Automata Evaluation and Text Search Protocols with Simulation Based Security

Rosario Gennaro* Carmit Hazay† Jeffrey S. Sorensen‡

June 3, 2010

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 joint computation of some function over private inputs. This joint computation must satisfy at least *privacy* (no other information is revealed beyond the output of the function) and *correctness* (the correct output is computed). In order to achieve this, the parties engage in a communication protocol. Today's standard definition (cf. [1] following [2, 3, 4]) formalizes security by comparing the execution of such 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; of course, 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 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

*IBM T.J. Watson Research Center, Yorktown Heights, NY Email:rosario@us.ibm.com.

†Dept. of Computer Science and Applied Mathematics, Weizmann Institute and IDC, Israel. Email: carmit.hazay@weizmann.ac.il.

‡Google, New York, NY Email: sorenj@google.com.

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 a pattern p, where the size of 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 with the text T. We develop a protocol for two parties (one holding an automaton Γ and another holding an input text T) to securely compute the evaluation of Γ on T. This protocol can be of independent interest beyond the pattern matching application and can be considered an extension of the work by Ishai and Paskin [9]. In their work, Ishai and Paskin considered the model of obliviously evaluating branching programs (a deterministic automaton is a special case of a branching program). In this model, the communication is limited to an amount that is proportional to the input for the branching program and independent of the description of the program. Still, only privacy is guaranteed. Our protocol achieves full security, but the amount of work is proportional to the size of the automaton (program).

Pattern matching has been widely studied for decades due to its broad applicability. Yet, most of the suggested constructions do not achieve any level of security, even for the most limited settings. Therefore, we starting with an extremely efficient protocol that computes this function in the "honest-but-curious" setting.[1] This solution can be extended to be secure in the case of one-sided simulation, where full simulation is provided for one of the corruption cases, while only privacy (via computational indistinguishability) is guaranteed for the other corruption case. This protocol is comparable to the one-sided simulatable protocol of [10].

Moving to the malicious setting introduces quite a few subtleties and requires the use of a different technique. To achieve full simulatability, we introduce a second independent protocol, which employs several other novel sub-protocols, including, 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.

Our algorithm's security is based on the El Gamal encryption scheme [11, 12] and thus requires a relatively small security parameter (although any additively homomorphic threshold encryption scheme with secure two-party distributed protocols to generate shared keys and perform decryptions would work).

Motivation. Secure pattern matching has many potential applications. Consider, for example, the hypothetical case of 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 forbidden from releasing the DNA records to a third party. Likewise, the researcher may not want to reveal what specific gene she is working on, nor trust the hospital to perform the search correctly.

[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.

It would seem that existing honest-but-curious solutions 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 [10] 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, but 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 [13] 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 [14]. Their protocols, however, are only secure in the honest-but-curious setting. We extend these 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 "generic" solutions for secure two-party computation of *any* function already present in the literature. We are going to compare our protocols to the two most efficient generic two-party protocols against malicious adversaries, both based on the garbling-technique by Yao [5]. Recall that Yao's protocol (which is secure against semi-honest players) uses a Boolean circuit that computes the function, and its computational complexity is linear in the size of the circuit.

- Lindell and Pinkas [15]: in order to make Yao resistant to malicious adversaries, their scheme uses binary cut-and-choose strategy, so it requires running s copies in parallel 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. The main drawback of [15] the communications costs associated with $O(s|C| + s^2 n)$ symmetric-key encryptions. A recent paper by Pinkas et al. [16] showed that communication is the major bottleneck when implementing the malicious protocol of [15].

- Jarecki and Shmatikov [17] uses a special form of encryption for the garbling, and performs efficient zero-knowledge (ZK) proofs over that encryption scheme. The protocol however 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 external help. Furthermore, their protocol requires approximately 720 RSA exponentiations per gate, where these operations are computed modulo 2048 due to the Paillier's encryption scheme. This means that the bandwidth of [17] is relatively high as well.

Our protocols for oblivious automata evaluation and pattern matching operate in the standard model and require no CRS, nor do they 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. Finally, we note that our protocols apply the El Gamal encryption scheme [12] that can be implemented over an elliptic curve group. This, in turn, reduces the modulus value dramatically, typically only 160 bits for keys. We present detailed comparisons with these constructions in Section 3.2.1.

	Round Complexity	Communication Complexity	Asymmetric Computations	Symmetric Computations
Lindell-Pinkas	constant	$O(s\|C\| + s^2 n)$ times 128/256 bits	$\max(4n, 8s)$ OT's	$O(s\|C\| + s^2 \cdot n)$
Jarecki-Shmatikov	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

Figure 1: Comparison of Pattern Matching Protocols

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 $a \in \{0,1\}^*$.

2.1 Definition of Secure Two-Party Computation for malicious Adversaries

In this section we briefly present the standard definition for secure multiparty computation and refer to [7, Chapter 7] for more details and a motivating discussion.[2]

Two-party computation. A two-party protocol can be systematically analyzed by characterizing the protocol as a random process that maps pairs of inputs to pairs of outputs (one for each party). We refer to such a process as a functionality and denote it as $f : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^* \times \{0,1\}^*$, where $f = (f_1, f_2)$. That is, for every pair of inputs (x, y), the output is a random variable $(f_1(x,y), f_2(x,y))$ ranging over pairs of strings where P_1 receives $f_1(x,y)$ and P_2 receives

[2]This section is adapted, with permission, from [10].

$f_2(x, y)$. We sometimes denote such a functionality by $(x, y) \mapsto (f_1(x, y), f_2(x, y))$. Thus, for example, the oblivious transfer functionality is denoted by $((x_0, x_1), \sigma) \mapsto (\lambda, x_\sigma)$, where (x_0, x_1) is the first party's input, σ is the second party's input, and λ denotes the empty string (meaning that the first party receives no output).

Adversarial behavior. The aim of a secure multiparty protocol is to protect honest parties against dishonest behavior by other parties. However, in full-simulation based security we are also concerned with *malicious adversaries* who control some subset of the parties and may instruct them to arbitrarily deviate from the specified protocol. We also consider *static corruptions*, meaning that the set of corrupted parties is fixed at the onset.

Security of protocols (informal). The security of a protocol is analyzed by comparing what an adversary can do in a real protocol execution to what it can do in an ideal scenario that is secure by definition. This is formalized by considering an *ideal* computation involving an incorruptible *trusted third party* to whom the parties send their inputs. The trusted party computes the functionality on the inputs and returns to each party its respective output. A protocol is secure if any adversary interacting in the real protocol (where no trusted third party exists) can do no more harm than if it was involved in the above-described ideal computation. One technical detail that arises in the setting of no honest majority is that it is impossible to achieve fairness or guaranteed output delivery [18]. That is, it is possible for the adversary to prevent the honest party from receiving outputs. Furthermore, it may even be possible for the adversary to receive output while the honest party does not.

Execution in the ideal model. In an ideal execution, the parties send their inputs to the trusted party who computes the output. An honest party just sends the input that it received whereas a corrupted party can replace its input with any other value of the same length. Since we do not consider fairness, the trusted party first sends the output of the corrupted parties to the adversary, and the adversary then decides whether the honest parties receive their (correct) outputs or an abort symbol $\perp$. Let f be a two-party functionality where $f = (f_1, f_2)$, let $\mathcal{A}$ be a non-uniform probabilistic polynomial-time machine, and let $I \subseteq [2]$ be the set of corrupted parties (either P_1 is corrupted, or P_2 is corrupted, or neither). Then, the ideal execution of f on inputs (x, y), auxiliary input z to $\mathcal{A}$ and security parameter n, denoted $\text{IDEAL}_{f,\mathcal{A}(z),I}(x, y, n)$, is defined as the output pair of the honest party and the adversary $\mathcal{A}$ from the above ideal execution.

Execution in the real model. In the real model there is no trusted third party and the parties interact directly. The adversary $\mathcal{A}$ sends all messages in place of the the corrupted party, and may follow an arbitrary polynomial-time strategy. In contrast, the honest parties follow the instructions of the specified protocol π.

Let f be as above and let π be a two-party protocol for computing f. Furthermore, let $\mathcal{A}$ be a non-uniform probabilistic polynomial-time machine and let I be the set of corrupted parties. Then, the real execution of π on inputs (x, y), auxiliary input z to $\mathcal{A}$ and security parameter n, denoted $\text{REAL}_{\pi,\mathcal{A}(z),I}(x, y, n)$, is defined as the output vector of the honest parties and the adversary $\mathcal{A}$ from the real execution of π.

Security as emulation of a real execution in the ideal model. Having defined the ideal and real models, we can now define the security of protocols. Loosely speaking, the definition asserts

that a secure multi-party protocol (in the real model) emulates the ideal model (in which a trusted party exists). This is formulated by saying that adversaries in the ideal model are able to simulate executions of the real-model protocol.

Definition 1 *Let f and π be as above. Protocol π is said to* securely compute f with abort in the presence of malicious adversaries *if for every non-uniform probabilistic polynomial-time adversary $\mathcal{A}$ for the real model, there exists a non-uniform probabilistic polynomial-time adversary $\mathcal{S}$ for the ideal model, such that for every $I \subseteq [2]$,*

$$\left\{\text{IDEAL}_{f,\mathcal{S}(z),I}(x,y,n)\right\}_{x,y,z\in\{0,1\}^*,n\in\mathbb{N}} \stackrel{\text{c}}{\equiv} \left\{\text{REAL}_{\pi,\mathcal{A}(z),I}(x,y,n)\right\}_{x,y,z\in\{0,1\}^*,n\in\mathbb{N}}$$

where $|x| = |y|$.

2.2 Sequential Composition

Sequential composition theorems are important security goals in and of themselves. Sequential composition theorems are useful tools that help in writing proofs of security. The basic idea behind these composition theorems is that it is possible to design a protocol that uses an ideal functionality as a subroutine, and then analyze the security of the protocol when a trusted party computes this functionality. For example, assume that a protocol is constructed that uses the secure computation of some functionality as a subroutine. Then, first we construct a protocol for the functionality in question and then prove its security. Next, we prove the security of the larger protocol that uses the functionality as a subroutine in a model where the parties have access to a trusted party computing the functionality. The composition theorem then states that when the "ideal calls" to the trusted party for the functionality are replaced by real executions of a secure protocol computing this functionality, the protocol remains secure.

