Text Search Protocols with Simulation Based Security

Rosario Gennaro[1], Carmit Hazay[2], and Jeffrey S. Sorensen[1]

[1] IBM T.J. Watson Research Center, Hawthorne, New York, USA
{rosario,sorenj}@us.ibm.com
[2] Dept. of Computer Science and Applied Mathematics,
Weizmann Institute and IDC, Israel
carmit.hazay@weizmann.ac.il

Abstract. This paper presents an efficient protocol for securely computing the fundamental problem of *pattern matching*. This problem is defined in the two-party setting, where party P_1 holds a pattern and party P_2 holds a text. The goal of P_1 is to learn where the pattern appears in the text, without revealing it to P_2 or learning anything else about P_2's text. Our protocol is the first to address this problem with *full security* in the face of malicious adversaries. The construction is based on a novel protocol for *secure oblivious automata evaluation* which is of independent interest. In this problem party P_1 holds an automaton and party P_2 holds an input string, and they need to decide if the automaton accepts the input, without learning anything else.

1 Introduction

Secure two-party computation is defined as the joint computation of some function over private inputs using a communications protocol, satisying at least *privacy* (no other information is revealed beyond the output of the function) and *correctness* (the correct output is computed). Today's standard definition (cf. [1] following [2,3,4]) formalizes security by comparing the execution of such a protocol to an "ideal execution" where a trusted third party helps the parties compute the function. Specifically, in the ideal world the parties just send their inputs over perfectly secure communication lines to a trusted party, who then computes the function honestly and sends the output to the designated party. Informally, the real protocol is defined to be secure if all adversarial attacks on a real protocol can also be carried out in the ideal world. In the ideal world, the adversary can do almost nothing, and this guarantees that the same is also true in the real world. This definition of security is often called *simulation-based* because security is demonstrated by showing that a real protocol execution can be "simulated" in the ideal world.

Secure two-party computation has been extensively studied, and it is known that any efficient two-party functionality can be securely computed [5,6,7]. However, these are just feasibility results that demonstrate secure computation is possible, in principle, though not necessarily in practice. One reason is that the

P.Q. Nguyen and D. Pointcheval (Eds.): PKC 2010, LNCS 6056, pp. 332–350, 2010.

results mentioned above are generic, i.e. they do not exploit any structural properties of the specific function being computed. A long series of research efforts has been focused on finding efficient protocols for specific functions: constructing such protocols is crucial if secure computation is ever to be used in practice.

Our Contribution. In this paper we focus on the following problems:

- *Secure Pattern Matching.* We look at the basic problem of *pattern matching*. In this problem, one party holds a text T and the other party holds a pattern p, but $|T|$ and $|p|$ are mutually known. The aim is for the party holding the pattern to learn all the locations of the pattern in the text (and there may be many) while the other party learns nothing about the pattern.
- *Oblivious Automata Evaluation.* To solve the above problem we consider the approach of [8] which reduces the pattern matching problem to the composition of a pattern-specific automaton Γ with the text T. We develop a protocol for securely computing the evaluation of Γ on T.

The problem of pattern matching has been widely studied for decades due to its numerous applications. Yet, the problem of pattern matching in a secure setting has not received similar attention. Our starting point is an extremely efficient protocol that computes this function in the "honest-but-curious" setting.[1] This solution can be extended for one-sided simulation, or security even with corruption of the party with the pattern. This first protocol is independent of out protocol for the malicious setting, and is comparable to the one-sided simulatable protocol of [9]. However, while both protocols reach the same asymptotic complexity our protocol is much more practical since the concrete constants that are involved are much smaller ([9] requires $|p|$ oblivious transfers). Moreover, our protocol can be easily extended to address related problems such as approximate text search or text search with wildcards. The malicious setting introduces many subtleties beyond those considered in the previous settings and requires the use of a different technique. This includes the introduction of novel sub-protocols such as, for example, a protocol to prove that a correct pattern-specific automaton was constructed. We note that our protocols are the first efficient ones in the literature to achieve full simulatability for these problems with malicious adversaries. Security is based on the El Gamal encryption scheme [10,11] and thus requires a relatively small security parameter (although any additively homomorphic threshold encryption with secure two-party distributed protocols to generate shared keys and perform decryptions would work).

Motivation. Consider a hospital holding a DNA database of all the participants in a research study, and a researcher wanting to determine the frequency of the occurrence of a specific gene. This is a classical pattern matching application, which is however complicated by privacy considerations. The hospital may be

[1] In this setting, an adversary follows the protocol specification but may try to examine the messages it receives to learn more than it should.

forbidden from releasing the DNA records to a third party. Likewise, the researcher may not want to reveal what specific gene she is looking for, nor trust the hospital to perform the search correctly.

It would seem that basic honest-but-curious solutions (already present in the literature, see below) would work here. However, the parties may be motivated to produce invalid results, so a proof of accurate output might be as important as the output itself. Moreover, there is also a need to make sure that the data on which the protocol is run is valid. For example, a rogue hospital could sell "fake" DNA databases for research purposes. Perhaps some trusted certification authorities might one day pre-certify a database as being valid for certain applications. Then, the security properties of our protocol could guarantee that only valid data is used in the pattern matching protocol. (The first step of our protocol is for the hospital to publish an encryption of the data, this could be replaced by publication of encrypted data that was certified as correct.)

Related work. The problem of secure pattern matching was studied by Hazay and Lindell in [9] who used oblivious pseudorandom function (PRF) evaluation to evaluate every block of size m bits. However, their protocol achieves only a weaker notion of security called one-sided simulatability which guarantees privacy in all cases and requires that one of the two parties is never corrupted to guarantee correctness. It is tempting to think that a protocol for computing oblivious PRF evaluation with a committed key (where it is guaranteed that the same key is used for all PRF evaluations) for malicious adversaries [12] suffices for malicious security. Unfortunately, this is not the case since the inputs for the PRF must be consistent and it is not clear how to enforce this. Namely, for every i, the last $m-1$ bits of the ith block are supposed to be the first $m-1$ bits of the following block. The idea to use oblivious automata evaluation to achieve secure pattern matching originates in [13]. Their protocols, however, are only secure in the honest-but-curious setting. We improve their results to tolerate a malicious adversary.

The efficiency of our protocol. When presenting a two-party protocol for the secure computation of a specific function, one has to make sure that the resulting protocol is indeed more efficient than the known "generic" solutions for secure two-party computation of *any* function. We compare our protocols to the two most efficient generic two-party protocols secure against malicious adversaries, both based on the circuit garbling-technique by Yao [5]. Recall that Yao's protocol (which is secure against semi-honest players) uses a Boolean circuit to compute the fuction, and its computational complexity is linear in the size of the circuit.

- One result, in [14], to make Yao resistant to malicious adversaries uses a binary cut-and-choose strategy. This requires running s copies of Yao's protocol, where s is a statistical security parameter that must be large enough so that $2 * 2^{\frac{-s}{17}}$ is sufficiently small. This requires $O(s|C| + s^2 m)$ symmetric-key encryptions, and this communications overhead, as shown by [15], is a major obstacle.

– The other general result from [16] uses a special form of encryption for garbling and performs efficient zero-knowledge (ZK) proofs over that encryption scheme. The protocol requires a common reference string (CRS) which consists of a strong RSA modulus. To the best of our knowledge, there are currently no efficient techniques for generating a shared strong RSA modulus without incorporating external help. Furthermore, their protocol requires approximately 720 RSA exponentiations per gate where these operations are computed modulo 2048 due to the use of Paillier's encryption scheme. This means that the bandwidth of [16] is relatively high as well.

