSetAll}$. If the check succeeds, the simulator concludes that the query is honest and queries the functionality oracle for the answer. It encrypts the answer using AHE with the key generated from the chosen s-word, and sends this to A. Detailed proof of the same can be found in the appendix.

We presented our protocols for the case of conjunctions and proof of security against an adversarial server in the non-adaptive setting. We note that these limitations can be overcome as in [26] and [8]. Any query that is in the searchable normal form (SNF), i.e. queries of the type - $w_1 \wedge \phi(w_2, w_3 \ldots w_n)$, can be supported by choosing the s-word as w_1 and having the server compute whether some $\mathsf{ind} \in \mathsf{DB}(w_1)$ is part of the final answer by calculating $\nu_i \leftarrow$ boolean value of predicate $\mathsf{bxtoken}[c,i]^{y/\rho_i} \stackrel{?}{\in} \mathsf{XSet}$ for $2 \leq i \leq n$ and checking if $\phi(\nu_2, \nu_3 \ldots \nu_n) = \mathsf{True}$. Note that this modification does not change the search complexity or leakage (in addition to learning ϕ) compared to the case when the same keywords were used in pure conjunctive form. Also, queries of the type $w_1 \vee w_2 \vee \ldots w_n$ can be efficiently supported as in [8] by considering each w_i as a separate s-word and having the data owner provide an stag for each of the w_i. The client then runs the remaining protocol as before, but now switching to the next stag value on every stop received from the server. Finally, we note that we may also achieve privacy of the filtering predicate exactly as in [26].

4.2 Support for Range Queries

Here we describe how our basic protocol can be extended to deal with range filtering on attributes. For example, with this extension, our protocol will support queries of the type - "Compute average of attribute Income for all the citizens that have Age in range $[20 \ldots 40]$ and $\mathsf{State} = \mathsf{Timbaktu}$ and $\mathsf{Gender} = \mathsf{Male}$". In order to support such queries, we make use of the "Logarithmic-BRC/URC" schemes to support range queries, as proposed by Loannis et. al. in [13]. We proceed to describe the main ideas.

For simplicity, let us assume that there is only one attribute, say attr_R, on which range queries are performed, and that this attribute takes values from domain Dom. Also,

© 2016, Indian Institute of Technology Delhi

for simplicity we restrict attention to conjunctive queries as in the above example. Our protocol can easily be extended to include general queries involving range filtering on multiple attributes.

So far, we have assumed that keywords are represented by attribute-value pairs $(\mathsf{attr}, \mathsf{val})$ where val is a single value. But now to support ranges, we extend the meaning of val to include a range, so that a keyword can now correspond to a range for attribute attr_R. We define $\mathsf{Range}(\mathsf{w})$ to be the range associated with keyword w if any, and NULL otherwise.

During $\mathsf{EDBSetup}$, as in the Log-BRC/URC scheme of [13], the data owner builds a binary tree over the domain Dom. Each node in the tree is then assigned a unique keyword, which we refer to as rword. These rwords are similar to the keywords in the original protocol, except that each of them now correspond to a range of values in the domain Dom, instead of just a single value. Now, for each $(\mathsf{ind}_i, \mathsf{W}_i) \in \mathsf{DB}$, let val be the value of the attribute attr_R in the corresponding document. Append to W_i the rword for the nodes on the path from the root of the binary tree to val. This results in the original DB getting changed to a new database, which we call DB'. Now, the data owner builds EDB in a similar way as before by running $\mathsf{EDBSetup}$ of the original protocol over DB'.

In the $\mathsf{GenToken}$ procedure, the data owner on receiving the query $Q = ((w_1 \wedge w_2 \wedge \ldots w_n), \mathsf{attr}, f)$, checks if $\mathsf{Range}(w_i) \neq \mathsf{NULL}$ for some w_i. If not, that means that there are no range queries involved, and the protocol continues to work as before. If on the other hand, say WLOG that $\mathsf{Range}(w_1) \neq \mathsf{NULL}$, then the data owner modifies the query $(w_1 \wedge w_2 \wedge \ldots w_n)$ to incorporate the range filtering. It finds the nodes s.t. their associated ranges cover the entire range $\mathsf{Range}(w_1)$; this set is known as the "range cover" of $\mathsf{Range}(w_1)$. This step is performed using either Best Range Cover(BRC)/Uniform Range Cover (URC) in the binary tree created in $\mathsf{EDBSetup}$. For more details on BRC/URC, please refer to [13]. Let $\{\mathsf{rkw}_1, \mathsf{rkw}_2 \ldots \mathsf{rkw}_r\}$ be the corresponding keywords associated with these nodes. Then, it changes the query $(w_1 \wedge w_2 \wedge \ldots w_n)$ to $(\mathsf{rkw}_1 \vee \mathsf{rkw}_2 \vee \ldots \mathsf{rkw}_r) \wedge w_2 \wedge \ldots w_n$ and continues with the $\mathsf{GenToken}$ procedure from the previous protocol run on the new query. Intuitively, since the ranges of the nodes found by BRC/URC correspond to a partition of $\mathsf{Range}(\mathsf{w_1})$, so all the documents having $\mathsf{ind}[\mathsf{attr}_R] \in \mathsf{Range}(\mathsf{w_1})$ also satisfy $\mathsf{ind}[\mathsf{attr}_R] \in (\mathsf{rkw}_1 \vee \mathsf{rkw}_2 \vee \ldots \mathsf{rkw}_r)$ and vice versa. Thus, by simply replacing the query with a new one, our protocol can be made to handle range filtering.

Arguments for correctness, security and leakage follow as before, as the problem has

© *2016, Indian Institute of Technology Delhi*

been reduced to our earlier case. The complete protocol supporting conjunction of one range query and atleast one equality query, is provided in the appendix. As discussed earlier, handling pure range queries requires some additional work, as these may not be expressible in SNF form.