The hybrid model. The aforementioned composition theorems are formalized by considering a *hybrid model* where parties both interact with each other (as in the real model) and use trusted help (as in the ideal model). Specifically, the parties run a protocol π that contains "ideal calls" to a trusted party computing some functionalities $f_1, \ldots, f_m$. These ideal calls are just instructions to send an input to the trusted party. Upon receiving the output back from the trusted party, the protocol π continues. We stress that honest parties do not send messages in π between the time that they send input to the trusted party and the time that they receive back output (this is because we consider *sequential* composition here). Of course, the trusted party may be used a number of times throughout the π-execution. However, each time is independent (i.e., the trusted party does not maintain any state between these calls). We call the regular messages of π that are sent amongst the parties standard messages and the messages that are sent between parties and the trusted party ideal messages.

Let $f_1, \ldots, f_m$ be probabilistic polynomial-time functionalities and let π be a two-party protocol that uses ideal calls to a trusted party computing $f_1, \ldots, f_m$. Furthermore, let $\mathcal{A}$ be a non-uniform probabilistic polynomial-time machine and let I be the set of corrupted parties. Then, the $f_1, \ldots, f_m$-hybrid execution of π on inputs (x, y), auxiliary input z to $\mathcal{A}$ and security parameter n, denoted $\text{HYBRID}^{f_1,\ldots,f_m}_{\pi,\mathcal{A}(z),I}(x,y,n)$, is defined as the output vector of the honest parties and the adversary $\mathcal{A}$ from the hybrid execution of π with a trusted party computing $f_1, \ldots, f_m$.

Sequential modular composition. Let $f_1, \ldots, f_m$ and π be as above, and let $\rho_1, \ldots, \rho_m$ be protocols. Consider the real protocol $\pi^{\rho_1,\ldots,\rho_m}$ that is defined as follows: All standard messages of π are unchanged. When a party P_i is instructed to send an ideal message α_i to the trusted party to compute functionality f_j, it begins a real execution of ρ_j with input α_i instead. When this execution of ρ_j concludes with output β_i, party P_i continues with π as if β_i was the output received by the trusted party (i.e. as if it were running in the $f_1, \ldots, f_m$-hybrid model). Then, the composition theorem of [1] states that if ρ_j securely computes f_j for every $j \in \{1, \ldots, m\}$, then the output distribution of a protocol π in a hybrid execution with $f_1, \ldots, f_m$ is computationally indistinguishable from the output distribution of the real protocol $\pi^{\rho_1,\ldots,\rho_m}$. This holds for security in the presence of malicious adversaries [1] and one-sided simulation when considering the corruption case that has a simulator (an easy corollary from [1]).

2.3 One-Sided Simulation for Two-Party Protocols

Two of our protocols achieve a level of security that we call one-sided simulation. In these protocols, P_2 receives output while P_1 should learn nothing. In one-sided simulation, *full simulation* is possible when P_2 is corrupted. However, when P_1 is corrupted we only guarantee *privacy*, meaning that P_1 learns nothing whatsoever about P_2's input (this is straightforward to formalize because P_1 receives no output). This is a relaxed level of security and does not achieve everything we want; for example, independence of inputs and correctness are not guaranteed. Nevertheless, for this level of security we are able to construct highly efficient protocols that are secure in the presence of malicious adversaries.

Formally, let $\text{REAL}_{\pi,\mathcal{A}(z),i}(x, y, n)$ denote the output of the honest party and the adversary $\mathcal{A}$ (controlling party P_i) after a real execution of protocol π, where P_1 has input x, P_2 has input y, $\mathcal{A}$ has auxiliary input z, and the security parameter is n. Let $\text{IDEAL}_{f,\mathcal{S}(z),i}(x, y, n)$ be the analogous distribution in an ideal execution with a trusted party who computes f for the parties. Finally, let $\text{VIEW}^{\mathcal{A}}_{\pi,\mathcal{A}(z),i}(x, y, n)$ denote the view of the adversary after a real execution of π as above. Then, we have the following definition:

Definition 2 *Let f be a functionality where only P_2 receives output. We say that a protocol π* securely computes *f* with one-sided simulation *if the following holds:*

1. *For every non-uniform* PPT *adversary $\mathcal{A}$ controlling P_2 in the real model, there exists a non-uniform* PPT *adversary $\mathcal{S}$ for the ideal model, such that*

$$\left\{\text{REAL}_{\pi,\mathcal{A}(z),2}(x, y, n)\right\}_{x,y,z\in\{0,1\}^*,n\in N} \stackrel{\text{c}}{\equiv} \left\{\text{IDEAL}_{f,\mathcal{S}(z),2}(x, y, n)\right\}_{x,y,z\in\{0,1\}^*,n\in N}$$

   where $|x| = |y|$.

2. *For every non-uniform* PPT *adversary $\mathcal{A}$ controlling P_1, and every polynomial $p(\cdot)$*

$$\left\{\text{VIEW}^{\mathcal{A}}_{\pi,\mathcal{A}(z),1}(x, y, n)\right\}_{x,y,y',z\in\{0,1\}^*,n\in N} \stackrel{\text{c}}{\equiv} \left\{\text{VIEW}^{\mathcal{A}}_{\pi,\mathcal{A}(z),1}(x, y', n)\right\}_{x,y,y',z\in\{0,1\}^*,n\in N} \quad (1)$$

   where $|x| = |y| = |y'|$.

Note that the ensembles in Eq. (1) are indexed by two different inputs y and y' for P_2. The requirement is that $\mathcal{A}$ cannot distinguish between the case that P_2 used the first input y or the second input y'.

2.4 Finite Automata

A deterministic finite automaton is described by a tuple $\Gamma = (Q, \Sigma, \Delta, q_0, F)$, where Q is the set of states, Σ is an alphabet of inputs, $\Delta : Q \times \Sigma \to Q$ denotes a state-transition table, $q_0 \in Q$ is the initial state, and $F \subseteq Q$ is the set of final (or accepting) states. Without loss of generality, we consider only automata with complete transition tables, where there exists a transition at each state for every input $\sigma \in \Sigma$. We also consider the notation of $\Delta(q_0, (\sigma_1, \ldots, \sigma_\ell))$ to denote the result of the automaton evaluation on $\sigma_1, \ldots, \sigma_\ell$. Every automaton specifies a *language*, which is the (potentially infinite) set of strings accepted by the automaton.

2.5 The El Gamal Encryption Scheme

We consider the following modification of the El Gamal encryption scheme [12]. The public-key is the tuple $pk = \langle \mathbb{G}, q, g, h \rangle$ and the corresponding private key is $sk = \langle \mathbb{G}, q, g, x \rangle$, where $\mathbb{G}$ is a cyclic group of prime order q with a generator g (we assume multiplication and group membership can be performed efficiently in $\mathbb{G}$). In addition, it holds that $h = g^x$.

Encryption of a message $m \in [1, \ldots, q']$ (with $q' << q$) is performed by choosing $r \leftarrow_R \mathbb{Z}_q$ and computing $E_{pk}(m; r) = \langle g^r, h^r \cdot g^m \rangle$. Decryption of a ciphertext $c = \langle a, b \rangle$ is performed by computing $g^m = b \cdot a^{-x}$, and then finding m by exhaustive search, or some more efficient (but still super-polynomial) method such as the baby-step giant-step algorithm. So this scheme works only for small integer domains (i.e. q' must be small) which is the case for our protocol. Note that a zero encryption corresponds to $E_{pk}(0; r) = \langle g^r, h^r \cdot g^0 \rangle = \langle g^r, h^r \rangle$

The security of this scheme relies on the hardness of solving the DDH problem. The reason we modify El Gamal in this way (by encrypting g^m rather than m) is to make it additively homomorphic.

Homomorphic encryption. We abuse notation and use $E_{pk}(m)$ to denote the distribution $E_{pk}(m; r)$ where r is chosen uniformly at random.

Definition 3 *A public-key encryption scheme (G, E, D) is* homomorphic *if, for all n and all (pk, sk) output by $G(1^n)$, it is possible to define groups $\mathbb{M}, \mathbb{C}$ such that:*

- *The plaintext space is $\mathbb{M}$, and all ciphertexts output by E_{pk} are elements of $\mathbb{C}$.*
- *For any $m_1, m_2 \in \mathbb{M}$ and $c_1, c_2 \in \mathbb{C}$ with $m_1 = D_{sk}(c_1)$ and $m_2 = D_{sk}(c_2)$, it holds that*

$$\{pk, c_1, c_1 \cdot c_2\} \equiv \{pk, E_{pk}(m_1), E_{pk}(m_1 + m_2)\}$$

where the group operations are carried out in $\mathbb{C}$ and $\mathbb{M}$, respectively.

Our modification of El Gamal is homomorphic with respect to component-wise multiplication of ciphertexts. We denote by $c_1 \cdot_G c_2$ the respective multiplications of $c_1^1 \cdot c_2^1$ and $c_1^2 \cdot c_2^2$ where $c_i = \langle c_i^1, c_i^2 \rangle = E_{pk}(m_i)$, such that the multiplication result yields the encryption of $m_1 + m_2$.

Threshold encryption. We denote the key generation functionality by $\mathcal{F}_{\text{KEY}}$ and the corresponding protocol by π_{KEY}. the key generation functionality $\mathcal{F}_{\text{KEY}}$ is informally defined as follows:

$$(1^n, 1^n) \mapsto \Big((pk, sk_1), (pk, sk_2) \Big), \tag{2}$$

where $(pk, sk) \leftarrow_R G(1^n)$, and sk_1 and sk_2 are random shares of sk. The decryption functionality is defined by

$$(c, pk) \mapsto \Big((m : c = E_{pk}(m)), \lambda\Big), \tag{3}$$

An efficient threshold El Gamal scheme can be constructed based on the protocol of Diffie and Hellman [11]; see details below.

