Efficient Secure Pattern Search Algorithm

Gurgen, Khachatryan
American University of Armenia
Yerevan, Armenia
e-mail: gurgenkh@aua.am

Mihran, Hovsepyan
Russian Armenian University
Yerevan, Armenia
e-mail: hovsepyan.mihran@gmail.com

Aram, Jivanyan
American University of Armenia
Yerevan, Armenia
e-mail: ajivanyan@aua.am

ABSTRACT

In this paper we describe an efficient protocol for oblivious evaluation of a binary alphabet Deterministic Finite Automata (DFA) between the DFA owner (client) and the input text owner (server). The protocol requires only a single round of client-server communication. The number of server-side computations is linear to the text length and does not depend on the size of the DFA, and the number of client-side computations is linear to the multiplication of the number of the DFA states and text length, and it does not depend on the internal structure of the DFA. Our protocol uses white-box based 1-out-of-2 oblivious transfer protocol as a construction block. As a result, we have no public-key operations in our algorithm.
Also, we have developed a test program which implements the protocol and this paper includes the results of benchmarks done for different input data. These results demonstrate the efficiency of the construction and confirm the low computational overhead of server side operations.

Keywords

cryptography, secure function evaluation, white-box, oblivious transfer

1. INTRODUCTION

Consider the following oblivious Deterministic Finite Automata DFA evaluation problem: the first party (client) has a DFA (or regular expression) Γ as long as the second party (server) has a binary text string X. Their objective is to check whether the string X belongs to the language generated by the Γ allowing both parties to learn the answer so that neither the client nor the server learned any additional information about the input of each other.
This problem and its variations have numerous applications; lots of problems that associated with tow-party secure and private communication adduce this problem. One such example is searching a number of appearances of a specific DNA pattern among people of some specific group (for instance the Police (the server) wants to allow biogenetical researchers (clients) to find the number of people among the criminals who have a specific (featured by the researchers) pattern in their DNAs without providing the entire list of DNAs of such people, and without learning which patterns have been searched). Generally the *secure pattern matching* problem and its variants which have been actively discussed during recent years are such examples [3, 4, 5, 6, 7].
As in every two party communication protocol here also it is important to take into account the number of client-server communication rounds. Also depending on the real world application the ratio between the number of DFA states and the text length may be different; they both may have about the same size or vice versa, one of them may have a significantly bigger size than another. Because of these two factors the protocol is designed to run in a **single client-server communication round**, to do a **small number of server side computations** and to use **no asymmetric cryptography operations**.
Our protocol is similar to the "Efficient Protocol for Oblivious DFA Evaluation" construction described in [1]. But unlike it, we do not use any public-key operations in our algorithm, we use white-box based 1-out-of-2 oblivious transfer protocol (later OT) [2] as a construction block.

2. THE PROTOCOL

Notations: We assume that the client has a DFA $\Gamma(Q;\{0,1\};\ \Delta; s_1; F)$ with binary $\{0,1\}$ input alphabet; set of states Q; transitions matrix Δ of size $|Q|\times 2$, where $\Delta[q,b]\in Q$ is the state where the DFA will go if it sees the letter b being in state q; initial state s_1 and set of accepting (or final) states F, while the server has n – bit string $X\in\{0,1\}^n$. Saying the result of evaluation of Γ on the string X (or simply $\Gamma(X)$) we mean the Boolean value which is 1 (or true) if the state

$$\Delta\left[\dots\Delta\left[\Delta[s_1,X[1]],X[2]\right]\dots,X[n]\right]$$

is an accepting state (i.e., belongs to F) and 0 (or false) otherwise. By k we denote the security parameter of the protocol. The pipe-sign | is used to denote the concatenation of binary strings and the circled plus sign $\oplus$ is used to denote the XOR operation. By $\lceil r\rceil$ we denote the ceiling of a real number r.