4.3 Support for Variance and other functions

We note that our protocol FltAvg can be readily extended to support evaluation of functions such as variance and correlation by replacing the additive homomorphic scheme by a homomorphic scheme which is capable of evaluating degree 2 polynomials, such as [5]. In general, we can handle more sophisticated functions by increasing the homomorphic capabilities of the underlying encryption scheme. However, even the most efficient versions of fully homomorphic encryption are too slow in practice [17, 40, 32, 11], so for computing more general functions we provide a different protocol in Section 5.

© 2016, Indian Institute of Technology Delhi

Chapter 5

Computing General Functions

5.1 Protocol

In this section, we provide a protocol which can additionally support queries of format: "Compute function f on attribute attr for all data records that match keyword w". Again, we rely on the assumption that the number of records matching a given keyword will be a small fraction of the total number of records. We note that f can be arbitrary, but the filtering criteria is restricted to single keyword match rather than an arbitrary Boolean predicate.

For simplicity, let us assume that the only attribute supporting computation of general functions is denoted by attr, and that it takes Boolean values. We note that it is straightforward to lift these restrictions. Our protocol requires that the data owner maintain information about the frequency of each keyword, namely the number of records T_w that match a keyword w, i.e. $T_w = |\mathsf{DB}(w)|$. Our protocol is inspired from the single key functional encryption scheme of Sahai and Seyalioglu [36], and can be seen as a "dual" of the same – [36] supports a single function key and unbounded number of ciphertexts whereas our protocol can be interpreted as supporting a single ciphertext and an unbounded number of function queries. Details follow.

The data owner $\mathcal{D}$, during the setup phase, chooses two PRFs F_τ F_S with range $\{0,1\}^\tau$ as well as a symmetric key encryption scheme SKE. For each keyword w, $\mathcal{D}$ sets $K_g \leftarrow F_\tau(K_S, w)$, where K_S is a PRF key. For $c \in [T_w]$, $b \in \{0,1\}$, she constructs $2T_w$ secret keys of the symmetric key encryption scheme SKE as $\{\mathsf{GSK}_{b,c} \leftarrow \mathsf{SKE.KeyGen}\}$ using randomness generated by PRF $F_S(K_g, bc)$. For $c \in [T_w]$, she checks the value of attribute attr in the corresponding data record: if $\mathsf{ind}_c[\mathsf{attr}] = 0$, then she sets $\mathsf{GSK}_c = \mathsf{GSK}_{0,c}$, else $\mathsf{GSK}_c = \mathsf{GSK}_{1,c}$. The value GSK_c is now appended to the list $\mathsf{T}[w]$.

When the client makes a query $Q = (\phi(\bar{w}), \mathsf{attr}, f)$ of the aforementioned format, the data owner retrieves the s-word w and using the master key, generates the symmetric keys $(\mathsf{GSK}_{0,c}, \mathsf{GSK}_{1,c})$ for $c \in [T_w]$. Next, she generates a garbled circuit Γ for f, along

© 2016, Indian Institute of Technology Delhi

with the $2T_w$ labels denoted by $\{(L^f_{0,c}, L^f_{1,c})\}_{c\in[T_w]}$. Then she encrypts label $L^f_{b,c}$ with key $\mathsf{GSK}_{b,c}$ to obtain $\mathsf{GCT}_{b,c}$ for $c \in [T_w], b \in \{0,1\}$. She permutes the two encryptions for all T_w pairs, namely, she chooses T_w bits $\{b_c\}_{c\in[T_w]}$ and sets $\mathsf{GCT} = \left\{(\mathsf{GCT}_{b_c,c}, \mathsf{GCT}_{\bar{b}_c,c})\right\}_{c\in[T_w]}$. The garbled ciruit Γ, the label encryptions GCT and the map M of the output wire labels of the circuit to 0 and 1 and are provided to the client. The client provides the garbled circuit and the $2T_w$ label encryptions to the server along with the stag required for search. The server retrieves the list t using stag as before, and obtains $\{\mathsf{GSK}_c\}_{c\in[T_w]}$. These T_w secret keys decrypt exactly T_w labels: those corresponding to the attribute values $\mathsf{ind}_c[\mathsf{attr}]$ for $c \in [T_w]$. Using these, the server evaluates the garbled circuit, and recovers an output label. It returns this label to the client as the answer. The client, having received the map from the output label to the bit 0 and 1 recovers the output of the function in the clear. The protocol is described in the follwing figure.

5.2 Correctness and Security

Correctness follows directly from correctness of the search algorithm, correctness of symmetric key encryption and correctness of garbled circuits. Security follows from the security of garbled circuits, semantic security of symmetric key encryption and security of the search protocol. Let us examine security against an honest but curious server. Intuitively, in addition to the leakage resulting from search, the server sees a garbled circuit, all pairs of encrypted labels of the garbled circuit, and one set of decrypted labels, corresponding to the values $\mathsf{ind}_c[\mathsf{attr}]$ for $c \in [T_w]$. From this, he can evaluate the output label of the garbled circuit which is random and fresh for each query. Since he is not provided the mapping of this label to the output, he does not learn anything. We define the simulator in the ideal world so that it constructs the garbled circuit and decrypted labels using the simulator for garbled circuits. Then, given an adversary $\mathcal{A}$ who can distinguish the output of the real and ideal game (Definition 2), we can construct an adversary $\mathcal{B}$ who can distinguish between the real and simulated garbled circuit.

More formally, $\mathcal{B}$ runs $\mathcal{A}$ who outputs DB, from which the set $\mathbf{x} = (\mathsf{ind}_c[\mathsf{attr}])_{c\in[T_w]}$ may be obtained. $\mathcal{B}$ constructs EDB as in the real world, choosing a random set of T_w symmetric key encryption keys to be inserted in the list and provides this to $\mathcal{A}$. Next $\mathcal{A}$ specifies a query $Q = (\phi(\bar{w}), \mathsf{attr}, f)$. $\mathcal{B}$ returns $(\mathbf{x}, f)$ as the garbled circuits challenge and receives a garbled circuit, one set of labels corresponding to $\mathbf{x}$ and a map from the output labels

© 2016, Indian Institute of Technology Delhi

$\underline{\mathsf{EDBSetup}(\mathsf{DB}, \mathcal{F})}$: Let F_τ and F_S be PRFs with range $\{0,1\}^\tau$ and SKE be a symmetric key encryption scheme. Pick key K_S for PRF F_τ.