2 Tools and Definitions

Throughout the paper, we denote the security parameter by n. Although not explicitly specified, input lengths are always assumed to be bounded by some polynomial in n. A probabilistic machine is said to run in *polynomial-time* (PPT) if it runs in time that is polynomial in the security parameter n alone.

A function $\mu(\cdot)$ is *negligible* in n (or simply *negligible*) if for every polynomial $p(\cdot)$ there exists a value N such that $\mu(n) < \frac{1}{p(n)}$ for all $n > N$; i.e., $\mu(n) = n^{-\omega(1)}$. Let $X = \{X(n)\}_{n\in N, a\in\{0,1\}^*}$ and $Y = \{Y(n)\}_{n\in N, a\in\{0,1\}^*}$ be distribution ensembles. We say that X and Y are *computationally indistinguishable*, denoted $X \stackrel{c}{\equiv} Y$, if for every polynomial non-uniform distinguisher D there exists a negligible $\mu(\cdot)$ such that

$$\Big|\Pr[D(X(n,a)) = 1] - \Pr[D(Y(n,a)) = 1]\Big| < \mu(n)$$

for every $n \in N$ and $a \in \{0,1\}^*$.

Due to space considerations we defer the formal definitions for two-party secure computations in the presence of malicious adversaries and one-sided simulation security to the extended version of this paper.

Zero-knowledge proofs. Our protocols use the several standard zero-knowledge proofs, as summarized in Table 1. We also employ the following additional zero-knowledge proofs.

1. A zero-knowledge proof of knowledge π_{ENC} for the following relation: Let $C_i = [c_{i,1}, ..., c_{i,m}]$ for $i \in \{0,1\}$ and $C' = [c'_1, ..., c'_m]$ be three vectors of m ciphertexts each. We want to prove that C' is the "re-encryption" of the

Table 1. Zero knowledge proofs referenced by our protocols

Protocol	Relation/Language	Reference
π_{DL}	$\mathcal{R}_{\text{DL}} = \{((\mathbb{G}, g, h), x) \mid h = g^x\}$	[17]
π_{DDH}	$\mathcal{R}_{\text{DDH}} = \{((\mathbb{G}, g, g_1, g_2, g_3), x) \mid g_1 = g^x \wedge g_3 = g_2^x\}\}$	[18]
π_{NZ}	$\text{L}_{\text{NZ}} = \{(\mathbb{G}, g, h, \langle\alpha, \beta\rangle) \mid \exists\ (m \neq 0, r)$ s.t. $\alpha = g^r,\ \beta = h^r g^m\}$	[19]

same messages encrypted in either C_0 or C_1, or, in other words, that there exists an index $i \in \{0,1\}$ such that for all j, c'_j was obtained by multiplying $c_{i,j}$ by a random encryption of 0. More formally,

$$\mathcal{R}_{\text{ENC}} = \left\{(\mathbb{G}, g, m, C_0, C_1, C'), (i, \{r_j\}_j) \middle| \text{ s.t. for all } j : c'_j = c_{i,j} \cdot_G E_{pk}(0; r_j)\right\}.$$

This involves the parties computing the sets $c_0 = \left\{\prod(c_{0,j} \cdot_G (1/c'_j))^{r_{0,j}}\right\}_{j=1}^{Q}$ and $c_1 = \left\{\prod(c_{1,j} \cdot_G (1/c'_j))^{r_{1,j}}\right\}_{j=1}^{Q}$, where the sets $\{r_{0,j}\}_j$ and $\{r_{1,j}\}_j$ are public randomness. The prover then proves that either $\langle pk, c_0\rangle$ or $\langle pk, c_1\rangle$ is a Diffie-Hellman tuple.

2. Let $C = \{c_{i,j}\}_{j,i}$ and $C' = \{c'_{i,j}\}_{j,i}$ be two sets of encryptions, where $j \in \{1, \ldots, |Q|\}$ and $i \in \{0,1\}$. Then we consider a zero-knowledge proof of knowledge π_{PERM} for proving that C and C' correspond to the same decryption vector up to some random permutation. Meaning that,

$$\mathcal{R}_{\text{PERM}} = \left\{\left(pk, C, C'\right), \left(\pi, \{r_{j,i}\}_{j,i}\right) \middle| \forall\ i,\ \{c_{j,i} = c'_{\pi(j),i} \cdot E_{pk}(0; r_{j,i})\}_j\right\}$$

where π is a random one-to-one mapping over the elements $\{1, \ldots, |Q|\}$. Basically we prove that C' is obtained from C by randomizing all the ciphertexts and permuting the indices (i.e., the columns). We require that the same permutation is applied for both vectors. The problem in which a single a vector of ciphertexts is randomized and permuted is defined by

$$\mathcal{R}^1_{\text{PERM}} = \left\{\left((c_1, \ldots, c_Q), (\tilde{c}_1, \ldots, \tilde{c}_Q), pk\right), \left(\pi, (r_1, \ldots, r_Q)\right) \middle| \forall\ i, \tilde{c}_j = c_{\pi(j)} \cdot E_{pk}(0; r_j)\right\}.$$

and has been widely studied. The state-of-the-art protocol is in [20]. We use a simpler, though slightly less efficient (but still good for our purposes) protocol, from [21], which is an efficient zero-knowledge proof π^1_{PERM} for $\mathcal{R}^1_{\text{PERM}}$ with linear computation and communication complexity and constant number of rounds. We use this slightly less efficient protocol because its proof proof applies to the case where the same permutation is applied to multiple vectors of ciphertexts.

3 Secure Text Search Protocols

Text search involves scanning a text sequentially, looking for instances of a particular pattern. Efficient text search requires analysis of the pattern string to enable $O(\ell)$ scanning that skips over regions of text whenever possible matches are provably not possible, as in KMP [8] which we employ here.

Pattern matching is defined as follows: given a binary string T of length ℓ and a binary pattern p of length m, find all the locations in the text where pattern p appears in the text. Stated differently, for every $i = 1, \ldots, \ell - m + 1$, let T_i

be the substring of length m that begins at the ith position in T. Then, the basic problem of pattern matching is to return the set $\{i \mid T_i = p\}$. Formally, we consider the functionality $\mathcal{F}_{\text{PM}}$ defined by

$$(p,(T,m)) \mapsto \begin{cases} (\{i \mid T_i = p\}, \lambda) & \text{if } |p| = m \\ (\{i \mid T_i = p_1 \dots p_m\}, \lambda) & \text{otherwise} \end{cases}$$

Note that P_2, who holds the text, learns nothing about the pattern held by P_1, and the only thing that P_1 learns about the text held by P_2 is the locations where its pattern appears. Our starting point is an extremely efficient protocol that computes $\mathcal{F}_{\text{PM}}$ in the "honest-but-curious" setting, where the adversary follows the protocol specification but tries to gain useful information about the honest party's input.

3.1 "Honest-But-Curious" Secure Text Search

The protocol shown in Fig. 1 uses homomorphic encryption to ensure the privacy of the two parties' inputs. Informally, party P_1 computes a matrix Φ of size $2 \times m$ that includes an encryption of zero in position (i,j) if $p_j = i$ or an encryption of one otherwise. Given Φ, party P_2 creates a new encryption e_k for every text location k that corresponds to the product of the encryptions at locations (t_{k+j-1}, j) for all $j \in \{1,\dots,m\}$. Since e_k is the Hamming distance between p and T_k, iff p matches T_k, e_k will be a random encryption of zero. We remark that even though we consider here the problem of exact matching, this solution can be easily applied to the problem of text search with mismatches, or for larger alphabets.