2.6 Zero-Knowledge Proofs

Our protocols use the following standard zero-knowledge proofs:

Protocol	Relation/Language	Reference
π_{DL}	$\mathcal{R}_{\text{DL}} = \{((\mathbb{G}, g, h), x) \mid h = g^x\}$	[19]
π_{DDH}	$\mathcal{R}_{\text{DDH}} = \{((\mathbb{G}, g, g_1, g_2, g_3), x) \mid g_1 = g^x \wedge g_3 = g_2^x\}\}$	[20]
π_{NZ}	$\text{L}_{\text{NZ}} = \{(\mathbb{G}, g, h, \langle\alpha, \beta\rangle) \mid \exists\ (m \neq 0, r)$ s.t. $\alpha = g^r,\ \beta = h^r g^m\}$	[21]

It further employs the following zero-knowledge proofs for which we provide detailed constructions

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 same messages encrypted in either C_0 or C_1, or, in other words, 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,

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

In the proof the joint statement is a collection of three vectors, and the prover produces proofs that the third vector is a randomized version of either the first or the second vector. We continue with our protocol,

Protocol 1 (zero-knowledge proof of knowledge for $\mathcal{R}_{\text{ENC}}$):

- **Joint statement:** *The set* $(\mathbb{G}, g, Q, C_0, C_1, C')$.
- **Auxiliary input for the prover:** *An index* i *and a set* $\{r_j\}_j$ *as in* $\mathcal{R}_{\text{ENC}}$.
- **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:**
  (a) *Let* $C_i = [c_{i,1}, ..., c_{i,Q}]$ *for* $i \in \{0, 1\}$ *and* $C' = [c'_1, ..., c'_Q]$. *Then the parties compute 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.*
  (b) *The prover then proves that either* $\langle pk, c_0\rangle$ *or* $\langle pk, c_1\rangle$ *is a Diffie-Hellman tuple.*

Proposition 2.1 *Assume that the DDH assumption holds relative to* $\mathbb{G}$. *Then Protocol* 1 *is a statistical zero-knowledge proof of knowledge for* $\mathcal{R}_{\text{ENC}}$ *with perfect completeness and negligible soundness.*

It is easy to verify that the verifier is always convinced by an honest prover. The arguments for zero-knowledge and knowledge extraction can be derived from [22].

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,

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

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

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

and has been widely studied in the literature. The state-of-the-art protocol is in [23] (we refer the reader to [23] also for a complete list of relevant work in the area). We are going to use a simpler, slightly less efficient (but still good for our purposes) protocol by Groth and Lu [24]. They presented 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. The reason we use a slightly less efficient protocol is due to the fact that it is easy to show that this proof is applicable to the case where the same permutation is applied to more than one vector of ciphertexts (as we require), and because it can be applied to the El Gamal encryption scheme.

3 Secure Text Search Protocols

Text search algorithms can be broadly broken down into algorithms that are *sequential* where the text is searched by scanning for all occurrences of a particular *pattern*. Efficient variants of this approach analyze the pattern string to enable $O(\ell)$ scanning that skips over regions of text whenever possible matches are provably not possible. This includes the widely studied Knuth-Morris-Pratt [8], Boyer-Moore [25], and most recently, Factor Oracle [26] based algorithms.

Alternatively, algorithms based upon analysis of the text to be searched are categorized as *index* based search. In this category are *suffix tree* based algorithms which build a data structure in $O(\ell)$ time and storage. In practice, the storage requirements for suffix trees can be quite a large multiple of the text length ℓ, but indexes that are sub-linear, but bounded by the entropy of the text, is surveyed in [27].

Finally, for completeness, there also exists algorithms, for searching that build partial or "fuzzy" indexes based upon n-grams or Bloom filters [28]. This includes *inverted index* based approaches. These probabilistic algorithms can not easily be bounded in security properties nor in running time for general texts. For some applications areas, such as natural language search, these approaches may yield more practical solutions, and this type of indexing was suggested in [29] in a similar security context. This paper is, however, concerned with general texts.

The pattern matching problem is defined as follows: given a binary text 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

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

where λ is an empty string, T_i is defined as above.

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. As discussed above, this problem has been intensively studied and can be solved optimally in time that is linear in size of the text and the number of occurrences, independent of the pattern size.

3.1 Honest-But-Curious Secure Text Search

The protocol employs the properties of homomorphic encryption to compute the sum of the differences between the pattern and the text. Informally, party P_1 computes a matrix Φ of size $2 \times m$ that includes an encryption of zero in position (i, j) if $p_j = i$ and 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 $j \in \{1, \ldots, m\}$. Now, since e_k is equal to the Hamming distance between p and T_k, this guarantees that if p matches T_k, e_k is indeed a random zero encryption. Figure 2 illustrates the approach schematically.

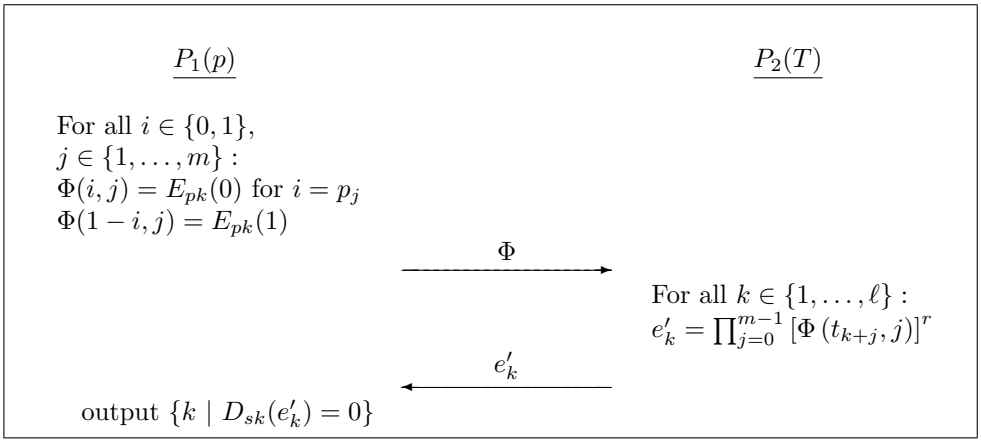

Figure 2: Text search in the honest-but-curious setting

Formally, **Protocol π_{SIMPLE}**

- **Inputs:** The input of P_1 is a binary search string $p = p_1, \ldots, p_m$ and P_2 a binary text string $T = t_1, \ldots, t_\ell$
- **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 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). Then we state the following,

Theorem 4 *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, that $\Phi(0, j), \Phi(1, j)$ 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, concluding that the protocol achieves one-sided simulation; see Section 2.3 for the formal security definition. 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 5 *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.*

Proof Sketch: Assume P_1 is malicious. Then we need to present a simulator S which plays the role of P_2 and builds a view for P_1 which is (computationally) indistinguishable from the one of the real protocol without knowing the real input text held by P_2.

The crucial point is that at the end of Step 1 (which in π'_{SIMPLE} includes the above zero-knowledge proof) the simulator can learn the input of P_1 (i.e. the pattern p) by extracting it from the proof of knowledge π_{PERM}. At this point the simulator is also given the output of the protocol by the ideal-model trusted party, i.e. S knows in which locations of the input text of P_2 the pattern appears. S then chooses a text T' which contains the pattern in the exact same locations but is otherwise an arbitrary string $p' \neq p$. It then runs the rest of the protocol using T'. It is not hard to see that the view of P_1 produced by S is actually identically distributed as the real one.

As for the case that P_2 is corrupted, we only need to prove that the privacy of P_1 is preserved. This can be shown via a reduction to the security of El Gamal . Moreover, it can be proven that for every two bit strings p, p' of length m, a corrupted P_2 cannot distinguish an execution where P_1 enters p, from an execution where P_1 enters p'. ■

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. Thus, the total amount of work is $O(\ell \cdot m)$.

Because this protocol does not seem to have a natural extension to address malicious adversaries, we propose a quite different protocol in the next section.

We conclude by remarking that protocol π'_{SIMPLE} takes a different approach than the one-sided simulatable protocol of [10]. In particular, while both protocols reach the same asymptotic complexity, our protocol is much more practical since the concrete constants that are involved are much smaller (as the protocol of [10] requires $|p|$ oblivious transfer evaluations). Moreover, our protocol can be easily applied to closely related problems such as approximate text search (matching with errors) or text search with wildcards.

We present three generalizations of the classic pattern matching problem to other problems of practical interest. Due to the similarities to protocol π'_{SIMPLE} we omit the proofs.

Approximate text search. In text search with mismatches, there exists an additional public parameter ρ which determines the number of mismatched that can be tolerated. Specifically, P_1 should learn all the text locations in which the Hamming distance between the pattern and the substring at these text locations is smaller equal to ρ. More formally, we consider the functionality for approximate text search $\mathcal{F}_{\text{APM}}$ that is defined by

$$((p,\rho),(T,m,\rho')) \mapsto \begin{cases} (\{i \mid d(T_i,p) \leq \rho\},\lambda) & \text{if } |p| = m \text{ and } \rho = \rho' \\ (\lambda,\lambda) & \text{otherwise} \end{cases}$$

where $d(x,y)$ denotes the Hamming distance of two binary strings. The most recent algorithm for solving this problem in an insecure setting is the solution by Amir et. al. [30] who introduces a solution in $O(\ell\sqrt{\rho \log \rho})$ time. A simple extension to our protocol yields a protocol that computes this functionality as well. That is, upon completing its computations and before masking the terms as in Step 3 of π'_{SIMPLE}, party P_2 produces $\rho+1$ encryptions from each encryption e_k by subtracting from it all the values between $[0,...,\rho]$. Finally, it masks these encryptions and randomly shuffles the result. Clearly, if both parties are honest then P_2's output is correct with overwhelming probability.

Furthermore, we remark that the simulation for corrupted P_1 does not change, since now the simulator receives from the trusted party all the text locations where the pattern matches with at most ρ mismatches. The proof for the case that P_2 is corrupted is identical to the proof above. Let π_{APM} denote protocol π'_{SIMPLE} with the above modification, then it holds that

Theorem 6 *Assume that (G,E,D) is the semantically secure El Gamal encryption scheme. Then Protocol π_{APM} securely computes $\mathcal{F}_{\text{APM}}$ with one-sided simulation.*

The communication and computation complexities are $O(\rho \cdot \ell)$.

Text search with wildcards. A wildcard symbol matches against any character when searching the text. Note that in protocol π'_{SIMPLE}, a wildcard can be emulated by having P_1 send two encryptions of zero instead of encryptions of zero and one. By doing so, we make sure that regardless of the text bit, P_2 will not count it as a mismatch. Note that the number of exponentiations of this modified protocol, π_{DC}, is linear in the length of the text just as in the standard insecure setting [31]. More formally,

Theorem 7 *Assume that (G, E, D) is the semantically secure El Gamal encryption scheme. Then Protocol π_{DC} securely computes the pattern matching problem with wildcards with one-sided simulation.*

The security proof is as above except that P_1 uses a slightly different proof of knowledge. In particular, it proves the statement in which each encryption within Φ belongs to the set $\{E_{pk}(0), E_{pk}(1)\}$ and that there does not exists $1 \leq j \leq m$ such that $\{\Phi(0, j), \Phi(1, j)\}$ denote a pair of encryptions $\{E_{pk}(1), E_{pk}(1)\}$. These proofs are standard for the El Gamal encryption scheme.