- Initialize T to an empty array indexed by $w \in \mathsf{W}$.
- For $w \in \mathsf{W}$, build $\mathsf{T}[w]$ as follows:
  - Initialize t to an empty list. Let $K_g \leftarrow F_\tau(K_S, w)$ and $T_w = |\mathsf{DB}(w)|$.
  - Initialize $c \leftarrow 0$ and for all $\mathsf{ind} \in \mathsf{DB}(w)$ in random order, do the following:
    - For $b \in \{0,1\}$, generate $2T_w$ SKE secret keys $(\mathsf{GSK}_{0,c}, \mathsf{GSK}_{1,c})$ using $\mathsf{SKE.KeyGen}$, where $F_S(K_g, c)$ is used to generate the randomness used by $\mathsf{SKE.KeyGen}$.
    - If $\mathsf{ind}_c[\mathsf{attr}] = 0$, then let $\mathsf{GSK}_c = \mathsf{GSK}_{0,c}$, else $\mathsf{GSK}_c = \mathsf{GSK}_{1,c}$.
    - Append GSK_c to t.
  - Set $\mathsf{T}[w] = t$.
- Let $(\mathsf{TSet}, K_T) \leftarrow \mathsf{TSetSetup}(\mathsf{T})$ and $K_M \leftarrow \mathsf{AuthEnc.KeyGen}(1^\tau)$.
- Output $\mathsf{EDB} = \mathsf{TSet}$ and $\mathsf{MSK} = (K_S, K_T, K_M)$.

$\underline{\mathsf{GenToken}(Q, \mathsf{EDB})}$: Parse Q as (w, attr, f) and MSK as (K_S, K_T, K_M).

- $\mathcal{D}$ computes $\mathsf{stag} \leftarrow \mathsf{TSetGetTag}(K_T, w)$, and $K_g = F_\tau(K_S, w)$
- For $c \in [T_w]$, $\mathcal{D}$ generates $\mathsf{GSK}_{0,c}, \mathsf{GSK}_{1,c}$ by invoking $\mathsf{SKE.KeyGen}$ using randomness $F_S(K_g, c)$.
- $\mathcal{D}$ generates a garbled circuit Γ for f, along with the $2T_w$ labels denoted by $\{(L^f_{0,c}, L^f_{1,c})\}_{c\in[T_w]}$ and the map M that maps the output labels $\{L^{out}_b, L^{out}_{\bar{b}}\}$ to $\{0,1\}$.
- $\mathcal{D}$ encrypts label $L^f_{b,c}$ with key $\mathsf{GSK}_{b,c}$ to obtain $\mathsf{GCT}_{b,c}$ for $c \in [T_w], b \in \{0,1\}$.
- $\mathcal{D}$ permutes the two encryptions for all T_w pairs, namely, she chooses T_w bits $\{b_c\}_{c\in[T_w]}$ and sets $\mathsf{GCT} = \left\{(\mathsf{GCT}_{b_c,c}, \mathsf{GCT}_{\bar{b}_c,c})\right\}_{c\in[T_w]}$.
- $\mathsf{env} \leftarrow \mathsf{AuthEnc.Enc}(K_M, (\Gamma, \mathsf{GCT}, \mathsf{stag}))$.
- Output $\mathsf{SK}_Q = (\mathsf{env}, M)$.

$\underline{\mathsf{SCompute}(\mathsf{EDB}, \mathsf{SK}_Q)}$: Client $\mathcal{C}$ on input the token $\mathsf{SK}_Q = (\mathsf{env}, M)$, does the following:

- Send to the server $\mathcal{E}$ the message env.
- When the server sends L^{out}, use map M to determine the bit b it corresponds to. Output b.

Server $\mathcal{E}$ with input $\mathsf{EDB} = (\mathsf{TSet}, K_M)$ responds as follows:

- $(\Gamma, \mathsf{GCT}, \mathsf{stag}) \leftarrow \mathsf{AuthEnc.Dec}(K_M, \mathsf{env})$.
- Get $t \leftarrow \mathsf{TSetRetrieve}(\mathsf{TSet}, \mathsf{stag})$.
- For $c \in [T_w]$, do:
  - Let the c^{th} tuple of t contain GSK_c.
  - Try to decrypt $\mathsf{GCT}_{b_c,c}, \mathsf{GCT}_{\bar{b}_c,c}$ with GSK_c. Exactly one ciphertext decrypts to yield a label L^f_c.
- Execute the garbled circuit Γ with labels $\{L^f_c\}_{c\in[T_w]}$ to obtain output label L^{out}. Return L^{out} to the client.

to 0 and 1. He encrypts this set of labels with the keys provided in EDB and encrypts random labels using random keys for the remaining T_w positions. He returns this garbled circuit and these encryptions to the server $\mathcal{A}$. This is repeated for every query. Finally $\mathcal{A}$ outputs a bit, which $\mathcal{B}$ also outputs. If $\mathcal{A}$ can distinguish between the real and ideal

© *2016, Indian Institute of Technology Delhi*

world, then it follows that either the semantic security of symmetric key encryption does not hold, or the security of garbled circuits does not hold.

