Practical Private Regular Expression Matching

Florian Kerschbaum

SAP Research
Karlsruhe, Germany
Florian.Kerschbaum@sap.com

Abstract. Regular expressions are a frequently used tool to search in large texts. They provide the ability to compare against a structured pattern that can match many text strings and are common to many applications, even programming languages. This paper extends the problem to the private two-party setting where one party has the text string and the other party has the regular expression. The privacy constraint is that neither party should learn about the input of the other party, i.e. the string or the regular expression, except the result of the computation which is wether the string matches the regular expression or not. Secure Multiparty Computation provides general algorithms for any such problem, but it has been recommended to develop special protocols for important cases that provide better performance using the domain knowledge of that problem. This paper presents two protocols: One with perfect secrecy that provides a lower-bound on protocols using circuit construction and a fast one that provides better performance bounds, but the secrecy it provides is limited and tuned for practical applications. The fast protocol presented here uses permutation and commutative encryption as its only building blocks.

1 Introduction

Regular expressions provide a very powerful tool to search in text databases. They can search for structured patterns matching up to an infinite number of strings. Therefore they also have often been used in genome database searches [4, 12]. Especially in genome searches there may be a privacy requirement, since the search pattern may reveal sensitive data, such as the searcher's identity, predisposition for certain health risks, or simply patented material. Also the database may need protection, since the data can be economically valuable and privacy–sensitive, because of person or health–related data. A private regular expression search protects searcher and database equally well.

Regular expressions are also used in programming languages [26]. Regular expressions can be extended with powerful matching techniques using special algorithms. In this paper the capabilities following in table 1 have been considered.

The regular expression $a?(b|c)(b|c)*$ matches any string that is constructed from b and c only, but may begin with an optional a, e.g. c, $bbbccc$, abc and many more. Regular expression matching is to determine whether a given string can be constructed from the regular expression.

Imagine a setting where one party has the regular expression and another party has the string, but they don't want to exchange their information. A common situation

Please use the following format when citing this chapter:
Author(s) [insert Last name, First-name initial(s)], 2006, in IFIP International Federation for Information Processing, Volume 201, Security and Privacy in Dynamic Environments, eds. Fischer-Hubner, S., Rannenberg, K., Yngstrom, L., Lindskog, S., (Boston: Springer), pp. [insert page numbers].

Table 1. Symbols for regular expressions.

Symbol	Description
a	Matches the character exactly: "a"
ab	Matches the character sequence in this order: "a" then "b", so "ab"
$a?$	Matches the character zero or one time: "" or "a"
$a*$	Matches the character zero to infinite times: "", "a", "aa", ...
$a\|b$	Matches either of both characters: "a" or "b"
(x)	Groups the pattern x, such that it can be used in subsequent operations

would be where one party has a text database and the other party wants to search for the occurrences of a regular expression. The privacy requirement is that neither learns about the input of the other party except what can be inferred by the result. If Alice has the string and Bob the regular expression, Alice should not learn the regular expression, its size or composition and Bob should not learn the string or its characteristics. They should only learn whether the regular expression matches the string or not. Secure Multiparty Computation provides the ability to solve any such problem by constructing a binary circuit and executing it privately, but it is recommended to find specific solutions to important problems. The solutions presented in this paper uses only permutation, randomization and commutative encryption. No circuit emulation techniques or slow, e.g. homomorphic, encryption are being used as building blocks. As the commutative encryption scheme RSA [22] with a common modulus could be used. The very similar Pohlig-Hellman encryption [21] can be used, as well. The common modulus attack [10] is not possible, since public keys are never exchanged during the protocol. Semantically secure (probabilistic) encryption is not needed, since the plain text will be randomized when required. To learn more about RSA and commutative encryption, see [23].

