Mathematical Problems of Computer Science 45, 77--89, 2016.

Two-Party Regular Expression Matching Protocol without Asymmetric Cryptography Operations

Gurgen H. Khachatryan[1], Mihran M. Hovsepyan[2], Aram H. Jivanyan[3]

[1] American University of Armenia
[2] Russian-Armenian (Slavonic) University
[3] Onecryptor CJSC
e-mail: gurgenkh@aua.am, hovsepyan.mihran@gmail.com, aram@skycryptor.com

Abstract

In this paper we describe a new protocol for secure evaluation of Deterministic Finite Automata (DFA) between two parties (client and server). The protocol has no restrictions on the DFA's input alphabet and runs in a single client-server communication round. It uses $O(mn)$ operations for client-side computations; $O(mn|Q|)$ operations for server-side computations, and the network communication bandwidth is $O(mnk|Q|)$ bytes where k is the security parameter of the protocol, m is the size of the DFA's input alphabet, n is the length of the input text and $|Q|$ is the number of the DFA states. As a building block our algorithm uses white-box based 1-out-of-n oblivious transfer protocol, which results that the protocol does no public-key operations. Apart from the description of the protocol the paper also contains results of efficiency benchmarks done on our implementation of the protocol.

Keywords: Cryptography, Secure function evaluation, Secure pattern matching, Oblivious transfer.

1. Introduction

The problem of two-party oblivious (secure) evaluation of DFA is a subproblem of more generic two-party secure function evaluation problem.
In two-party secure function evaluation problem there is a known for two parties function $F: U \times V \to Z$, the first party has an input argument $u \in U$ and the second party has an input argument $v \in V$ and the objective of these two parties to calculate $F(u, v)$ allowing one or both parties to learn the result, but none of them should learn any additional information about the other's input.
This problem has various real life applications. One such example is searching a number of appearances of a specific DNA pattern among people of some specific group (the FBI 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). In this example the input argument of the first party is the pattern and the input of the second party is the database of DNAs.

The next application is in banking: a person wants to take a credit from the bank. The bank needs to check whether he fits their requirements, particularly they want to check his credit history stored in Credit Report Agencies (CRA). But full credit report of a person may contain lots of private information as long as the criteria of the bank for giving a credit for the person also may be private. So the bank may want to check his criteria with the CRA without learning the full credit history and without providing the criteria to the CRA. The range of applications of the two-party secure function evaluation problem is not limited to the privacy-preserving genomic computations and credit checking, it can also be used in remote diagnostics, graph algorithms, data mining, medical diagnostics, face recognition, policy checking and in other areas. For more details see [3], it gives entire overview of the problem and its applications.

Here we consider a problem where the first party (client) has a DFA (regular expression) Γ as long as the second party (server) has a text string X which consists of the letters of the DFA's input alphabet. Their objective is to check whether the string X belongs to the language generated by the Γ or, in other words, whether the string X matches 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 may have some variations:

a) The client wants to hide the answer from the server.
b) The client or both parties want to learn whether the string X has a substring y which matches Γ.
c) The client or both parties want to learn positions of all substrings of the string X which match Γ.
d) The client or both parties want to learn the number of substrings of the string X which match Γ.

In this paper we describe a protocol which solves the initial problem and it is shown that after a little modification the protocol can be applied to these variations as well.

Similar to the "Efficient Protocol for Oblivious DFA Evaluation" [1] construction our protocol runs in a single client-server communication round and in both client and server sides it has the same as the protocol from [1] complexity of symmetric operations when the input alphabet of the DFA is binary. But unlike [1], our protocol has **no restriction on the DFAs input alphabet** and it uses **no asymmetric operations** anymore.