Brief description of the protocol: The main steps of the protocol are the following:

a) *Client*: Create a special evaluation matrix (DFA matrix) M_Γ of size $n\times|Q|\times 2$ intended for evaluation of Γ on n-bit strings.
b) *Client*: Create garbled DFA matrix GM_Γ by permuting each row of the M_Γ matrix, then encrypting each cell of the matrix using one time pad. As a result, to calculate $\Gamma(X)$ it will be enough for the server to have the garbled DFA matrix GM_Γ and a key for each position $i; 1\le i\le n$ of the string X corresponding to the letter of that position.
c) *Server*: For each letter of string X create an OT query token [2] and send them all along with the OT initialization data to the client.

d) *Client*: For each OT query token create an OT response token for the corresponding key [2] and send them together with the garbled DFA matrix GM_Γ to the Server.

e) *Server*: From OT response tokens invoke the keys for positions of the string X, then compute $\Gamma(X)$ using those keys and GM_Γ garbled DFA matrix.

Now let us look at these steps in more detail.

Step a): For evaluating Γ on any n-bit string we create a DFA matrix M_Γ of size $n \times |Q| \times 2$ such that for each state q and letter b $M_\Gamma[i,q,b] = \Delta[q,b]\ \forall i; 1 \le i \le n-1$ and $M_\Gamma[n,q,b] = 1$ if $\Delta[q,b]$ is a final state and $M_\Gamma[n,q,b] = 0$ if $\Delta[q,b]$ is not final. It is easy to see that in such notation $\Gamma(X)$ is equal to

$$M_\Gamma\Big[n, \dots M_\Gamma\Big[2, M_\Gamma\big[1, s_1, X[1]\big], X[2]\Big] \dots, X[n]\Big]$$

Figure 1 (a, b) below illustrates an example of DFA with its DFA matrix.

Step b): It is worth to note that for any permutation $P: Q \to Q$ the DFA Γ is equal (accepts the same set of strings) to the DFA $\Gamma_P(Q; \{0,1\}; \Delta_P; P[s_1]; F_P)$ where for each state q and letter b $\Delta_P[P[q], b] = P\big[\Delta[q,b]\big]$ and $F_P = \{q : P^{-1}[q] \in F\}$. The first stage of DFA matrix garbling is based on this fact. Namely, we first create n random permutations of the DFA states Q and fill by them n rows of an $n \times |Q|$ permutations matrix PER, then we create permuted DFA matrix PM_Γ such that for each state q and letter b

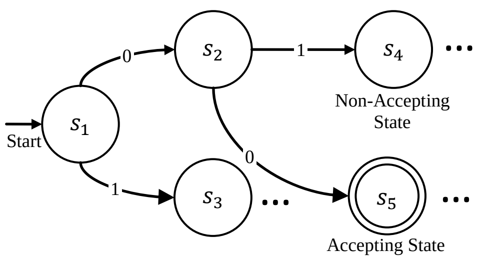
$$PM_\Gamma[i, PER[i,q], b] = PER\big[i+1, M_\Gamma[i,q,b]\big]$$

for each i; $1 \le i \le n-1$ and

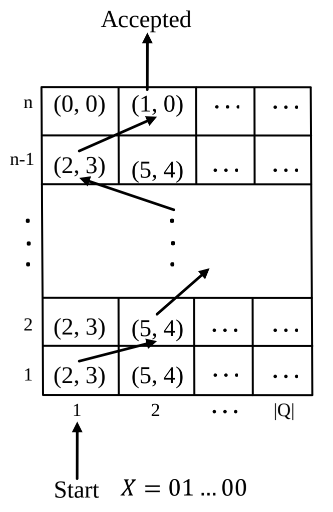
$$PM_\Gamma[n, PER[n,q], b] = M_\Gamma[n,q,b]$$

Figure 1 (c) below illustrates an example of a PM_Γ matrix. Here it is not hard to observe that in terms of PM_Γ matrix $\Gamma(X)$ is equal to