Q-ary alphabet. Recalling that protocol π'_{SIMPLE} compares binary strings and computes the pattern matching functionality for the binary alphabet. However, in some scenarios the pattern and the text are defined over a larger alphabet Σ, (e.g., when searching in a DNA database the alphabet is of size four.) Note that when no security properties are required, dealing with larger alphabet is rather straightforward.

When T and p are drawn from a q-ary alphabet, the algorithm π_{SIMPLE} can be extended to the case where Φ is a q by m matrix. In this case, P_1 must prove that each row of Φ is a permutation of a vector of $\{E_{pk}(0), E_{pk}(1), \ldots, E_{pk}(1)\}$, using π_{PERM} with a single encryption of zero $q - 1$ encryptions of one. The size of the alphabet appears as a multiplicative cost for both computations and communications. The security proof in this extension is not appreciably different from the binary case.

State	Prefix	$\Upsilon(q_i)$	Fail State	$\Gamma(q_i, j)$ $j = 0$	$j = 1$
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	$\Gamma(q_2, 0)$	$\Gamma(q_2, 1)$

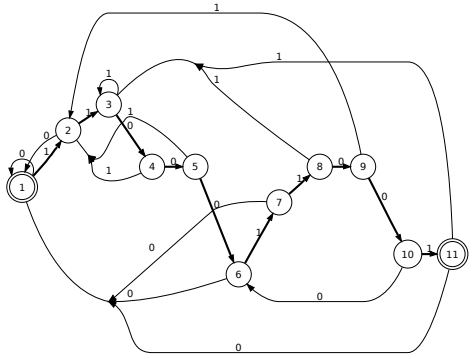

Figure 3: Construction of determinized KMP automaton for pattern 1100011001

3.2 Secure Text Search in the Presence of Malicious Adversaries

We consider a secure version of the KMP algorithm [8]. The KMP algorithm 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 $p_{\langle i-1 \rangle}$. Namely, the table points to the largest prefix $p_{\langle i' \rangle}$ that matches a suffix of $p_{\langle i-1 \rangle}$. Assume that we have already successfully compared the first $i-1$ bits of p against the text, yet we encounter a mismatch when comparing the ith bit of p (we store the length of this prefix). The automaton encodes the appropriate transition to begin comparing the next potential match. If we ever reach the last state, we output a match.

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 $i \in \{1, \ldots, m\}$, $\Delta(q_{i-1} \times p_i) \to q_i$ and $\Delta(q_{i-1} \times (1-p_i)) \to \Upsilon(i)$ 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 $F = q_m$.

Now, we need a way for P_1 and P_2 to jointly compute the result of running the automaton on P_2's text, such that no information is revealed about either the text or the automaton (besides knowing if the final state is accepting or not). In the next section we show a general protocol to perform such a *secure* and *oblivious* evaluation of an automaton. The protocol works for any automaton (not just a KMP one) and therefore can 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.

3.2.1 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

$$\mathcal{F}_{\text{AUTO}} : (\Gamma = (Q, \Sigma, \Delta, q_0, F), t) \mapsto \begin{cases} (\text{accept}, \lambda) & \text{if } \Gamma(t) \in F \\ (\text{no-accept}, \lambda) & \text{otherwise} \end{cases}$$

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 this protocol, we consider a binary alphabet. It therefore holds that 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 set membership using techniques from below.

Recall that our starting point is the protocol from [14] which was the first to use oblivious automata evaluation, secure in the honest-but-curious setting. 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 input for the automaton 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 fact that the parties are honest-but-curious significantly simplifies their construction.

Unfortunately, we cannot see any natural way to extend their technique to the malicious adversary case (even when using oblivious transfer that is resilient to malicious attacks). Coping with such behavior is much more challenging, starting with requiring a proof that a valid automaton is indeed used, and verifying that the intermediate computations do not leak any information about the parties' inputs. Thus, our construction takes a different approach.

A high level description. we begin by briefly motivating our construction; see Figure 4 as well. At the beginning of the protocol P_1 and P_2 jointly 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. Note that this immediately 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 ℓ iterations (the length of the text).[3]

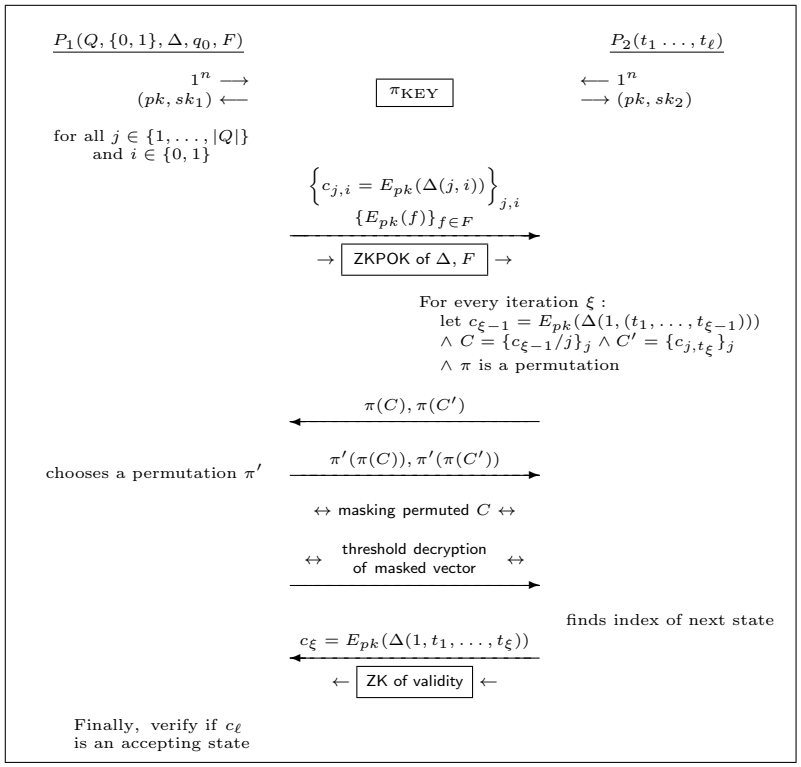

Figure 4: A high-level diagram of π_{AUTO}.

At the outset of each iteration the parties know a randomized version of the encryption of the current state, and their goal is to find an encryption of the next state. Informally, at iteration i, P_2 selects from the matrix the entire encrypted column of all possible $|Q|$ next states for its input t_i (as it only knows an encryption of the current state). Then, using the homomorphic properties

[3]Unfortunately, these iterations are not independent and thus cannot be employed in parallel. This is due to the fact that the parties must start every iteration with an encryption of the current state. Nevertheless, we show in Section 4 how to minimize the number of rounds into $O(|Q|)$ when performing a secure text search, which is typically quite small.

of El Gamal , the parties obliviously select the correct next state; this stage involves the following computations. Let $c_{\xi-1}$ denote an encryption of $\Delta(1, t_1, \ldots, t_{\xi-1})$. Then 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. Note that only one ciphertext in this set will be an encryption of 0, and that 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. But, as a final remark we note 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 π_{AUTO}:

- **Inputs:** The input of P_1 is a description of an automaton $\Gamma = (Q, \{0,1\}, \Delta, q_0, F)$, 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.

  We also make the following modification: recall that 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 add 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 transition table Δ under pk component-wise; $\Delta_{\mathsf{E}} = \left\{c_{j,i} = E_{pk}(\Delta(j,i))\right\}_{j,i}$. 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 $c_{3\cdot|Q|}$ (note that this encryption always denotes 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 $c_{\xi-1}$ denote 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 also chooses a random permutation π over $\{1, \ldots, |Q|\}$ and sends P_1, a randomized version of $\{\pi(C), \pi(C')\}$ (i.e. the ciphertexts are not only permuted but also randomized by multiplication with $E_{pk}(0)$). The parties engage in a zero-knowledge proof π_{PERM} for which 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 above step. Recall that C^1_π corresponds to the permuted ϵ column from the transition matrix, which denotes the labels of the states minus the label of the current state. Then the parties take turn 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}. Notice that the ciphertext that denotes an encryption of 0 value (i.e. an encryption of g^0), will not be influenced by the masking, while the others are mapped to a random value. P_1 repeats this step and masks the result.
   (e) **Additive masking:** Let $\widetilde{C}_1, \widetilde{C}_2$ denote the masked columns from the above step. Recall that $\widetilde{C}_1$ corresponds to the permuted ϵ column from the transition matrix, which has by now been masked by both parties. Then P_2 chooses $|Q|$ random values $\mu_1, \ldots, \mu_{|Q|}$ and encrypts them; $\gamma_i = E_{pk}(\mu_i)$. It also computes $\bar{c}_i = (\tilde{c}_i \cdot_G \gamma_i) \cdot_G E_{pk}(0)$ for every $\tilde{c}_i \in \widetilde{C}_1$, and proves in zero-knowledge that the masking is correct by proving that for every i the ciphertext
   $$\left\langle \frac{\bar{c}_{i,a}}{\tilde{c}_{i,a} \cdot \gamma_{i,a}}, \frac{\bar{c}_{i,b}}{\tilde{c}_{i,b} \cdot \gamma_{i,b}} \right\rangle$$
   is an encryption of zero, using π_{DDH}. Notice that the ciphertext $\tilde{c}_i$ that denotes an encryption of 0 (i.e. an encryption of g^0), is now mapped to the ciphertext $\bar{c}_i$ that contains an encryption of μ_i. The others are mapped to random values. P_1 repeats this step and masks the result.
   (f) **Decrypting column** ϵ**:** Let $\overline{C} = [\bar{c}_1, \ldots, \bar{c}_{|Q|}]$ denote the masked vector computed in the previous step. 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; $\mu_1^{P_i}, \ldots, \mu_{|Q|}^{P_i}$, and proves their correctness via π_{DDH}. The parties chooses the index j for which there exists $D_{sk}(\bar{c}_j) = \mu_j^{P_1} + \mu_j^{P_2}$ (with high probability there will be only one such index).

5. **Checking output:** After the ℓth iteration we have that c_ℓ denotes the encryption of $\Delta(1,t)$. To check if this is an accepting state or not 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. In other words, 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. P_1 accepts if one of the decryptions equals one of the additive masks 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. The reason for that is due to the fact that in every iteration ξ, the parties agree on the correct encrypted next state; c_ξ, with probability close to 1 (since the probability that there exists more than one element in $\overline{C}$ that is an encryption of zero is negligible).

We continue with the following claim,

Theorem 8 *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 8 is actually quite involved. Consider for example, the case of the proof 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}.

Proof: We separately prove security in the case that P_1 is corrupted and the case that P_2 is corrupted. Our proof is in a hybrid model where a trusted party computes the ideal functionalities for $\mathcal{R}_{\text{DL}}$, $\mathcal{R}_{\text{DDH}}$ and $\mathcal{R}_{\text{PERM}}$.