5.3 Discussion

We note that even though the protocol can support arbitrary functions and does not use any "heavyweight" cryptography, it incurs the disadvantages that search is only restricted to a single keyword and the data owner $\mathcal{D}$ must now maintain the size of matching documents for each keyword, namely $|\mathsf{DB}(w)|$ for all $w \in \mathsf{W}$. These restrictions are required since the data owner must know the input size for the garbled circuit she generates. Note that even for SSE, $\mathcal{D}$ was required to maintain some information regarding $|\mathsf{DB}(w)|$ so as to compute s-words, but this was mitigated by the observation that in most datasets the number of very frequent words is very small and the data owner can keep a small state with sufficient information to choose light s-terms (using Bloom filters for example) [8, Sec 3.1.1]. We remark that in the multi-client setting that we consider, the data owner may not be as resource limited as the client considered by SSE,. Still, it is desirable to overcome these restrictions; we leave this as an open problem.

© 2016, Indian Institute of Technology Delhi

Chapter 6

Related Work

In this section, we describe some related work. Functional Encryption (FE) [38, 37] enables computing of arbitrary functions on encrypted data but function computation takes time polynomial in the data set size even for simple functions [4, 12, 7, 6, 21]. Additionally, the security model of FE assumes a single malicious adversary that encompasses the server and client. By contrast, we follow the norm in SSE and assume that the server and client do not collude. We believe this is a reasonable assumption, especially for large data applications and deserves to be explored more in FE constructions. A variant of FE, called "Controlled FE" [33] does provide efficient protocols, but also suffers from computation time polynomial in dataset size. Another primitive related to our work is structured encryption [9]. Structured encryption generalizes symmetric searchable encryption (SSE) to the setting of arbitrarily structured data. In the context of cloud storage, structured encryption allows a client to encrypt data without losing the ability to query and retrieve it efficiently. Finally, we note that recent work has studied functionalities other than search (see [42, 10, 35] and references therein) but these are restricted to the single client setting.

© *2016, Indian Institute of Technology Delhi*

Bibliography

[1] C. Arun. http://www.thehindu.com/opinion/lead/lead-article-on-aadhaar-bill-by-chinmayi-arun-\privacy-is-a-fundamental-right/article8366413.ece.

[2] M. Bellare, V. T. Hoang, and P. Rogaway. Foundations of Garbled Circuits. In *CCS*, 2012.

[3] D. Boneh, G. D. Crescenzo, R. Ostrovsky, and G. Persiano. Public key encryption with keyword search. In *EUROCRYPT*, 2004.

[4] D. Boneh and M. K. Franklin. Identity-based encryption from the weil pairing. In *CRYPTO*, 2001.

[5] D. Boneh, E.-J. Goh, and K. Nissim. Evaluating 2-dnf formulas on ciphertexts. In *TCC*, 2005.

[6] D. Boneh and B. Waters. Conjunctive, subset, and range queries on encrypted data. In *TCC*, 2007.

[7] X. Boyen and B. Waters. Anonymous hierarchical identity-based encryption (without random oracles). In *CRYPTO*, pages 290–307, 2006.

[8] D. Cash, S. Jarecki, C. S. Jutla, H. Krawczyk, M. Rosu, and M. Steiner. Highly-scalable searchable symmetric encryption with support for boolean queries. In *CRYPTO*, 2013.

[9] M. Chase and S. Kamara. Structured encryption and controlled disclosure. In *ASIACRYPT*, 2010.

[10] M. Chase and E. Shen. Pattern matching encryption. http://elastic.org/~fche/mirrors/www.cryptome.org/2014/08/pattern-matching-encryption.pdf, 2014.

[11] D. D. Chen, N. Mentens, F. Vercauteren, S. S. Roy, R. C. Cheung, D. Pao, and I. Verbauwhede. High-speed polynomial multiplication architecture for ring-lwe and she cryptosystems. *Circuits and Systems I: Regular Papers, IEEE Transactions on*, 62(1):157–166, 2015.

© *2016, Indian Institute of Technology Delhi*

[12] C. Cocks. An identity based encryption scheme based on quadratic residues. In *IMA Int. Conf.*, pages 360–363, 2001.

[13] I. Demertzis, S. Papadopoulos, O. Papapetrou, A. Deligiannakis, and M. Garofalakis. Practical private range search revisited. In *sigmod*, 2016.

[14] W. A. Drazen, E. Ekwedike, and R. Gennaro. Highly scalable verifiable encrypted search. In *Communications and Network Security (CNS), 2015 IEEE Conference on*, pages 497–505. IEEE, 2015.

[15] T. El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Proceedings of CRYPTO 84 on Advances in Cryptology*, 1985.

[16] D. Evans, J. Katz, and Y. Huang. Quid-pro-quo-tocols: Strengthening semi-honest protocols with dual execution. *2014 IEEE Symposium on Security and Privacy*, 0:272–284, 2012.

[17] C. Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, pages 169–178, 2009.

[18] I. Government. http://www.censusindia.gov.in/2011-prov-results/paper2-vol2/data_files/kerala/Analysis_Census_Data.pdf.

[19] U. Government. http://www.census.gov/ces/dataproducts/index.html.

[20] U. Government. http://factfinder.census.gov/.

[21] V. Goyal, O. Pandey, A. Sahai, and B. Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *ACM Conference on Computer and Communications Security*, pages 89–98, 2006.

[22] J. Groth. On the size of pairing-based non-interactive arguments. In M. Fischlin and J.-S. Coron, editors, *Advances in Cryptology – EUROCRYPT 2016: 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II*, 2016.

[23] A. Hamlin, N. Schear, E. Shen, M. Varia, S. Yakoubov, and A. Yerukhimovich. Cryptography for big data security. In *Big Data: Storage, Sharing, and Security*. Auerbach Publications, 2016. https://eprint.iacr.org/2016/012.pdf.

© *2016, Indian Institute of Technology Delhi*

[24] Y. Huang, D. Evans, J. Katz, and L. Malka. Faster secure two-party computation using garbled circuits. In *Proceedings of the 20th USENIX Conference on Security*, SEC'11, pages 35–35, 2011.