2. Preliminaries

2.1 Notations

We will assume that the client has a DFA $\Gamma(Q; \Sigma; \Delta; s_1; F)$ with input alphabet $\Sigma = \{b_1, b_2, \dots, b_{|\Sigma|}\}$; set of states $Q = \{q_1, q_2, \dots, q_{|Q|}\}$; table of transitions $\Delta_{|Q|\times|\Sigma|}$, where $\Delta[q, b] \in Q$ is the state following after state q through letter b; initial state s_1 and set of accepting (or final) states F, while the server has an n length string $X \in \Sigma^n$. 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 concatenation of strings and the circled plus sign $\oplus$ is used to denote per-bit XOR operation. By $\lceil r \rceil$ we denote the ceiling of a real number r. In matrix indexing sometimes we use letters of the input alphabet Σ and states Q instead of their numbers (i.e., b_j instead of j, q_j instead of j).

2.2 Definitions:

We will use the following concepts for describing the protocol and its security.
One round 1-out-of-n Oblivious Transfer protocol (later OT): A protocol is intended for solving the following problem. The client has a number i from 1 to n and the server has a set of n secrets $\{s_1, s_2, \dots, s_n\}$. They should communicate and after that the client should learn the i-th secret s_i, but the server should not learn i and the client should not learn any other secret apart from the s_i. There are various implementations of OT protocols, here we will use white box based 1-out-of-n oblivious transfer described in [2].
Secure Two-party Computation: Let $f = (f_1, f_2)$ of form $f: \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^* \times \{0,1\}^*$ be a two-party computation and π be a two-party protocol for computing f between the parties p_1 and p_2. The input of p_1 is x and the input of p_2 is y. Here we will briefly define two notions of security.
a) Full security (simulation-based security) against malicious adversaries: This security level is defined by requiring indistinguishability (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.
b) Privacy against malicious adversaries: This level of security guarantees that a corrupted party will not learn any information about the honest parties input. However, this does not always guarantee that the joint outputs of the parties in the real world is simulatable in an ideal world.
For more detailed discussion of these security notations refer to [12].
Looking ahead we want to note that in our protocol, we prove full-secure against one malicious party and private against the other.

3. The Protocol for Binary DFAs

At first let us consider the case when the input alphabet of the DFA $\Sigma = \{0,1\}$ is binary, we call such DFAs “binary DFA”.
*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-length binary 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-length binary 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) bellow illustrates an example of DFA with its DFA matrix.

Step b): Here it is worth to note that for any permutation $P: Q \to Q$ the DFA Γ is equal (i.e., 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 $\Delta_P[P[q], b] = P\big[\Delta[q,b]\big]$; 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

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

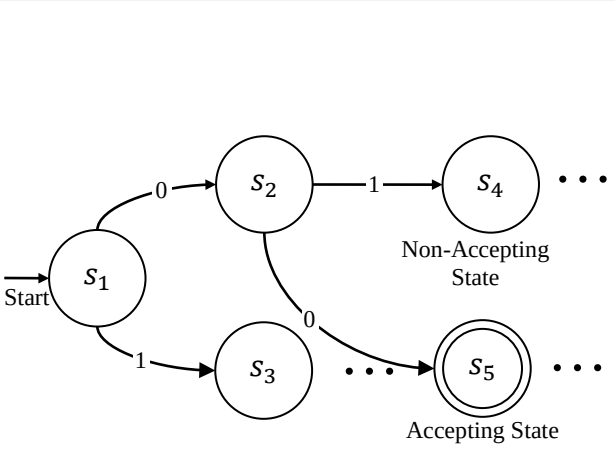
Figure 1 (c) bellow illustrates an example of a PM_Γ matrix. Here it is not hard to observe that in terms of PM_Γ matrix $\Gamma(X)$ is equal to