Party P_1 is corrupted. Let $\mathcal{A}$ denote an adversary controlling P_1. We construct a simulator $\mathcal{S}$ as follows,

1. $\mathcal{S}$ is given a description of an automaton $\Gamma = (Q, \{0,1\}, \Delta, q_0, F)$ and $\mathcal{A}$'s auxiliary input and invokes $\mathcal{A}$ on these values.

2. $\mathcal{S}$ completes the key setup stage as the honest P_2 would. It receives the input from the ideal computation of $\mathcal{R}_{\text{DL}}$, i.e. the value r_1 used by P_1 in the key setup.

3. $\mathcal{S}$ receives from $\mathcal{A}$ the encryptions Δ_{E} and $\mathsf{E}(F)$ and verifies its proofs for a valid automaton. If the verification fails $\mathcal{S}$ sends $\perp$ to the trusted party for $\mathcal{F}_{\text{AUTO}}$ and halts.

4. $\mathcal{S}$ receives the inputs for the ideal computations of $\mathcal{R}_{\text{DL}}$; where for every $\langle c_1, c_2\rangle \in \Delta_{\mathsf{E}} \cup \mathsf{E}(F)$, $\mathcal{A}$ sends $((\mathbb{G}, g, \langle c_1, c_2\rangle), t)$. $\mathcal{S}$ first verifies that $c_1 = g^t$ and sends $\perp$ to the trusted party for $\mathcal{F}_{\text{AUTO}}$ in case it does not hold, otherwise it defines $\mathcal{A}$'s input as follows. For every $c_{j,i} \in \Delta_{\mathsf{E}}$ it sets $sk' = t_{j,i}$ (that denotes the witness of $c_{j,i}$ for π_{DL}) and computes $\Delta(j,i) = D_{sk'}(c_{j,i})$.[4] $\mathcal{S}$ computes the set of accepting states F the same way (as $\mathcal{A}$ may send encryptions of invalid accepting states, $\mathcal{S}$ records only the valid states that correspond to values within $[1, \ldots, Q]$). If the recorded set Δ does not constitute a valid transition matrix, $\mathcal{S}$ outputs fail.

5. $\mathcal{S}$ sends $\Gamma = (Q, \{0,1\}, \Delta, q_0, F)$ to its trusted party, where Δ and F are as recorded in Step 4 of the simulation. If it receives back the message "accept" it chooses an arbitrary string $t' = t'_1 \ldots t'_\ell$ for which $\Gamma(t') \in F$. Else it chooses a string t' such that $\Gamma(t')$ is not an accepting state.

6. $\mathcal{S}$ completes the execution as the honest P_2 would on this input. Specifically, in the first iteration, $\mathcal{S}$ chooses c_{1,t'_1} and sends $\mathcal{A}$, $c_1 = c_{1,t'_1} \cdot E_{pk}(0)$. It then emulates the trusted party for $\mathcal{R}_{\text{DDH}}$ and hands $\mathcal{A}$ the value 1 for proving that it computed c_1 correctly.

7. In every iteration ξ, $\mathcal{S}$ plays the role of the honest P_2, emulating the ideal computations for $\mathcal{R}_{\text{PERM}}$ and $\mathcal{R}_{\text{DDH}}$.

8. $\mathcal{S}$ outputs whatever $\mathcal{A}$ does.

We first note that $\mathcal{S}$ outputs fail with negligible probability due to the negligible soundness of the proof. In particular, if $\mathcal{A}$ sends an encryption of a value not in $[1, \ldots, Q]$ then the proof fails since either there exists an index i in which $\bar{c}_i$ is not an encryption of zero or one, or $c_{3\cdot|Q|}$ is not an encryption of $|Q|$ (where the decryption of $c_{3\cdot|Q|}$ is needed to prevent from $\mathcal{A}$ to use a sequence of $|Q|$ labels that do not correspond to $[1, \ldots, |Q|]$). Next we show that the output distribution of $\mathcal{A}$ in the hybrid and the simulated executions are computationally indistinguishable. Recall that $\mathcal{S}$ plays against $\mathcal{A}$ with input t' so that $\Gamma(t) \in F$ if and only if $\Gamma(t') \in F$ where t is the input of the real P_2. The intuition of the proof follows from the security of El Gamal, where the adversary should not be able to distinguish between an encrypted path of the automaton that was computed relative to t or t'. Formally, we define a sequence of hybrid games and denote by the random variable $\mathrm{H}_\ell^{\mathcal{A}(z)}(\Gamma = (Q, \{0,1\}, \Delta, q_0, F), t, n)$ (for a fixed n) the output of $\mathcal{A}$ in hybrid game H_ℓ.

Recall that the difference between the simulated and the real executions is due to the fact that $\mathcal{S}$ enters $t' = t'_1 \ldots, t'_\ell$ whereas P_2 enters $t = t_1 \ldots, t_\ell$. Assume for contradiction the existence of a distinguisher circuit D between the real and simulated execution. A natural hybrid argument

[4] We exploit here the symmetry of El Gamal where instead of decrypting using the original secret-key, the simulator decrypts using the discrete logarithm of c_1 as follow. Let $c = \langle c_1, c_2\rangle$ and let t denotes $\log_g c_1$, then $\mathcal{S}$ computes c_2/h^t where $h \in pk$. Note that $c_2 = h^t \cdot m$ and thus we get the correct decryption. Note also that at this point $\mathcal{S}$ knows the secret key so it can decrypt directly, but in some of our hybrid arguments we will make a reduction to the security of the encryption scheme, and in this case $\mathcal{S}$ will not know the secret key.

implies that there must exist an iteration i in which D distinguishes between an execution on the input $\tilde{t}_i = t'_1 \ldots, t'_{i-1}, t_i, \ldots, t_\ell$ and an execution on the input $\tilde{t}_{i+1} = t'_1 \ldots, t'_i, t_{i+1}, \ldots, t_\ell$. Our goal will be to show that for any i the views of those two executions are computationally indistinguishable.

Game H_0**:** The simulated execution, as described above.

We begin with the following *intermediate game*, as it may hold that the automaton accepts the real and the simulated inputs but not the hybrid's strings, or vice-versa.

Game H'_0**:** In this game there is no trusted party and no honest P_2. Instead, we define a new simulator $\mathcal{S}'$ which is given the real input t of party P_2 and evaluates the automaton on t by itself. Furthermore, $\mathcal{S}'$ runs exactly like $\mathcal{S}$ in H_0 using t' as its input except for the last step, when P_1 checks the output, it changes the way it masks. Remember that in Step 4e for every ciphertext $c \in \widetilde{C}_1$, party P_2 (as well as the simulator $\mathcal{S}$) sends the values $\{\mu_j\}_j$, $\{\gamma_j\}_j$ and $\{\bar{c}_j\}_j$, and proves correctness via π_{DDH}. Then the simulator $\mathcal{S}'$ publishes a set of masks that is independent of $\bar{C}$. That is, in Step 4e, for every j, $\mathcal{S}'$ sends a random μ_j that is independent of γ_j, and then proves that the collection of these encryptions was computed correctly by emulating the ideal execution of $\mathcal{R}_{\mathrm{DDH}}$. Nevertheless, $\mathcal{S}'$ computes the set of encryptions $\{\bar{c}_j\}_j$ consistently with the masks $\{\mu_j\}_j$. We note that this game is required as it may be that the automaton's outputs on the hybrid strings may not be consistent with its output on t (and t').

The output distributions in games H_0 and H'_0 are computationally indistinguishable via a reduction to the hardness of DDH. Assume, for contradiction, the existence of a distinguisher circuit D for H_0 and H'_0. Construct a distinguisher circuit D_{ddh} that distinguishes between a Diffie-Hellman tuple and a a non Diffie-Hellman tuple.[5] D_{ddh} receives a tuple $\bar{c} = \langle g, h, (h_1^1, h_2^1), \ldots, (h_1^{|Q|}, h_2^{|Q|}) \rangle$ and works as follows. It first sets $h = pk$ by sending the value h/g^{r_1} in the key setup phase. Next it chooses random masks $(\mu_1, \ldots, \mu_{|Q|})$ and computes $\hat{h}_2^i = h_2^i \cdot \mu_i$ and $\gamma_i = \langle h_1^i, \hat{h}_2^i \rangle$ for all i. D_{ddh} completes the execution using these values. Note that D_{ddh} is able to decrypt in Step 5d even without the knowledge of the discrete logarithm of its "share" h/g^{r_1}, as it knows the decrypted values and the share of the adversary. Then the output distribution generated by D_{ddh} is either identical to the simulation or to the current game.

Next, we prove that every two consecutive executions on $\tilde{t}_i$ and $\tilde{t}_{i-1}$ are computationally indistinguishable. Formally, we denote by the random variable $\mathrm{H}_\ell^{\mathcal{A}(z),i}(\Gamma = (Q, \{0,1\}, \Delta, q_0, F), t, n)$ (for a fixed n and i) the output of $\mathcal{A}$ in the hybrid game H_ℓ^i. We focus on the ith iteration for the following sequence of games,

Game H_0^i**:** In this game the simulator behaves as in H'_0, but enters the string $\tilde{t}_i$ as its input. Note that when $i = \ell$ this game is identical to game H'_0.

Game H_1^i**:** In this game the simulator $\mathcal{S}_1^i$ publishes a set of masks that is independent of $\bar{C}$ as in game H'_0 above.

Game H_2^i**:** The simulator $\mathcal{S}_2^i$ is identical to $\mathcal{S}_1^i$ except that it uses the bit t_i in Step 4b of the protocol instead of t'_i, that is $\mathcal{S}_2^i$ sends to $\mathcal{A}$ column $\{c_{j,t_i}\}_j$ instead of column $\{c_{j,t'_i}\}_j$. Notice that the rest of the execution is unchanged. In particular, $\mathcal{S}_2^i$ behaves exactly as in the previous game placing random ciphertexts and masks in Step 4e. Also, it still concludes iteration i with c_i (which denotes the next state as computed in the ith iteration) equal $\Delta(1, t'_1, \ldots, t'_{i-1}, t'_i)$ as in the previous hybrid game (therefore ignoring the fact that t_i was entered previously).

[5] We consider a game where a distinguisher D is given a tuple of the form $\langle g, h, (h_1^1, h_2^1), \ldots, (h_1^\ell, h_2^\ell) \rangle$ where for all i, $\langle g, h, h_1^i, h_2^i \rangle$ is either a Diffie-Hellman tuple or not. In particular, D outputs its guess and wins the game if its guess is correct. The hardness of this game can be reduced to the hardness of DDH using a standard hybrid argument.

