Secure Outsourced Pattern Matching based on Bit-Parallelism

Mohammad Hasan Samadani and Mehdi Berenjkoub

***Abstract*—Secure outsourcing is essential to growth of cloud usage. There are some protocols allowing any functionality to be outsourced. However, specific constructions are necessary in order to do so in an efficient way. In this paper, we consider the problem of secure outsourced pattern matching. Our solution is based on Bit-Parallel Shift-ADD algorithm. The properties of this insecure algorithm allow our construction to search in an outsourced text, without revealing any non-trivial information to the computing server. We achieve a round optimal protocol that allows us to search for patterns with wildcards and handles the Hamming distance computation. Since the protocol has no leakage to the server, it cannot be optimal considering communication complexity; however, we suggest efficient techniques to achieve communication optimality through outsourcing of decryption as well. The security of our protocol is proved in the semi-honest setting. Then, in order to retain the efficiency of the protocol, we omit the correctness property in the malicious setting and prove that the scheme remains private in the presence of malicious adversaries.**

***Index Terms*— Bit-Parallel Shift-ADD, Outsourcing, Secure pattern matching, Two-party computation.**

I. Introduction

In the simplest form, the pattern matching problem consists in finding all the positions in a text T of length n that matches a pattern P of length m, both from a finite alphabet Σ. Due to vast applicability of this problem it has been studied for decades. Recently, with the trending attention into secure multi-party computation there is growing interest in the secure pattern matching problem. In a typical secure two-party setting, the pattern owner wants to search for her private pattern in the private text of the text owner, while the pattern owner learns nothing about the text more than the matching positions and the text owner learns nothing about the pattern.

Du and Attalah [1] were the first to formalize the secure pattern matching problem. They categorized this problem into four distinct scenarios; in the first two scenarios, Client wants to search for her private pattern either in the private or public text of Server. In the third scenario, Client wants to search for her private pattern in her private text (in an encrypted form) outsourced to Server, which we refer to as outsourced pattern matching. In the last scenario, Client is going to delegate her search capabilities on her outsourced private text to Carl who has a private pattern, which we refer to as delegated pattern matching. The first scenario received much attention in the literature [2-7] while the others are rarely considered [8, 9].

The focus of this paper is on the outsourced pattern matching. To be precise, Client has a private text that she stores in an encrypted form with Server. She later issues queries in the form of private search patterns, which Server evaluates on the outsourced text. As part of this functionality, Server should not learn anything about Client's data (either the text or the patterns) besides the publicly available information such as the text and pattern lengths.

Our protocol is intuitively based on an insecure pattern matching algorithm called the Bit-Parallel Shift-ADD algorithm [10], which is a non-comparison based algorithm [11]. Then we design a specific construction for secure pattern matching outsourcing by exploiting the functional structure of this algorithm.

This protocol supports wildcards in the pattern and works for any finite alphabet. Also, its required storage space, communication and computation complexity are independent of the alphabet size which makes it suitable for very large alphabets, e.g. UTF-8 [12]. Moreover, our protocol can be used when the outsourced text is a database of keywords (keyword search), a stream of characters (text search), and a live stream of characters (live text search). The text is outsourced to a single server, besides it can be deleted from the outsourcer side since it can be fetched and decrypted later. It also take $O(n \cdot \kappa)$ space to store a text of size n, where κ is the security parameter. It is also very easy to update or delete the whole or any portions of the outsourced text as well as adding any new characters or keywords to any desired position in the text.

The most notable property of this protocol is that there is no need to predict, designate, or fix the patterns to be queried later while preprocessing the text for outsourcing, despite all other known protocols. This enables the outsourcer to query for any new or unpredicted pattern with arbitrary length after outsourcing.

The proposed protocol is round-optimal; only one round (one-way) for outsourcing the text and two rounds (one-way) for searching. In the pattern matching problem, the optimal communication complexity during query phase for the receiver is linear in the number of matches. Since the worst

Manuscript received August 6, 2016; accepted January 24, 2017.
(Corresponding Author) Mohammad Hasan Samadani is a PhD candidate at the Department of Electrical and Computer Engineering, Isfahan University of Technology, Isfahan, Iran (e-mail:mohammad.samadani@gmail.com).

Mehdi Brenjkoob is with the Department of Electrical and Computer Engineering, Isfahan University of Technology, Isfahan, Iran (e-mail: brnjkb@cc.iut.ac.ir).

case output size is proportional to the length of the text and a circuit that tolerates this worst case is needed, it is not clear how to achieve optimal communication complexity during query phase in two rounds even using Fully Homomorphic Encryption [13]. On the other hand, there is no way to achieve the optimal communication complexity unless giving the possibly corrupted server some leakages about the pattern, text, or matched positions. Intuitively, one can think of this case as if Server is able to send sublinear information to Client in the query phase without sacrificing any matched position, he has been able to know which parts of the text did not match the pattern and which parts possibly did. This at least means that Server has some information about the pattern, text, or matched positions. In the literature, researchers [8, 9] allowed some leakages in order to avoid linear communication complexity. We emphasize that our protocol is able to avoid this leakage at the price of this complexity. However, one can choose to outsource the result extraction phase in order to avoid the communication complexity.

The paper is organized as follow: the rest of this section consists of problem statement, related work, and our contributions. We covered some basic concepts that are used through the paper in Section 2. Then, in Section 3, we proposed our secure protocol for pattern matching outsourcing. The security and performance of the protocol are discussed in Sections 4 and 5 respectively. Finally, Section 6 points some future works and Section 7 concludes the paper.

A. Related work

With the recent growth in cloud computing services, the problem of secure outsourcing of computations gained more attention. There are some works in the area of outsourcing and delegatable computation which enable clients to outsource any functionality to an untrusted server [14-16]. These general constructions often have poor efficiency and high computation overhead due to the use of fully homomorphic encryption [8]. The previous experience in the literature of two/multi-party computation proved that to move towards more practical schemes one may use the special properties of each functionality to design specific constructions rather than using such general solutions [17].

The problem of secure pattern matching has received a great attention in the literature of two-party computation motivated by its critical and broad applicability in computer science. Du and Atallah [1] were the first to formalize the secure pattern matching problem and defined four scenarios of secure pattern matching. They proposed a basic secure protocol for computing the inner product to vectors with the help of an untrusted third party. Then, they used this basic protocol to design secure solutions for each of those scenarios. However, almost all of these protocols are just feasibility results and impractical. Also, they suffer from major limitations; the help of an untrusted third party is needed, text must be stored in the form of equal-length keywords, and only the same length patterns can be evaluated.