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

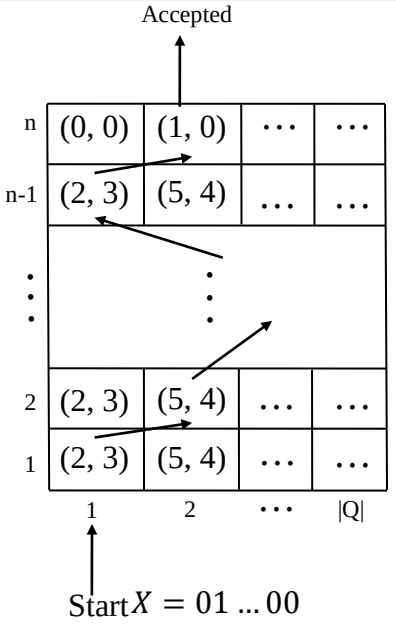
For the second stage of DFA matrix garbling we generate an $n \times 2$ size matrix of random k-bit keys K and an $(n+1) \times |Q|$ size matrix of k'-bit pads PAD; where for each state q, and for each $i; 1 \le i \le n$ the $PAD[i,q]$ is a random k'-bit string and $PAD[n+1,q] = \underbrace{00 \dots 0}_{k'}$. Here 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 these, we create the garbled DFA matrix GM_Γ such that

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

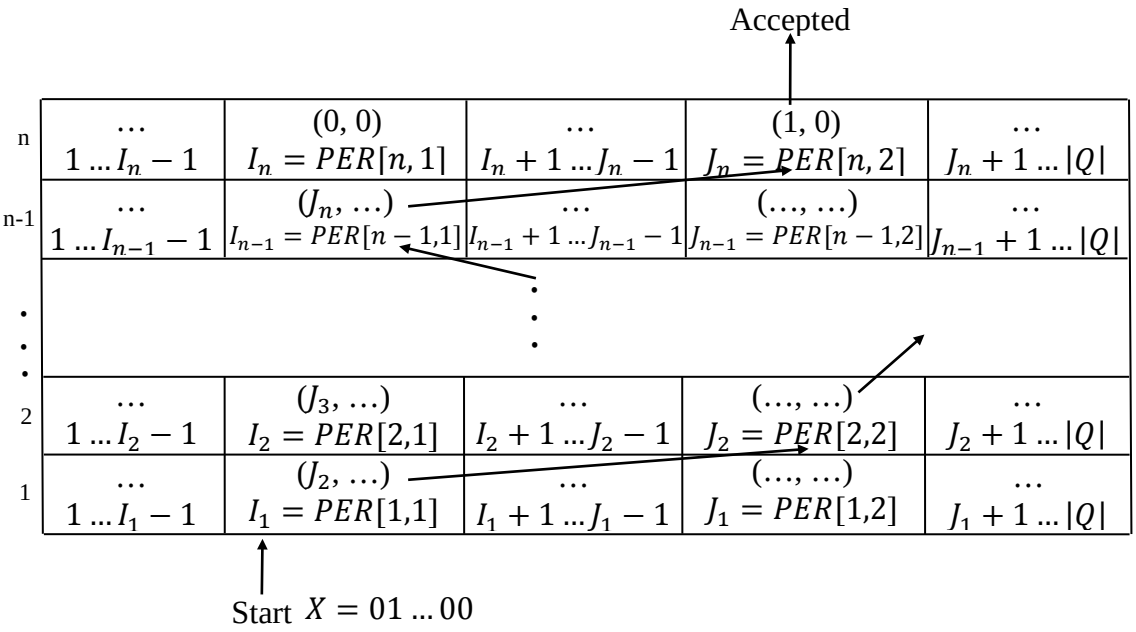
for each $i; 1 \le i \le n$, letter b and state q.



(a) A general binary DFA Γ



(b) M_Γ (DFA Matrix of Γ)



(c) Permuted DFA Matrix PM_Γ

Fig. 1. DFA Γ, DFA matrix M_Γ, Permuted DFA matrix PM_Γ.