A regular expression can be converted (via a non deterministic finite automaton) into a deterministic finite automaton (DFA). The DFA has a transition matrix M with a row for each state and a column for each symbol in the alphabet Σ. Each field contains the number of the state being transited to. Each state has an attribute attached to it: $accept$ or $non - accept$. Accepting states indicate a successful match, while non-accepting states indicate that the sub-string up to the current position does not match. The result of our computation is whether the automaton is in an accepting state at the last character of the input string. Each automaton has a start state. The start state can already be an accepting state and is the first state of each matching. The protocol presented here can be used to evaluate any DFA in a private manner, where one party has its input and the other party has the automaton. For further information about regular expressions and automatons, the reader is referred to [18].

The next section describes previous work in private computations and secure multiparty computation. The third section describes notation. Then follow the protocols: First the perfect-secret one, then the fast, yet not so secure protocol as the main result of the paper.

2 Related Work

Secure Multi-Party Computation (SMC) was introduced in [27]. SMC computes a function $f(a, b)$ where Alice has input a and Bob has input b, such that Alice learns nothing about b, except what can be inferred from $f(a, b)$ and a, and Bob learns nothing about a, except what can be inferred from $f(a, b)$ and b. Any such problem can be solved using a trusted third party, but the goal of SMC is to replace that third party by a protocol between Alice and Bob. Clever general solution to SMC are described in [3, 8, 16, 17]. In [16] a method to transform any protocol secure in the semi-honest setting into a protocol secure in the malicious setting is provided. This solution emulates a circuit constructed for function $f(\cdot, \cdot)$ and executes that circuit privately. Since circuit emulation is slow, [15] suggests to find faster solutions for important problems. One such problem is Private Information Retrieval (PIR). In PIR Alice has a set of data items v and Bob has an index i. Bob retrieves v_i and learns nothing about the other data items and Alice does not learn i. Several elegant techniques for PIR exist that provide better performance than circuit emulation [6, 9, 14, 20]. There is also a large amount of literature on other specific problems. Recently, protocols for database operations such as join, have been published [1]. These protocols provide perfect-secrecy, and use techniques such as commutative encryption and permutation similarly to the presented protocol. In [24] schemes for searching on encrypted data have been presented. They provide perfect secrecy with cleverly adapted encryption schemes, but allow only for exact searches of entire words in an outsourcing model. Not computationally-secure protocols for mathematical problems have been presented in [5, 11]. These provide solutions for secure dot-product and division. A fast protocol with weak security requirements has been recommended for privacy-preserving data mining in [25]. Similar protocols exist for many algorithmic problems, e.g. binary search [13] or edit distance [2].

3 Notation

Let $E(x)$ denote the encrypted value of x. Encryption by Alice and Bob is denoted as $E_A(\cdot)$ and $E_B(\cdot)$, respectively. All encryption is done using a commutative encryption scheme, i.e. $E_A(E_B(x)) = E_B(E_A(x))$. Similarly decryption is denoted by $D_A(\cdot)$ and $D_B(\cdot)$.

Permutations are denoted as Π. Application of permutation is denoted as $\Pi(\cdot)$. Π^{-1} denotes the inverse permutation that returns the items to their original ordering: $\Pi^{-1}(\Pi(\boldsymbol{v})) = \Pi(\Pi^{-1}(\boldsymbol{v})) = \boldsymbol{v}$. Permutation is not commutative, i.e. $\Pi_A(\Pi_B(\boldsymbol{v})) \neq \Pi_B(\Pi_A(\boldsymbol{v}))$.

Let $M, M^1, M^2, M^3, \ldots$ denote matrices and $m^k_{i,j}$ denote the i-th element of the j-th row of matrix M^k. Let $(v_1, v_2, ..., v_n)$ denote vector $\boldsymbol{v}$ of length n with elements v_1 through v_n.

A mapping Ψ is a set of pairs $\{(a, b), \ldots\}$ where $a \mapsto b$. It is not required that a be unique.

Let the variable t range over the states of the automaton.

