Secure Text Processing with Applications to Private DNA Matching[*]

Jonathan Katz
University of Maryland
College Park, MD
jkatz@cs.umd.edu

Lior Malka
University of Maryland
College Park, MD
lior@cs.umd.edu

ABSTRACT

Motivated by the problem of private DNA matching, we consider the design of efficient protocols for *secure text processing*. Here, informally, a party P_1 holds a text T and a party P_2 holds a pattern p and some additional information y, and P_2 wants to learn $\{f(T, j, y)\}$ for all locations j where p is found as a substring in T. (In particular, this generalizes the basic *pattern matching* problem.) We aim for protocols with full security against a malicious P_2 that also preserve privacy against a malicious P_1 (i.e., *one-sided security*). We show how to modify Yao's garbled circuit approach to obtain a protocol where the size of the garbled circuit is linear in the number of occurrences of p in T (rather than linear in $|T|$). Along the way we show a new keyword search protocol that may be of independent interest.

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: General—*security and protection*

General Terms

Security, Theory

1. INTRODUCTION

Text-processing algorithms are fundamental to computer science. They are used on the Internet to classify web pages, monitor traffic, and support search engines. In the field of bio-informatics they play a critical role in DNA matching and analysis. This paper studies text processing in the setting of secure two-party computation. We consider the scenario where a party P_1 holds a text T and a party P_2 holds a pattern p and possibly some additional information y. The goal is to process T based on p and y, with P_2 learning nothing more than the agreed-upon result (and P_1 learning nothing about p or y).

In recent years, several efficient two-party protocols for various tasks related to text-processing have been given; we provide a more complete discussion further below. None of the existing solutions, however, appears to be flexible or general enough to encompass natural problems that arise, e.g., in private DNA matching (which was the motivation for this work). By way of illustration, let $\Sigma = \{\texttt{A},\texttt{C},\texttt{T},\texttt{G}\}$ and $T, p \in \Sigma^*$ (where T represents a DNA sequence and p represents a nucleotide pattern). Let $\ell_{\max}(T, p) \geq 0$ denote the largest integer ℓ' for which $p^{\ell'}$ appears as a substring in T. For integers ϵ, ℓ, define

$$M(T, p, \epsilon, \ell) = \begin{cases} 1 & |\ell_{\max}(T,p) - \ell| \leq \epsilon \\ 0 & \text{otherwise} \end{cases} . \quad (1)$$

Say P_1 holds T, while P_2 holds a pattern p and $y = \langle \epsilon, \ell \rangle$ as additional information; P_2 wants to compute $M(T, p, \epsilon, \ell)$. This sort of computation is exactly what is used in the Combined DNA Index System[1] (CODIS), run by the FBI for DNA identity testing. Existing protocols for secure text processing do not appear to readily yield solutions for this sort of problem.

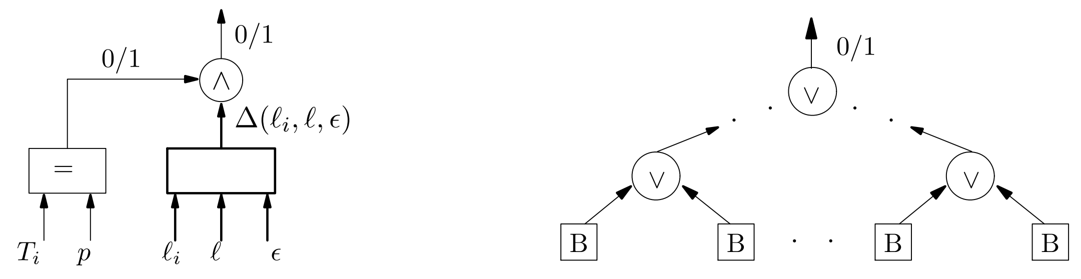
Faced with any problem in secure two-party computation, the first thing to check is whether a *generic* approach based on Yao's garbled circuit [21] might be applicable. Indeed, in recent years it has been demonstrated [16, 15, 19, 18] that secure two-party protocols based on Yao's garbled circuit approach may be practical when the function being computed can be expressed as a small boolean circuit. Unfortunately, that is *not* the case here, at least for the applications we are interested in where $|T|$ can be large.

To see why, consider one natural circuit for computing M. Let $m = |p|$ denote the length of the pattern, and assume m is known. Fixing T, let T_i denote the m-character substring of T beginning at position i. For $i \in \{1, \ldots, |T| - m + 1\}$, define $\ell_i \geq 1$ to be the largest integer such that $T_i^{\ell_i}$ occurs as a substring of T beginning at position i. (Observe that P_1 can pre-compute ℓ_i for all i without any interaction with P_2.) Define next the basic circuit B_i that takes inputs from both parties and returns 1 iff both $|\ell_i - \ell| \leq \epsilon$ and $T_i = p$ (cf. Figure 1(a)). Finally, M can be expressed as the OR of the outputs of all the B_i (cf. Figure 1(b)).

The conclusion of the exercise is this: even if the circuit B_i for computing the "basic" functionality is small, *the circuit for computing the "actual" functionality* (M in this case) *is*

[*]Research supported by NSF CAREER award #0447075 and DARPA. The contents of this paper do not necessarily reflect the position or the policy of the US Government, and no official endorsement should be inferred.

[1]`http://www.fbi.gov/hq/lab/codis`.



(a) The basic circuit B (b) The combined circuit C

Figure 1: A circuit computing M (cf. Equation (1)).

large, and in particular has size $O(|T|-m+1)$ (since it is the OR of $|T|-m+1$ smaller circuits). In applications where $|T|$ is large, then, applying generic approaches will lead to inefficient protocols.

Let $n = |T|$ be the length of T. Our main observation is that in many cases there is no pattern that appears in the text T more than some bounded number of times, and this bound may be much smaller than n. This gives hope that instead of duplicating the basic circuit $O(n)$ times, we can use only as many copies as necessary. The question then is how to achieve this without having P_1 learn something about the pattern p held by P_2. We explain next how this is possible.