Formally,

Protocol 1. $\boldsymbol{\pi_{\text{SIMPLE}}}$

– **Inputs:** The input of P_1 is a binary search string $p = p_1,\dots,p_m$ and P_2 a binary text string $T = t_1,\dots,t_\ell$

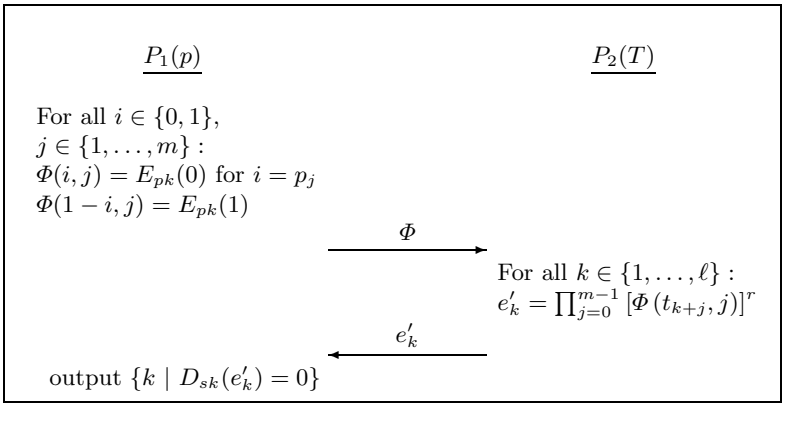

Fig. 1. Text search in the honest-but-curious setting

- **Conventions:** The parties jointly agree on a group $\mathbb{G}$ of prime order q and a generator g for the El Gamal encryption. Party P_1 generates a key pair $(pk, sk) \leftarrow G$ and publishes pk. Finally, unless written differently, $j \in \{1, \ldots, m\}$ and $i \in \{0, 1\}$.
- **The protocol:**
  1. **Encryption of pattern.** Party P_1 builds a $2 \times m$ matrix of ciphertexts Φ defined by,
  $$\Phi(i,j) = \begin{cases} E_{pk}(0;r) & p_j = i \\ E_{pk}(1;r) & \text{otherwise} \end{cases}$$
  where each r is a uniformly chosen random value of appropriate length. The matrix Φ is sent to party P_2.
  2. **Scanning of text.** For each offset $k \in \{1, \ldots, \ell - m + 1\}$, P_2 computes
  $$e_k = \prod_{j=1}^{m} \Phi\left(t_{k+j-1}, j\right)$$
  Then for each offset k, it holds that T_k matches pattern p if and only if $e_k = E_{pk}(0)$.
  3. **Masking of terms.** Due to the fact that the decryption of e_k reveals the number of matched elements at text location k, party P_2 masks this result through scalar multiplication. In particular, P_2 sends the set $\{e'_k = (e_k)^{r_k}\}_k$ where r_k is a random string chosen independently for each k.
  4. **Obtaining result.** P_1 uses sk to decrypt the values of e'_k and obtains
  $$\{k \mid D_{sk}(e'_k) = 0\}$$

Clearly, if both parties are honest then P_1 outputs a correct set of indexes with overwhelming probability (an error may occur with negligible probability if $(e_k)^r$ is an encryption of zero even though e_k is not), as the parties execute a naive solution for $\mathcal{F}_{\text{PM}}$. Then we state the following,

Theorem 1. *Assume that (G, E, D) is the semantically secure El Gamal encryption scheme. Then protocol π_{SIMPLE} securely computes $\mathcal{F}_{\text{PM}}$ in the presence of honest-but-curious adversaries.*

The proof is straightforward via a reduction to the security of (G, E, D) and is therefore omitted.

Furthermore, if party P_1 proves that it computed matrix Φ correctly, we can also guarantee full simulation with respect to a corrupted P_1. This can be achieved by having P_1 prove, for every j, $\Phi(0, j), \Phi(1, j)$ is a permuted pair of the encryptions $E_{pk}(0), E_{pk}(1)$, using π_{PERM}. Constructing a simulator for the case of a corrupted P_2 is more challenging since the protocol does not guarantee that P_2 computes $\{e'_k\}_k$ relative to a well defined bit string p. In particular, it may compute every encryption e'_k using a different length m string. Thus, only

privacy is guaranteed for this case. Let π'_{SIMPLE} denote the modified version of π_{SIMPLE} with the additional zero-knowledge proof of knowledge π_{PERM} of P_1. We conclude with the following claim,

Theorem 2. *Assume that (G, E, D) is the semantically secure El Gamal encryption scheme. Then protocol π'_{SIMPLE} securely computes $\mathcal{F}_{\text{PM}}$ with one-sided simulation.*

The proof sketch is in the extended version of this paper.

Efficiency. We first note that the protocol π'_{SIMPLE} is constant round. The overall communication costs are of sending $O(m+\ell)$ group elements, and the computation costs are of performing $O(m+\ell)$ modular exponentiations, as P_1 sends the table Φ and P_2 replies with a collection of ℓ encryptions. The additional cost of π_{PERM} is linear in the length of the pattern.

The fact that this protocol does not seem to be naturally extendable for the malicious setting has led us to search for the techniques presented in the next section.

3.2 Secure Text Search in the Presence of Malicious Adversaries

We consider a secure version of the KMP algorithm [8], that searches for occurrences of a pattern p within a text T by employing the observation that when a mismatch occurs, the pattern itself embodies sufficient information to determine where the next match could begin, thus bypassing re-examination of previously matched characters. More formally, P_1, whose input is a pattern p, constructs an automaton Γ_p as follows: We denote by $p_{\langle i \rangle}$ the length-i prefix $p_1, \ldots, p_i$ of p. P_1 first constructs a table Υ with m entries where its ith entry denotes the largest prefix of p that matches a suffix of $p_{\langle i-1 \rangle}$. This table (as in Fig. 2) maintains the appropriate partial match state if a mismatch occurs in the ith bit of p. The algorithm indicates for each bit of the text, every input that reaches the final state, or one bit for each text location.