4 Protocols

4.1 Setup

In the remainder of this paper, it will be assumed that Alice has a string $\boldsymbol{x} = (x_1, x_2, x_3, ..., x_l)$ of length l and Bob has the regular expression R. Bob converts his regular expression R into a deterministic finite automaton represented by transition matrix M and start state s. M has m rows, one for each state in the automaton and n column, one for each symbol in the alphabet Σ. As in any secure multi-party computation (or even encryption) the lengths of the inputs is not hidden. So the parameters l, m, and n are public. The only technique (as in encryption) to hide the length of the inputs is padding, but, as it comes at the expense of runtime, its discussion is not expanded here. The protocols operate in the "Honest-but-Curious" model, i.e. both parties follow the protocol as intended, but are allowed to record information along the way to make inferences about the input of the other party.

4.2 Perfect Secret Protocol

The Perfect Secret Protocol is used as a reference, so the output is defined as all (ending) position where the regular expression matches. In section 4.3 protocols to compute other results, such as does the regular expression match at all, are presented for the fast protocol. Similar techniques could obviously be applied to the Perfect Secret Protocol, but are left out due to space constraints. The main operation the protocol has to provide is the indexing into the automaton iterating the state of the automaton. Let t be the current state of the automaton, then this operation can be written as

$$t := M[t, x[i]]$$

This operation only contains t as a state variable, that needs to be shared between Alice and Bob. In a circuit construction this state (t) would be shared in a XOR-like fashion and the indexing would be done using binary gates in an iterative or recursive algorithm. The obvious lower bound on that is $O(\log m)$ operations for the indexing. The operation would need to be repeated for each symbol x_i and the overall complexity is $O(m \cdot n \cdot l \cdot \log m)$. This can already be optimized by sharing the state not in an XOR-like fashion, but by using the modular operation. Let Alice have t_A and Bob have t_B, then the state would be $t = t_A + t_B \bmod m$. This sharing is perfectly secret, since t can be any value for any given value of t_A or t_B that Alice or Bob, respectively, has. Bob needs to implement two operations: $add(h)$ and $rotate(h)$. In the $add(h)$ operation Bob adds $h \bmod m$ to each value in the transition matrix M. In the $rotate(h)$ operation Bob rotates the rows of the transition matrix M by h rows, i.e. row i becomes row $i + h \bmod m$. The protocol then works as follows:

1. Alice gets start state s and sets $t_A = s (t_B = 0)$.
2. Bob chooses a random number r and applies $add(r)$.
3. Alice and Bob engage in an Oblivious Transfer (OT) protocol for all items in the transition matrix M and Alice retrieves the element corresponding to index t_A, x_i as its new value for t_A. *Note:* A special, unused bit can indicate whether the state is *accept*. Alice communicates this bit back to Bob.

4. Bob applies $rotate(r)$ and they repeat from step 2 for each character in x.

Theorem 1. *The "Perfect Secret Protocol" is secure in the semi-honest model.*

Proof. The composition theorem for secure multi-party computation [7] states, loosely speaking, that a protocol using secure building blocks protocols is secure if it is secure when those are replaced by invocations to ideal trusted third party. In this case the building block protocol is OT and the security of the protocol must be proven if those were done by ideal functionality. It will be shown that the view of each party can be simulated using a probabilistic polynomial time (PPT) algorithm from the inputs and outputs of each party alone. Let o be the output at each party (the bits indicating the state attribute). The view of each is the messages collected (assuming that OT is done using the ideal model). Bob is sent the output by Alice, so the simulator for his view $VIEW_{Bob}$ is simply: $VIEW_{Bob} = o$. Alice's view is the shares t_A and the simulator for view is to uniformly select l random numbers r'_i, $0 \leq r'_i < m$ for $i = 0, \ldots, l$. The simulator and Alice's view are computationally indistinguishable, because the probability that a certain number x appears is equal in the view and the simulator and independent of previous events in both cases:

$$\begin{aligned} Pr[VIEW^i_{Alice} = x] &= Pr[x = t - t_B \bmod m] \\ &= \frac{1}{m} \\ &= Pr[r'_i = x] \\ &= Pr[SIMULATOR^i_{Alice} = x] \end{aligned}$$