Assume now, for simplicity, that each m-character pattern appears at most once in T. Our idea is to de-couple the "pattern matching" portion of the computation (e.g., the search for a location i such that $T_i = p$) from the subsequent computation of the "basic" circuit. In a bit more detail (though still somewhat informally), P_1 prepares a garbled version of the basic circuit and sends it to P_2. Recall, this basic circuit takes inputs p, ℓ, ϵ from P_2, and inputs T_i, ℓ_i from P_1. Note that we only need to evaluate this circuit for the position i (if one exists) for which $T_i = p$. To exploit this, we have P_1 prepare appropriate input-wire labels corresponding to every possible match; i.e., it prepares input-wire labels for each pair (T_i, ℓ_i), for $i = 1$ to $n - m + 1$. The parties then use a variant of keyword search that allows P_2 to learn the appropriate input-wire labels only for i such that $T_i = p$. (If no such i exists, then P_2 learns this fact and simply outputs 0.) The parties also use oblivious transfer, as usual, to enable P_2 to learn the input-wire labels corresponding to its own inputs p, ℓ, ϵ. The net result is that P_2 is able to evaluate the garbled circuit for the "right" set of inputs and thereby obtain the desired result of the computation. Note that, besides the keyword search sub-routine, the rest of the protocol has complexity proportional the size $|B|$ of the basic circuit, rather than proportional to $n \cdot |B|$) as in the naive approach. (In general the rest of the protocol would have complexity proportional to $r_{\max} \cdot |B|$, where $r_{\max} \leq n$ is an upper bound on the number of repetitions of any pattern, as opposed to $n \cdot |B|$.)

1.1 Organization of the Paper

In Section 2 we introduce some functionalities we will rely on, and briefly review the (standard) definition of one-sided security. We describe protocols for some basic text-processing tasks in Section 3. In that section, which serves as a warm-up to our main results, we first (re-)consider the task of keyword search [4] and show an immediate application to the problem of *pattern matching*. Our resulting pattern-matching protocol improves in some respects on the protocol of Hazay and Lindell [9]; we defer further discussion to that section. We also observe that our approach extends to yield protocols for a wide class of text-processing problems; specifically, it enables computation of any functionality where a party P_1 holds a text T and a party P_2 party has a pattern p, and P_2 should learn $\{f(T, i) \mid T_i = p\}$ for some arbitrary function f. We give a few applications where the desired functionality can be written in this way.

Our main result is described in Section 4. There, we give a protocol that can securely compute any functionality of the form described with complexity (excepting the keyword-search sub-routine) linear in an upper bound on the number of occurrences of p in T (rather than linear in $|T|$). We then discuss how to apply this protocol to text-processing problems such as approximate tandem repeats (motivated by DNA matching, as described previously).

1.2 Related Work

Broadly speaking, there are two approaches to constructing protocols for secure computation. *Generic* protocols can be used to evaluate arbitrary functions, given a description of the function as a circuit; *special-purpose* protocols are tailored to specific functions of interest. Yao's "garbled circuit" approach [21] (extended in [8] to handle malicious adversaries) gives a generic protocol for secure two-party computation. In recent years several implementations and improvements of Yao's garbled circuit protocol have been shown [16, 10, 14, 15, 19, 18]. Regardless of any improvements, however, a fundamental limitation of this approach is that the garbled circuit has size linear in the size of the circuit being computed.

More efficient, special-purpose protocols have been developed for several functionalities of interest. Several efficient protocols for keyword search are known [4, 12, 9, 3]; there also exist efficient protocols for pattern matching [20, 9, 6] but, as discussed previously, these protocols do not seem to extend to more complex functionalities such as the ones we consider here. While several researchers have also investigated specific problems related to DNA matching [2, 20, 11, 5], none of these works seem to apply to our specific problem. Finally, we also mention recent work on oblivious evaluation of finite automata [20, 5, 6]. Applying such protocols directly to our setting seems to yield less efficient protocols. Moreover, our approach allows for the computation of functions that cannot be computed by finite automata.

2. PRELIMINARIES

Throughout, we use k to denote the security parameter. We rely on secure protocols for several functionalities that we briefly describe here. In all cases, we use these functionalities in a black-box manner, and so can use any of the known protocols for securely computing these functionalities. For concreteness, we note that efficient constructions of all the necessary protocols (with varying security guarantees) exist based on the decisional Diffie-Hellman assumption (e.g., [1, 17, 4, 13]).

Oblivious transfer (OT). In a 1-out-of-2 (string) OT protocol, one party holds two equal-length strings m_0, m_1 and the second party holds a bit b; the second party learns m_b and the first party learns nothing. For one of our protocols we use a parallel variant of the OT functionality, where one party holds k pairs of strings $((m_0^1, m_1^1), \ldots, (m_0^k, m_1^k))$ and the second party holds a k-bit string y; the second party should learn $\{m_{y_i}^i\}_{i=1}^k$.

Oblivious pseudorandom function evaluation (OPRF). Let F be some fixed pseudorandom function. In an OPRF protocol one party holds a key s and the second party holds an input x; the second party learns $F_s(x)$ and the first party learns nothing. We denote this functionality by $\mathcal{F}_{\text{OPRF}}$.

2.1 Garbled Circuits

Yao's garbled circuit methodology [21] provides a key building block for secure two-party computation. We do not provide the details, but instead describe Yao's approach in an abstract manner that suffices for our purposes. Let C be a circuit, and assume for concreteness here that C takes a k-bit input from each party and provides a single-bit output to the second party. Yao gives algorithms $(\mathsf{Garble}, \mathsf{Eval})$ such that:

- $\mathsf{Garble}(1^k, C)$ outputs a *garbled circuit* gC and two sets of *input-wire labels* $\vec{X} = (X^{1,0}, X^{1,1}, \ldots, X^{k,0}, X^{k,1})$ and $\vec{Y} = (Y^{1,0}, Y^{1,1}, \ldots, Y^{k,0}, Y^{k,1})$. The value $X^{i,0}$ (resp., $X^{i,1}$) is the "0-label" (resp., "1-label") for the ith input wire of one of the parties. ($Y^{i,0}, Y^{i,1}$ analogously serve as the input-wire labels for the second party.)
- $\mathsf{Eval}(\mathsf{gC}, X^1, \ldots, X^k, Y^1, \ldots, Y^k)$ outputs a bit.