Due to the fact that the only difference between H_2^i and H_1^i is which vector $\mathcal{S}_2^i$ sends to $\mathcal{A}$ in Step 4b, clearly the adversary views in these two games are computationally indistinguishable via a reduction to the semantic security of El Gamal .

Game $\mathrm{H}_2^{i,k}$ for $k = 1, ..., \ell - i$: This is a series of $\ell - i$ games. For ease of notation, let $\mathcal{S}_2^{i,0} = \mathcal{S}_2^i$ and $\mathrm{H}_2^{i,0} = \mathrm{H}_2^i$. We are going to show that $\mathrm{H}_2^{i,k}$ is indistinguishable from $\mathrm{H}_2^{i,k+1}$.

The simulator $\mathcal{S}_2^{i,k}$ in game $\mathrm{H}_2^{i,k}$ is identical $\mathcal{S}_2^{i,k-1}$ except that in Step 4b of iteration $i + k$, it does not send the permuted vector C' of column t_{i+k}, but rather it modifies it into a new vector C'' for which it replaces the entry *in position* $\Delta(1, t'_1, \ldots, t'_{i-1}, t'_i, \ldots, t_{i+k-1})$ with the *state value* $\Delta(1, t'_1, \ldots, t'_{i-1}, t_i, \ldots, t_{i+k-1})$.

The views of the adversary in games $\mathrm{H}_2^{i,k}$ and $\mathrm{H}_2^{i,k+1}$ are computationally indistinguishable via a reduction to the semantic security of El Gamal (since we are modifying the distribution of the plaintexts in the permuted column).

Game H_3^i: The simulator $\mathcal{S}_3^i$ is identical to $\mathcal{S}_2^{i,\ell-i}$ except that it modifies Step 4b of iteration $i + 1$ (not i like before). Recall that in this step P_2 sends a randomized and permuted version of C, the vector of ciphertexts that contains 0 in the current state location. In game H_2^i the current state location at iteration $i + 1$ is $\Delta(1, t'_1, \ldots, t'_{i-1}, t'_i)$. The simulator $\mathcal{S}_3^i$ instead sends a vector which has an encryption of 0 in location $\Delta(1, t'_1, \ldots, t'_{i-1}, t_i)$, and random ciphertexts everywhere else. It emulates the $\mathcal{R}_{\mathrm{PERM}}$ ideal functionality to simulate the zero-knowledge proof.

Again the adversary's views in games H_2^i and H_3^i are computationally indistinguishable, via the semantic security of El Gamal .

Game H_4^i: The simulator $\mathcal{S}_4^i$ identical to $\mathcal{S}_3^i$ except that in Step 4f of the ith iteration it decrypts according to $\Delta(1, t'_1, \ldots, t'_{i-1}, t_i)$. More specifically: the simulator extracts the permutation used by P_1 in Step 4c, and therefore knows in which position j the state $\Delta(1, t'_1, \ldots, t'_{i-1}, t_i)$ should be in Step 4d. For that position the simulator will then claim that $\bar{c}_j$ encrypts the same value as γ_j by simulating π_{DDH} (recall that, since game H_1^i, all the masks and ciphertexts are random.)

In this case, the adversary's views in games H_3^i and H_4^i are identical, as the simulator is "cheating" in the selection of $\bar{c}_j$ in both.

Game H_5^i: The simulator $\mathcal{S}_5^i$ identical to $\mathcal{S}_4^i$ except that it modifies Step 4b of iteration $i + 1$ (not i) by reversing the actions of $\mathcal{S}_3^i$. In other words this time the randomized and permuted version of C (the vector of ciphertexts that contains 0 in the current state location) is computed correctly according to $\Delta(1, t'_1, \ldots, t'_{i-1}, t_i)$.

The adversary's views in games H_4^i and H_5^i are computationally indistinguishable, via a reduction to the semantic security of El Gamal .

Game $\mathrm{H}_6^{i,k}$ for $k = \ell - i - 1, ..., 0$: This is a series of $\ell - i$ games. For ease of notation, let $\mathcal{S}_6^{i,\ell-i} = \mathcal{S}_5^i$ and $\mathrm{H}_6^{i,\ell-i} = \mathrm{H}_6^i$. We are going to show that $\mathrm{H}_6^{i,k}$ is indistinguishable from $\mathrm{H}_6^{i,k+1}$.

The simulator $\mathcal{S}_6^{i,k}$ in game $\mathrm{H}_6^{i,k}$ is identical $\mathcal{S}_2^{i,k+1}$ except that in Step 4b of iteration $i + k$, it really sends the permuted vector C' of column t_{i+k} (therefore reversing the actions of $\mathrm{H}_2^{i,k}$ where this column had been modified.)

The views of the adversary in games $\mathrm{H}_6^{i,k}$ and $\mathrm{H}_6^{i,k+1}$ are computationally indistinguishable via a reduction to the semantic security of El Gamal (since we are modifying the distribution of the plaintexts in the permuted column).

Game H_7^i: We conclude with a game in which simulator $\mathcal{S}_7^i$ which behaves exactly like $\mathcal{S}_6^{i,0}$, except that it computes the masks and ciphertexts in Step 4e as in the real execution. Clearly, by the

same indistinguishability argument for H_1^i, the views of the adversary in games $\mathrm{H}_6^{i,0}$ and H_7^i are computationally indistinguishable, because of the security of El Gamal .

It is also easy to see that the game H_7^i is identical to the simulated game in which the simulator enters $\tilde{t}_i = t'_1 \ldots, t'_{i-1}, t_i, \ldots, t_\ell$, i.e. H_0^{i-1}

This concludes the proof for the case when P_1 is corrupted.

Party P_2 is corrupted. This case is much simpler because P_2 does not learn anything, so that is needed is to make sure that the automaton input by P_1 remains secret, insuring P_1's privacy is preserved. Intuitively, the simulator enters an arbitrary automaton with the appropriate number of states and plays the role of the honest P_1. Moreover, it extracts the adversary's input within the zero-knowledge proof of knowledge π_{ENC}. Formally, let $\mathcal{A}$ denote an adversary controlling P_2, we construct a simulator $\mathcal{S}$ for P_1 as follows,

1. $\mathcal{S}$ is given a string $t_1, \ldots, t_\ell$ and $\mathcal{A}$'s auxiliary input and invokes $\mathcal{A}$ on these values.
2. $\mathcal{S}$ completes the key setup stage as the honest P_1 would.
3. $\mathcal{S}$ encrypts an arbitrary automaton $\Gamma' = (Q', \{0,1\}, \Delta', q_0, F')$ with $|Q'| = |Q|$ and $|F'| = |F|$ and sends its encryption.
4. In every iteration ξ, $\mathcal{S}$ fixes t_ξ by extracting it from the proof of knowledge π_{ENC} that $\mathcal{A}$ runs in Step 4b (for the first iteration $\mathcal{S}$ extracts t_1 in Step 3b).
5. $\mathcal{S}$ completes the execution as the honest P_1 would on this input.
6. $\mathcal{S}$ outputs whatever $\mathcal{A}$ does.

The only difference between this simulated execution and the hybrid execution is within the fact that the simulation runs on a different encrypted automaton. Therefore the reduction is almost immediate to the security of the El Gamal encryption scheme. More formally, recall that the parties carry out a decryption in Step 4f and Step 5d, where the later decryptions are only viewed by P_1. Therefore our goal is to prove that privacy is preserved in spite of these decryptions. We denote by the random variable $\mathrm{H}_\ell^{\mathcal{A}(z)}(\Gamma = (Q, \{0,1\}, \Delta, q_0, F), t, n)$ (for a fixed n) the view of $\mathcal{A}$ in the hybrid game H_ℓ.

Game H_0: The simulated execution.

Game H_1: In this game we define a new simulator $\mathcal{S}_1$ that is identical to simulator $\mathcal{S}$ except as follows. Recall that in Step 4f where the parties decrypt column ϵ, it includes one encryption of zero; c_{zero} and random encryptions, then $\mathcal{S}_1$ "decrypts" c_{zero} into a random value. Furthermore, it randomly chooses an encryption other than c_{zero} and "decrypts" it into zero. More formally when decrypting $c_{\text{zero}} = \langle c^1_{\text{zero}}, c^2_{\text{zero}} \rangle$, $\mathcal{S}_1$ sends a random element in $\mathbb{G}$ instead of $(c^1_{\text{zero}})^{r_1}$ where r_1 denotes its share within sk. In addition, it randomly chooses an encryption $c = \langle c_1, c_2, \rangle$ and sends $\mathcal{A}$, $c_2/(c_1)^{r_2}$ where r_2 denotes $\mathcal{A}$'s share within sk. Now, assuming that $\mathcal{A}$ replies with the respective values $(c^1_{\text{zero}})^{r_2}$ and $c_1^{r_2}$, the result of the decryption process yields a random element in $\mathbb{G}$ and g^0, respectively.

The adversary views in H_0 and H_1 are computationally indistinguishable via a reduction to the hardness of DDH. Before constructing a distinguisher, we note that if the DDH problem is hard, then it is also hard to distinguish between a mixed pair of a Diffie-Hellman tuple and a non-Diffie-Hellman tuple of the specific form $(g, h, g^{r_1}, h^{r_1}, g^{r_2}, h^{r'_2})$ and a tuple that is giving in the opposite order using a standard hybrid argument. Now, a distinguisher D_{ddh} that receives a

tuple $\langle h, h_1, h_2, h_3, h'_2, h'_3\rangle$ continues as follows. It uses h_1 as its secret key share (with h being the generator), and sets $c_{\text{zero}} = \langle h_2, h_3\rangle$ and $c = \langle h'_2, h'_3\rangle$ (note that these encryptions are the result of the masking in Step 4d that are determined by $\mathcal{S}_1$). Finally, in Step 4f it "decrypts" c_{zero} into $h_3/(h_2)^{r_2}$ and the ciphertext c into $h'_3/(h'_2)^{r_2}$.

Now, if $\langle h, h_1, h_2, h_3\rangle$ is a Diffie-Hellman tuple and $\langle h, h_1, h_2, h_3\rangle$ is a non-Diffie-Hellman tuple then $\mathcal{A}$'s view is as in the previous game, whereas in case it is not a Diffie-Hellman tuple $\mathcal{A}$'s view is as in this current game. Thus the indistinguishability between these games can be reduced to solving the DDH problem.