□

The communication cost is $O(m \cdot n \cdot l)$ and the $\log m$ factor has been replaced by a constant for the encryption system as the protocol works on integer numbers rather than on bits. This protocol also provides a lower bound on circuit construction protocols as they have to implement the indexing in a similar fashion and it already has been combined with the OT protocol in the above protocol. But the communication cost for this protocol can be prohibitively large, especially for large texts where the text length l dominates the cost. This paper therefore presents a second protocol that provides better performance characteristics for large texts, but at the expense of some privacy.

4.3 Fast Protocol

The fast protocol proceeds in the following general steps which will be detailed in later sections:

1. Bob prepares and encrypts the matrix M, such that Alice cannot extract any information about the regular expression. Alice then permutes the matrix, such that Bob cannot relate any column index to an input symbol of Alice. This relies on the fact that undoing a permutation in a graph is a hard problem related to the graph isomorphism problem. This is called the Obfuscation Protocol.
2. Then Alice and Bob engage in the State Transition Protocol for each character in Alice's string and proceed through the states of the automaton.
3. When all characters of Alice have been processed Alice and Bob engage in a short protocol to find the result.

Obfuscation Protocol The entire Obfuscation Protocol is composed of a number of building blocks. Each building block will be presented separately and by itself. The protocol is very complex, because it is a combination of so many parts and the reader be referred to [19] for a detailed, mathematical description of the protocol.

The basic protocol permutes the transition matrix of Bob. The general idea of the protocol is to create a permutation as a mapping of double-encrypted values. The mapping can then be applied to the encrypted values, such the permutation takes place without Alice knowing the original values. The protocol proceeds as follows:

1. Bob encrypts each entry of transition matrix M with $E_B(\cdot)$ and sends them to Alice.
2. Alice (double-)encrypts each entry with $E_A(\cdot)$.
3. Alice permutes the rows of M with a random permutation Π_A.
4. Alice creates a mapping Ψ of $i \mapsto \Pi_A(i)$. She encrypts i with $E_A(\cdot)$ and $\Pi_A(i)$ with $E_{A'}(\cdot)$. She sends the result to Bob.
5. Bob encrypts the first element of each pair with $E_B(\cdot)$ and the second element with $E_{B'}(\cdot)$. He applies a random permutation Π_B to the mapping and sends the result back to Alice.
6. Alice replaces each occurrence of $E_A(E_B(i))$ in M with its mapping $E_{A'}(E_{B'}(\Pi_A(i)))$.
7. She decrypts each value with $D_{A'}(\cdot)$ and sends the result to Bob.
8. Bob decrypts each value with $D_{B'}(\cdot)$.

Bob obtains the permuted transition matrix and could use it in the protocol. The first obvious problem is that this does not generate an isomorphic graph, since each edge has an assigned symbol. Alice needs to permute each row and permute each row differently. Adding such a step into the protocol is not difficult, but the transition matrix cannot be used "as-is" anymore for the evaluation of the regular expression. So Alice needs to maintain a copy of each such permutation (or the random seed that generated it) and undo it during the State Transition protocol.

The second problem is that the graph of the automaton might not be big enough to generate a difficult isomorphism problem. Therefore Alice needs to increase the size of the graph. She can split each node into k nodes where k is a security parameter agreed to by both parties. Of course, edges are not copied, but a target (from the split nodes) is chosen randomly. Details can be found in [19]. This also creates obfuscation in the graph and increases the difficulty of undoing the permutation. Introducing this into the protocol, Alice can simply copy the nodes when they are encrypted. The mapping then needs to be increased to $k \cdot m$ pairs where each i is mapped to k new values. She then randomly picks one of the pairs.