State	Prefix	$\Upsilon(q_i)$	Fail State	$\Gamma(q_i, j)$ $j=0$	$\Gamma(q_i, j)$ $j=1$
q_1			q_1	q_1	q_2
q_2	1		q_1	$\Gamma(q_1, 0)$	q_3
q_3	11	1	q_2	q_4	$\Gamma(q_2, 1)$
q_4	110		q_1	q_5	$\Gamma(q_1, 1)$
q_5	1100		q_1	q_6	$\Gamma(q_1, 1)$
q_6	11000		q_1	$\Gamma(q_1, 0)$	q_7
q_7	110001	1	q_2	$\Gamma(q_2, 0)$	q_8
q_8	1100011	11	q_3	q_9	$\Gamma(q_3, 1)$
q_9	11000110	110	q_4	q_{10}	$\Gamma(q_4, 1)$
q_{10}	110001100	1100	q_5	$\Gamma(q_5, 0)$	q_{11}
q_{11}	1100011001	1	q_2	$\Gamma(q_2, 0)$	$\Gamma(q_2, 1)$

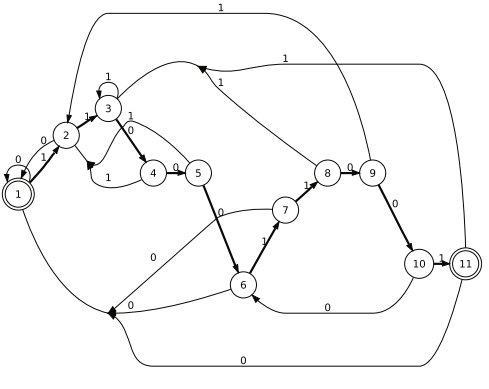

Fig. 2. Construction of determinized KMP automaton for pattern 1100011001

We remark that Υ can be easily constructed in time $O(m^2)$ by comparing p against itself at every alignment. P_1 constructs its automaton Γ_p based on Υ. It first sets $|Q| = |p| + 1$ and constructs the transition table Δ as follows. For all $i \in \{1, \ldots, m\}$, $\Delta(q_{i-1} \times p_i) \rightarrow q_i$ and $\Delta(q_{i-1} \times (1 - p_i)) \rightarrow \Upsilon(i)$ where $\Upsilon(i)$ denotes the ith entry in Υ (we denote the labels of the states in Q by the sequential integers starting from 1 to $m + 1$. This way, if there is no matching prefix in the ith entry, the automaton goes back to the initial state q_1). P_1 concludes the construction by setting $F = q_m$.

In the next section we show a general protocol evalue to perform a *secure* and *oblivious* evaluation of P_1's automaton on P_2's text. The protocol works for any automaton (not just a KMP one) and therefore may be of independent interest. After showing the automata evaluation protocol, we also show how to prove in zero-knowledge that the automaton P_1 constructs is a correct KMP automaton.

3.3 Secure Oblivious Automata Evaluation

In this section we present a secure protocol for oblivious automata evaluation in the presence of malicious adversaries. In this functionality P_1 inputs a description of an automaton Γ, and P_2 inputs a string t. The result of the protocol is that P_1 receives $\Gamma(t)$, while P_2 learns nothing. Formally, we define this problem via the functionality

$$\mathcal{F}_{\text{AUTO}} : (\Gamma = (Q, \Sigma, \Delta, q_1, F), t) \mapsto \begin{cases} (\mathsf{accept}, \lambda) & \text{if } \Gamma(t) \in F \\ (\mathsf{no-accept}, \lambda) & \text{otherwise} \end{cases}$$

where λ is the empty string (denoting that P_2 does not receive an output) and $\Gamma(t)$ denotes the final state in the evaluation of Γ on t. For a binary inpuy, the transition table contains $|Q|$ rows and two columns. Furthermore, we assume that the names of the states are the integers $\{1, \ldots, |Q|\}$. For simplicity, we assume that $|Q|$ and $|F|$ are public. This is due to the fact that this information is public anyway when reducing the problem of pattern matching into oblivious polynomial evaluation. For the sake of generality we note making $|F|$ private again can be easily dealt by having P_1 send a vector of encryptions for which the ith encryption is a zero encryption only if $q_i \notin F$. Otherwise, it is an encryption of q_i (this can be verified using a simple zero-knowledge proof). The final verification can be done by checking membership in this set using techniques from below.

Recall that our starting point is the protocol from [13]. Their idea is to have the parties share the current machine state, such that by the end of the kth iteration the party with the automaton knows a random string r^k whereas the party with the text learns $q^k + r^k$. The parties complete each iteration by running an oblivious transfer in which the next state is now shared between them. The honest-but-curious setting significantly simplifies their construction. Unfortunately, we cannot see any natural way to extend their technique into the malicious case (even using oblivious transfers resilient to malicious attacks).

A high level description. We begin by briefly motivating our construction; see Fig. 3 as well. Loosely speaking, at the beginning of the protocol P_1 and P_2 jointly

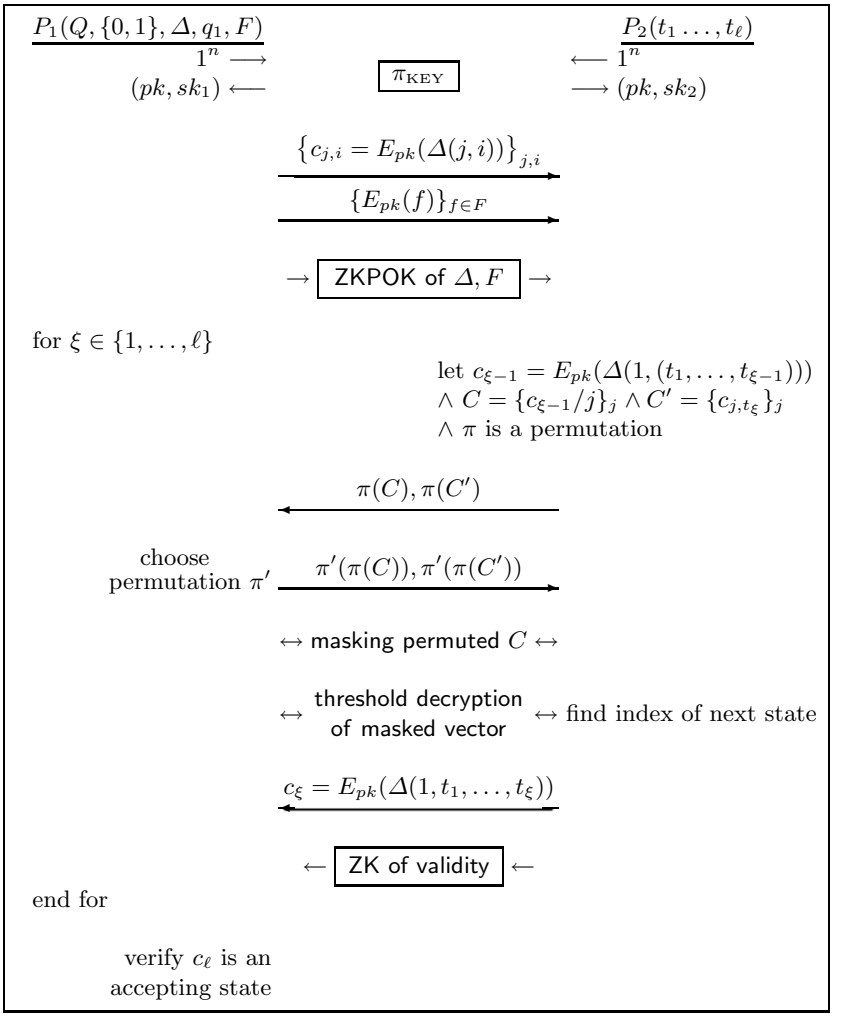

Fig. 3. A high-level diagram of π_{AUTO}

generate a public-key (G, E, D) for the threshold El Gamal encryption scheme (denoted by the sub-protocol π_{KEY}). Next, party P_1 encrypts its transition table Δ and the set of accepting states F, and sends it to P_2. This allows P_2 to find the encryption of the next state $c_1 = \Delta(1, t_1)$, by selecting it from the encrypted matrix. P_2 re-randomizes this encryption and shows it to P_1. The protocol continues in this fashion for the whole text with ℓ iterations.[2]

[2] Unfortunately, these iterations are not independent. Each requires an encryption of the current state, and thus cannot be performed in parallel. We show in Sect. 4 how to minimize the number of rounds into $O(|Q|)$ when performing a secure text search, which is typically quite small.