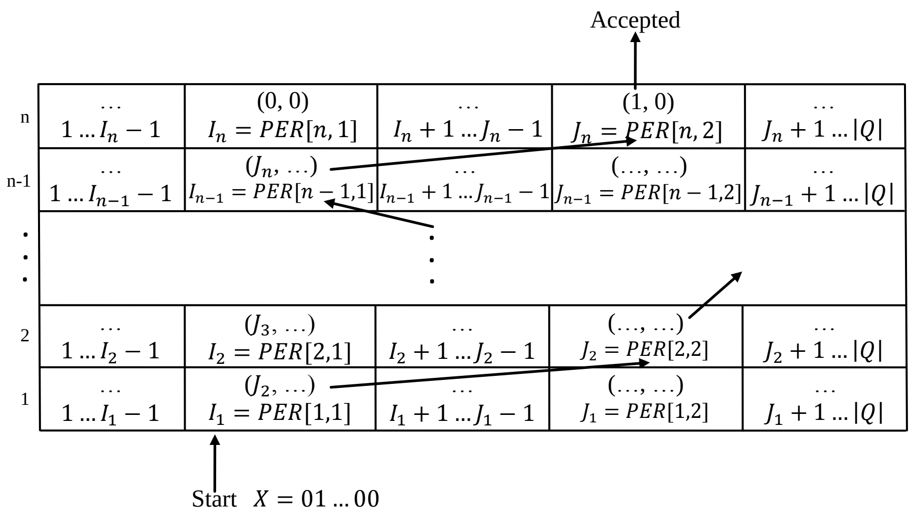
$$PM_\Gamma\Big[n, \dots PM_\Gamma\Big[2, PM_\Gamma\big[1, PER[1,s_1], X[1]\big], X[2]\Big] \dots, X[n]\Big]$$



(a) A general DFA Γ



(b) M_Γ (DFA Matrix of Γ)



(c) Permuted DFA Matrix PM_Γ

Figure 1: DFA Γ, DFA matrix M_Γ, Permuted DFA matrix PM_Γ

For the second stage of DFA matrix garbling we generate an $n \times 2$ size matrix of random k-bit keys K and an $(n+1) \times |Q|$ size matrix of k'-bit pads PAD; where for each state q, $PAD[i,q]$ is a random k'-bit string for each $i; 1 \le i \le n$ and $PAD[n+1,q] = \underbrace{00\ldots0}_{k'}$. By k' we denote $k - \lceil \log_2 |Q| \rceil$.
For garbling the permuted DFA matrix we also need some CSPRNG (cryptographically secure pseudo-random number generator) $PRG: \{0,1\}^{k'} \to \{0,1\}^{2k}$. Since the PRG receives a k'-bit string as an input and returns $2k$-bit string as an output by $PRG(Y,0)$ we denote the first k-bits of $PRG(Y)$ and by $PRG(Y,1)$ we denote the second k-bits of $PRG(Y)$. Having all this, we create the garbled DFA matrix GM_Γ such that

$$GM_\Gamma[i,q,b] = \big(PM_\Gamma[i,q,b] \mid PAD\big[i+1, PM_\Gamma[i,q,b]\big]\big) \oplus K[i,b] \oplus PRG(PAD[i,q],b)$$

for each $i; 1 \le i \le n$, letter b and state q.
Step c): Having garbled DFA matrix GM_Γ and $K\big[i,X[i]\big]$ for each $i; 1 \le i \le n$ the server will be able to calculate $\Gamma(X)$ (we will see that in step **e**). Hence, here for each letter $X[i]$ of its input string the server generates an OT query token $\text{OT}_{Query}[i]$ and along with OT initialization info OT_{Init} sends them to the client. For the details of OT initialization info and OT query token generation see [2].
Step d): For each OT query the token $\text{OT}_{Query}[i]$ $(1 \le i \le n)$ the client, using OT initialization info OT_{Init} generates an OT response token $\text{OT}_{Responce}[i]$ which carries sufficient info to invoke $K\big[i,X[i]\big]$ (see [2]). Then the client sends those response tokens, garbled DFA matrix GM_Γ, $PER[1,s_1]$ and $PAD\big[1,PER[1,s_1]\big]$ to the server.
Step e): At this step first the server for each $i; 1 \le i \le n$ invokes $K\big[i,X[i]\big]$ key stored in the corresponding response token $\text{OT}_{Responce}[i]$ (see [2]). Then having the keys, garbled DFA matrix GM_Γ, $PER[1,s_1]$ and $PAD\big[1,PER[1,s_1]\big]$,it runs the following algorithm to calculate $\Gamma(X)$:

```
Evaluation of garbled DFA matrix

$cur\_state\_id := PER[1,s_1]$;
$cur\_pad := PAD[1,cur\_state\_id]$;
for each row $i = 1$ to $n$ do
  $cur\_state\_id | cur\_pad := GM_\Gamma\big[i,cur\_state\_id,X[i]\big] \oplus$
        $K\big[i,X[i]\big] \oplus PRG(cur\_pad,X[i])$;
end for
return $cur\_state\_id$
```