[25] Y. Huang, J. Katz, and D. Evans. Efficient secure two-party computation using symmetric cut-and-choose. In *CRYPTO*, 2013.

[26] S. Jarecki, C. S. Jutla, H. Krawczyk, M. Rosu, and M. Steiner. Outsourced symmetric private information retrieval. In *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*, pages 875–888, 2013.

[27] R. Kerbaj and J. Ungoed-Thomas. http://www.thesundaytimes.co.uk/sto/news/uk_news/National/article1258476.ece?CMP=OTH-gnws-standard-2013_05_11.

[28] B. Kreuter, A. Shelat, B. Mood, and K. R. B. Butler. PCF: A portable circuit format for scalable two-party secure computation. In *Proceedings of the 22th USENIX Security Symposium, Washington, DC, USA, August 14-16, 2013*, pages 321–336, 2013.

[29] B. Kreuter, A. Shelat, and C. Shen. Billion-gate secure computation with malicious adversaries. In *Proceedings of the 21th USENIX Security Symposium, Bellevue, WA, USA, August 8-10, 2012*, pages 285–300, 2012.

[30] Y. Lindell. Fast cut-and-choose based protocols for malicious and covert adversaries. In *CRYPTO*, 2013.

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

[32] M. Naehrig, K. Lauter, and V. Vaikuntanathan. Can homomorphic encryption be practical? In *CCSW*, 2011.

[33] M. Naveed, S. Agrawal, M. Prabhakaran, X. Wang, E. Ayday, J.-P. Hubaux, and C. Gunter. Controlled functional encryption. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, 2014.

[34] I. M. official statement. https://www.ipsos-mori.com/newsevents/latestnews/1390/Ipsos-MORI-response-to-the-Sunday-Times.aspx.

© *2016, Indian Institute of Technology Delhi*

[35] R. A. Popa. *Building practical systems that compute on encrypted data*. PhD thesis, Massachusetts Institute of Technology, 2014.

[36] A. Sahai and H. Seyalioglu. Worry-free encryption: Functional encryption with public keys. In *Proceedings of the 17th ACM Conference on Computer and Communications Security*, CCS '10, 2010.

[37] A. Sahai and B. Waters. Functional encryption:beyond public key cryptography. Power Point Presentation, 2008. http://userweb.cs.utexas.edu/bwaters/presentations/ files/functional.ppt.

[38] A. Sahai and B. Waters. Fuzzy identity-based encryption. In *EUROCRYPT*, pages 457–473, 2005.

[39] A. Sinha and H. T. Pranesh Prakash.

[40] E. Stefanov and E. Shi. Oblivistore: High performance oblivious cloud storage. In *Security and Privacy (SP), 2013 IEEE Symposium on*, pages 253–267. IEEE, 2013.

[41] P. Swabey. http://www.information-age.com/technology/mobile-and-networking/123457043/ee-and-ipsos-mori-face-privacy-backlash-over

[42] S. Tu, M. F. Kaashoek, S. Madden, and N. Zeldovich. Processing analytical queries over encrypted data. In *Proceedings of the 39th international conference on Very Large Data Bases*, PVLDB'13, pages 289–300. VLDB Endowment, 2013.

[43] Wikipedia. https://en.wikipedia.org/wiki/Aadhaar.

[44] A. C.-C. Yao. How to generate and exchange secrets. In *Proceedings of the 27th Annual Symposium on Foundations of Computer Science*, 1986.

[45] S. Zahur and D. Evans. Circuit structures for improving efficiency of security and privacy tools. In *IEEE Symposium on Security and Privacy*, 2013.

[46] A. Zelin. https://www.mrs.org.uk/pdf/Andrew_Zelin_presentation.pdf.

© *2016, Indian Institute of Technology Delhi*

Appendix A

Protocol supporting Range Queries

The protocol supporting range queries is provided in the following figures. Changes from our original protocol are highlighted in blue.

$\mathsf{EDBSetup}(\mathsf{DB}, \mathcal{F})$: Let F_τ and F_p be 2 PRFs with range in $\{0,1\}^\tau$ and $\mathbb{Z}_p^*$ respectively, where τ is the security parameter. Pick keys K_S for PRF F_τ and K_X, K_I for PRF F_p.