At the beginning of each iteration, the parties know a randomized encryption of the current state, and their goal is to find an encryption of the next state. At iteration i, P_2 selects from the matrix the entire encrypted column of all possible $|Q|$ next states for its input t_i (only knowing an encryption of the current state). Then, using the homomorphic properties of El Gamal , the parties obliviously select the correct next state: Let $c_{\xi-1}$ denote an encryption of $\Delta(1, t_1, \ldots, t_{\xi-1})$. The parties compute first the set $C = \{c_{\xi-1}/c_{j,\epsilon}\}_j$, where $\{c_{j,\epsilon}\}_j$ correspond to encryptions of the labels in Q; see below for more details. Only one ciphertext in this set will be an encryption of 0, it indicates the position corresponding to the current state. The protocol concludes by the parties jointly checking if the encrypted state that is produced within the final iteration is in the encrypted list of accepting states.

There are several technical challenges in constructing such a secure protocol. In particular the identification of the next encrypted state without leaking additional information requires a couple of rounds of interaction between the parties in which they mask and permute the ciphertext vector containing all possible states, in order to "destroy any link" between their input and the next encrypted state. Moreover, in order to protect against malicious behavior, zero-knowledge proofs are included at each step to make sure parties behave according to the protocol specifications. We are now ready to present a formal description of our protocol.

For a final remark we denote that due to technicalities that arise in the security proof, our protocol employs an unnatural masking technique, where instead of multiplying or adding a random value to each encryption, it uses both. The reason for this becomes clear in the proof.

Protocol 2. $\boldsymbol{\pi_{\text{AUTO}}}$

- **Inputs:** The input of P_1 is a description of an automaton $\Gamma = (Q, \{0,1\}, \Delta, q_1, F)$, and the input of P_2 is a binary string $t = t_1, \ldots, t_\ell$.
- **Auxiliary Inputs:** $|Q|$ and $|F|$ for P_2, ℓ for P_1, and the security parameter 1^n for both.
- **Conventions:** We assume that the parties jointly agree on a group $\mathbb{G}$ of prime order q and a generator g for the threshold El Gamal encryption scheme. Both parties check every received ciphertext for validity, and abort if an invalid ciphertext is received.

  The transition table Δ is defined as $\Delta(j, i)$ which denotes the state that follows state j if the input letter is i. We augment this table by adding a column to Δ labeled by ϵ: we define $\Delta(j, \epsilon) = j$ (the reader can think of it as the "label" of the jth row of the transition table).

  Since we are assuming a binary alphabet, unless written differently, $j \in \{1, \ldots, |Q|\}$ and $i \in \{\epsilon, 0, 1\}$. Finally we assume that the initial state is labeled 1.
- **The protocol:**
  1. **El Gamal key setup:**
     (a) Party P_1 chooses a random value $r_1 \leftarrow_R \mathbb{Z}_q$ and sends party P_2 the value $g_1 = g^{r_1}$. It proves knowledge of r_1 using π_{DL}.

(b) Party P_2 chooses a random value $r_2 \leftarrow_R \mathbb{Z}_q$ and sends party P_1 the value $g_2 = g^{r_2}$. It proves knowledge of r_2 using π_{DL}. The parties set $pk = \langle \mathbb{G}, q, g, h = g_1 \cdot g_2 \rangle$ (i.e., the secret key is $(r_1 + r_2) \bmod q$).

2. **Encrypting P_1 transition table and accepting states:**

(a) P_1 encrypts its (augmented) transition table Δ under pk componentwise; $\Delta_{\mathsf{E}} = \left\{c_{j,i} = E_{pk}(\Delta(j,i))\right\}_{j,i}$. Notice that $c_{j,\epsilon}$ is an encryption of the state j. P_1 also sends the list of encrypted accepting states denoted by $\mathsf{E}(F) = \{E_{pk}(f)\}_{f \in F}$.

(b) For every encryption $\langle c_1, c_2 \rangle \in \Delta_{\mathsf{E}} \cup \mathsf{E}(F)$, P_1 proves the knowledge of $\log_g c_1$ using π_{DL}.

(c) *Proving the validity of the encrypted transition matrix.* P_1 proves that Δ_E is a set of encryptions for values from the set $\{1, \ldots, |Q|\}$. It first sorts the encryptions according to their encrypted values, denoted by $c_1, \ldots, c_{3 \cdot |Q|}$. P_1 multiplies every encryption in this set with a random encryption of 0, sends it to P_2 and proves: firstly, that this vector is a permutation of Δ_E, using π_{PERM}, further that $\bar{c}_i = c_i / c_{i-1} \in \{E_{pk}(0), E_{pk}(1)\}$ by proving that either $(pk, \bar{c}_i)$ or $(pk, \bar{c}_i / E_{pk}(1))$ is a Diffie-Hellman tuple, and, finally, that $\prod_i \bar{c}_i = E_{pk}(|Q|)$. P_1 also decrypts $c_{3 \cdot |Q|}$, which is always an encryption of $|Q|$.

3. **First iteration:**

(a) P_2 chooses the encryption of the next state $c_{1,t_1} = E_{pk}(\Delta(1, t_1))$. It then defines $c_1 = c_{1,t_1} \cdot_G E_{pk}(0)$, i.e. a random encryption of the next state and sends it to P_1.

(b) P_2 proves that $D_{sk}(c_1) \in \{D_{sk}(c_{1,0}), D_{sk}(c_{1,1})\}$ using the zero-knowledge proof π_{ENC} for $m = 1$.

4. **Iterations $\{2, \ldots, \ell\}$:** for every $\xi \in \{2, \ldots, \ell\}$, let $c_{\xi-1}$ denotes the encryption of $\Delta(1, (t_1, \ldots, t_{\xi-1}))$; the parties continue as follows:

(a) **Subtracting the current state from the state labels in Δ:** The parties compute the vector of encryptions $C = \{c_{\xi-1} / c_{j,\epsilon}\}$ for all j. Note that only one ciphertext will denote an encryption of 0, and that indicates the position corresponding to the current state.

(b) P_2 **permutes C and column t_ξ:**

- P_2 computes $C' = \{c_{j,t_\xi} \cdot_G E_{pk}(0)\}$ for all j (note that C' corresponds to column t_ξ in the transition matrix – i.e. the encryptions of all the possible next states given input bit t_ξ) and sends C' to P_1. It also proves that C' were computed correctly using π_{ENC}.
- P_2 chooses a random permutation π over $\{1, \ldots, |Q|\}$ and sends P_1 a randomized version of $\{\pi(C), \pi(C')\}$ That is, the ciphertexts are permuted and randomized by multiplication with $E_{pk}(0)$. The parties engage in zero-knowledge proof π_{PERM} where P_2 proves that it computed this step correctly.

(c) P_1 **permutes $\pi(C)$ and column t_ξ:** Let $C^2_\pi, {C'}^2_\pi$ denote the permuted columns that P_2 sent. If P_1 is accepts the proof π_{PERM} it continues similarly by permuting and randomizing $C^2_\pi, {C'}^2_\pi$ using a new random permutation π'. P_1 proves its computations using π_{PERM}.