Their work is followed up with other researches in various directions. The first scenario, where Client has a private pattern and Server has a private text, has gained much more attention [2-7]. Also, various kinds of pattern, from a simple string of characters to the complex form of regular expression [14], are considered. On the other hand, the third scenario (outsourcing) which is the focus of this paper, is rarely considered in the literature.

The most significant work in this regard is the work of Faust, Hazay, and Venturi [8], which covers the forth scenario (delegation) as well. Their work considers a trapdoor for each pattern of length m in the form of an easy to solve instance of the subset sum problem [18]. Namely, Client outsources her text in the form of a random vector in which all positions that match a specific pattern equals an instance of the subset sum problem, where giving a trapdoor for that pattern is easy to solve. Any party that is able to generate such a trapdoor is able to search for that pattern. This makes this scheme applicable in the case that Client wants delegate her search power to other parties; this delegation is done through a secure protocol in which the other party privately learns the trapdoor.

They proved this protocol using simulation-based security [17], in the presence of semi-honest and malicious adversaries. Also, during the query phase the overall communication and computation complexity done by Client is linear in the number of matches, however, it allows some leakages to Server. More precisely, Server learns the matched positions and more importantly the repetitions in the text; Server is not able to learn the context of these positions, however, this enables him to run statistical analysis on the text and learn even more information about it. Moreover, pattern's length m must be known and fixed prior preprocessing a text for outsourcing and the pattern must be in the simplest form, i.e. a simple string of characters and no wildcard.

Another work by Wei and Reiter considered more complex patterns in the form of Deterministic Finite Automata (DFA) [19]. This work is an interactive protocol with round complexity linear in the size of the text.

We are also aware of the context of Searchable Encryption (SE) [20]. In a SE scheme a server is allowed to search in an encrypted data on behalf of a client without learning information about the data. There are two high level approaches to achieve a SE scheme; some schemes implement this via a ciphertext that allows searching [21], while in most others the client generates a list of encrypted (possibly non-decryptable, e.g. using a hash function) predefined keywords. A main problem with the first type is that the queried pattern is revealed to the server if matched. Also, the problem with the second type is that the patterns to be queried must be known and fixed at the time of outsourcing; if a pattern is present in the text but is not considered as a keyword to be search later when outsourcing, it cannot be found during the search.

The main problem with SE is that searchable encryption usually does not ensure the privacy of the searched pattern [20]. While this issue is addressed in some schemes, it arises a more severe problem in which all the plaintext in the specific positions should be associated with the pattern (keyword) ahead of time. Since the simulator does not know the text, it is

not possible to produce a consistent preprocessed text in the simulation. It is actually unclear how to generate such an indistinguishable view even in the random oracle model [8].

B. Our contribution

We propose an efficient secure protocol for outsourcing the pattern matching problem. This protocol supports wildcards, text and keyword search, and is able to handle the Hamming distance. The protocol is round optimal; only one round (one-way) for outsourcing the text and two rounds (one-way) for searching and there is no need for any more interactions.

Also, we proved that no nontrivial information is leaked to the computing server. We also proposed a technique to outsource the fetching and decryption phase of the protocol, which enables us to achieve optimal communication and computation complexity while preserving the round optimality for the client and the computing server.

II. Basic Concepts

In this section, we give a high level overview on the basic concepts and preliminaries required throughout the paper. Also, we provide the security definitions and the ideal functionality.

A. Bit-Parallel Shift-ADD Algorithm

Our central tool for providing an outsourced pattern matching protocol is Bit-Parallelism [10,11], a powerful family of insecure pattern matching algorithms to represent all possible states of the search at each step, so as to update all of them simultaneously with a few logical and arithmetic operations when observing the next character of the text (next step). All members of this family have the Shift operation is common, though based on the second operation they are instantiated as the Shift-AND, Shift-OR, or Shift-ADD algorithms. In this paper, we used the logic behind the Shift-ADD algorithm. We describe here what is essential to know about the Shift-ADD protocol in this paper with some minor modifications to better fit our requirements; we refer the readers to [11] for more details about Bit-Parallel pattern matching.

Recall that *T* is a text of length *n* and *P* is a pattern of length *m*, both from a finite alphabet Σ. The Shift-ADD algorithm considers m parallel search states and updates all of them simultaneously based on the seen character of the text [10]. Each of these states is the current distance between a pair of corresponding (sub-pattern, sub-string).

Let define the distance between each pair of characters as either zero or non-zero, standing for match or mismatch. Until precise definition of distance matrix *D*, let $d(\sigma_1,\sigma_2)$ be a distance function that takes two characters (σ_1,σ_2) and outputs their defined distance. Further, the distance between two equal length strings is defined as the sum of distances between the pairs of characters at the same positions. This implies that these strings are matched if their distance is zero.

Consider a matrix *S* of size $m \times n$, initialized with zero. Each entry S_j[i] in row i of column j, stores the distance between sub-pattern $p_1,\ldots,p_i$ and sub-string $t_{j-i+1},\ldots,t_j$. Therefore, the last entry of each column, $S_j[m]$, is the distance between the pattern and sub-string $T_j = t_{j-m+1},...,t_j$ of length m.

In a recursive manner, the value of column S_j can be computed solely based on the previous column S_{j-1} and the distance between the seen character at position *j* of the text and each character of the pattern. More precisely we have the following, while initializing all undefined cells $S_0[i]$, $S_j[0]$ with zero.

$$S_j[i] = S_{j-1}[i-1] + d(t_j, p_i), \quad j = 1,\ldots,n \text{ and } i = 1,\ldots,m \tag{1}$$

In other words, after seeing character t_j, the values in S_j are updated by shifting down (assumed as a column) the previous values, while entering a zero from the top, and adding the distance between t_j and each character p_i of the pattern.

In the following, we redefine the Shift-ADD algorithm in three phases. This description is useful in the next sections, where a secure outsourcing protocol is presented.

Phase 1. Pattern preprocessing

In this phase, pattern *P* is represented in the form of a distance matrix (denoted by matrix *D*) of size $m \cdot |\Sigma|$, i.e. a row for each symbol of *P* and a column for symbol of Σ. Each entry of *D* is set to zero if the corresponding symbol of *P* and Σ are equal (match), and non-zero otherwise (mismatch). This matrix is an instantiation of distance function *d* that we used above. We denote by $D[\sigma]$ a column of *D* indexed by character σ.

Phase 2. Text evaluation

We can consider two cases for a text; a stream of characters or a set of keywords. In the first case, we assume the text as an unstructured sequence of characters. While in the keyword case, we assume the text is constructed as a set of keywords. In this section, for the sake of generality, we consider text *T* as a stream of characters. Later, we show how our protocol can be used in the case of keywords.