- Initialize $\mathsf{XSet}, \mathsf{XSetAll}$ to empty set and T to an empty array indexed by $w \in \mathsf{W}$; Parse $(\mathsf{ind}_i, \mathsf{W}_i)_{i=1}^d \leftarrow \mathsf{DB}$.
- Build a new database DB' as follows:
  - As in the Log-BRC/URC scheme, build a binary tree over the domain Dom.
  - For each node in the tree, assign a unique keyword, referred to as rword.
  - For $(\mathsf{ind}_i, \mathsf{W}_i) \in \mathsf{DB}$, do: $val \leftarrow$ value of the attribute attr_R in the document with index ind_i; $\mathsf{W}'_i \leftarrow \mathsf{W}_i \cup \{\mathsf{rword}$ for nodes on the path in the tree from the root to $val\}$; Add $(\mathsf{ind}_i, \mathsf{W}'_i)$ to DB'.
- $\mathsf{W}' \leftarrow \cup_i \mathsf{W}'_i$; For $w \in \mathsf{W}'$, build $\mathsf{XSet}, \mathsf{XSetAll}$ and $\mathsf{T}[w]$ as follows:
  - Initialize t to an empty list.
  - $\mathsf{strap} \leftarrow F_\tau(K_S, w)$ and $(K_z,\ K_h) \leftarrow (F_\tau(\mathsf{strap}, 1),\ F_\tau(\mathsf{strap}, 2))$.
  - $\forall (1 \le i \le d)$, add $g^{F_p(K_X, w) \cdot F_p(K_I, \mathsf{ind}_i)}$ to $\mathsf{XSetAll}$.
  - Initialize $(\mathsf{sum}_1, \mathsf{sum}_2 \ldots \mathsf{sum}_k)$ with zeros. Initialize $c \leftarrow 0$ and for all $\mathsf{ind} \in \mathsf{DB}'(\mathsf{w})$ in random order, do:
    - Let $\mathsf{xind} \leftarrow F_p(K_I, \mathsf{ind})$. Let $c = c + 1$, $z_c \leftarrow F_p(K_z, c)$, and $y \leftarrow \mathsf{xind} \cdot z_c^{-1}$.
    - Set $\mathsf{xtag} \leftarrow g^{F_p(K_X, w) \cdot \mathsf{xind}}$ and add xtag to XSet.
    - for $j \in [k]$, let $\mathsf{val} \leftarrow \mathsf{ind}[\mathsf{attr}_j]$, $K_h^j \leftarrow F_\tau(K_h, j)$, $\mathsf{ct}_j \leftarrow \mathsf{AHE.Enc}(K_h^j, -\mathsf{val})$ and $\mathsf{sum}_j = \mathsf{sum}_j + \mathsf{val}$.
    - Let $\mathsf{ct} \leftarrow (\mathsf{ct}_1, \mathsf{ct}_2 \ldots \mathsf{ct}_k)$ and append (y, ct) to t.
  - for $j \in [k]$, let $K_h^j \leftarrow F_\tau(K_h, j)$, $\mathsf{sct}_j \leftarrow \mathsf{AHE.Enc}(K_h^j, \mathsf{sum}_j)$.
  - Append $(0, \mathsf{sct}_1, \mathsf{sct}_2 \ldots \mathsf{sct}_k)$ to head of t and set $\mathsf{T}[w] = t$.
- Let $(\mathsf{TSet}, K_T) \leftarrow \mathsf{TSetSetup}(\mathsf{T})$ and $K_M \leftarrow \mathsf{AuthEnc.KeyGen}(1^\tau)$.
- Output $\mathsf{EDB} = (\mathsf{TSet}, \mathsf{XSet}, K_M)$ and $\mathsf{MSK} = (K_S, K_X, K_I, K_T, K_M)$.

© 2016, Indian Institute of Technology Delhi

$\underline{\mathsf{GenToken}(Q, \mathsf{MSK})}$: Parse Q as $(\phi(\bar{w}), \mathsf{attr}, f)$ and MSK as $(K_S, K_X, K_I, K_T, K_M)$. Assume $\phi(\bar{w}) = w_1 \wedge w_2 \ldots \wedge w_n$, where WLOG $\mathsf{Range(w_1)} \neq \mathsf{NULL}$ and $n \geq 2$. Also, let attr be the j_0^{th} computable attribute.

- Let $\mathsf{rkw}_1, \mathsf{rkw}_2 \ldots \mathsf{rkw}_r$ be the keywords corresponding to the nodes given by $RC(\mathsf{Range(w_1)})$, where RC is one of BRC/URC.
- $\phi'(\bar{w}) \leftarrow ((\mathsf{rkw}_1 \vee \mathsf{rkw}_2 \vee \ldots \mathsf{rkw}_r) \wedge w_2 \wedge \ldots w_n)$
- Rename $\phi'(\bar{w})$ to $((w_1 \vee w_2 \vee \ldots w_r) \wedge w_{r+1} \wedge \ldots w_{r+n-1}$.
- WLOG, assume w_{r+n-1} as the sword out of $\{w_{r+1}, w_{r+2} \ldots w_{r+n-1}\}$ (chosen by $\mathcal{D}$).
- Compute $\mathsf{stag} \leftarrow \mathsf{TSetGetTag}(K_T, w_{r+n-1})$, $\mathsf{strap} \leftarrow F_\tau(K_S, w_{r+n-1})$.
- For $i = 1$ to $r + n - 2$, do : $\rho_i \xleftarrow{\$} \mathbb{Z}_p^*$ and set $\mathsf{bxtrap}_i \leftarrow g^{F_p(K_X, w_i) \cdot \rho_i}$.
- $\mathsf{env} \leftarrow \mathsf{AuthEnc.Enc}(K_M, (j_0, \mathsf{stag}, \rho_1, \rho_2, \ldots \rho_{r+n-2}))$.
- Output $\mathsf{SK}_Q = (j_0, \mathsf{env}, \mathsf{strap}, \mathsf{bxtrap}_1, \mathsf{bxtrap}_2 \ldots \mathsf{bxtrap}_{r+n-2})$.

$\underline{\mathsf{SCompute}(\mathsf{EDB}, \mathsf{SK}_Q)}$: Client $\mathcal{C}$ on input the token $\mathsf{SK}_Q = (j_0, \mathsf{env}, \mathsf{strap}, \mathsf{bxtrap}_1, \mathsf{bxtrap}_2 \ldots \mathsf{bxtrap}_{r+n-2})$, does the following:

- $(K_z, K_h) \leftarrow (F_\tau(\mathsf{strap}, 1), F_\tau(\mathsf{strap}, 2))$.
- Send to the server $\mathcal{E}$ the message $(\mathsf{env}, \mathsf{bxtoken}[1], \mathsf{bxtoken}[2] \ldots)$, where $\mathsf{bxtoken}[.]$ is computed as follows :
  - For $c = 1, 2 \ldots$, until $\mathcal{E}$ sends stop, do:
    - $z_c \leftarrow F_p(K_z, c)$ and $\mathsf{bxtoken}[c, i] \leftarrow \mathsf{bxtrap}_i^{z_c} \quad \forall\ 1 \leq i \leq r + n - 2$.
    - Set $\mathsf{bxtoken}[c] \leftarrow (\mathsf{bxtoken}[c, 1] \ldots \mathsf{bxtoken}[c, r + n - 2])$.
- $K_h^{j_0} \leftarrow F_\tau(K_h, j_0)$.
- When the server sends stop, along with it, it sends (temp, n). Output $\mathsf{AHE.Dec}(K_h^{j_0}, \mathsf{temp})/n$.