(d) **Multiplicative masking:** Let $C^1_\pi, {C'}^1_\pi$ denote the permuted columns from the previous step. C^1_π corresponds to the permuted ϵ column from the transition matrix, which contains the labels of the states minus the label of the current state. The parties take turns in masking C^1_π as follows: for every $\langle c_{j,a}, c_{j,b}\rangle \in C^1_\pi$, P_2 chooses $x \leftarrow_R \mathbb{Z}_q$ and computes $c'_j = \langle c^x_{j,a}, c^x_{j,b}\rangle$. It then proves that $(\mathbb{G}, c_{j,a}, c^x_{j,a}, c_{j,b}, c^x_{j,b})$ is a Diffie-Hellman tuple using π_{DDH}. (Encryptions of zero will be unaffected by this step, while non-zero values will be mapped to random values.) P_1 repeats this step and masks the result. Let $\widetilde{C}_1, \widetilde{C}_2$ be the resulting masked columns for P_1 and P_2 respectively.

(e) **Additive masking:** $\widetilde{C}_1$ corresponds to the permuted ϵ column from the transition matrix, which has by now been masked by both parties. P_2 chooses $|Q|$ random values $\mu^{P_2}_1, \ldots, \mu^{P_2}_{|Q|}$ and encrypts them; $\gamma^{P_2}_i = E_{pk}(\mu^{P_2}_i)$. P_2 also computes $\bar{c}^{P_2}_i = (\tilde{c}_i \cdot_G \gamma^{P_2}_i) \cdot_G E_{pk}(0)$ for every $\tilde{c}_i \in \widetilde{C}_2$, and proves in zero-knowledge that the masking is correct by proving that for every i

$$\left\langle \frac{\bar{c}_{i,a}}{\tilde{c}_{i,a} \cdot \gamma_{i,a}}, \frac{\bar{c}_{i,b}}{\tilde{c}_{i,b} \cdot \gamma_{i,b}} \right\rangle$$

is an encryption of zero, using π_{DDH}. The ciphertext $\tilde{c}_i$ that denotes an encryption of zero is now mapped to the ciphertext $\bar{c}_i$ that contains an encryption of μ_i. The others are mapped to random values. Let $\overline{C}_2 = [\bar{c}_1, \ldots, \bar{c}_{|Q|}]$ denote this masked vector. P_1 performs this step as well to obtain $\overline{C}_1$.

(f) **Decrypting column ϵ:** The parties decrypt it using their shared knowledge of sk. In particular, for every $\bar{c}_j \in \overline{C}$ in which $\bar{c}_j = \langle \bar{c}_{j,a}, \bar{c}_{j,b}\rangle$, P_1 computes $c'_j = \bar{c}^{r_1}_{j,a}$ and proves that $(\mathbb{G}, g, g^{r_1}, c'_j, \bar{c}_{j,a})$ is a Diffie-Hellman tuple. Next P_2 computes $c''_j = \bar{c}^{r_2}_{j,a}$ and proves that $(\mathbb{G}, g, g^{r_2}, c''_j, \bar{c}_{j,a})$ is a Diffie-Hellman tuple. The parties decrypt $\bar{c}_j$ by computing $D_{sk}(\bar{c}_j) = \bar{c}_{j,b}/(c'_j \cdot c''_j)$. Each party P_i sends its additive shares; $\mu^{P_i}_1, \ldots, \mu^{P_i}_{|Q|}$, and proves their correctness via π_{DDH}. The parties chooses the index j for which there exists $D_{sk}(\bar{c}_j) = \mu^{P_1}_j + \mu^{P_2}_j$ (with high probability there will be only one such index).

5. **Checking the output:** After the ℓth iteration c_ℓ contains the encryption of $\Delta(1,t)$. To check if this is an accepting state without revealing any other information (in particular which state it is) the parties do the following:

(a) They compute the ciphertext vector $C_F = \{c_\ell / c\}_{c \in \mathsf{E}(F)}$. Notice that $\Delta(1,t)$ is accepting if and only if one of these ciphertexts is an encryption of 0.

(b) P_2 masks the ciphertexts as in Steps 4d and 4e. Let C'_F be the resulting vector.
(c) P_2 randomizes and permutes C'_F. It also proves correctness using π_{PERM}. Let C''_F be the resulting vector.
(d) The parties decrypt all the ciphertexts in C''_F with the result going only to P_1; for every ciphertext $c = \langle c_a, c_b \rangle \in C''_F$, the party P_2 sends $c'_a = c_a^{r_2}$ and proves that $(\mathbb{G}, g, g^{r_2}, c_a, c'_a)$ is a Diffie-Hellman tuple. This information allows P_1 to decrypt the ciphertexts, and P_1 accepts if one of the decryptions equals one of the additive mask of P_2.

Before turning to the security proof we show that if both parties are honest then P_1 outputs $\Gamma(t)$ with probability negligibly close to 1. This is because with each iteration ξ the parties agree upon the correct encrypted state c_ξ with probability close to 1. We continue with the following claim,

Theorem 3. *Assume that π_{DL}, π_{DDH}, π_{ENC} and π_{PERM} are as described above and that (G, E, D) is the semantically secure El Gamal encryption scheme. Then π_{AUTO} securely computes $\mathcal{F}_{\text{AUTO}}$ in the presence of malicious adversaries.*

Intuitively it should be clear that the automaton and the text remain secret, if the encryption scheme is secure. However, a formal proof of Theorem 3 is actually quite involved. Consider, for example, the case in which P_1 is corrupted and we need to simulate P_2. The simulator is going to choose a random input and run P_2's code on it. Then, to prove that this view is indistinguishable, we need a reduction to the encryption scheme. A straightforward reduction does not work for the following reason: in order for the simulator to finish the execution correctly it must "know" the current state, at every iteration i; but when we do a reduction to El Gamal we need to plug in a ciphertext for the current state for which we do not know a decryption, and this prevents us from going forward to iteration $i + 1$. A non-trivial solution to this problem is to prove that the real and simulated views are indistinguishable via a sequence of hybrid games, in which indistinguishable changes are introduced to the way the simulator works, but still allowing it to finish the simulated execution.

As for the case that P_2 is corrupted, the proof is rather simple mainly because P_2 does not receive an output. Specifically, the simulator extracts P_2's input in every iteration using the extractor for π_{ENC}.

Efficiency. We present an analysis of our protocol and compare its efficiency to the generic protocols of [14,16] for secure two-party computation in the presence of malicious adversaries. We introduced the efficency of the generic protocols in Sec. 1. A circuit that computes $\mathcal{F}_{\text{AUTO}}$ would require $O(\ell Q \log Q)$ gates. While there exists a sequential circuit of size $O(\ell Q)$ that computes a Q-state automaton over a binary input of size ℓ, the generic protocol applies only to combinatorial circuits. So, our protocol improves the computational complexity over the generic solution by a factor of $n \log Q$ or $\log Q$ operations compared to [14] and [16] respectively.[3] In comparison to [14,16], we have the following:

[3] Although not presented here, our protocol generalizes to strings over a larger alphabet, and this provides another $\log \Sigma$ factor improvement where Σ is the size of the alphabet.