The third problem is that one node may have multiple in-edges, i.e. a number i can appear multiple times in M and the cipher text would reveal that. The in-degrees of all nodes would be revealed, such that one can more easily guess the automaton. This can be prevented by randomizing each entry in the transition matrix, such that its encrypted value is unique. Bob can choose such a randomization before sending the transition matrix to Alice, but then the mapping needs to be adapted. In the mapping process Bob needs to send the unencrypted values (remember they are just randomized versions

of $1, \ldots, m$) to Alice. As the mapping needs to hide the in-degrees of each node, the mapping needs to be padded with random values for nodes that are not of the maximum degree. This increases the size of the mapping to $k \cdot m \cdot \delta$ where δ is the maximum in-degree of any node. The proposed protocol in [19] suggests leaking this information, but for perfect secrecy $\delta = m \cdot n$ can be chosen. Now there is an entry in the mapping for each entry in the (split) transition matrix with additional random elements that are discarded.

The final protocol has communication complexity $O(k \cdot n \cdot m^2)$.

State Transition Protocol The State Transition protocol has to compute the function $t := M[t, x[i]]$ where t is the current state of the automaton. Recall that Alice's has applied different permutations to each row, so there is a different permutation to be undone for each distinct t. The general outline of the protocol is as follows:

1. Alice sends an encryption of the inverse permutations to Bob, i.e. she encrypts the original index of each permuted element. This step has communication complexity $O(m \cdot n)$.
2. For each element x_i of the string:
   (a) Bob sends the double-encrypted inverse permutation to Alice. This step has communication complexity $O(n)$.
   (b) Alice selects the index i and returns the decrypted element.
   (c) Bob decrypts and obtains the permuted index to obtain the next element.

A detailed mathematical description can be found in [19]. This protocol has communication complexity $O(l \cdot n)$, if $m < l$, since the steps 2(a) to 2(c) are executed l times.

Combating the Frequency Attack Bob obtains the current state during the State Transition Protocol. (Alice does so via the encrypted permutation, as well, but does not have the original automaton.) This state has been permuted from the original automaton, but Bob can try to deduce information about the permutation by observing the intermediate values. Bob can try to identify certain patterns, such as repeating states, but the split operation and permutation can transform any such pattern into a number of similar ones. But such patterns are not only determined by the automaton, but by the string, as well. Furthermore, Bob does not have complete freedom in choosing the automaton, since he wants to compute the matching of the regular expression. The structure of the automaton is tied to the regular expression, although Bob can influence the structure.

There is another technique Bob can use to relate permuted to initial states. Each state has an a priori probability p of occurring depending on some distribution of the string x. If that distribution is known to Bob, he can use the frequency attack on the State Transition Protocol. In the worst case, imagine Bob searching for the letter e. The automaton has two states and the probability of the state corresponding to an occurrence of e has significantly lower probability than the other, namely the probability of e occurring in the text. This probability can be refined for a second matching depending on what information was leaked by the result of the first (e.g. the number of actual

appearing e's). Bob can then rank the permuted states by their occurrence. He adds them up starting at the least frequent state and stops when he has reached probability p. Although, through splitting and permutation he will probably get a number of wrong states in this selection, the probability of these states of belonging to the original low probability state (e.g. e) is probably higher than an uniform distribution, i.e. the protocol leaks some information.

To counter this problem, the following solutions are suggested. Ideally each state would be split into a number of states proportional to p. Bob would transmit a probability of occurrence for each state along with M^1 to Alice, but he might not be trusted with the computation of these probabilities, since he can gain from incorrect information. In another approach, Alice might apply remedial action if she detects derivation from an uniform distribution. Bob sends the current state t along with each double-encrypted inverse permutation v in the State Transition Protocol. Alice monitors the statistical distribution of the states, and if one state deviates from the expected value $\frac{1}{m}$ by a fixed value, she will re-apply the Obfuscation Protocol, but this time split this state into a constant number c states. This will also permute all states again, such that the gathered statistics so far, don't relate to statistics gathered later. If the number of such re-obfuscation is bounded by the security parameter k, the overall cost of obfuscation would rise to $O(k^2 \cdot m^2 \cdot n)$.