Step c): Having the garbled DFA matrix GM_Γ and $K[i, X[i]]$ 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 $\mathrm{OT}_{Query}[i]$ 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 $\mathrm{OT}_{Query}[i]$ $(1 \le i \le n)$ the client, using OT initialization info OT_{Init} generates an OT response token $\mathrm{OT}_{Responce}[i]$ which carries sufficient info to invoke $K[i, X[i]]$ (see [2]). Then the client sends those response tokens, garbled DFA matrix GM_Γ, $PER[1, s_1]$ and $PAD[1, PER[1, s_1]]$ to the server.
Step e): At this step first the server for each $i; 1 \le i \le n$ invokes $K[i, X[i]]$ key stored in the corresponding response token $\mathrm{OT}_{Responce}[i]$ (see [2]). Then having the keys, garbled DFA matrix GM_Γ, $PER[1, s_1]$ and $PAD[1, PER[1, s_1]]$, 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**
  $cur\_state\_id | cur\_pad := GM_\Gamma\left[i, cur_{state_{id}}, X[i]\right] \oplus K[i, X[i]] \oplus PRG(cur\_pad, X[i])$;
end for
return cur_state_id

The step by step construction of the garbled DFA matrix implies that this algorithm will give the same result as evaluation of the initial DFA matrix, so no additional proof is required here.
These were all steps of the protocol for binary DFAs, now let us move to the case where there is no restriction on the input alphabet of the DFA.

4. Case of Non-Binary DFAs

Here again the protocol consists of 5 steps and they are almost the same as in the case of binary DFAs, the main difference is that the size of the third dimension of M_Γ, PM_Γ and GM_Γ matrixes is $|\Sigma|$ instead of 2.
Let us look at these steps in detail.
Step a): Create a DFA matrix M_Γ of size $n \times |Q| \times |\Sigma|$ such that for each state q and letter $b \in \Sigma$ $M_\Gamma[i, q, b] = \Delta[q, b]$ $\forall i; 1 \le i \le n-1$ and $M_\Gamma[n, q, b]$ is equal to 1 if $\Delta[q, b]$ is a final state and it is 0, otherwise. In such notation $\Gamma(X)$ is equal to

$$M_\Gamma\left[n, \ldots M_\Gamma\left[2, M_\Gamma[1, s_1, X[1]], X[2]\right] \ldots, X[n]\right].$$

Figure 2 (a, b) bellow illustrates an example of DFA with its DFA matrix.
Step b): Here again we 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 of size $n \times |Q| \times |\Sigma|$ such that for each state q and letter $b \in \Sigma$

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

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

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

Figure 2 (c) bellow illustrates an example of a PM_Γ matrix. In terms of PM_Γ matrix $\Gamma(X)$ is equal to

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

Then we create an $n \times |\Sigma|$ size matrix of random k-bit keys K and an $(n+1) \times |Q|$ size matrix of k'-bit pads PAD; here again for each state q, and for each $i; 1 \le i \le n$ the $PAD[i,q]$ is a random k'-bit string and $PAD[n+1,q] = \underbrace{00\dots0}_{k'}$, where k' is $k - \lceil \log_2|Q| \rceil$. We also need some CSPRNG $PRG: \{0,1\}^{k'} \to \{0,1\}^{k\cdot|\Sigma|}$, and by $PRG(Y,\ j)$ we denote the j-th k-bits of $PRG(Y)$ for each j; $1 \le j \le |\Sigma|$. After all, 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 $b \in \Sigma$ and state $q \in Q$.

Step c): And again having the 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 $\Gamma(X)$. So here we start OT phase between the client and the server for transferring the corresponding keys and GM_Γ.

For each letter $X[i]$ of its input string the server generates an OT query token $\mathrm{OT}_{Query}[i]$ and along with OT initialization info OT_{Init} sends them to the client [2].