1. *Rounds of Communication:* Our protocol runs $O(\ell)$ rounds where ℓ is the length of the text, compared to the constant round complexity of [14,16]. This is due to the fact that the parties cannot initiate a new iteration before completing the previous one. By applying known techniques, the round complexity can be reduced to $O(m)$ (i.e., the length of the pattern) by breaking the text into blocks of size $2m$; see Sect. 4 for more details. We note that the length of the pattern is typically very small, usually up to a constant. An additional improvement can be achieved if some leakage is allowed. In particular, since the number of rounds are determined by the length of the pattern, the pattern can be broken into smaller blocks and the output combined out of these results. This means that additional information about the text is released, as it is then possible to identify appearances in the text that correspond to substrings of the input pattern.
2. *Asymmetric computations:* The overall number of exponentiations in protocol $\pi_{\text{VALIDAUTO}}$, including the zero-knowledge proofs is $O(m \cdot \ell)$. As for [14], the number of such computations depends on the number of oblivious transfer executions, which is bounded by $\max(4\ell, 8s)$ where and the number of commitments $2\ell s(s+1)$, where s must be large enough so that $2 * 2^{\frac{-s}{17}}$ is sufficiently small. Finally, [16] requires $O(|C|)$ such operations. However, after carefully examining these costs, it seems that the parties compute approximately 720 RSA exponentiations per gate, where the number of gates is $O(m \cdot \ell)$, which is clearly not practical. Furthermore, the protocol of [16] also requires a security parameter, usually of size of at least 1024, due to the strong RSA assumption. Consequently, all the public-key operations are performed over groups modulus this value (or higher, such as N^2). In contrast, our protocol uses the El Gamal encryption scheme which can be implemented over elliptic curve group, typically, using only 160 bit keys.
3. *Symmetric operations:* [14], due to the use of a symmetric encryption scheme, also requires $O(s|C| + s^2 \cdot n)$ such symmetric computations.
4. *Bandwidth:* In our protocol, the parties send each other $O(m \cdot \ell)$ encryptions. The bandwidth of the protocol in [16] is similar (again, with relatively high constants), whereas in [14], the bandwidth is dominated by the $O(s|C|+s^2 n)$ symmetric-key encryptions. In [15], it is shown that communication is the main bottleneck when implementing the malicious protocol of [14].

In order to conclude the comparison, we note that implementing a circuit that solves the basic pattern matching problem may be significantly harder than implementing our protocol.

3.4 A Zero-Knowledge Proof of Knowledge for a KMP Automaton

Space does not permit a full treatment of our protocol for $\pi_{\text{VALIDAUTO}}$, but we we include a description of the protocol here:

Protocol 3. $\pi_{\text{VALIDAUTO}}$

- **Auxiliary input for the prover:** A collection $\{Q_{i,j}, r_{i,j}\}_{i,j}$ of Q sets, each set is of size 2, such that $c_{i,j} = E_{pk}(Q_{i,j}; r_{i,j})$ for all $i \in \{0,1\}$ and $j \in \{1,\ldots,|Q|\}$.
- **Auxiliary inputs for both:** A prime p such that $p-1 = 2q$ for a prime q, the description of a group $\mathbb{G}$ of order q for which the DDH assumption holds, and a generator g of $\mathbb{G}$.
- **The protocol:**
  1. For every $c_{i,j} = \langle \alpha_{i,j}, \beta_{i,j} \rangle$, the prover P proves the knowledge of $\log_g \alpha_{i,j}$ using π_{DL}.
  2. **For every row $\Delta_j = \{c_{j,0}, c_{j,1}\}_{j=1}^{|Q|}$ in the transition matrix P proves the following:**
     (a) It first randomly permutes $c_{j,0}$ and $c_{j,1}$ and employs π_{PERM} to prove its computations.
     (b) It proves that there exists $b \in \{0,1\}$ in which $c_{j,b} = E_{pk}(j+1)$ by proving that $(pk, c_{j,b}/E_{pk}(j+1))$ is a Diffie-Hellman tuple.
     (c) P **proves the correctness of $c_{j,1-b}$ in two steps:** (i) It first proves that it corresponds to a valid prefix of $p_{\langle j-1 \rangle}$, (ii) then it proves the maximality of this prefix ($p_{\langle r \rangle}$ denotes the rth length prefix $p_1, \ldots, p_r$ of p).
        i. Let $|Q|-1 = m$, then the verifier V chooses m random elements $u_\alpha \leftarrow_R \mathbb{Z}_q$ and sends $\{u_\alpha\}_\alpha$ to P. Next, both parties use the homomorphic properties of the encryption scheme to compute an encryption $v_{\alpha',\alpha''} = E_{pk}(\sum_{k=\alpha'}^{\alpha''} u_k \cdot p_k)$ for all $\alpha', \alpha'' \in \{1,\ldots,m\}$ with $\alpha' < \alpha''$.
        ii. **Proving the existence of a prefix that matches a suffix of $p_{\langle j-1 \rangle}$:** For all $1 \le k \le j-1$ the parties compute an encryption $v'_k = (v_{j-k,j-1}/v_{1,k}) \cdot (c_{j,1-b}/k)$.
        P then proves that there exists k for which v'_k is a Diffie-Hellman tuple.
        iii. **Proving that $p_{\langle D_{sk}(c_{i,b}) \rangle}$ corresponds to the longest suffix of $p_{\langle j-1 \rangle}$:**
        Next P proves that there does not exist an index $D_{sk}(c_{i,1-b}) < k \le j-1$ in which $v_{j-k,j-1}/v_{1,k} = 0$ yet $c_{j,1-b}/k \neq 0$, as this would imply that $p_{\langle k \rangle}$ is a larger suffix of $p_{\langle j-1 \rangle}$ that matches however, $D_{sk}(c_{j,1-b}) \neq k$.
        - Therefore, for every $1 \le k \le j-2$ and for every $2 \le k' \le j-1$ the parties compute an encryption $e_{k,k'} = v'_k \cdot (v_{j-k',j-1}/v_{1,k'})$ for which P then proves that $e_{k,k'}$ is not an encryption of zero using π_{NZ}.
  3. If all the proofs are successfully completed, V outputs 1. Otherwise it outputs 0.

We refer the reader to the extended version of this paper for a detailed definition and proof.

4 Text Search Protocol with Simulation Based Security

In this section we present our complete and main construction for securely evaluating the pattern matching functionality $\mathcal{F}_{\text{PM}}$ defined by

$$(p,(T,m)) \mapsto \begin{cases} (\{i \mid T_i = p\}, \lambda) & \text{if } |p| = m \\ (\{i \mid T_i = p_1 \dots p_m\}, \lambda) & \text{otherwise} \end{cases}$$

Recall that our construction is presented in the malicious setting with full simulatability and is modular in the sub-protocols π_{AUTO} and $\pi_{\text{VALIDAUTO}}$. Having described the sub-protocols incorporated in the our scheme we are now ready to describe it formally. Our protocol is comprised out of two main phases: first the parties first engage in an execution of $\pi_{\text{VALIDAUTO}}$ for which P_1 proves that it indeed sent a valid KMP automaton, followed by an execution of π_{AUTO} which in an evaluation of Γ on P_2's private input.

In order to reduce the round complexity of the protocol, long texts are partitioned into $2m$ pieces and are handled separately so that the KMP algorithm is employed on each block independently (thus all these executions can be run in parallel). That is, let $T = t_1, \dots, t_\ell$ then the text is partitioned into the blocks $(t_1, \dots, t_{2m})$,$(t_m, \dots, t_{3m})$,$(t_{2m}, \dots, t_{4m})$ and so on, such that every two consecutive blocks overlap in m bits. This ensures that all the matches will be found. Therefore, the total number of blocks is ℓ/m.

Protocol 4. $\boldsymbol{\pi_{\text{PM}}}$

- **Inputs:** The input of P_1 is a binary pattern $p = p_1, \dots, p_m$, and the input of P_2 is a binary string $T = t_1, \dots, t_\ell$.
- **Auxiliary Inputs:** the security parameter 1^n, and the input sizes ℓ and m.