A matrix of size $m \times n$, denoted by *R*, is used to evaluate the preprocessed pattern over *T*. In each step of the evaluation, in order to realize Equation 1, read character t_j of *T* and compute the *j*th column R_j of *R* as:

1. Initialize column R_j by cloning the shifted elements of R_{j-1} while entering 0 from the top.
2. Add $D[t_j]$ to R_j element-wise.

Phase 3. Result extraction

Any $R_j[m] = 0$ ($R[m, j] = 0$) indicates a match. This is due to the fact that in the computed matrix *R*, the cell $R[k, j]$ indicates the distance between the sub-string $t_{j-k+1},...,t_j$ and

Functionality F_{OPM}

Functionality F_{OPM} runs between Client and Server as follows:

- Upon receiving a message (text, *T*, *tid*) from Client, stores *T* under ID *tid* and sends (outsource, *tid*, |*T*|) to Server.
- Upon receiving a message (query, *P*, *tid*, *qid*) from Client, checks whether it has a text stored under *tid* and sends (*tid*,*qid*,|*P*|) to Server. It then retrieves *T* stored under *tid*, computes *Matched* = {*j*|*P* matches T_j} by searching for pattern *P* at each location in *T*, and stores *Matched* under index (*tid*,*qid*).
- Upon receiving a message (approve, *tid*, *qid*) from Server, retrieves *Matched* stored under (*tid*,*qid*) and sends (result, *tid*, *qid*, *Matched*) to Client. Otherwise, it sends ⊥.

Fig. 1. ideal functionality for secure outsourced pattern matching.

the prefix $p(k) = p_1,...,p_k$ of *P*; the last row ($R[m, j]$) indicates the distance between each substring and the pattern.

B. Security Model and Definition

In this section we proceed with describing the security definitions used in this paper as well as the necessary cryptographic background.

Additively Homomorphic Semantically Secure Encryption. Throughout this paper, we use a public-key encryption scheme $E := (Gen, Enc, Dec)$ with an additive homomorphism and semantic security property. The key generation algorithm Gen outputs a public-private key pair *(pk,sk)*, taking a security parameter 1^{κ}. The encryption algorithm *Enc* outputs a ciphertext *c*, based on inputs *pk* and a plaintext message *M*. The decryption algorithm *Dec* takes a ciphertext *c* and private key *sk* and outputs a decrypted message *M* (or an error).

The additive homomorphism property of this scheme allows addition of the corresponding plaintexts of two ciphertexts by applying an efficient operation, without the need of the *sk* and decryption. This operation is denoted by $Enc_{pk}(X+Y) = Enc_{pk}(X) +_h Enc_{pk}(Y)$ for plaintexts *X* and *Y*. This also implies that the corresponding plaintext *X* of a ciphertext *c* can be multiplied by a known integer, which we denote by $Enc_{pk}(v \cdot X) = v \cdot_h Enc_{pk}(X)$ for a known integer *v*. In practice, $+_h$ and $\cdot_h$ respectively correspond to multiplication and exponentiation; we use this notation in the rest of this paper

We also define the encryption of an array to be an elementwise encryption of it. We can then define the addition of two encrypted arrays of the same dimensions by homomorphically adding their corresponding elements.

Moreover, the semantic security of the scheme implies that the adversary is not able to distinguish one message from another once they are encrypted with more probability that random guessing.

Security Definitions. We follow the ideal/real world paradigm in order to prove the security of our protocol. We use two different definitions of security to preserve efficiency of our protocol in the presence of semi-honest and malicious adversaries. Both definitions follow the simulation of the real world in the ideal world. Roughly speaking, according to simulation paradigm, security is guaranteed if any real world adversary is not able to harm more in comparison to an ideal world adversary. In the ideal world the parties privately send their input to an ideal functionality, where he does the computation and privately returns to each party its defined output. The security in the ideal world is guaranteed by definition. In the real world the parties engage in a protocol to obtain their output. We say that this protocol is secure if for any adversary in the real world there is a simulator in the ideal world that a distinguisher is not able to distinguish between the generated transcripts, i.e. views, of two worlds [17]. For inputs *(x,y)*, we denote the ideal world transcript by *IDEAL* and the real world transcript by *REAL*. Note that in the literature this definition is equivalent to the definition of indistinguishability of views, for semi-honest adversaries [17]. Definition 1 formalizes this security definition.

Definition 1. We say that a two-party protocol π securely realizes functionality *F* if for any PPT adversary *adv* in the real world there exists a PPT simulator *Sim* in the ideal world such that for any tuple of inputs *(x,y)* and auxiliary input *z*,

$$IDEAL_{F,Sim(z)}(x, y) \overset{c}{\approx} REAL_{\pi,\mathrm{adv}(z)}(x, y)$$

Where $\overset{c}{\approx}$ denotes computational indistinguishability.

The second definition of security is helpful when one party has no input and no output in the protocol, the same as our case. In this setting, we relieve the security definition, for the sake of efficiency. Here, the definition requires that the party with no input and output is unable to distinguish between any two inputs of the other party. Note that this definition does not provide correctness of output. This is formulated via indistinguishability of two views of the adversary in the real world for any two inputs, denoted by $VIEW^{adv}_{\pi,adv(z),B}(\mathrm{x}, -)$ and $VIEW^{adv}_{\pi,adv(z),B}(\mathrm{x}', -)$, respectively.

Definition 2. Let *F* be a functionality run between parties *A* and *B* where party *B* has no input and output. We say that a two-party protocol π provides privacy of the input/output of party *A*, according to functionality *F*, if for any input *x* and auxiliary input *z* the following holds:

$$VIEW^{adv}_{\pi,adv(z),B}(\mathrm{x}, -) \overset{c}{\approx} VIEW^{adv}_{\pi,adv(z),B}(\mathrm{x}', -).$$

Ideal Functionality. The inputs of our pattern matching outsourcing problem are a text *T* of length *n* and a pattern *P* of size *m*; Client outsources the text once and may query for

several patterns later. The goal is to find all positions in text in which the pattern matches the sub-string of the same length ending at those positions. This problem is formalized via an ideal functionality in the ideal world, shown in Figure 1.

III. Protocol Specification

Recall that Client has a private text that she stores in an encrypted form with Server. Client later requests Server to search for some private patterns (queries) in Client's text. Besides the trivial information such as the text and the pattern length, the functionality implies that Server should not learn anything about Client's data, i.e. the text or the patterns. We proceed to present a secure protocol that realizes this functionality, which is intrinsically based on the bit-parallel shift-add algorithm.

Roughly speaking, in this protocol, Server stores an encrypted representation of the text which is preprocessed and outsourced by Client. Client later sends a query in the form of encrypted pattern and Server evaluates the query by obliviously computing the difference between the corresponding characters of the pattern and each sub-string of the text, which is solely efficient computing on encrypted data.