By way of notation, for $\vec{X}$ as above and an input $x \in \{0,1\}^k$ we let $\vec{X}(x) \overset{\text{def}}{=} (X^{1,x_1}, \ldots, X^{k,x_k})$ (and similarly for $\vec{Y}(y)$). The *correctness* guarantee is that for any C and any inputs x, y, if $\mathsf{Garble}(1^k, C)$ outputs $(\mathsf{gC}, \vec{X}, \vec{Y})$ then

$$\mathsf{Eval}(\mathsf{gC}, \vec{X}(x), \vec{Y}(y)) = C(x, y).$$

In typical usage, one party (holding input x) runs $\mathsf{Garble}(1^k, C)$ to obtain $(\mathsf{gC}, \vec{X}, \vec{Y})$ and sends gC and $\vec{X}(x)$ to the second party. In addition, the second party (holding input y) obtains $\vec{Y}(y)$ using k invocations of oblivious transfer. The second party can then compute $C(x, y)$ using Eval as indicated above.

The *security* guarantee (informally stated) is that there exists an efficient simulator $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$ such that

- $\mathcal{S}_1(1^k, C)$ outputs input-wire labels $X^1, \ldots, X^k, Y^1, \ldots, Y^k$ and state s.
- $\mathcal{S}_2(y, C(x, y), s)$ outputs gC.
- For any C, x, y, the following distributions are computationally indistingishable:

$$\left\{ \begin{array}{c} X^1, \ldots, X^k, Y^1, \ldots, Y^k, s \leftarrow \mathcal{S}_1(1^k, C); \\ \mathsf{gC} \leftarrow \mathcal{S}_2(y, C(x,y), s) \end{array} : (\mathsf{gC}, X^1, \ldots, X^k, Y^1, \ldots, Y^k) \right\}$$

and

$$\left\{ \mathsf{gC}, \vec{X}, \vec{Y} \leftarrow \mathsf{Garble}(1^k, C) : (\mathsf{gC}, \vec{X}(x), \vec{Y}(y)) \right\}.$$

(The above is slightly stronger than what is usually required, but is satisfied by the standard construction.) Since the computation of $\mathcal{S}_1, \mathcal{S}_2$ depends only on y and $C(x, y)$, the above ensures that the second party learns nothing beyond what is implied by its own input and output. For technical reasons, we also extend the definition of $\mathcal{S}_2$ so that $\mathcal{S}_2(y, \perp, s)$ outputs gC such that, for any x, y, the resulting $(\mathsf{gC}, Y^1, \ldots, Y^k)$ is computationally indistinguishable from $(\mathsf{gC}, \vec{Y}(y))$ (where $(\mathsf{gC}, \vec{X}, \vec{Y}) \leftarrow \mathsf{Garble}(1^k, C)$). This implies that if the first party *doesn't* send the input-wire labels $\vec{X}(x)$, then the second party does not even learn the output $C(x, y)$.

2.2 Secure Computation

We use standard notions of secure computation, and refer the reader elsewhere (e.g., [7]) for a detailed discussion. Briefly, security is defined by requiring indistinguishability between a real execution of the protocol and an ideal world in which there is a trusted party who evaluates the function in question on behalf of the parties. More formally, fix a protocol π that computes a functionality $\mathcal{F}$. Because it holds throughout this paper, assume $\mathcal{F}$ provides output to P_2 only. (Note, however, that $\mathcal{F}$ may be randomized.) Consider first an execution in the real world. Here, party P_1 holds an input x while party P_2 holds an input y, and one of these parties is corrupted by an adversary $\mathcal{A}$ that has auxiliary input z. The honest party runs protocol π as instructed (using some fixed value for the security parameter k), responding to (arbitrary) messages sent by $\mathcal{A}$ on behalf of the other party. (We stress that $\mathcal{A}$ is *malicious* and can arbitrarily deviate from the protocol specification.) Let $\text{VIEW}_\pi^{\mathcal{A}}(x, y, k)$ denote the view of $\mathcal{A}$ throughout this interaction, and let $\text{OUT}_\pi(x, y, k)$ denote the output of the honest party (which is empty if P_1 is honest). Set

$$\text{REAL}_{\pi, \mathcal{A}(z)}(x, y, k) \overset{\text{def}}{=} \left(\text{VIEW}_\pi^{\mathcal{A}}(x, y, k),\ \text{OUT}_\pi(x, y, k) \right).$$

An ideal execution of the computation of $\mathcal{F}$ proceeds as follows. Once again, party P_1 holds an input x while party P_2 holds an input y, and one of these parties is corrupted by an adversary $\mathcal{A}$ that has auxiliary input z. The honest party sends its input to the trusted party, while the corrupted party can send anything. Let x', y' be the values received by the trusted party; it then computes $\mathcal{F}(x', y')$ and gives the result to P_2. (There are no issues of fairness here since P_1 receives no output.) The honest party outputs whatever it was sent by the trusted party, and $\mathcal{A}$ outputs an arbitrary function of its view. Let $\text{OUT}_{\mathcal{F}}^{\mathcal{A}}(x, y, k)$ (resp., $\text{OUT}_{\mathcal{F}}(x, y, k)$) denote the output of $\mathcal{A}$ (resp., the honest party) following an execution in the ideal model as described above. Set

$$\text{IDEAL}_{\mathcal{F}, \mathcal{A}(z)}(x, y, k) \overset{\text{def}}{=} \left(\text{OUT}_{\mathcal{F}}^{\mathcal{A}}(x, y, k),\ \text{OUT}_{\mathcal{F}}(x, y, k) \right).$$

Protocol π_{search}

Input to P_1: A database $\{(p_i, x_i)\}_{i=1}^n$.

Input to P_2: A pattern p.

Output: P_2 learns $\{x_i \mid p_i = p\}$, the database size n, and the payload length $|x_i|$.

1. Party P_1 chooses a key $s \leftarrow \{0,1\}^k$, and then for each $i \in \{1, \ldots, n\}$ computes $\alpha_i := F_s(p_i)$ and $c_i \leftarrow \mathsf{Enc}_{\alpha_i}(x_i)$.
2. P_1 and P_2 execute an OPRF protocol, where the input of P_1 is s and the input of P_2 is p. Party P_2 obtains $\alpha = F_s(p)$.
3. P_1 sends the $\{c_i\}$, in random permuted order, to P_2.
4. P_2 outputs $\{\mathsf{Dec}_\alpha(c_i) \mid \mathsf{Dec}_\alpha(c_i) \neq \perp\}$.