3. PROTOCOL SECURITY

In the protocol we use a white-box based 1-out-of-2 OT protocol. Such *OT protocol is considered to be secure* if the underlying white-box encryption schema is secure. In our case we use white-box encryption schema based on SAFER+ encryption algorithm. White-box scheme can be considered secure if no computationally bounded adversary is able to extract the master encryption key from the white-box encryption tables and no computationally bounded adversary is able to make decryption functionality with the help of only white-box encryption tables. So the white-box scheme is considered to be secure if it is secure against key-recovery and reverse-engineering attacks.
Full-security for a two-party computation is defined by requiring indistinguishability (either perfect, statistical or computational) between a real execution of the protocol and an ideal execution in which there is a TTP (trusted third party) who receives the parties input, evaluates the function and outputs the results to them. *Privacy against malicious adversaries* guarantees that a corrupted party will not learn any information about the honest parties input. However, this does not guarantee that the joint output of the parties in the real world is simulatable in an ideal world.
Readers can refer to [1] for more detailed discussion on these two levels of security.
Summing up all definitions above we see that according to the theorem 1 from [1] our protocol is *fully-secure when only one party is malicious* and it is *private when both parties are malicious*.

4. IMPLEMENTATION AND BENCHMARKS

Implementation: We implemented the protocol and a test application for it in C++. The implementation of the protocol is parameterized by the security parameter k and by the CSPRNG PRG. The test application acts as both a client and a server. As a server, it receives a text string as an input, and as a client, its input is a regular expression which it converts to the equivalent DFA, first using Thompson's construction algorithm [8] to create NDFA equivalent to the regular expression and then using subset construction algorithm [9] to convert the NDFA to the DFA. After all, the test application runs the steps a) – e) of the protocol and calculates the result of the evaluation of the DFA on the text string. Besides the result of the DFA evaluation, it prints out the time spent for different parts of computations.
Benchmarks: From the description of the protocol it is clear that the number of computations done by the algorithm depends only on the security parameter k, the number of the DFA states and the length of the text string. Apart from that, in our algorithm we have generation of random pads and keys, as well as usage of CSPRNG, and, depending on the approach used for generation of random pads and keys and chosen CSPRNG the efficiency of the algorithm may differ for the same input data (k, Γ and X). We did our benchmarks on a 64-bit Windows 7 PC with Intel® Core™ 2 Quad Q6600 2.4 GHz processor and 4GB RAM, using SHA-256 as a $\{0,1\}^{k'} \to \{0,1\}^{2k}$ CSPRNG and C++ standard library's rand() function for random pad/key generation. The security parameter k was 128. The method of garbled DFA matrix construction clearly shows and our benchmarks confirmed that the number of operations hence spent time for garbled DFA matrix creation is proportional to the multiplication of the number of the DFA states and text length ($|Q| * n$). The first table shows performance of garbled DFA matrix generation of our implementation on inputs of different sizes. The table contains averaged results from 10 runs for each pair of inputs.

Table 1. Garbled DFA creation time (sec):

$\lvert Q\rvert$ \ n	10	100	1000	10000	100000
10	<0.001	0.002	0.017	0.17	1.7
100	0.002	0.017	0.17	1.7	17
1000	0.017	0.17	1.7	17	>100
10000	0.17	1.7	17	>100	>100
100000	1.7	17	>100	>100	>100
1000000	17	>100	>100	>100	>100

And the second table shows the performance of the OT phase of the protocol and performance of the garbled DFA matrix evaluation by the server.

Table 2. Time spent in OT phase (sec):

n	OT query generation and response extraction (server)	OT response generation (client)	Garbled DFA evaluation (server)
10	<0.001	<0.001	<0.001
100	0.002	<0.001	<0.001
1000	0.021	0.007	<0.001
10000	0.21	0.07	0.002
100000	2.1	0.7	0.02
1000000	21	7	0.2

The table shows only the dependency of efficiencies of the operations from the length of the server's input string X, since the number of OT queries / responses and the number of steps for garbled DFA matrix evaluation is equal to n and does not depend on structure of DFA.

All these results show that the described protocol is usable for real applications.