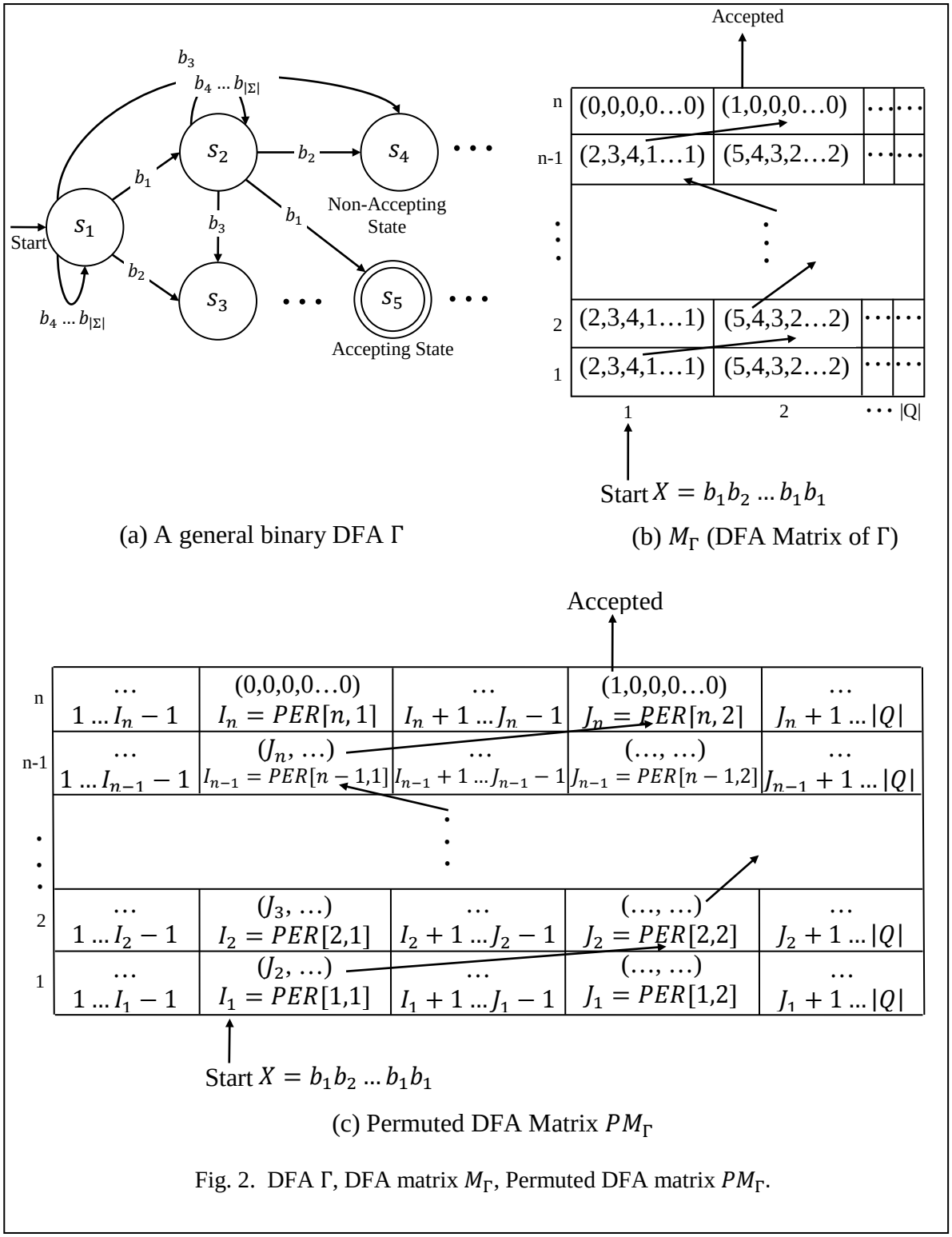
Step d): For each OT query the token $\mathrm{OT}_{Query}[i]$ $(1 \le i \le n)$ the client, using OT initialization info OT_{Init} generates an OT response token $\mathrm{OT}_{Responce}[i]$ which carries sufficient info to invoke $K\big[i, X[i]\big]$ [2]. Then the client sends those response tokens, 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 firstly the server invokes $K\big[i, X[i]\big]$ keys stored in the corresponding response token $\mathrm{OT}_{Responce}[i]$ for each $i; 1 \le i \le n$ [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

Here again the step by step construction of the garbled DFA matrix implies that this algorithm will give the same result as evaluation of the initial DFA matrix.