Figure 2: A protocol for keyword search.

The strongest notion of security — *full security* — requires that for every polynomial-time adversary $\mathcal{A}$ in the real world, there exists a corresponding polynomial-time adversary $\mathcal{S}$ in the ideal world such that $\text{REAL}_{\pi,\mathcal{A}}(x, y, k)$ and $\text{IDEAL}_{\mathcal{F},\mathcal{S}}(x, y, k)$ are computationally indistinguishable. Here, we achieve a slightly weaker definition known as *one-sided security*. Specifically, we achieve full security *when P_2 is corrupted.* When P_1 is corrupted, though, we only achieve *privacy* for the honest P_2; namely, a malicious P_1 learns nothing about the input of P_2. If P_1 is corrupted, however, there are no guarantees that the *output* of P_2 is correct.

Definition 2.1. *Let π be a two-party protocol computing a functionality $\mathcal{F}$ where only P_2 receives output. We say π is* one-sided secure *if:*

1. *For every non-uniform polynomial-time adversary $\mathcal{A}$ corrupting P_2 in the real world, there is a corresponding non-uniform polynomial-time adversary $\mathcal{S}$ corrupting P_2 in the ideal world such that*

$$\left\{\text{REAL}_{\pi,\mathcal{A}(z)}(x, y, k)\right\} \overset{c}{\equiv} \left\{\text{IDEAL}_{\mathcal{F},\mathcal{S}(z)}(x, y, k)\right\}.$$

2. *For every non-uniform polynomial-time adversary $\mathcal{A}$ corrupting P_1 in the real world, there is a corresponding non-uniform polynomial-time adversary $\mathcal{S}$ corrupting P_1 in the ideal world such that*

$$\left\{\text{VIEW}_{\pi,\mathcal{A}(z)}(x, y, k)\right\} \overset{c}{\equiv} \left\{\text{VIEW}_{\mathcal{F},\mathcal{S}(z)}(x, y, k)\right\}.$$

3. A KEYWORD-SEARCH PROTOCOL

In this section we present a protocol for *keyword search*, and then show an application to *pattern matching* (both of these are defined more formally below). Our constructions yield some advantages relative to prior protocols for these tasks [4, 9] that will be discussed at the end of this section.

3.1 Keyword Search

In this setting party P_1 has as input a database $\{(p_i, x_i)\}$ of tuples, where we refer to each p_i as a *keyword* and each x_i as the associated *payload.* (We assume the number of tuples is known, and all payloads have the same [known] length.) Party P_2 has as input a keyword p, and should learn $\{x_i \mid p_i = p\}$. We denote this functionality by $\mathcal{F}_{\text{ks}}$, and stress that we allow $p_i = p_j$ for $i \neq j$ (something that is disallowed[2] in prior work).

The basic idea of our protocol is similar to that of previous work, but differs in the details. P_1 begins by choosing a random key s for a pseudorandom function F. Then for each tuple (p_i, x_i) party P_1 computes $\alpha_i := F_s(p_i)$ and $c_i \leftarrow \mathsf{Enc}_{\alpha_i}(x_i)$ (where Enc represents a symmetric-key encryption scheme whose properties will be discussed below), and sends all the $\{c_i\}$ (in random permuted order) to P_2. Next, P_1 and P_2 run an OPRF protocol that enables P_2 to learn $\alpha := F_s(p)$. Finally, P_2 attempts decryption of each of the received c_i using α, and outputs the resulting plaintext for any successful decryption. See Figure 2.

In addition to the standard notion of indistinguishability against chosen-plaintext attacks, we impose two additional requirements on the encryption scheme used in the protocol. First, we require that decryption with a random (incorrect) key fails except with negligible probability; i.e., that $\Pr_{\alpha,\alpha'}[\mathsf{Dec}_{\alpha'}(\mathsf{Enc}_\alpha(x)) \neq \perp]$ is negligible for any x. Second, we require a notion of *key indistinguishability* which, roughly, means that it is impossible to distinguish two ciphertexts encrypted using the same key from two ciphertexts encrypted using different keys. Formally, we require the following to be negligible for any polynomial-time distinguisher D:

$$\left|\Pr_\alpha[D^{\mathsf{Enc}_\alpha(\cdot),\mathsf{Enc}_\alpha(\cdot)}(1^k) = 1] - \Pr_{\alpha,\alpha'}[D^{\mathsf{Enc}_\alpha(\cdot),\mathsf{Enc}_{\alpha'}(\cdot)}(1^k) = 1]\right|.$$

Both of these requirements are achieved by many standard encryption schemes, e.g., counter mode where the plaintext is padded with 0^k.

Theorem 3.1. *If $(\mathsf{Enc}, \mathsf{Dec})$ satisfies the properties outlined above, and the OPRF sub-protocol is one-sided secure (cf. Definition 2.1), then π_{search} is a one-sided secure protocol for $\mathcal{F}_{\text{ks}}$.*

Proof. We need to show that π_{search} achieves privacy against a malicious P_1, and is fully secure against a malicious P_2. Privacy when P_1 is corrupted follows easily from the assumed privacy of the OPRF sub-protocol, since P_2 sends no other messages in π_{search}. Next, we show security against a malicious P_2 in the $\mathcal{F}_{\text{OPRF}}$-hybrid model. (By standard composition theorems, this implies security of π_{search}.) To do so we describe a *simulator* $\mathcal{S}$ that is given access to an ideal functionality computing $\mathcal{F}_{\text{ks}}$. The simulator runs the

[2]Keyword repeats could be handled in prior work by padding all payloads to some maximum length, but this would be less efficient than what we propose.

adversarial P_2 as a sub-routine and extracts from P_2 its input p to $\mathcal{F}_{\text{OPRF}}$. Then $\mathcal{S}$ sends p to the functionality $\mathcal{F}_{ks}$, and receives in return the database size n, the payload length $|x|$, and a set $\{x_i\}_{i=1}^t$ for some t (possibly $t = 0$). The simulator chooses random $\alpha, \alpha' \leftarrow \{0,1\}^k$, and gives α to P_2 as its output from $\mathcal{F}_{\text{OPRF}}$. Finally, $\mathcal{S}$ prepares t ciphertexts $c_i \leftarrow \mathsf{Enc}_\alpha(x_i)$ and $n - t$ ciphertexts $c_i \leftarrow \mathsf{Enc}_{\alpha'}(0^{|x|})$ and gives these (in random permuted order) to P_2. This generates a view for P_2 that is computationally indistinguishable from the view of P_2 running π_{search} in the $\mathcal{F}_{\text{OPRF}}$-hybrid model. $\square$

3.2 Applications to Pattern Matching

We can use any protocol for keyword search as a subroutine in various text-processing tasks. We illustrate with the example of pattern matching, and then describe a more general class of functions we can compute.