Server $\mathcal{E}$ with input $\mathsf{EDB} = (\mathsf{TSet}, \mathsf{XSet}, K_M)$ responds as follows:

- Let $(j_0, \mathsf{stag}, \rho_1, \ldots \rho_{r+n-2}) \leftarrow \mathsf{AuthEnc.Dec}(K_M, \mathsf{env})$. Let $t \leftarrow \mathsf{TSetRetrieve}(\mathsf{TSet}, \mathsf{stag})$.
- Let $(0, \mathsf{sct}_1, \mathsf{sct}_2 \ldots \mathsf{sct}_k)$ be the first tuple of t. $\mathsf{temp} \leftarrow \mathsf{sct}_{j_0}$ and $\mathsf{cnt} \leftarrow 0$.
- For $c = 1, 2, \ldots, |t| - 1$, do:
  - Let the $c + 1^{th}$ tuple of t be $(y, \mathsf{ct}_1, \ldots, \mathsf{ct}_k)$.
  - For $\mathsf{bxtoken}[c]$ as received from $\mathcal{C}$, do for $i \in [1 \ldots r + n - 2]$:
    - If $\mathsf{bxtoken}[c, i]^{y/\rho_i} \notin \mathsf{XSetAll}$, then return $\perp$.
    - Define $\nu_i =$ boolean value of the predicate $(\mathsf{bxtoken}[c, i]^{y/\rho_i} \in \mathsf{XSet})$
  - If $((\nu_1 \vee \nu_2 \vee \ldots \nu_r) \wedge \nu_{r+1} \wedge \ldots \nu_{r+n-2}) = False$, then $\mathsf{temp} = \mathsf{temp} + \mathsf{ct_{j_0}}$ and $\mathsf{cnt} = \mathsf{cnt} + 1$.
- When the last tuple in t is reached, send $(\mathsf{stop}, (\ \mathsf{temp}, |t| - 1 - \mathsf{cnt}\ ))$ to $\mathcal{C}$.

© 2016, Indian Institute of Technology Delhi

Appendix B

Proof of theorem 6: Security against adversarial clients

Here, we provide a proof of theorem 6, which shows that the protocol FltAvg is secure against adversarial clients. First, we provide the simulator for the same (as shown in following figure), followed by a series of indistinguishable games which starts from the real world and ends in the ideal world.

Let G_0 denote $\mathsf{Real}_A^{\Pi}(1^\lambda)$ and G_7 denote $\mathsf{Ideal}_{A,\mathsf{Sim}}^{\Pi}(1^\lambda)$ as defined in the section **??**. We will create a series of modifications to G_0, say G_i. Let P_i denote the probability that game G_i outputs 1. We show that the difference between P_0 and P_7 is a negligible function of the security parameter λ.

For simplicy, we assume that the query is a conjunction.

Game G_1 : In game G_1 we add an abort if any of the ciphertexts env accepted by the server in SCompute has not been generated by the data owner during GenToken protocol. By *Strong-UF-CMA* unforgeability of the AuthEnc scheme, $P_1 \overset{c}{\approx} P_0$.

Game G_2 : In game G_2 we add an abort if any two of the ciphertexts env generated by the GenToken protocol are the same. By the properties of AuthEnc, we have that a collision in ciphertext implies a collision in plaintext and as we are generating ρ_is randomly from $\mathbb{Z}_p^*$, the probability of a collision of all ρ_is is negligible. So, $P_2 \overset{c}{\approx} P_1$.

Game G_3 : We modify game G_2 by adding an abort if there exists any two different keywords $w, w' \in W$ such that $F_p(K_T, w') = F_p(K_T, w)$ or $F_p(K_X, w') = F_p(K_X, w)$. This happens with negligible probability as F_p is a PRF and hence $P_3 \overset{c}{\approx} P_2$.

Game G_4 : In the GenToken protocol, instead of encrypting the ρ_is, which are used to generate bxtraps, we encrypt independently random ρ_i' values to get env. We also maintain a list called QueryTable, which is indexed by the env ciphertexts and stores the original $\rho_i s$ along with the query $\bar{w} = (w_1, w_2, \ldots w_n)$. We modify the SCompute protocol accordingly

© *2016, Indian Institute of Technology Delhi*

$S_0(1^\tau)$:

- Select key K_S for PRF F_τ and keys K_X, K_I, K_T for PRF F_p and K_M for AuthEnc.
- Initialize QueryTable as an empty table. It will be indexed by env ciphertexts.
- Initialize the state to be $\mathsf{st} \leftarrow (\mathsf{K_S}, \mathsf{K_X}, \mathsf{K_I}, \mathsf{K_T}, \mathsf{K_M}, \mathsf{QueryTable})$.

$S_1(\mathsf{st})$:

- On receiving $(\phi(\bar{w}), \mathsf{attr}, f)$ as input from A, Abort if query is not valid.
- Let $\bar{w}$ be $(w_1, w_2, \ldots w_n)$.
- Choose $\rho_i, \rho_i' \xleftarrow{\$} \mathbb{Z}_p^* \ \forall i \in (1, 2, \ldots n)$
- $\mathsf{stag} \leftarrow F_p(K_T, w_1)$, $\mathsf{strap} \leftarrow F_\tau(K_S, w_1)$.
- $\mathsf{bxtrap}_i \leftarrow g^{F_p(K_X, w_i)\cdot\rho_i} \ \forall i \in (2, 3, \ldots n)$
- Output $(\mathsf{env}, \mathsf{strap}, \mathsf{bxtrap}_2, \ldots \mathsf{bxtrap}_n)$ where $\mathsf{env} \leftarrow \mathsf{AuthEnc}(K_M, (j_0, \mathsf{stag}, \rho_2', \ldots \rho_n')$ and set $\mathsf{QueryTable}(env) \leftarrow (w_1, w_2, \ldots w_n, \rho_1, \rho_2, \ldots \rho_n, j_0)$