Game H_2: In this game there is no trusted party and no honest P_1. Instead, we define a new simulator $\mathcal{S}_2$ which is given the real input $\Gamma = (Q, \{0,1\}, \Delta, q_0, F)$ of party P_1 and encrypts in Step 2 of the protocol the transition table Δ and the set F instead of Δ' and F'.

The views of the adversary in these two games are computationally indistinguishable via a reduction to the semantic security of El Gamal . Informally, a distinguisher D_E can be constructed as follows. D_E sends its oracle two descriptions of automata Γ and Γ' and forwards $\mathcal{A}$ the oracle's response. Note that D_E is able to decrypt correctly in Step 4f as it sets the permuted vectors in Step 4c.

Note that the differences between game H_2 and the hybrid executions are in Steps 4d and 4f (as defined in game H_1). We conclude that these executions are computationally indistinguishable via a reduction to the hardness of DDH similarly to the reduction presented above. ∎

Efficiency. We present an analysis of our protocol and compare its efficiency to the generic protocols of [17, 15] for secure two-party computation in the presence of malicious adversaries. In [17], Jarecki and Shmatikov revisit the problem of constructing a protocol for securely computing any two-party Boolean circuit and present a new variant of Yao's protocol [5] on committed inputs using a public-key scheme. The protocol however 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 an external help. In [15], the scheme uses binary cut-and-choose strategy, so it requires running s copies in parallel 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.

Note first that a circuit that computes $\mathcal{F}_{\text{AUTO}}$ would require $O(\ell Q \log Q)$ gates. We note that there is a circuit of size $O(\ell Q)$ that computes a Q-state automaton over a binary input of size ℓ, but this circuit depends on the automaton, and therefore cannot be used in this context. Since we want to preserve the secrecy of the automaton, we need a circuit that takes as input *any* Q-state automaton and ℓ-bit string, and computes the automaton on the input string. This accounts for the extra $\log Q$ factor.

Thus, our protocol improves the computational complexity over the generic solution by a factor of $n \log Q$ or $\log Q$ operations compared to [15] and [17] respectively.[6] In comparison to [17, 15], we have the following:

1. *Rounds of communication:* Our protocol runs $O(\ell)$ rounds where ℓ is the length of the text. This round complexity is inherent from 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 into $O(m)$ (i.e., the length of the pattern) by breaking the text into blocks of

[6] Although we are not presenting it here, our protocol works also when the input string is over a larger alphabet. This would account for another $|\Sigma|$ factor improvement where Σ is the size of the alphabet.

size 2ℓ; see Section 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 text can be broken into smaller blocks and the output would be combined out of these results. This means that additional information about the text is released, as now it is possible to identify appearances in the text that correspond to substring of the input pattern. Finally, we note that the round complexity of [17, 15] is constant.

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 the analysis of the protocol of [15], the number of such computations depends on the number of oblivious transfer executions, which is bounded by $\max(4n, 8s)$ where n denotes the input's size of P_2, and the number of commitments $2ns(s+1)$, where s must be large enough so that $2 * 2^{\frac{-s}{17}}$ is sufficiently small. Finally, the protocol of [17] 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 $O(m \cdot \ell)$. This is probably impractical. Furthermore, the protocol of [17] also requires a security parameter, usually of size of at least 1024, since it assumes 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 bits for the key size.

3. *Symmetric operations:* We also consider the approximate number of symmetric computations that are required for the computation of the protocol of [15] (these computations are required due to the usage of a symmetric encryption scheme). Then their protocol requires $O(s|C| + s^2 \cdot n)$ such computations.

4. *Bandwidth:* Finally, we consider the communication complexity. In our protocol, the parties send each other $O(m \cdot \ell)$ encryptions. The bandwidth of the protocol in [17] is similar (again, with relatively high constants), whereas in [15] the bandwidth is $O(s|C| + s^2 n)$ symmetric-key encryptions. A recent paper by Pinkas et al. [16] showed that communication is the major bottleneck when implementing the malicious protocol of [15].

Finally, we note that implementing a circuit that solves the basic pattern matching problem may be significantly harder than implementing our protocol. We conclude the comparison by saying that based on the above our protocol offers an alternative efficient solution for computing the automata evaluation functionality.

Dealing with arbitrary size alphabet. Protocol π_{AUTO} can be naturally extended for the case arbitrary size alphabet by simply have P_1 send a larger table with a row for each letter. The rest of the protocol is naturally modified. Note that this will introduce a $|\Sigma|$ overhead to the communication and computation costs, where Σ is the alphabet.

3.2.2 A Zero-Knowledge Proof of Knowledge for $\mathcal{R}_{\text{VALIDAUTO}}$

In this section we present a zero-knowledge proof of knowledge for the relation $\mathcal{R}_{\text{VALIDAUTO}}$ defined by:

$$\left(\left(\{Q_{i,j}, r_{i,j}\}_{i,j}\right), \left(\{c_{i,j}\}_{i,j}, pk\right)\right) \mapsto \begin{cases} (\lambda, 1) & \forall i,j\ c_{i,j} = E_{pk}(Q_{i,j}; r_{i,j}) \text{ and} \\ & \{Q_{i,j}\}_{i,j} \text{ is a valid KMP automaton} \\ (\lambda, 0) & \text{otherwise} \end{cases}$$

where $i \in \{0,1\}, j \in \{1, \ldots, |Q|\}$. This proof is needed in Protocol π_{PM} to ensure the validity of the encrypted automaton that P_1 sends. We remark that it is unnecessary for this proof to be a proof of knowledge, as the knowledge extraction of the automaton can be performed within protocol π_{AUTO}; formally described in Section 4. Nevertheless, for the sake of modularity we consider this property here as well. Our proof is modular with the zero-knowledge proof for the following language,

$$\text{L}_{\text{NZ}} = \{(\mathbb{G}, g, q, h, h_1, h_2) \mid \exists\ (m \neq 0, r) \text{ s.t. } \alpha = g^r,\ \beta = h^r g^m\}$$

An efficient proof π_{NZ} can be found in [21].

Essentially, this proof is focused on showing the automaton corresponds to a valid string $p = p_1, \ldots, p_{|Q|-1}$ and is computed correctly according to table Υ; formally defined in Section 3.2. That is, for every prefix $p_1, \ldots, p_j$ of p the prover proves that there exists an index $i \in \{0,1\}$ in which the encryption $c_{i,j}$ corresponds to the table value $\Upsilon(j+1)$. Recalling that the proof must not leak any information about p, these checks must be performed obliviously of the prefix. We therefore conduct a brute force search on the matched prefix of every suffix in which ultimately, the verifier accepts only if the conditions for $\mathcal{R}_{\text{VALIDAUTO}}$ are met. For simplicity, our proof is not optimized; see below for further discussion. We now continue with the formal description of our proof $\pi_{\text{VALIDAUTO}}$ and its proof of security,

Protocol 2 (zero-knowledge proof of knowledge $\pi_{\text{VALIDAUTO}}$ for $\mathcal{R}_{\text{VALIDAUTO}}$):

- **Joint statement:** *A public-key pk and a collection $\{c_{i,j}\}_{i,j}$ of $|Q|$ sets, each set is of size 2 which corresponds to a row in the transition matrix Δ.*
- **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 $\mathbb{G}$.*
- **Convention:** *Both parties check every received ciphertext for validity (i.e, that it is in $\mathbb{G}$), and abort if an invalid ciphertext is received. Unless written differently, $i \in \{0,1\}$ and $j \in \{1, \ldots, |Q|\}$.*
- **Notation:** *We use non-standard notation and write $E_{pk}(m)$ (instead of $E_{pk}(g^m)$) for the encryption of g^m. Using this notation, the El Gamal encryption scheme is* additively homomorphic. *We note, however, that using our notation the result of decrypting $E_{pk}(m)$ is g^m, i.e., $D_{sk}(E_{pk}(m)) = g^m$.*
- **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.*[7]

(c) *P proves the correctness of* $c_{j,1-b}$ *in two steps: It first proves that it corresponds to a valid prefix of* $p_{\langle j-1\rangle}$*, then it proves the maximality of this prefix (recall that* $p_{\langle r\rangle}$ *denotes the* r*th 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* $\alpha' < \alpha''$*. (This set of encryptions is computed once).*

ii. **Proving the existence of a prefix that matches a suffix of** $p_{\langle j-1\rangle}$**:** *For all* $1 \leq k \leq j-1$ *the parties compute an encryption* $v'_k = (v_{j-k,j-1}/v_{1,k}) \cdot (c_{j,1-b}/g^k)$*.*
P then proves that there exists k *for which* v'_k *is a Diffie-Hellman tuple.*[8] *(That is, P proves that there exists an encryption that corresponds to a zero encryption. The parties set* $v'_0 = c_{j,1-b}$ *when there is no matching prefix for any suffix of* $p_{\langle j-1\rangle}$ *which means* $c_{j,1-b}$ *denotes the encryption of the initial state* q_0*.)*

iii. **Proving that** $p_{\langle D_{sk}(c_{j,b})\rangle}$ **corresponds to the longest suffix of** $p_{\langle j-1\rangle}$**:**
Next P proves that there does not exist an index $D_{sk}(c_{j,1-b}) < \gamma \leq j-1$ *in which* $v_{j-\gamma,j-1}/v_{1,\gamma} = 0$ *yet* $c_{j,1-b}/g^\gamma \neq 0$*, as this would imply that there exists a larger string* $p_{\langle\gamma\rangle}$ *that matches a suffix of* $p_{\langle j-1\rangle}$ *yet,* $D_{sk}(c_{j,1-b}) \neq \gamma$*.*

– *Therefore, for every* $1 \leq k \leq j-2$ *and* $2 \leq k' \leq j-1$ *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*.*

Theorem 9 *Assume that* π_{DL}*,* π_{PERM}*,* π_{DDH} *and* π_{NZ} *are as described above and that* (G, E, D) *is the semantically secure El Gamal encryption scheme. Then* $\pi_{\text{VALIDAUTO}}$ *is a computational zero-knowledge proof of knowledge for* $\mathcal{R}_{\text{VALIDAUTO}}$ *with perfect completeness.*

Efficiency. Note first that the round complexity of $\pi_{\text{VALIDAUTO}}$ is constant, as the zero-knowledge proofs can be implemented in constant rounds and run in parallel. As for the number of asymmetric computations, we note that an optimized construction achieves computation cost of $O(m^2)$ operations. Informally, this is due to the fact that there are m^2 distinct encryptions in the set $\{v_{\alpha',\alpha''}\}_{\alpha',\alpha''}$. Furthermore, the multiplications computed in Step 2(c)iii can be computed "on the fly" when the above set is being computed. Thus, we conclude that the overall number of exponentiations is $O(m^2)$.