The task of *pattern matching* is as follows. P_1 holds some text $T \in \Sigma^n$ and P_2 holds a pattern $p \in \Sigma^m$ for some alphabet Σ. (We assume n and m are known by both parties.) P_2 should learn the indices i (if any) where p occurs as a pattern in T. If we let T_i denote the m-character substring of T starting at position i, then P_2 should learn $\{i \mid T_i = p\}$.

Pattern matching can be reduced to keyword search[3] by having P_1 process the text T to generate the "database" $D = \{(T_i, i)\}_{i=1}^{n-m+1}$, and then having P_1 and P_2 compute $\mathcal{F}_{\text{ks}}$ using inputs D and p, respectively [9]. The primary advantage of using our keyword-search protocol here is that *it naturally allows for keyword repeats*, which in the present context means that it automatically handles the case where patterns may repeat in T (and so it may occur that $T_i = T_j$ for distinct i, j). In contrast, Hazay and Lindell [9] need to introduce additional modifications in order to deal with such repeats in the text. As a consequence, our resulting protocol for pattern matching makes only a *single* call to the underlying sub-protocol for OPRF, whereas the Hazay-Lindell protocol for pattern matching requires $n-m+1$ calls to an underlying OPRF protocol.[4]

More generally, we can use keyword search for secure computation of any text-processing task of the following form: P_1 holds a text T, and P_2 holds a pattern p; the goal is for P_2 to learn $\{f(T,i) \mid T_i = p\}$ for an arbitrary function f. (Pattern matching is a special case where $f(T,i) \overset{\text{def}}{=} i$.) This can be done by having P_1 process T to generate a database $D = \{(T_i, f(T,i))\}_{i=1}^{n-m+1}$, and then having P_1 and P_2 compute $\mathcal{F}_{\text{ks}}$ using inputs D and p, respectively. Once again, it is crucial that $\mathcal{F}_{\text{ks}}$ be able to handle keyword repeats.

We list some applications of the above technique:

Checking for tandem repeats in DNA. A *tandem repeat* in a DNA snippet T is a pattern that is repeated multiple times in adjacent locations of T. Rephrasing, a given nucleotide pattern p is said to occur as a tandem repeat in T if p^ℓ (for $\ell > 1$) occurs as a substring of T. Say P_1 holds a DNA snippet T and P_2 holds an m-character nucleotide pattern p, and P_2 wants to learn all locations i where p appears in T and, for each such location i, the largest ℓ for which p^ℓ occurs as a substring beginning at position i. Letting T_i, as always, denote the m-character substring beginning at position i, this task can be done easily within our framework by defining the function $f(T,i) = (i, \ell_{\max})$, where $\ell_{\max}$ is the largest integer such that $T_i^{\ell_{\max}}$ appears as a substring in T beginning at position i. (We remark that in this particular case efficiency can be improved by using the fact that p^ℓ occurs at position i iff $p^{\ell-1}$ occurs at position $i + m$.)

Text statistics. Our basic framework can also be used for secure computation of various statistics about T. As one example, consider the case where P_2 wants to find out the number of occurrences of p in T (without learning the locations of these occurrences). This can be done within our framework by defining $f(T,i) = N$, where N is the number of times the m-character string T_i occurs in T.

In the case when $f(T,i) = f'(T_i, T)$ for some function f' (as is the case in the preceding example), we can also use a variant of the reduction given previously. Namely, P_1 can exhaustively enumerate the space of possible patterns p and, for each p that occurs as a substring of T, add the tuple $(p, f'(p,T))$ to its database D. As before, P_1 and P_2 then compute $\mathcal{F}_{\text{ks}}$ using inputs D and p, respectively. This variant approach has complexity $O(|\Sigma|^m)$, whereas the original approach sketched earlier in this subsection has complexity $O(n-m)$. Depending on the relevant parameters, the variant approach might in some cases be preferred. (By way of example, if T represents a DNA snippet and p is a nucleotide pattern consisting of four base pairs, then $m, |\Sigma| = 4$ while we might have $T \approx 500{,}000$; the variant approach just discussed would then be preferred.)

Non-numeric values. So far we focused on computing numeric values, but clearly non-numeric values can be computed as well. For example, suppose that P_2 is interested in searching T for all occurrences of a pattern p, and whenever the pattern is found P_2 would like to learn the next t characters. This is easily handled in our framework by setting $f(T,i) = T'_{i+m}$, where T'_i denotes the substring of T of length t starting at location i.

4. GENERAL TEXT PROCESSING

In this section we give a general protocol for secure text processing by combining the keyword search functionality (described in Section 3) and Yao's garbled circuit approach. The resulting protocol has two attractive features: it can compute a wide variety of functions on a text T and a pattern p, and it does so using a number of circuits that is linear in (an upper bound on) the number of occurrences of p in T (rather than linear in $|T|$). We discuss applications and extensions of this protocol in Section 5.

4.1 An Overview of the Protocol

As usual, P_1 has a text T, and P_2 has a pattern p. In addition, we now allow P_2 to also have some private parameters y. We consider a general class of functionalities defined by two functions g and h known to both parties. Formally, define a class of functionalities $\mathcal{F}_{g,h}$ as:

$$\mathcal{F}_{g,h}(T,p,y) = \left\{ h\left(g(T,i), y \right) \mid T_i = p \right\}, \tag{2}$$

where, as usual, T_i is the substring of T of length $m = |p|$ starting at location i. (Also, we continue to assume that only

[3] Recall we aim for one-sided security only. The reductions described in this section do *not* suffice to achieve full security against a malicious P_1.