Let a mapping Ψ map each character of the alphabet to a distinct integer, i.e. each character of the alphabet is represented using a numerical value denoted by $\Psi(\sigma)$ for character σ. We define the distance between two characters as the difference of their numeric values; the distance function is defined as $d[\sigma_1,\sigma_2] = \Psi(\sigma_1) - \Psi(\sigma_2)$, where σ_1 and σ_2 are characters. More precisely, we realize this definition by adding a NULL symbol $\emptyset$ to the alphabet (typically $\Psi(\emptyset)=0$) and assign an index to each symbol of the alphabet which is treated as its distance from the $\emptyset$ character. Now, the distance of each pair of characters can be computed as the difference of their distances from the $\emptyset$ character:

$$\begin{aligned} d[\sigma_1,\sigma_2] &= d[\sigma_1,\emptyset] - d[\sigma_2,\emptyset] \\ &= \Psi(\sigma_1) - \Psi(\sigma_2) \end{aligned} \quad (2)$$

An instant implication of such definition is that the distance between two same length strings can be the sum of distances of their corresponding characters (Equation 3); two strings are matching if their distance is zero.

$$\begin{aligned} &d[\sigma_1\sigma_2...\sigma_n, \sigma'_1\sigma'_2...\sigma'_n] = \\ &d[\sigma_1,\sigma'_1] + d[\sigma_2,\sigma'_2] + ... + d[\sigma_n,\sigma'_n] \end{aligned} \quad (3)$$

Note that the solution we are describing here is applicable only to the case when two characters/strings are considered either a match or mismatch for the purposes of the search (i.e., the distance matrices are binary) and simple patterns consisting of only alphabet characters. Later, we will demonstrate how to modify this solution to support wildcards as well.

We emphasize that in this approach, by definition, distances between any two given characters can be negative, positive, or zero. Since the algorithm adds the distances between a character of the text and a character of the pattern, it may produce false positives, i.e. total zero distance while some partial distance are non-zero, where positive distances are compensated by negative distances. As a simple example, strings GT and TG is considered as matched since $d[G,T] + d[T,G] = 0$.

We see three options to mitigate this shortcoming: The first option is to ignore it by definition, since in some contexts this kind of match may be useful. The second option is to compute squared distances between two characters prior to adding them together. Implementing this option will necessitate the use of pairing operations [22]. The third option is to randomize non-zero distances over a large space prior to adding them with new ones. More precisely, Server randomizes each entry of the previous column by homomorphically multiplying them by a fresh randomness. This will randomize the non-zero values while making the zero ones unchanged. Since by construction zero indicates a match and any mismatch can be represented by a random value, this will preserve the properties of the solution while making the probability of false positives negligible.

Since the third option can be realized more efficiently than the second one, we use it in our construction.

We proceed with protocol specifications. In order to outsource a text, Client opts a private arbitrary mapping Ψ to index all symbols of the alphabet and the NULL character, e.g. $\{\emptyset = 0, A = 1, C = 2, T = 3, G = 4\}$. The mapping function can be even a Pseudo Random Function (PRF) [23]. Then, using a semantically secure non-deterministic additively homomorphic encryption scheme, she substitute each character of the text with an encryption of its corresponding index, resulting in vector C. Client stores this encrypted representation of the text with Server, while keeping the key and the mapping locally.

Note that the text is not needed anymore to be kept locally since it can be restored by downloading it from Server and decrypting.

After outsourcing the text to Server, whenever Client wants to search for a private pattern P in her text, she simply creates a vector of size m and set each of its elements to $\Psi(\emptyset) - \Psi(p_i)$, for each character p_i of P. Then she encrypts this vector element-wise and sends resulted vector $D^{\emptyset}$ to Server.

Server is now able to evaluate the pattern, in form of vector $D^{\emptyset}$, on encrypted text C inspired by the fact that $C[j] +_h D^{\emptyset}[i]$ is the distance between jth character of the text and ith character of pattern P, in an encrypted form. In other words, Server is able to obliviously compute the randomized sum of the distances of each character of the text with the corresponding characters of P, which can be used to compute the distance of each sub-string with the pattern.

Protocol π_{OPM}

This protocol runs between Client and Server. Client outsources a text $T \in \Sigma^n$ and later may query patterns $P \in \Sigma^m$ for a finite alphabet Σ. The distances between the alphabet character and pattern characters are considered to be binary.

Setup phase.

– *Setup*(1^κ): Client chooses a semantically secure additively homomorphic encryption scheme E = (*Gen*,*Enc*,*Dec*) and runs the *Gen*(1^κ) to produce a key pair (*sk*,*pk*). She publishes *pk*. Client also creates a mapping $\Psi : \Sigma \cup \emptyset \rightarrow Z$ (typically $\Psi(\emptyset) = 0$).

– *GenText*(*pk*,*T*): Client substitute each character t_j of T with $c_j = Enc(\Psi(t_j) - \Psi(\emptyset))$ and stores $C = \{c_j | j = 1,...,n\}$ with Server.

Query phase.

– *GenPattern*(*pk*,*P*): Client creates an empty vector $D^\emptyset$ of size $m = |P|$ and fills the ith element $D^\emptyset[i]$ with $Enc_{pk}(\Psi(\emptyset) - \Psi(p_i))$, where p_i is the ith symbol of pattern P. Client then sends $D^\emptyset$ to Server.

– *Eval*(*C*,$D^\emptyset$,*pk*): For each encrypted character c_j, Server maintains a vector R_j of size $m = |P|$.

For each j, Server randomly samples r_j from the plaintext space of *Enc* and computes:

$R_j[1] = c_j +_h D^\emptyset[1]$

$R_j[k] = r_j \cdot_h (R_{j-1}[k-1]) +_h c_j +_h D^\emptyset[k]$ for $2 \le k \le m$

$result_j = R_j[m]$

Extract phase.

– *Extract*(*Res*,*sk*): Server sends *Res* = $\{result_j | j = 1,...,n\}$ to Client. Client parses *Res* as $\{result_j | j = 1,...,n\}$ and decrypts each $result_j$. Client returns all indices j as matching locations where $Dec_{sk}(result_j) = 0$.

Fig. 2. Secure protocol for outsourced pattern matching in the semi-honest model.

More precisely, Server creates matrix R of size $m \times n$, initialized with zero. In each step j of the evaluation, Server needs to compute the distance of each sub-string of the text ending at position j and each prefix of the pattern. To this end, Server reads next encrypted character c_j of the encrypted text, homomorphically adds it to each element of vector $D^\emptyset$, randomizes the resulted vector, shifts down R_{j-1}, while entering an encryption of zero from the top, and finally adds the randomized vector with R_{j-1} to obtain vector R_j. Note that Server is able to compute an encryption of zero since he has the public key of Client, however he is not able to distinguish any encrypted value from another since the encryption scheme is semantically secure.