Identifying the start state Alice and Bob need to identify the start state of the automaton after the Obfuscation Protocol. Alice therefore sends the encrypted inverse permutation (similar to the State Transition Protocol step 1) to Bob. Bob picks randomly one of the start states and sends it to Alice who decrypts and returns it Bob. This weakens the isomorphism problem for Bob, but since Alice also has split the start state there are still many possibilities for Bob to choose from. The detailed description can again be found in [19].

Obtaining the result There are different computations that a regular expression match (or search) can answer. First we will try to answer the question whether the (entire) text matches the regular expression. Bob has state attributes $accept, non-accept$ for each state in M. The answer to the question above is the state attribute of the last state. Alice and Bob need to track the attributes of the states. Therefore Bob encrypts them randomly: a random number with parity 0 for $accept$, a random number with parity 1 for $non-accept$, each encrypted with $E_B(\cdot)$. He sends them along with M^1 to Alice who tracks them during the "Obfuscation" protocol. She double encrypts with $E_A(\cdot)$ and returns them to Bob. Bob can then decrypt the state attribute of the final state and send it to Alice who announces it after decrypting. Note that the randomization is removed by Alice, and can therefore not be used to track the values. This protocol has communication complexity $O(m)$. For details, the reader is referred to [19] again.

An automaton for detecting a regular expression can be modified in such a way that it can be used for an entire text and each input symbol is processed only once. This provides a speed-up compared reading up to l_{max} symbols for each position in the text, but the results need to be obtained with a different protocol. The idea is to collect all state attributes in encrypted form during the "State Transition" protocol and then run

a protocol to obtain the results. First each encrypted state attribute is accompanied by an encrypted index by Bob and both vectors are randomly permuted. Then Alice filters the vectors, such that only the encrypted indices of matches remain. The following questions can then be answered easily. Protocols with detailed steps can again be found in [19].

1. *What are the position of matches?*
   Alice returns the encrypted indices to Bob who decrypts and announces them.
2. *What is the minimum position of a match?*
   Bob splits the indices into two additive shares. He sends one part unencrypted and the other part encrypted using a homomorphic encryption. Alice re-randomizes Bob's shares after filtering, such that he cannot track individual indices. In [2] a protocol is presented to find the minimum of a vector split in this way.
3. *How many matches are there?*
   Alice counts the number of matches and announces the result.
4. *Is there any match at all?*
   Bob generates a number of fake entries. He obtains Alice's encryption by sending the values encrypted under his key first and then after Alice responded with the double-encrypted values, he in the second step sends them encrypted only under Alice's key. Let the number of false matches be r. Then Alice counts the number of matches in the resulting vector and stores it in r'. Alice and Bob compare privately (using double encryption) r and r'. If they match, the regular expression did not match.

5 Conclusion

This paper presented two protocols for evaluating an automaton. One perfect-secret and one adapted to provide better performance at cost of such some privacy. The second protocol has been extended into a protocol for privately searching a text using a regular expression. It uses double-encryption, randomization and permutation as its only building blocks. The communication cost was kept low and only practical building blocks were used. The construction of the protocol shows that the security relies on the computational difficulty of the graph isomorphism problem. Future research would be to extend the regular expression matching algorithm with more powerful matching techniques, e.g. look-ahead. The problem of repeated searches using different regular expressions is subject to further investigation.

References

1. R. Agrawal, A. Evfimievski, and R. Srikant. Information sharing across private databases. *Proceedings of the ACM SIGMOD international conference on Management of data*, 2003.
2. M. Atallah, F. Kerschbaum, and W. Du. Secure and Private Sequence Comparisons. *Proceedings of the 2nd Workshop on Privacy in the Electronic Society*, 2003.
3. M. Ben-Or, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. *Proceedings of the 20th ACM symposium on Theory of computing*, 1988.