[4] Hazay and Lindell show that this overhead can be avoided by using a *specific* OPRF protocol for a *specific* (number-theoretic) PRF. We avoid any additional overhead while using OPRF as a black box.

Protocol π_{txt}

Input to P_1: A text $T \in \Sigma^n$.
Input to P_2: A pattern $p \in \Sigma^m$ $(m < n)$ and parameters $y \in \{0,1\}^k$.

Common input: The input lengths n, m and an upper bound u on the number of times p appears as a substring in T.

Output: P_2 learns $\left\{ h\left(g(T,i), y\right) \mid T_i = p \right\}$.

Let $t = n - m + 1$. The parties do:

1. P_1 runs u invocations of $\mathsf{Garble}(1^k, H)$ to obtain $(\mathsf{gH}_1, \vec{X}_1, \vec{Y}_1)$, $\ldots$, $(\mathsf{gH}_u, \vec{X}_u, \vec{Y}_u)$.
2. P_1 and P_2 execute k (parallel) instances of OT. In the ith instance, the inputs of P_1 are the two strings $(Y_1^{i,0}, \ldots, Y_u^{i,0})$ and $(Y_1^{i,1}, \ldots, Y_u^{i,1})$, and the input of P_2 is y_i. At the end of this step, P_2 holds $\vec{Y}_1(y), \ldots, \vec{Y}_u(y)$.
3. P_1 sets $x_i = g(T,i)$ for all i. Then P_1 defines a database D as follows. Choose a random permutation π of $\{1, \ldots, u\}$. Then for $i = 1, \ldots, t$ do:
   (a) Say this is the Nth time T_i has been encountered as a substring in T. (Note that $1 \le N \le u$.) Let $j = \pi(N)$.
   (b) Add $\left(T_i, \left(j, \vec{X}_j(x_i)\right)\right)$ to D.
4. P_1 and P_2 compute functionality $\mathcal{F}_{\text{ks}}$ using inputs D and p, respectively. As a result, P_2 obtains a set $\{(j, \vec{X}'_j)\}_{j \in U}$ for some $U \subseteq \{1, \ldots, u\}$.
5. P_1 sends $\mathsf{gH}_1, \ldots, \mathsf{gH}_u$ to P_2.
6. Party P_2 outputs $\left\{ \mathsf{Eval}\left(\mathsf{gH}_j, \vec{X}'_j, \vec{Y}_j(y)\right) \right\}_{j \in U}$.

Figure 3: A protocol for secure text processing.

P_2 obtains output.) As described, $\mathcal{F}_{g,h}$ allows P_2 to learn a *set* of values, but in some applications it is desirable to instead compute a single result based on these values (e.g., the example from the Introduction). In the following section we describe how to apply our techniques to that case.

The idea behind our protocol is to compute $\mathcal{F}_{g,h}(T,p,y)$ using keyword search and Yao's garbled circuit. We motivate how this is done, postponing discussion of several technical details to the following section. Let u denote a (known) upper bound on the number of times any m-character pattern repeats in T. At the beginning of the protocol P_1 constructs u garbled circuits $\mathsf{gH}_1, \ldots, \mathsf{gH}_u$ from the circuit H for the function h. We want P_2 to be able to evaluate these garbled circuits on P_2's input y, as well as the (at most u) values $x_i \overset{\text{def}}{=} g(T,i)$ for which $T_i = p$. Note that P_1 can compute all the x_i without any interaction with P_2. Thus, all we need to do is provide a way for P_2 to learn the appropriate input-wire labels.

It is easy for P_2 to learn the labels of the wires corresponding to its own input y using oblivious transfer. In fact, because P_2's input to each of the (garbled) circuits is the same we can accomplish this using just $|y|$ invocations of (string) oblivious transfer. A further optimization would be for P_1 to use the same labels for the wires corresponding to P_2's input, in each of the u circuits. (For simplicity, this optimization is not applied in the following section.)

To enable P_2 to learn the labels of the wires corresponding to P_1's input(s) to the u garbled circuits, we rely on keyword search as a sub-routine. Essentially (but omitting some technical details), P_1 prepares a database D with entries of the form $(T_i, \vec{X}_j(x_i))$ where $\vec{X}_j$ denotes the labels on the input wires of P_1 in the jth garbled circuit. When P_1 and P_2 then run keyword search (with P_2 using p as its keyword), P_2 learns exactly the input-wire labels for those indices i satisfying $T_i = p$. To complete the protocol, P_2 needs only to evaluate each garbled circuit using the input-wire labels it has obtained for that circuit.

4.2 The Protocol

Our protocol is described in Figure 3, where we use H to denote a circuit for computing the function h. We remark that the permutation π is used to hide the ordering in which the results $h\left(g(T,i), y\right)$ are computed; this ordering should be hidden because the output of $\mathcal{F}_{g,h}$ (cf. Equation (2)) is an unordered *set*. If we instead define the output of $\mathcal{F}_{g,h}$ to be an ordered *list*, then this permutation would be unnecessary.

Note also that although our description of the protocol assumes that g, h are deterministic it would not be difficult to modify the protocol to handle probabilistic functionalities: g is anyway computed locally by P_1, and P_1 could hardwire (independent) randomness into each circuit H before garbling it. (These fixes apply when considering one-sided security, but we note that they would not suffice to guarantee full security against a malicious P_1.)

THEOREM 4.1. *Fix (probabilistic) polynomial-time functions g and h, and consider protocol π_{txt} from Figure 3. If the sub-protocols for the parallel OT and $\mathcal{F}_{\text{ks}}$ are each one-sided secure, then π_{txt} is a one-sided secure protocol for the functionality $\mathcal{F}_{g,h}$.*

PROOF. We need to show that π_{txt} is private even against a malicious P_1, and fully secure against a malicious P_2. Privacy when P_1 is corrupted follows easily from the assumed privacy of the sub-protocols used, since P_2 sends no messages in π_{txt} other than the messages it sends during executions of the OT and keyword-search sub-protocols. Next, we show security against a malicious P_2 in a hybrid model where the parties have access to ideal functionalities computing $\mathcal{F}_{\text{ks}}$ and the parallel OT functionality. We prove security by briefly describing a *simulator* that is given access to an ideal functionality computing $\mathcal{F}_{g,h}$. (By standard composition theorems, this implies security of π_{txt}.)