As discussed earlier, the last row of matrix Result at position j, indicates the randomized sum of the pattern with the sub-string of the same length ending at position j. This row should be sent back to Client, where she can decrypt and find if there is any matched position.

IV. Security Evaluation

The detail of the construction are given in Figure 2. In this construction, all necessary operation performed by Server is done without allowing him to obtain any information about the outsourced text, pattern, or the distances. Thus, we obtain the following:

Theorem 1. Given an additively homomorphic semantically secure encryption scheme *(Gen,Enc,Dec)*, protocol π_{OPM} in Figure 2 securely realizes the functionality F_{OPM} in Figure 1 in the presence of semi-honest adversaries.

Proof. We show both correctness and privacy.

Correctness. We need every single pattern P queried by Client is correctly being answered, with overwhelming probability, with respect to the outsourced text T. Our protocol achieves correctness if and only if any time $T_j = P$ it returns the position j as a match. We next show that the algorithm in Figure 2 returns all indices j when $T_j = P$.

The server computes $result_j$ for each index j of T. As shown in Equation 4 below, $result_j$ is the randomized sum of the distances of each character of T_j with the relevant characters of P. The distances will add to 0 if all of them are actually 0, which indicates a match. With a negligible probability, $result_j$ may be add to 0, while $T_j \neq P$, which is a false positive.

$$\begin{aligned} result_j &= State_j[m] \\ &= (r_j \cdot_h State_{j-1}[m-1]) +_h c_j +_h D^\emptyset[m] \\ &= Enc(0) +_h r_{j-m+2} \cdot_h (c_{j-m+1} +_h D^\emptyset[1]) +_h \cdots \\ &\quad r_j \cdot_h (c_{j-1} +_h D^\emptyset[m-1]) +_h (c_j +_h D^\emptyset[m]) \\ &= Enc\ (r_{j-m+2}.(\Psi(t_{j-m+1}) - \Psi(p_1)) + ... \\ &\quad +(\Psi(t_j) - \Psi(p_m))) \\ &= \begin{cases} Enc(0) & if \quad T_j = P \\ Enc(random) & if \quad T_j \neq P \end{cases} \end{aligned} \tag{4}$$

This concludes the correctness part of the proof.

Privacy. In order to show the security of the protocol, we construct two simulators Sim_S, Sim_C for adversarial Server and Client, respectively. Without loss of generality we assume that the alphabet is public.

Case 1: Adversarial Client Since Server has no input in this protocol, the output of Client is solely dependent on her own input. Sim_C for adversarial Client is naïve; Sim_C executes the same instructions as Server does in the real world.

Case 2: Adversarial Server Since the adversary controlling Server learns no information about the pattern and the text and observes only encrypted values, the simulation is straightforward:

- Sim_S initially runs $(sk^0, pk^0) \leftarrow KeyGen(1^\kappa)$ and sends pk^0 to *B*.
- Upon receiving a message (outsource, *tid*, *n*) from the trusted party (i.e., Client wants to outsource text *T* of length *n*), Sim_S generates an arbitrary text *T'* of length *n* from alphabet $\sum$ and stores it under ID *tid*. Sim_S runs $C' \leftarrow GenText(pk,T)$ and sends *C'* to Server.
- Upon receiving a message (*tid*, *qid*, *M*) from the trusted party, Sim_S generates an arbitrary pattern *P'* of length *m* from Σ. Sim_S then runs $D_{p'}^{\varnothing} \leftarrow GenPattern(pk', P')$ for this pattern and sends vector $D_{p'}^{\varnothing}$ to Server.

To show that the two views are indistinguishable, consider that it is possible to distinguish between them, namely, $(C, D_p^{\varnothing})$ and $(C', D_{p'}^{\varnothing})$. It is obvious that all of these elements are ciphertexts of a semantically secure encryption scheme. Thus, any non-negligible advantage in distinguishing the views can be used to build a distinguisher for breaking security of the encryption with a non-negligible probability. This contradicts semantic security of the encryption scheme. This concludes the proof.

Before we conclude this section, we comment on the security of this protocol in the presence of malicious adversaries. The protocol of Figure 2 in its current description is not able to provide correctness against an adversarial server. Correctness is achievable using the known techniques in the literature but will burden the communication and computational complexity of the protocol.

On the other hand, since the adversarial server has no input to the protocol and receives no output as well, it is rather simple to define and prove the privacy of the protocol against a malicious server. We obtain the following:

Theorem 2. Given an additively homomorphic semantically secure encryption scheme *(Gen,Enc,Dec)*, protocol π_{opm} in Figure 2 preserves the privacy of the pattern, outsourced text, and matching results, according to functionality F_{OPM} in Figure 1 in the presence of a malicious server.

Proof. Our proof for privacy against a malicious server is based on the indistinguishability of two inputs based on Definition 2. More precisely, based on this definition, if the adversary is not able to distinguish between two different inputs, he gains no non-trivial information about the input and output. Intuitively, the indistinguishability of the views for two different sets of inputs (pattern/text) is evident, since the encryption scheme is semantically secure; formally, we show that if the adversary is able to distinguish between two inputs, we can use him to build a distinguisher that distinguishes between two plaintexts once they are encrypted using a semantically secure encryption scheme.

We use three views in our proof; $V = VIEW((\mathrm{P},\mathrm{T}),-)$ is the view for pattern P and text T, $\mathrm{V}_2 = \mathrm{VIEW}''((\mathrm{P},\mathrm{T}),-)$ is the view for the same pattern and text with different randomness, and $V_1 = VIEW((\mathrm{P}',\mathrm{T}'),-)$ is the view for pattern P' and text T' of the same lengths:

$$V = VIEW((\mathrm{P},\mathrm{T}),-) := \{\mathrm{V}_l \mid l = 1,..., \mathrm{n}+\mathrm{m}+\mathrm{m}.\mathrm{n}\}$$
$$= \{\mathrm{c}_1,..., \mathrm{c}_n, D^{\varnothing}[1],..., D^{\varnothing}[\mathrm{m}], R_1[1],..., R_n[\mathrm{m}]\},$$
$$V_1 = VIEW'((\mathrm{P}',\mathrm{T}'),-) := \{\mathrm{V}'_l \mid l = 1,..., \mathrm{n}+\mathrm{m}+\mathrm{m}.\mathrm{n}\}$$
$$= \{\mathrm{c}'_1,..., \mathrm{c}'_n, D^{\varnothing'}[1],..., D^{\varnothing'}[\mathrm{m}], R'_1[1],..., R'_n[\mathrm{m}]\},$$
$$V_2 = VIEW''((\mathrm{P},\mathrm{T}),-) := \{\mathrm{V}''_l \mid l = 1,..., \mathrm{n}+\mathrm{m}+\mathrm{m}.\mathrm{n}\}$$
$$= \{\mathrm{c}''_1,..., \mathrm{c}''_n, D^{\varnothing''}[1],..., D^{\varnothing''}[\mathrm{m}], R''_1[1],..., R''_n[\mathrm{m}]\}$$