- **The protocol:**
  1. P_1 constructs its automaton $\Gamma = (Q, \Sigma, \Delta, q_1, F)$ according to the KMP specifications based on its input p and sends P_2 encryptions of the transition matrix Δ and the accepting states F, denoted by E_Δ and E_F.
  2. The parties engage in an execution of the zero-knowledge proof $\pi_{\text{VALIDAUTO}}$ where P_1 proves that Γ was constructed correctly. That is, P_1 proves that the set E_Δ corresponds to a valid KMP automaton and a well defined input pattern of length m. If P_2's output from this execution is 1 the parties continue to the next step. Otherwise P_2 aborts.
  3. P_2 sends an encryption of T to P_1 and the parties partition T into ℓ/m blocks of length $2m$ in which every two consecutive blocks overlap in m bits.
  4. The parties engage in ℓ/m parallel executions of π_{AUTO} on these blocks.[4] For every $1 \le i \le \ell/m$, let $\{\mathsf{output}_j^i\}_{j=1}^{m+1}$ denotes the set of P_1's outputs from the ith execution. Then P_1 returns $\{j \mid \mathsf{output}_j^i = \text{``}\mathsf{accept}''\}$.

[4] The parties run a slightly modified version of π_{AUTO} where they carry out Step 5 for verifying acceptance $m+1$ times, as each block contains potentially $m+1$ matches. This step can be executed in parallel for all block locations.

Theorem 4. *Assume that* π_{AUTO} *and* $\pi_{\text{VALIDAUTO}}$ *are as described above and that* (G, E, D) *is the semantically secure El Gamal encryption scheme. Then* π_{PM} *securely computes* $\mathcal{F}_{\text{PM}}$ *in the presence of malicious adversaries.*

The security proof for π_{PM} is a combination of the proofs described for for π_{AUTO} and $\pi_{\text{VALIDAUTO}}$ and is therefore omitted here.

Efficiency. Since the costs are dominated by the costs of π_{AUTO}, we refer the reader to the detailed analysis presented in Sect. 3.3. The overall costs are amount to $O(m \cdot \ell + m^2) = O(m \cdot \ell)$ since in most cases $m << \ell$.

5 Conclusion

Our protocols for oblivious automata evaluation and pattern matching operate in the standard model and require no CRS nor apply a cut-and-choose strategy. Having P_1, P_2 hold strings of lengths ℓ, m for the text and the pattern respectively, both protocols incur communication and computation costs of $O(\ell \cdot m)$ which is even asymptotically better then the general construction for the oblivious automata evaluation that requires a circuit with $O(\ell \cdot m \log m)$ gates. Table 2 summarizes and compares the computational and communications costs of each of these schemes.

Table 2. Summary of results

	Round Complexity	Communication Complexity	Asymmetric Computations	Symmetric Computations
Lindell-Pinkas [14]	constant	$O(s\|C\| + s^2 m)$ times 128/256 bits	$\max(4m, 8s)$ OT's	$O(s\|C\| + s^2 \cdot m)$
Jarecki-Shmatikov [16]	constant	$O(\ell \cdot m)$ times 2048 bits	720 exp. per gate	None
Our Protocol	$O(m)$	$O(\ell \cdot m)$ times 160 bits	$O(\ell \cdot m)$	None

References

1. Canetti, R.: Security and composition of multi-party cryptographic protocols. Journal of Cryptology 13, 2000 (1998)
2. Goldwasser, S., Levin, L.A.: Fair computation of general functions in presence of immoral majority. In: Menezes, A., Vanstone, S.A. (eds.) CRYPTO 1990. LNCS, vol. 537, pp. 77–93. Springer, Heidelberg (1991)
3. Beaver, D.: Foundations of secure interactive computing. In: Feigenbaum, J. (ed.) CRYPTO 1991. LNCS, vol. 576, pp. 377–391. Springer, Heidelberg (1992)
4. Micali, S., Rogaway, P.: Secure computation (abstract). In: Feigenbaum, J. (ed.) CRYPTO 1991. LNCS, vol. 576, pp. 392–404. Springer, Heidelberg (1992)
5. Yao, A.C.C.: How to generate and exchange secrets. In: SFCS 1986: Proceedings of the 27th Annual Symposium on Foundations of Computer Science, Washington, DC, USA, pp. 162–167. IEEE Computer Society, Los Alamitos (1986)

6. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game. In: STOC 1987: Proceedings of the nineteenth annual ACM symposium on Theory of computing, pp. 218–229. ACM, New York (1987)
7. Goldreich, O.: Foundations of Cryptography: Basic Applications, vol. 2. Cambridge University Press, New York (2004)
8. Knuth Jr., D.E., Morris, J.H., Pratt, V.R.: Fast pattern matching in strings. SIAM J. Comput. 6(2), 323–350 (1977)
9. Hazay, C., Lindell, Y.: Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries. In: Canetti, R. (ed.) TCC 2008. LNCS, vol. 4948, pp. 155–175. Springer, Heidelberg (2008)
10. Diffie, W., Hellman, M.E.: New directions in cryptography. IEEE Transactions on Information Theory IT-22(6), 644–654 (1976)
11. El Gamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. In: Blakely, G.R., Chaum, D. (eds.) CRYPTO 1984. LNCS, vol. 196, pp. 10–18. Springer, Heidelberg (1985)
12. Jarecki, S., Xiaomin, L.: Efficient oblivious pseudorandom function with applications to adaptive ot and secure computation of set intersection. In: Reingold, O. (ed.) TCC 2009. LNCS, vol. 5444, pp. 577–594. Springer, Heidelberg (2009)
13. Troncoso-Pastoriza, J.R., Katzenbeisser, S., Celik, M.: Privacy preserving error resilient dna searching through oblivious automata. In: CCS 2007: Proceedings of the 14th ACM conference on Computer and communications security, pp. 519–528. ACM, New York (2007)
14. Lindell, Y., Pinkas, B.: An efficient protocol for secure two-party computation in the presence of malicious adversaries. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 52–78. Springer, Heidelberg (2007)
15. Pinkas, B., Schneider, T., Smart, N.P., Williams, S.C.: Secure two-party computation is practical. In: ASIACRYPT 2009, Tokyo, Japan. LNCS, pp. 250–267. Springer, Heidelberg (2009)
16. Jarecki, S., Shmatikov, V.: Efficient two-party secure computation on committed inputs. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 97–114. Springer, Heidelberg (2007)
17. Schnorr, C.P.: Efficient identification and signatures for smart cards. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 239–252. Springer, Heidelberg (1990)
18. Chaum, D., Pedersen, T.P.: Wallet databases with observers. In: Brickell, E.F. (ed.) CRYPTO 1992. LNCS, vol. 740, pp. 89–105. Springer, Heidelberg (1993)
19. Hazay, C., Nissim, K.: Efficient set operations in the presence of malicious adversaries (2010)
20. Groth, J., Ishai, Y.: Sub-linear zero-knowledge argument for correctness of a shuffle. In: Smart, N.P. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 379–396. Springer, Heidelberg (2008)
21. Groth, J., Lu, S.: Verifiable shuffle of large size ciphertexts. In: Okamoto, T., Wang, X. (eds.) PKC 2007. LNCS, vol. 4450, pp. 377–392. Springer, Heidelberg (2007)