Let $(\mathcal{S}_1, \mathcal{S}_2)$ be the simulators guaranteed for the Yao garbled circuit construction. Our simulator begins by running u independent copies of $\mathcal{S}_1(1^k, H)$ to obtain $(\vec{X}'_1, \vec{Y}'_1, s_1)$, ..., $(\vec{X}'_u, \vec{Y}'_u, s_u)$ (note that here each $\vec{X}', \vec{Y}'$ is a k-tuple of strings, cf. Section 2.1). The simulator then extracts from P_2 its input y to the parallel OT functionality, and provides to P_2 in return the vectors $\vec{Y}'_1, \ldots, \vec{Y}'_u$. Next, the simulator extracts from P_2 its input p to the $\mathcal{F}_{\text{ks}}$ functionality. The simulator sends (p, y) to the ideal functionality computing $\mathcal{F}_{g,h}$ and receives in return a set $\{z_i\}_{i=1}^{u'}$ (where $0 \leq u' \leq u$).

The simulator then chooses a random permutation π of $\{1, \ldots, u\}$ and gives to P_2 the values $\{(\pi(i), \vec{X}'_{\pi(i)})\}_{i=1}^{u'}$ as the output from the invocation of $\mathcal{F}_{\text{ks}}$. To complete the simulation, the simulator computes $\mathsf{gH}_{\pi(i)} \leftarrow \mathcal{S}_2\left((p, y), z_i, s_{\pi(i)}\right)$ for $i = 1$ to u', and $\mathsf{gH}_{\pi(i)} \leftarrow \mathcal{S}_2(p, y, \perp, s_{\pi(i)})$ for $i = u' + 1$ to u. It then sends $\mathsf{gH}_1, \ldots, \mathsf{gH}_u$ as the final message to P_2.

We omit the proof that this generates a view for P_2 that is computationally indistinguishable from the view of P_2 when running π_{txt} in the specified hybrid model. $\square$

4.3 Efficiency

The most notable feature of our protocol is that it uses only u garbled circuits, rather than $O(|T|)$ garbled circuits as in a naive application of Yao's methodology. To see the resulting improvement, let us focus on the communication complexity (though a similar calculation applies to the computational complexity also) and concentrate on terms that depend on $|T|$ — a reasonable choice if we assume $|T|$ dominates all other parameters. Any "naive" application of Yao's approach to computing the functionality in Equation (2) will involve garbling a circuit containing (among other things) $|T|$ copies of H. The communication required for transmitting the resulting garbled circuit is thus lower bounded by (roughly) $4k|T||H|$ bits. In our protocol, on the other hand, the only dependence on $|T|$ is in the sub-protocol for keyword search. Looking at the keyword-search protocol from Section 3 (and ignoring the OPRF sub-protocol there, whose complexity is independent of $|T|$), we see that the communication complexity used by that protocol will be (roughly) $|T| \cdot (k\,|g_{\text{out}}| + k)$ bits, where $|g_{\text{out}}|$ denotes the output length of the function g.

5. APPLICATIONS AND EXTENSIONS

In Section 4 we have described a protocol π_{txt} for secure text processing. In this section we describe some extensions and potential applications of that protocol.

Let us return to the functionality considered in the Introduction (cf. Equation (1)). That function can "almost" be viewed as an instance of the class of functions $\mathcal{F}_{g,h}$ described in the previous section if we set $g(T, i) \overset{\text{def}}{=} \ell_{\max}(T, T_i)$ and

$$h(\ell', (\epsilon, \ell)) = \begin{cases} 1 & |\ell' - \ell| \leq \epsilon \\ 0 & \text{otherwise} \end{cases}. \tag{3}$$

Two problems arise in formulating the problem this way and applying the protocol from the previous section: first, the resulting protocol is *inefficient* since P_2 (possibly) gets the same answer u times rather than just once; second, it is *insecure* since it reveals how many times p occurs as a substring in T (whereas the functionality as described in Equation (1) does not reveal this information).

Nevertheless, we can address both these issues with a protocol constructed using the same general paradigm employed in the previous section. Specifically, we now have P_1 construct only a single garbled circuit gH for H (reflecting the fact that a single evaluation of h is sufficient for computing the desired functionality). We need P_2 to evaluate this circuit on P_2's inputs ϵ, ℓ as well as the value $\ell' = g(T, p)$. Once again, the problem reduces to finding a way for P_2 to learn all the appropriate input-wire labels.

As before, it is easy for P_2 to learn the labels of the wires corresponding to its own inputs ϵ, ℓ using oblivious transfer. To enable P_2 to learn the appropriate input-wire labels, we use keyword search: P_1 prepares a "database" of entries of the form $(p^*, g(T, p^*))$, and P_1 learns only $g(T, p)$. An additional subtlety here is that we need to prevent P_2 from learning *how many times* the pattern p appears in T; this can be handled by "padding" the database using entries with random keys. See Figure 4.

In Figure 4, P_1 "pads" the database D so that it always contains exactly $t = n - m + 1$ entries; this prevents P_2 from learning how many distinct patterns occur as substrings of T. An alternative approach, which may be more efficient depending on the relative sizes of $|T|$ and $|\Sigma|^m$, is described next. First, both parties re-define h as:

$$h(b|\ell', (\epsilon, \ell)) = \begin{cases} b & |\ell' - \ell| \leq \epsilon \\ 0 & \text{otherwise} \end{cases}.$$

Then for each pattern $p \in \Sigma^m$, party P_1 does:

- If p occurs as a substring in T, then add the tuple $\left(p, \vec{X}(1|\ell_{\max}(T, p))\right)$ to D.
- If p does not occur as a substring in T, then add the tuple $\left(p, \vec{X}(0|0^{|\ell_{\max}|})\right)$ to D.

Now the database always has exactly $|\Sigma|^m$ entries.

On another note, we remark that the protocol in Figure 4 returns $\perp$ to P_2 in case its pattern p does not occur as a substring of T. This is avoided when using the technique described in the previous paragraph (and can be avoided in the protocol from Figure 4 via similar techniques).