Server sees V in a protocol run. Then, Client engages in another run with Server using either (P, T) or $(\mathrm{P}',\mathrm{T}')$ as input and Server will see view $\mathrm{V_b}$. Server is able to distinguish between two views if he can guess b with more probability than random guessing. Consider that Server is able to distinguish between these views, which means that Sever is able to learn if $\mathrm{V}=\mathrm{V}_2$ or $\mathrm{V}\neq\mathrm{V}_1$, where notation = means that the decrypted value of each element of these sets at the same position is equal and ≠ means that they differ in at least one position.

We have two cases: in the first case, if Server is able to learn that $\mathrm{V}=\mathrm{V}_2$, he is able to learn that for every position $l = 1,..., \mathrm{n}+\mathrm{m}+\mathrm{m}.\mathrm{n}$, $\mathrm{V}_l = \mathrm{V}''_l$. This is in contradiction with semantic security of the encryption scheme; however the inputs for these two views are identical, Client used a fresh randomness for each of them and the values V_l and V''_l are uniformly distributed in the range of the encryption scheme.

In the second case, Server is able to learn that there exist at least one position α where $\mathrm{V}_\alpha \neq \mathrm{V}'_\alpha$. This is also in contradiction with the semantic security of the encryption scheme. This concludes the proof of Theorem 2.

A. Variants

Before we conclude this section, we comment on the applicability of this construction to some different variants of pattern matching; the Hamming distance computation, wildcards, and the keyword search.

An important result of pattern matching that one may seek is the Hamming distance [19]. In this case, the evaluator is to know how many positions of a pattern and a sub-string of the same length differ. Binary or non-binary Hamming distance is referred to the cases that the alphabet is binary or non-binary, respectively.

Using the proposed protocol, computing the Hamming distance (in both cases) is not straightforward since Client uses virtual distances instead of actual ones. However, the client is able to learn the Hamming in some cases by using well-defined indices. Also, in this regard, Client can send to Server a power of ten that is greater than the maximum distance in vector $D^{\varnothing}$. Server exponentiates the previous distance (shifted element of matrix *R*) with it, prior adding the currently computed distance. Also, the randomization step is eliminated. Using this modification, Client is able to compute the Hamming distance simply after decrypting the returned result;

zeros indicates a match, while non-zero values can repeatedly divide on that power of ten, while any non-zero factor means a character mismatch in the corresponding position.

Another important variant of pattern matching is the case of wildcards (don't care). A wildcard character in a pattern matches any character of the alphabet. For the sake of efficiency, we consider two cases. In the first case, Server learns from Client the positions of wildcards in the pattern. This leakage should be reflected in the ideal functionality. In each step of the search, after computing the distance of characters, Server simply substitutes those positions with an encryption of zero and continues the evaluation like before. This substitution compensates the distance of those positions to zero.

In the second case, Client considers the wildcard positions as private. Client may use a masking mechanism to void the effect of those positions on the search by compensating their distance to zero. This mechanism leaves the ideal functionality unchanged, but comes at the price of pairing operation. This mask is a vector of length m set with encryptions of zero in wildcard positions and with encryptions of one elsewhere. In each step of the search, Server multiplies this mask vector to the computed vector of distances, using a pairing operation, prior adding that vector to the shifted previous distances.

The other variant that we would like to emphasize is the case of keyword search. In the protocol, we assumed the text as a stream of characters of size n, where we refer to it as the text search. In keyword search case, we assume Client stores with Server a set of ω keywords of different lengths (a simpler variant is when the lengths are equal). The ideal functionality can simply be modified to reflect this case. In the protocol, Client outsources each keyword as she outsources a text. Server evaluates each keyword as he evaluates a text in the protocol. In this case, those keywords which are shorter that the pattern are eliminated from the evaluation. For each keyword $w_l, l = 1, \dots, \omega$ of length $|w_l| \geq m$ Server evaluates the keyword; this means that in this case one can search for a pattern inside the longer keywords. Moreover, it is only necessary to compute the last $|w_l| - m$ elements of matrix R and send them to Client, which reduces the computational and communication complexity of the search in practice.

B. Outsourcing of Decryption

An important topic that we would like to bring up is fetching and extracting the results by the client. We discussed two cases of text search and keyword search. In both cases, Server learns no information about the text/keywords, pattern, and matched position/keywords; however, this privacy is obtained with a price.

In the text search case, Server evaluates the result as an encrypted vector of size$O(n)$, which requires $O(n.\kappa)$ communication and $O(n)$ decryptions from Client in order to learn the matched positions. This seems to be *inefficient* especially when m is small; however, the word "inefficient" is fairly misleading. In theory, as we mentioned in Section 1, it is not possible to achieve sublinear communication complexity without any leakage about the pattern or the text to the server. So, the minimum theoretical bound in this case is $O(n.\kappa)$. On the other hand, this complexity is as large as the following trivial solution:

- Outsourcing: Client stores the encrypted text (using a symmetric or an asymmetric scheme) with Server.
- Query: Client downloads the text (as a whole or part by part), decrypts it, and searches for the private pattern. (Note that in this solution, Client must search for the pattern herself which implies the search complexity for Client)

Therefore, one may argue that the secure outsourcing of pattern matching without any leakages is absurd, due to the same communication complexity as this trivial solution; however, the communication complexity of our protocol can be mitigated using the following well-known technique (Note that this technique is not applicable to the trivial solution and necessitates a multi-party searching protocol).

In order to mitigate this problem, Client could also outsource the result extraction phase (fetching and decryption) to a set of servers. We suggest the following modifications: Client uses a (k,k)-threshold cryptosystem and shares the private key between k servers. Also, Client chooses a fresh key pair (pk_2,sk_2) and sends pk_2 to Server. Server encrypts each index j using public key pk_2, creates set ER of pairs $M_j=(Enc(j), result_j), j=1,\dots,n$, and sends it to decryption servers. These k servers re-randomize the encryptions and randomly permute the set members in a distributed way. Finally they jointly decrypt the $result_j$ of each member and send the $Enc(j)$ part to Client whenever $Dec(result_j)=0$. Client can simply decrypt $Enc(j)$ and find the matched positions. In this case, we achieve optimal communication complexity for Client which is linear in the number of matches.