4. D. Betel, and C. Hogue. Kangaroo – A pattern-matching program for biological sequences. *Bioinformatics 3(20)*, 2002.
5. M. Bykova, M. Atallah, J. Li, K. Frikken, and M. Topkara. Private Collaborative Forecasting and Benchmarking. *Proceedings of the 3rd Workshop on Privacy in the Electronic Society*, 2004.
6. C. Cachin, S. Micali, and M. Stadler. Computationally private information retrieval with polylogarithmic communication. *Proceedings of EUROCRYPT*, 1999.
7. R. Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology 13(1)*, 2000.
8. D. Chaum, C. Crepeau, and I. Damgard. Multiparty unconditionally secure protocols. *Proceedings of the 20th ACM symposium on Theory of computing*, 1988.
9. B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. Private Information Retrieval. *Proceedings of the 36th Symposium on Foundations of Computer Science*, 1995.
10. J. DeLaurentis. A further weakness in the common modulus protocol for the RSA cryptoalgorithm. *Cryptologia 8(3)*, 1984.
11. W. Du, and M. Atallah. Privacy-Preserving Cooperative Scientific Computations. *Proceedings of the 14th IEEE Computer Security Foundations Workshop*, 2001.
12. B. Eckman, A. Kosky, L. Laroco. Extending traditional query-based integration approaches for functional characterization of post-genomic data. *Bioinformatics 17(7)*, 2001.
13. K. Frikken, and M. Atallah. Privacy Preserving Electronic Surveillance. *Proceedings of the 2nd Workshop on Privacy in the Electronic Society*, 2003.
14. Y. Gertner, Y. Ishai, and E. Kushilevitz. Protecting data privacy in private information retrieval schemes. *Proceedings of the 30th ACM Symposium on Theory of Computing*, 1998.
15. S. Goldwasser. Multi party computations: past and present. *Proceedings of the 16th ACM symposium on Principles of distributed computing*, 1997.
16. O. Goldreich. Secure Multi-party Computation. Available at *http://www.wisdom.weizmann.ac.il/~oded/pp.html*, 2002.
17. O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. *Proceedings of the 19th ACM conference on Theory of computing*, 1987.
18. J. Hopcroft, R. Motwani, and J. Ullman. Introduction to Automata Theory, Languages, and Computation. *Addison Wesley*, 2000.
19. F. Kerschbaum. Practical Private Regular Expression Matching. *Technical Report, University of Dortmund*, available at *http://www4.cs.uni-dortmund.de/RVS/FK/*, 2005.
20. E. Kushilevitz, and R. Ostrovsky. Replication is not needed: single database, computationally–private information retrieval. *Proceedings of the 38th Symposium on Foundations of Computer Science*, 1997.
21. S. Pohlig, and M. Hellman. An improved algorithm for computing logarithms over GF(p) and its cryptographic significance. *IEEE Transactions on Information Theory 24*, 1978.
22. R. Rivest, A. Shamir, and L. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM 21(2)*, 1978.
23. B. Schneier. Applied Cryptography, 2nd Edition. *John Wiley & Sons*, 1996.
24. D. X. Song, D. Wagner, and A. Perrig. Practical Techniques for Searches on Encrypted Data. *Proceedings of IEEE Symposium on Security and Privacy*, 2000.
25. J. Vaidya, and C. Clifton. Privacy Preserving Association Rule Mining in Vertically Partitioned Data. *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2002.
26. L. Wall, T. Christiansen, J. Orwant. Programming Perl, 3rd Edition. *O'Reilly*, 2000.
27. A. Yao. Protocols for Secure Computations. *Proceedings of the IEEE Symposium on Foundations of Computer Science* 23, 1982.