6. REFERENCES

[1] W. Aiello, Y. Ishai, and O. Reingold. Priced oblivious transfer: How to sell digital goods. In *Advances in Cryptology — Eurocrypt 2001*, volume 2045 of *LNCS*, pages 119–135. Springer, 2001.

[2] M. Atallah, F. Kerschbaum, and W. Du. Secure and private sequence comparisons. In *Proc. ACM Workshop on Privacy in the Electronic Society (WPES)*, pages 39–44. ACM, 2003.

[3] E. De Cristofaro and G. Tsudik. Practical private set intersection protocols with linear complexity. In *Financial Cryptography and Data Security 2010*. Available at `http://eprint.iacr.org/2009/491`.

[4] M. J. Freedman, Y. Ishai, B. Pinkas, and O. Reingold. Keyword search and oblivious pseudorandom

Input to P_1: A text $T \in \Sigma^n$.
Input to P_2: A pattern $p \in \Sigma^m$ $(m < n)$ and parameters ϵ, ℓ.

Common input: The input lengths n, m.

Output: P_2 learns $M(T, p, \epsilon, \ell)$ (see Equation (1)).

Let h be as in Equation (3), and let H be a circuit computing h. Let $t = n-m+1$. The parties do:

1. P_1 runs $\mathsf{Garble}(1^k, H)$ to obtain $(\mathsf{gH}, \vec{X}, \vec{Y})$.
2. P_1 and P_2 execute (parallel) instances of OT to enable P_2 to learn the input-wire labels $\vec{Y}'$ corresponding to its inputs ϵ, ℓ.
3. P_1 defines a database D as follows.
   (a) For each $p \in \Sigma^m$ that is a substring of T, add $\left(p|0^k, \vec{X}(\ell_{\max}(T,p))\right)$ to D. (We assume $\ell_{\max}$ is always a fixed number of bits.)
   (b) Let d denote the number of elements in D. Then add an additional $t-d$ elements to D of the form $(p^* r, \vec{X}(0^{|\ell_{\max}|}))$, where $r \leftarrow \{0,1\}^k$ and $p^* \in \Sigma^m$ is arbitrary.
4. P_1 and P_2 compute $\mathcal{F}_{\mathrm{ks}}$ on D and $p|0^k$, respectively. As a result, P_2 obtains $\vec{X}'$ or nothing. (In the latter case, P_2 outputs $\perp$. See text for discussion.)
5. P_1 sends gH to P_2.
6. P_2 outputs $\mathsf{Eval}\left(\mathsf{gH}, \vec{X}', \vec{Y}'\right)$.

Figure 4: Computing the functionality of Equation (1).

functions. In *2nd Theory of Cryptography Conference*, volume 3378 of *LNCS*, pages 303–324. Springer, 2005.

[5] K. Frikken. Practical private DNA string searching and matching through efficient oblivious automata evaluation. In *Data and Applications Security*, volume 5645 of *LNCS*, pages 81–94. Springer, 2009.

[6] R. Gennaro, C. Hazay, and J. S. Sorensen. Text search protocols with simulation based security. In *13th Intl. Conference on Theory and Practice of Public Key Cryptography (PKC 2010)*, volume 6056 of LNCS, pages 332–350. Springer, 2010.

[7] O. Goldreich. *Foundations of Cryptography, vol. 2: Basic Applications.* Cambridge University Press, Cambridge, UK, 2004.

[8] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game, or a completeness theorem for protocols with honest majority. In *19th Annual ACM Symposium on Theory of Computing (STOC)*, pages 218–229. ACM Press, 1987.

[9] C. Hazay and Y. Lindell. Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries. In *5th Theory of Cryptography Conference — TCC 2008*, volume 4948 of *LNCS*, pages 155–175. Springer, 2008.

[10] S. Jarecki and V. Shmatikov. Efficient two-party secure computation on committed inputs. In *Advances in Cryptology — Eurocrypt 2007*, volume 4515 of *LNCS*, pages 97–114. Springer, 2007.

[11] S. Jha, L. Kruger, and V. Shmatikov. Towards practical privacy for genomic computation. In *IEEE Symp. Security & Privacy*, pages 216–230. IEEE, 2008.

[12] L. Kissner and D. X. Song. Privacy-preserving set operations. In *Advances in Cryptology — Crypto 2005*, volume 3621 of *LNCS*, pages 241–257. Springer, 2005.

[13] A. Lindell. Efficient fully-simulatable oblivious transfer. In *Cryptographers' Track — RSA 2008*, volume 4964 of *LNCS*, pages 52–70. Springer, 2008.

[14] Y. Lindell and B. Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In *Advances in Cryptology — Eurocrypt 2007*, volume 4515 of *LNCS*, pages 52–78. Springer, 2007.

[15] Y. Lindell, B. Pinkas, and N. Smart. Implementing two-party computation efficiently with security against malicious adversaries. In *6th Intl. Conf. on Security and Cryptography for Networks (SCN '08)*, volume 5229 of *LNCS*, pages 2–20. Springer, 2008.

[16] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella. Fairplay — a secure two-party computation system. In *Proc. 13th USENIX Security Symposium*, pages 287–302. USENIX Association, 2004.

[17] M. Naor and B. Pinkas. Computationally secure oblivious transfer. *J. Cryptology*, 18(1):1–35, 2005.

[18] A. Paus, A.-R. Sadeghi, and T. Schneider. Practical secure evaluation of semi-private functions. In *Conf. on Applied Cryptography and Network Security*, volume 5536 of *LNCS*, pages 89–106. Springer, 2009.

[19] B. Pinkas, T. Schneider, N. Smart, and S. Williams. Secure two-party computation is practical. In *Advances in Cryptology — Asiacrypt 2009*, volume 5912 of *LNCS*, pages 250–267. Springer, 2009.

[20] J. R. Troncoso-Pastoriza, S. Katzenbeisser, and M. Celik. Privacy preserving error resilient DNA searching through oblivious automata. In *14th ACM Conf. on Computer and Communications Security (CCCS)*, pages 519–528. ACM Press, 2007.

[21] A. C.-C. Yao. How to generate and exchange secrets. In *27th Annual Symp. on Foundations of Computer Science (FOCS)*, pages 162–167. IEEE, 1986.