Note that we have been able to merge this decryption protocol with protocol π_{OPM}, but intentionally avoided it; because, it can be used as a black-box, its security properties is different from π_{OPM}, someone may want to change it without effecting the whole protocol, and this step is not necessary in keyword search case.

In the keyword case, communication complexity is $O(\omega_m.\kappa)$ where ω_m is the number of keywords with length equal to or longer than the pattern (considering that all differences of their lengths with the pattern's length are limited to a constant). In other words, for each keyword with the same length as the pattern only one encrypted element should be transferred and decrypted. Therefore, this protocol could be considered as an efficient protocol in the keyword search case in practice. Also, one may decide to outsource the decryptions as well.

V. Performance Analysis

Table 1 shows the communication and computational complexity of our protocol. As shown, the communication complexity of the protocol can be optimized to the order of number of matches, if Client chooses to outsource the decryption as well. Also, the round complexity of the protocol is one round (one-way) for outsourcing and two rounds (one-

TABLE I
COMPLEXITY OF PROTOCOL Π_{OPM} IN THE TEXT AND KEYWORD SEARCH SETTING. μ IS THE NUMBER OF MATCHED POSITIONS.

	Client		Server		
	Enc.	Dec.	Exp.	Mult.	Bandwidth
Out. phase	$O(n)$	-	-	-	$O(n.\kappa)$
Query phase	$O(m)$	-	$O(m.n)$ $O(m.\omega)$	$O(m.n)$ $O(m.\omega)$	$O(m.\kappa)$
Dec. Phase	-	$O(n)$ $O(\omega)$	-	-	$O(n.\kappa)$ $O(\omega.\kappa)$
Dec. Phase (outsourced)	-	$O(\mu)$	$O(n)$ or $O(\omega)$ partial decryption for each decryption server.		$O(n.\kappa)$ or $O(\omega.\kappa)$ for decryption servers. $O(\mu)$ for Client

way) for the rest of the protocol.

Before continuing with the implementation result of our scheme, we like to compare the security and performance of our protocol with one of the most recent works [8] which claims to be the first work considering the outsourced pattern matching problem. The main advantage of [8] over our work is that it can be used directly to delegate the search capability to other parties, while we did not consider this setting in this paper. On the other hand, our protocol outstands [8] in several properties. Table 2 summarizes some of these properties.

On the other hand, computational complexity of [8] during query phase is $O(n.m^2)$ modular multiplications and $O(n.\mathrm{m})$ modular exponentiations. The communication complexity of [8] is reduced to be linear in the number of matches since it allows the server to learn the matching positions, as otherwise there is no way to be achieved.

TABLEII
COMPARISON BETWEEN SOME FUNCTIONAL PROPERTIES OF [8] AND OUR PROTOCOL

[8]	Our protocol
Alphabet is binary	Alphabet is arbitrary with negligible effect on the performance
Text must be fully known before outsourcing	Text can be change or grow even during the query phase
Patterns must be predicted before outsourcing	No information is needed about the pattern before query phase
Only keyword search	Text search and keyword search
Maximum size of the pattern must be predicted before outsourcing	No limitation for pattern size
Small patterns cannot be used due to vulnerability to brute force attack	Security does not depend on the pattern size; any size can be queried.
Simple predicted patterns can be queried	More complex patterns, e.g. wildcard, can be queried
Security depends on several conflicting parameters (there is a trade-of between them)	Security depends on the security parameter of the encryption scheme
Several kinds of information leakage, e.g. some repetitions and matching positions can be learnt, some un-queried patterns can be queried.	No information leakage regarding the Ideal Functionality.

Table 3 and Table 4 show performance results of the implementation of the proposed protocol in the text search and keyword search respectively. These experiments were performed on two Intel Core i7 (3770K) 3.5GHz machines with 4GB of memory running Ubuntu 14.04. These machines were connected via Ethernet (100Mbps). Implementation was done as a single-threaded program in C++ utilizing libpaillier library [24] for Paillier Cryptosystem [25] using 1024-bit key. We used random patterns and texts, and tested alphabets of sizes 100 (printable) and 2 (binary). The protocol's computational complexity does not depend on the alphabet size, as shown in the tables.

TABLE III
PERFORMANCE RESULTS OF Π_{OPM} FOR DIFFERENT SETTING OF ALPHABET, PATTERN, AND TEXT. ALL THE TIMES ARE WALL-CLOCK TIME ROUNDED AND SHOWN IN SECONDS.

$\lvert\Sigma\rvert$	$\lvert P\rvert$	$\lvert T\rvert$	Out. (sec)	Query (sec)	Dec. (sec)	B.W. (bit)
100	1K	100K	1070	750	1000	200M
100	1K	10K	105	60	100	18M
100	1K	1K	11	0.007	10	2K
100	100	100K	1070	60	1000	200M
100	100	10K	105	7.5	100	18M
100	100	1K	11	0.75	10	1.7M
100	100	100	1	0.000	1	2K
100	10	100K	1070	7.5	1000	200M
100	10	10K	105	0.75	100	19M
100	10	1K	11	0.075	10	1.8M
100	10	100	1	0.0075	1	180K
2	1K	100K	1070	750	1000	200M
2	100	100K	1070	60	1000	200M
2	10	100K	1070	7.5	1000	200M

A. Practical Optimizations

Note that the outsource phase is run once, it can be executed offline, and can benefit from precomputation techniques, e.g. we can precompute random numbers and their exponentiations required in the Paillier cryptosystem. Also, since the characters of each alphabet are known, exponentiation of each character can be computed once and reused in every position of the text that this character appears (randomness is different in every repetition).

Also, almost all steps of the protocol can be parallelized. Bit-Parallel Shift-ADD algorithm can benefit from parallelism in the instruction level. This is due to the fact that in each step of the search all new states are independent of each other and can be computed simultaneously. The proposed protocol inherits this property in the thread-level and can be parallelized to run on multiple cores simultaneously; each new state can be computed independently and simultaneously on a separate thread running on a separate CPU core. Parallelism is also applicable on different portions of the text at the same time. This implies that the computation can be parallelized by a factor of number of cores.

Another optimization is related to the memory consumption of the protocol. We mentioned that Server creates a matrix of size $m \times n$, in order to compute the states of the search in the query phase. As another inherited property of Shift-ADD algorithm, the new states in each step of the search are merely

TABLE IV
PERFORMANCE RESULTS OF Π_{OPM} FOR DIFFERENT PATTERN SIZES AND DIFFERENT NUMBER OF KEYWORDS. ALL THE TIMES ARE ROUNDED WALL-CLOCK TIME SHOWN IN SECONDS. WE ASSUMED ω KEYWORDS WITH THE SAME LENGTH OF THE PATTERN. ALSO, WE FIXED THE ALPHABET SIZE TO 100 CHARACTERS.