5. CONCLUSION

Unlike other protocols for oblivious DFA evaluation published in recent years [1, 3, 4, 6, 10, 11], our protocol is totally free of public-key operations, and it makes it somewhat unique among others. Table 3 below illustrates the complexities of rounds, client and server computations and network communication bandwidth usage of our and other recent protocols.

It is also worth to note that the protocol described above allows both parties to learn only $\Gamma(X)$, but in some applications it may be inconvenient or insufficient. In [1] it is shown that for each of the following problems it is possible to solve them after modifying the protocol a little, and that those modifications have no security leakage:

a) The client wants to hide the result $\Gamma(X)$ from the server.

b) The client or both parties want to learn whether X has substring Y such that $\Gamma(Y) = 1$.

c) The client or both parties want to learn all positions of X from where starts a substring Y such that $\Gamma(Y) = 1$.

d) The client or both parties want to learn the number of positions of X from where starts a substring Y such that $\Gamma(Y) = 1$.

Table 3. Complexity of protocols

	Round Complexity	Client Computations		Server Computations		Network Communication Bandwidth
		Asymmetric	Symmetric	Asymmetric	Symmetric	
Troncoso [3]	$O(n)$	$O(n\lvert Q\rvert)$	None	$O(n\lvert Q\rvert)$	$O(n\lvert Q\rvert)$	$O(n\lvert Q\rvert k)$
Frikken [4]	2	$O(n+\lvert Q\rvert)$	$O(n\lvert Q\rvert)$	$O(n+\lvert Q\rvert)$	$O(n\lvert Q\rvert)$	$O(n\lvert Q\rvert k)$
Gennaro [6]	$O(\min\{\lvert Q\rvert, n\})$	$O(n\lvert Q\rvert)$	None	$O(n\lvert Q\rvert)$	None	$O(n\lvert Q\rvert k)$
Yao [10]	1	$O(n)$	$O(n\lvert Q\rvert \log\lvert Q\rvert)$	$O(n)$	$O(n\lvert Q\rvert \log\lvert Q\rvert)$	$O(kn^2)$
Ishai [11]	1	$O(n)$	None	$O(n\lvert Q\rvert)$	None	$O(n\lvert Q\rvert k)$
Mohassel [1]	1	$O(n)$	$O(n)$	$O(n)$	$O(n\lvert Q\rvert)$	$O(n\lvert Q\rvert k)$
This Protocol	1	None	$O(n\lvert Q\rvert)$	None	$O(n)$	$O(n\lvert Q\rvert k)$

REFERENCES

[1] P. Mohassel, S. Niksefat, S. Sadeghian and B. Sadeghiyan. An Efficient Protocol for Oblivious DFA Evaluation and Applications. In *Proceedings of Cryptographers' Track at the RSA Conference*, pages 398-415, 2012.

[2] G. Khachatryan, A. Jivanyan. Efficient Oblivious Transfer Protocols based on White-Box Cryptography, *Personal Communications*.

[3] J.R. Troncoso-Pastoriza, S. Katzenbeisser, and M. Celik. Privacy preserving error resilient DNA searching through oblivious automata. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 519–528. ACM, 2007.

[4] K. Frikken. Practical private DNA string searching and matching through efficient oblivious automata evaluation. *Data and Applications Security XXIII*, pages 81–94, 2009.

[5] J. Katz and L. Malka. Secure text processing with applications to private DNA matching. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 485–492. ACM, 2010.

[6] R. Gennaro, C. Hazay, and J. Sorensen. Text search protocols with simulation based security. *Public Key Cryptography–PKC 2010*, pages 332–350, 2010.

[7] C. Hazay and T. Toft. Computationally secure pattern matching in the presence of malicious adversaries. *Advances in Cryptology-ASIACRYPT 2010*, pages 195–212, 2010.

[8] K. Thompson. Programming Techniques: Regular expression search algorithm. *Communications of the ACM 11 (6)*, pages 419–422, 1968.

[9] S. Michael. Introduction to the Theory of Computation. *Theorem 1.19, section 1.2*, page 55. ISBN 0-534-94728-X.

[10] A.C. Yao. Protocols for secure computations. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, pages 160–164, Citeseer, 1982.

[11] Y. Ishai, J. Kilian, K. Nissim, and E. Petrank. Extending oblivious transfers efficiently. *Advances in Cryptology-CRYPTO 2003*, pages 145–161, 2003.