(a) A general binary DFA Γ

(b) M_Γ (DFA Matrix of Γ)

(c) Permuted DFA Matrix PM_Γ

Fig. 2. DFA Γ, DFA matrix M_Γ, Permuted DFA matrix PM_Γ.

5. 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 schema. 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 white-box scheme is considered to be secure if it is secure against key-recovery and reverse-engineering attacks.
Taking into account this and security definitions from chapter 2 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*.

6. Implementation and Benchmarks

6.1 Implementation

A C++ application has been created which implements the protocol. The implementation of the protocol is parameterized by the security parameter k and by the CSPRNG PRG. The 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 using Thompson's construction algorithm [8] to create NDFA equivalent to the regular expression, then using the subset construction algorithm [9] to convert the NDFA to the DFA. After all it runs the steps a) – e) of the protocol and calculates the result of evaluation of the DFA on the text string. During calculations it prints out time spent for different parts of computations.

6.2 Benchmarks

Algorithm construction implies that overall number of computations depends on the security parameter k, the number of the DFA states $|Q|$, the length of the text string n and the number of letters in the DFAs input alphabet $|\Sigma|$. Besides that, in the 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, $|Q|$, $|\Sigma|$ and $|X|$). We did our benchmarks for $k = 128$ and $|\Sigma| = 2$ on a 64-bit Windows 7 PC with Intel® Core™ 2 Quad Q6600 2.4 GHz processor and 4GB RAM, using SHA-256 as $\{0,1\}^{k'} \rightarrow \{0,1\}^{k\cdot|\Sigma|}$ CSPRNG and C++ standard library's rand() function for random pad/key generation. The method of garbled DFA matrix construction shows and our benchmarks confirmed that the number of operations, hence the spent time for garbled DFA matrix creation, is proportional to the $|Q| \cdot |\Sigma| \cdot n$ multiplication. 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 when (sec).

n / $\|Q\|$	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

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 when (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 n since the number of OT queries/responses and the number of steps for garbled DFA matrix evaluation is equal to n and do not depend on structure of DFA.

7. Conclusion

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

It is also worth to note that our protocol 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 modifications of the problem it is possible to solve it after modifying their protocol a little, and that those modifications have no security leakage. All those modifications of their protocol work for our protocol as well.

a) The client wants to hide the answer from the server.
b) The client or both parties want to learn whether the string x has a substring y which matches Γ.
c) The client or both parties want to learn positions of all substrings of the string x which match Γ.
d) The client or both parties want to learn the number of substrings of the string x which match Γ.

Table 3. Complexity of protocols.

	Round Complexity	Client Computations		Server Computations		Network Bandwidth
		Asymmetric	Symmetric	Asymmetric	Symmetric	
Troncoso [4]	$O(n)$	$O(n\|Q\|)$	None	$O(n\|Q\|)$	$O(n\|Q\|)$	$O(n\|Q\|k)$
Frikken [5]	2	$O(n+\|Q\|)$	$O(n\|Q\|)$	$O(n+\|Q\|)$	$O(n\|Q\|)$	$O(n\|Q\|k)$
Gennaro [6]	$O(\min\{\|Q\|,n$	$O(n\|Q\|)$	None	$O(n\|Q\|)$	None	$O(n\|Q\|k)$
Yao [10]	1	$O(n)$	$O(n\|Q\|\log\|Q$	$O(n)$	$O(n\|Q\|\log\|Q$	$O(kn^2)$
Ishai [11]	1	$O(n)$	None	$O(n\|Q\|)$	None	$O(n\|Q\|k)$
Mohassel [1]	1	$O(n)$	$O(n)$	$O(n)$	$O(n\|Q\|)$	$O(n\|Q\|k)$
This Protocol $\|\Sigma\|=2$	1	None	$O(n\|Q\|)$	None	$O(n)$	$O(n\|Q\|k)$

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*, pp. 398-415, 2012.