$\|P\|$	ω	Out. (sec)	Query (sec)	Dec. (sec)	B.W. (bit)
100	10K	10000	7500	100	20M
100	1K	1000	750	10	2M
100	100	100	75	1	200K
100	10	10	7.5	0.1	20K
10	10K	1000	750	100	20M
10	1K	100	75	10	2M
10	100	10	7.5	1	200K
10	10	1	0.7	0.1	20K

dependent on the states of the previous step and the current character of the text. This property can be utilized in order to reduce the required dynamic memory from a matrix of size $m \times n$ into two vectors of size m, without any negative effect on the computation performance.

VI. FUTURE WORKS

Before concluding the paper, we would like to bring up some further directions to follow our work. First, extending this work in order to support secure delegation of searching is very interesting. More precisely, this work considered the case that they only person who wants to search in the outsourced text is the outsourcer; however, she may want to delegate her search ability to other parties. This extension requires, at least, a protocol between the outsourcer and a third party in order to create a search token without revealing the third party's pattern to the outsourcer.

Another interesting direction, that we may follow, is to pack vector $D^{\emptyset}$ into a single encrypted value. More precisely, in this paper we used a vector of encrypted values, while we can pack them together and use a single encrypted value representing that vector. This would liken the Shift-ADD algorithm and our protocol even more, reducing the server-side computation complexity from $O(m.n)$ into $O(n)$, which is very interesting.

VII. CONCLUSION

In this work, we studied the problem of secure outsource pattern matching. To this end, we present a two round (one-way) protocol that enables a client to outsource her text or set of keywords to an untrusted server and search for her patterns in the outsource database later. Our construction does not force the outsourcer to specify her pattern when preprocessing the text or set of keywords, then she is able to search for any unpredicted pattern later. The scheme is secure against any semi-honest adversaries and retains the privacy of the pattern, text, and matched positions even in the presence of malicious adversaries. Unlike other relevant schemes, ours does not reveal any statistical information about the text, e.g. repetitions in the text. Also, this is the first scheme to allow wildcard search.

REFERENCES

[1] J. R. Troncoso-Pastoriza, S. Katzenbeisser, and M. Celik, "Privacy preserving error resilient DNA searching through oblivious automata," in the 14th ACM Conference on Computer and Communications Security. ACM, 2007, pp. 519–528.

[2] M. Sipser, Introduction to the Theory ofComputation. Thomson Course Technology Boston, 2006, vol. 2.

[3] R. Baeza-Yates and G. H. Gonnet, "A new approach to text searching," Communications of the ACM, vol. 35, no. 10, pp. 74–82, 1992.

[4] C. Hazay and Y. Lindell, "Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries," Journal of Cryptology, vol. 23, no. 3, pp. 422–456, 2010.

[5] C. Gentry, A fully homomorphic encryption scheme. Stanford University, 2009.

[6] W. Du and M. J. Atallah, "Protocols for secure remote database access with approximate matching," in E-Commerce Security and Privacy. Springer, 2001, pp. 87–111.

[7] M. J. Atallah, F. Kerschbaum, and W. Du, "Secure and private sequence comparisons," in the 2003 ACM workshop on Privacy in the Electronic Society. ACM, 2003, pp. 39–44.

[8] J. Baron, K. El Defrawy, K. Minkovich, R. Ostrovsky, and E. Tressler, "5PM: Secure pattern matching," Journal ofComputer Security, vol. 21, no. 5, pp. 601–625, 2013.

[9] R. Gennaro, C. Hazay, and J. S. Sorensen, "Automata evaluation and text search protocols with simulation-based security," Journal ofCryptology, pp. 1–40, 2010.

[10] C. Hazay and T. Toft, "Computationally secure pattern matching in the presence of malicious adversaries," Journal of Cryptology, vol. 27, no. 2, pp. 358–395, 2014.

[11] S. Faust, C. Hazay, and D. Venturi, "Outsourced pattern matching," in International Colloquium on Automata, Languages, and Programming. Springer, 2013, pp. 545–556.

[12] L. Wei and M. K. Reiter, "Third-party private DFA evaluation on encrypted files in the cloud." in ESORICS, vol. 7459. Springer 2012, pp. 523–540.

[13] S. Faro and T. Lecroq, "Twenty years of bit-parallelism in string matching," Festschrift for Borivoj Melichar, pp. 72–101, 2012.

[14] F. Yergeau, "UTF-8, a transformation format of ISO 10646," 2003. 15. B. Applebaum, Y. Ishai, and E. Kushilevitz, "From secrecy to soundness: Efficient verification via secure computation," Automata, languages and Programming, pp. 152–163, 2010.

[15] B. Applebaum, Y. Ishai, and E. Kushilevitz, "From secrecy to soundness: Efficient verification via secure computation," Automata, languages and Programming, pp. 152–163, 2010.

[16] K.-M. Chung, Y. T. Kalai, and S. P. Vadhan, "Improved delegation of computation using fully homomorphic encryption." in CRYPTO, vol. 6223. Springer, 2010, pp. 483–501.

[17] R. Gennaro, C. Gentry, and B. Parno, "Non-interactive verifiable computing: Outsourcing computation to untrusted workers," Advances in Cryptology–CRYPTO 2010, pp. 465–482, 2010.

[18] C. Hazay and Y. Lindell, Efficient secure two-party protocols: Techniques and constructions. Springer Science & Business Media, 2010.

[19] J. C. Lagarias and A. M. Odlyzko, "Solving low-density subset sum problems," Journal of the ACM (JACM), vol. 32, no. 1, pp. 229–246, 1985.

[20] C. B¨osch, P. Hartel, W. Jonker, and A. Peter, "A survey of provably secure searchable encryption," ACM Computing Surveys (CSUR), vol. 47, no. 2, p. 18, 2015.

[21] D. X. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in Security and Privacy, 2000. S&P 2000. Proceedings. 2000 IEEE Symposium on. IEEE, 2000, pp. 44–55.

[22] S. D. Galbraith, K. G. Paterson, and N. P. Smart, "Pairings for cryptographers," Discrete Applied Mathematics, vol. 156, no. 16, pp. 3113–3121, 2008.

[23] J. H˚astad, R. Impagliazzo, L. A. Levin, and M. Luby, "A pseudorandom generator from any one-way function," SIAM Journal on Computing, vol. 28, no. 4, pp. 1364–1396, 1999.

[24] J. Bethencourt, "Pallier library," 2010. [Online]. Available: http://acsc.cs.utexas.edu/libpaillier

[25] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in Advances in Cryptology-EUROCRYPT99. Springer, 1999, pp. 223–238.