Proof: We first show perfect completeness. This is derived from the fact that we conduct a brute force search for the matched prefix of every suffix.

Zero knowledge Let V^* be an arbitrary probabilistic polynomial-time strategy for V. Then a simulator $\mathcal{S}_{\text{VALIDAUTO}}$ for this proof can be constructed using the simulators $\mathcal{S}_{\text{DL}}$, $\mathcal{S}_{\text{PERM}}$, $\mathcal{S}_{\text{DDH}}$ and $\mathcal{S}_{\text{NZ}}$ from the corresponding proofs of π_{DL}, π_{PERM}, π_{DDH} and π_{NZ}. That is, $\mathcal{S}_{\text{VALIDAUTO}}$ invokes V^* and plays the role of the honest prover, except that in every zero-knowledge invocation it invokes

[7] We assume that the parties agree on an encryption of $j+1$, including its randomness.

[8] This proof is a simple extension of the standard proof for $\mathcal{R}_{\text{DDH}}$ using a general technique. In particular, the prover separates the challenge it is given by the verifier c into two values; c_1 and c_2 such that $c = c_1 \oplus c_2$. Assume w.l.o.g. that it does not have a witness for the first statement, then it always chooses c_1 in which it knows how to complete the proof (similarly to what the simulator for π_{DDH} does), and uses its witness for the other statement to complete the second proof on a giving challenge c_2. Note that the verifier cannot distinguish whether the prover knows the first or the second witness. See [22] for more details.

the appropriate simulator. The executions are computationally indistinguishable via standard reductions to the security of the zero-knowledge proofs.

Knowledge extraction It remains to show the existence of a knowledge extractor K. Let $P^*_{x,\zeta,\rho}$ be an arbitrary prover machine where $x = (\{c_{i,j}\}_{i,j}, pk)$, ζ is an auxiliary input and ρ is P^*'s random tape. Basically, the extractor K extracts $\mathcal{P}^*$'s input from the zero-knowledge proof π_{DL} at the beginning of the protocol. In particular for all i, j, P^* proves the knowledge of the randomness $r_{i,j}$ used for the computation of the ciphertext $c_{i,j}$. This, in turn, enables K to recover the plaintext $Q_{i,j}$ as well. It then continues playing the role of the honest verifier and aborts the execution if the honest verifier does. The fact that we perform a brute force search, combined with the fact that the randomness $\{u_\alpha\}_\alpha$ incorporated by the verifier preclude the event in which equality does not hold yet the sum of the encryptions amount to the same value. ■

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: (i) the parties first engage in an execution of $\pi_{\text{VALIDAUTO}}$ for which P_1 proves that it indeed sent a valid KMP automaton (ii) 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.

Details follow,

Protocol π_{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 $\Gamma = (Q, \Sigma, \Delta, q_0, F)$ according to the KMP specifications based on its input p and sends P_2 encryptions of the transition matrix Δ and the accepting states, denoted by E_Δ and E_F, respectively (recall that by our conventions $q_0 = 0$, $\Sigma = \{0, 1\}$, $Q = [0, \dots, m]$, and $F = \{q_m\}$).
  2. The parties engage in an execution of the zero-knowledge proof $\pi_{\text{VALIDAUTO}}$ for which P_1 proves that Γ was constructed correctly. That is, P_1 proves that the set E_Δ corresponds to a valid KMP automaton for a well defined input string 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.[9] For every $1 \leq i \leq \ell/m$, let $\{\mathsf{output}_{\mathsf{j}}^{\mathsf{i}}\}_{j=1}^{m+1}$ denotes the set of P_1's outputs from the ith execution. Then P_1 returns $\{j \mid \mathsf{output}_{\mathsf{j}}^{\mathsf{i}} = \text{``}\mathsf{accept}''\}$.

Theorem 10 *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 $\pi_{\text{VALIDAUTO}}$, we refer the reader to the detailed analysis presented in Section 3.2.1. The overall costs are amount to $O(m \cdot \ell + m^2) = O(m \cdot \ell)$ since in most cases $m << \ell$.

References

[1] Ran Canetti. Security and composition of multi-party cryptographic protocols. *Journal of Cryptology*, 13:2000, 1998.

[2] Shafi Goldwasser and Leonid A. Levin. Fair computation of general functions in presence of immoral majority. In *CRYPTO 90': Proceedings of the 10th Annual International Cryptology Conference on Advances in Cryptology*, pages 77–93, London, UK, 1990. Springer-Verlag.

[3] Donald Beaver. Foundations of secure interactive computing. In *CRYPTO 91': Proceedings of the 11th Annual International Cryptology Conference on Advances in Cryptology*, pages 377–391, London, UK, 1991. Springer-Verlag.

[4] Silvio Micali and Phillip Rogaway. Secure computation (abstract). In *CRYPTO 91': Proceedings of the 11th Annual International Cryptology Conference on Advances in Cryptology*, pages 392–404, 1991. This is preliminary version of unpublished 1992 manuscript.

[5] Andrew Chi-Chih Yao. How to generate and exchange secrets. In *SFCS 86': Proceedings of the 27th Annual Symposium on Foundations of Computer Science*, pages 162–167, Washington, DC, USA, 1986. IEEE Computer Society.

[6] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *STOC 87': Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 218–229, New York, NY, USA, 1987. ACM.

[7] Oded Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications.* Cambridge University Press, New York, NY, USA, 2004.

[8] Donald E. Knuth, James H. Morris Jr., and Vaughan R. Pratt. Fast pattern matching in strings. *SIAM J. Comput.*, 6(2):323–350, 1977.

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

[9] Yuval Ishai and Anat Paskin. Evaluating branching programs on encrypted data. In *TCC 07': Forth Theory of Cryptography Conference*, volume 4392 of *Lecture Notes in Computer Science*, pages 575–594. Springer-Verlag, 2007.

[10] Carmit Hazay and Yehuda Lindell. Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries. In Ran Canetti, editor, *TCC 08': Fifth Theory of Cryptography Conference*, volume 4948 of *Lecture Notes in Computer Science*, pages 155–175. Springer-Verlag, 2008.

[11] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, 1976.

[12] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *CRYPTO 85': Proceedings of the 11th Annual International Cryptology Conference on Advances in Cryptology*, pages 10–18, New York, NY, USA, 1985. Springer-Verlag.

[13] Stanislaw Jarecki and Liu Xiaomin. Efficient oblivious pseudorandom function with applications to adaptive ot and secure computation of set intersection. In *TCC 09': Sixth Theory of Cryptography Conference*, volume 5444 of *Lecture Notes in Computer Science*. Springer-Verlag, 2009.

[14] Juan Ramón Troncoso-Pastoriza, Stefan Katzenbeisser, and Mehmet Celik. Privacy preserving error resilient dna searching through oblivious automata. In *CCS 07': Proceedings of the 14th ACM conference on Computer and communications security*, pages 519–528, New York, NY, USA, 2007. ACM.

[15] Yehuda Lindell and Benny Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In *EUROCRYPT 07': Proceedings of the 26th annual international conference on Advances in Cryptology*, pages 52–78, Berlin, Heidelberg, 2007. Springer-Verlag.

[16] Schneider Thomas Smart Nigel P. Pinkas, Benny and Stephen C. Williams. Secure two-party computation is practical. In *ASIACRYPT 09'*, pages 250–267, Tokyo, Japan, 2009. Springer-Verlag.

[17] Stanislaw Jarecki and Vitaly Shmatikov. Efficient two-party secure computation on committed inputs. In *EUROCRYPT 07': Proceedings of the 26th annual international conference on Advances in Cryptology*, pages 97–114, Berlin, Heidelberg, 2007. Springer-Verlag.

[18] Richard Cleve. Limits on the security of coin flips when half the processors are faulty. In *STOC '86: Proceedings of the eighteenth annual ACM symposium on Theory of computing*, pages 364–369, New York, NY, USA, 1986. ACM.

[19] Claus P. Schnorr. Efficient identification and signatures for smart cards. In *CRYPTO 89': Proceedings on Advances in cryptology*, pages 239–252, New York, NY, USA, 1989. Springer-Verlag New York, Inc.

[20] David Chaum and Torben P. Pedersen. Wallet databases with observers. In *CRYPTO 92': Proceedings of the 12th Annual International Cryptology Conference on Advances in Cryptology*, pages 89–105, London, UK, 1992. Springer-Verlag.

[21] Carmit Hazay and Kobbi Nissim. Efficient set operations in the presence of malicious adversaries, 2010.

[22] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *CRYPTO '94: Proceedings of the 14th Annual International Cryptology Conference on Advances in Cryptology*, pages 174–187, London, UK, 1994. Springer-Verlag.

[23] Jens Groth and Yuval Ishai. Sub-linear zero-knowledge argument for correctness of a shuffle. In Nigel P. Smart, editor, *EUROCRYPT 08': Proceedings of the 27th annual international conference on Advances in Cryptology*, volume 4965 of *Lecture Notes in Computer Science*, pages 379–396. Springer, 2008.

[24] Jens Groth and Steve Lu. Verifiable shuffle of large size ciphertexts. In Tatsuaki Okamoto and Xiaoyun Wang, editors, *PKC 07': 10th International Conference on Practice and Theory in Public-Key Cryptography*, volume 4450 of *Lecture Notes in Computer Science*, pages 377–392. Springer, 2007.

[25] Robert S. Boyer and J. Strother Moore. A fast string searching algorithm. *Commun. ACM*, 20(10):762–772, 1977.

[26] Cyril Allauzen, Maxime Crochemore, and Mathieu Raffinot. Factor oracle: A new structure for pattern matching. In *SOFSEM '99: Proceedings of the 26th Conference on Current Trends in Theory and Practice of Informatics on Theory and Practice of Informatics*, pages 295–310, London, UK, 1999. Springer-Verlag.

[27] Gonzalo Navarro and Veli Mäkinen. Compressed full-text indexes. *ACM Comput. Surv.*, 39(1):2, 2007.

[28] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, 1970.

[29] Eu-Jin Goh. Secure indexes. Cryptology ePrint Archive, Report 2003/216, 2003. http://eprint.iacr.org/2003/216/.

[30] Amihood Amir, Moshe Lewenstein, and Ely Porat. Faster algorithms for string matching with mismatches. In *SODA 00'*, pages 794–803, San Francisco, California, USA, 2000.

[31] M. Sohel Rahman and Costas S. Iliopoulos. Pattern matching algorithms with don't cares. In *SOFSEM 07'*, pages 116–126, 2007.