[2] G. Khachatryan and A. Jivanyan, “Efficient oblivious transfer protocols based on white-box cryptography’, *(submitted for publication).* see more in http://cse.aua.am/applied-cryptography-laboratory/.

[3] V. Kolesnikov, A.-R. Sadeghi and T. Schneider, “From Dust to Dawn: Practically Efficient Two-Party Secure Function Evaluation Protocols and their Modular Design”, *Cryptology ePrint Archive*, Report 2010/079 (2010), https://eprint.iacr.org/2010/079.pdf.

[4] 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*, pp. 519–528, 2007.

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

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

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

[8] K. Thompson, “Programming Techniques: Regular expression search algorithm”, *Communications of the ACM* , vol. 11, no. 6, pp. 419–422, 1968.

[9] S. Michael, *Introduction to the Theory of Computation,* PWS Publishing Company, 1997.

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

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

[12] O. Goldreich, *Foundations of cryptography: Basic applications*, Cambridge Univ Pr, 2004.

S**ubmitted 04.08.2015, accepted 15.02.2016**

Առանց գաղտնագրման ասիմետրիկ գործողությունների կիրառման կանոնավոր արտահայտության գաղտնի համապատասխանեցման երկկողմանի պրոտոկոլ

Գ. Խաչատրյան, Մ. Հովսեփյան, Ա. Ջիվանյան

Ամփոփում

Այս հոդվածում նկարագրված է նոր երկկողմանի պրոտոկոլ, որը թույլ է տալիս առաջին կողմին՝ կլիենտին, գաղտնի կերպով ստուգել, արդյոք երկրորդ կողմի՝ սերվերի, մոտ եղած տողը համապատասխանում է իր մոտ եղած կանոնավոր արտահայտությանը: Պրոտոկոլը նախատեսված է կամայական այբբենարանով կանոնավոր արտահայտությունների համար և աշխատում է մի փուլով: Այն կատարում է $O(mn)$ գործողություն կլիենտի կողմում և $O(mn|Q|)$ գործողություն սերվերի կողմում, իսկ ցանցով տեղափոխվում է $O(mnk|Q|)$ բայթ ինֆորմացիա, որտեղ k-ն պրոտոկոլի անվտանգության պարամետրն է, m-ը կանոնավոր արտահայտության այբբենարանի տառերի քանակն է, n-ը սերվերի մոտ եղած տողի երկարությունն է՝ տառերի քանակը, իսկ $|Q|$-ն՝ կանոնավոր արտահայտությանը համապատասխանող մինիմալ դետերմինիստիկ վերջավոր ավտոմատի (ԴՎԱ) վիճակների քանակը:

Двухсторонний протокол тайного сопоставления регулярных выражений без применения операций асимметричного шифрования

Г. Хачатрян, М. Овсепян, А. Дживанян

Аннотация

В этой статье описан двухсторонний протокол, который позволяет первой стороне – клиенту, имеющего регулярное выражение, тайным образом проверять соответствует ли строка имеющееся у второй стороны – у сервера, его регулярному выражению. Протокол предназначен для регулярных выражений с произвольным алфавитом и работает за один раунд коммуникаций между клиентом и сервером. Протокол выполняет $O(mn)$ операций на клиенте, $O(mn|Q|)$ операций на сервере, а трафик сети $O(mnk|Q|)$ байтов, где k это параметр безопасности протокола, m количество букв в алфавите регулярного выражения, n длина входной строки сервера, а $|Q|$ количество состояний минимального детерминистического конечного автомата (ДКА) соответствующего регулярному выражению.