$S_2(\mathsf{st})$:

- On input $(\mathsf{env}, \mathsf{bxtoken}[1], \mathsf{bxtoken}[2], \ldots \mathsf{bxtoken}[n])$ from A, Abort if $\mathsf{QueryTable}(\mathsf{env}) = \perp$. Otherwise get $(w_1, w_2, \ldots w_n, \rho_1, \rho_2, \ldots \rho_n, j_0) \leftarrow \mathsf{QueryTable}(\mathsf{env})$.
- Send $Q = (\phi(\bar{w}), \mathsf{attr}, f)$ as input to the oracle which returns $(\mathsf{DB}(Q), (|\mathsf{DB}(w_1)|, \mathsf{cnt}))$.
- $\mathsf{strap} \leftarrow F_\tau(K_S, w)$ and $(K_z,\ K_h) \leftarrow (F_\tau(\mathsf{strap}, 1),\ F_\tau(\mathsf{strap}, 2))$.
- For $c = 1, 2, \ldots, |\mathsf{DB}(w_1)|$, do:
  - $z_c \leftarrow F_p(K_z, c)$.
  - For $\mathsf{bxtoken}[c] = (\mathsf{bxtoken}[c, 2] \ldots \mathsf{bxtoken}[c, n])$
    - If $\mathsf{bxtoken}[c, i] = g^{F_p(K_X, w_i)\cdot\rho_i \cdot z_c}$ continue, Else Abort.
- $K_h^{j_0} \leftarrow F_\tau(K_h, j_0)$, $\mathsf{ciph} \leftarrow \mathsf{AHE.Enc}(K_h^{j_0}, \mathsf{DB}(Q))$.
- After receiving $|\mathsf{DB}(w_1)|$ bxtokens from A send $(\mathsf{stop}, (\ \mathsf{ciph}, \mathsf{cnt}\ ))$ to $\mathcal{A}$.

to use ρ values from the QueryTable instead of decrypting the env ciphertext sent by A. We have $P_4 \overset{c}{\approx} P_3$ by the $IND\text{-}CCA$ property of AuthEnc. Also, the ciphertexts generated during the GenToken protocol are unique from game G_2 onwards. So, QueryTable has a unique entry for each env and hence the server can retrieve ρ_i values correctly for all queries.

Game G_5 : In this game, we modify the EDBSetup procedure. Instead of creating TSet from the array T, whenever the SCompute protocol has to find the list corresponding to a keyword from stag, it scans through all the keywords to find $w_1 \in \mathsf{W}$ such that

© 2016, Indian Institute of Technology Delhi

$\mathsf{stag} = F_p(K_T, w_1)$ (since $\mathsf{TSetGetTag}$ is implemented as $F_p(K_T, \cdot)$ in [26]). If such a keyword is found, then assign $t \leftarrow \mathsf{T}[w_1]$, otherwise Abort. From the no-false-negatives and no-false-positives properties of TSet [], It can be seen that t retrieved in G_5 is the same as t retireved in G_4 except for negligible probability. Hence, $P_5 \overset{c}{\approx} P_4$.

Game G_6 : In game G_6, once the server receives env from A, it retrieves $(w_1, w_2, \ldots w_n, \rho_1, \rho_2, \ldots \rho_n$ from $\mathsf{QueryTable}(\mathsf{env})$. It can now construct a query $Q = (\phi(\bar{w}), \mathsf{attr}, f)$ using the information retrieved. It sends this query to the ideal oracle to get $(\mathsf{DB}(Q), (|\mathsf{DB}(w_1)|, \mathsf{cnt}))$. We modify the way G_6 processes $\mathsf{bxtoken}$s received from A. For each c, it computes $z_c \leftarrow F_p(K_Z, c)$, where K_Z is calculated from w_1 retrieved in above games. On receiving $\mathsf{bxtoken}[c] = (\mathsf{bxtoken}[c, 2] \ldots \mathsf{bxtoken}[c, n])$, game G_6 checks if $\mathsf{bxtoken}[c, i] = g^{F_p(K_X, w_i) \cdot \rho_i \cdot z_c}$ to continue and aborts if the check fails. If this check succeeds in G_6, then the check for $\mathsf{bxtoken}[c, i]^{y_c/\rho_i}$ in $\mathsf{XSetAll}$ would have been a success in G_5 because, $\mathsf{bxtoken}[c, i] = g^{F_p(K_X, w_i) \cdot \rho_i \cdot z_c}$ implies $\mathsf{bxtoken}[c, i]^{y_c/\rho_i} = g^{F_p(K_X, w_i) \cdot \mathsf{xind}_c}$. Only difference comes when $g^{F_p(K_X, w_i) \cdot \mathsf{xind}_c} \in \mathsf{XSetAll}$ but $\mathsf{bxtoken}[c, i] \neq g^{F_p(K_X, w_i) \cdot \rho_i \cdot z_c}$. This is proven to be negligible in OSPIR paper. It implies that $P_6 \overset{c}{\approx} P_5$. For more details, we refer the reader to [26].

Game G_7 : In game G_7, we modify the way the server sends the result and *stop* to A. As the server receives $(\mathsf{DB}(Q), (|\mathsf{DB}(w_1)|, \mathsf{cnt}))$ from the oracle, it waits until it receives $|\mathsf{DB}(w_1)|$ $\mathsf{bxtoken}$s and then sends $(\mathsf{stop}, (\ \mathsf{ciph}, \mathsf{cnt}\ ))$, where $K_h^{j_0} \leftarrow F_\tau(K_h, j_0)$, $\mathsf{ciph} \leftarrow \mathsf{AHE.Enc}(K_h^{j_0}, \mathsf{DB}(Q))$. This creates a view exactly similar to that of G_6. From the property of AHE, ciph should not be distinguishable from the one created in previous games. Therefore, $P_7 \overset{c}{\approx} P_6$.

© 2016, Indian Institute of Technology Delhi