Computationally Secure Pattern Matching in the Presence of Malicious Adversaries

Carmit Hazay[1] and Tomas Toft[1]

Aarhus University
{carmit,ttoft}@cs.au.dk

Abstract. We propose a dedicated protocol for the highly motivated problem of secure two-party pattern matching: Alice holds a text $t \in \{0,1\}^*$ of length n, while Bob has a pattern $p \in \{0,1\}^*$ of length m. The goal is for Bob to learn where his pattern occurs in Alice's text. Our construction guarantees full simulation in the presence of malicious, polynomial-time adversaries (assuming that ElGamal encryption is semantically secure) and exhibits computation and communication costs of $O(n+m)$ in a constant round complexity.
In addition to the above, we propose a collection of protocols for variations of the secure pattern matching problem: The pattern may contain wildcards ($O(nm)$ communication in $O(1)$ rounds). The matches may be approximated, i.e., Hamming distance less than some threshold ($O(nm)$ communication in $O(1)$ rounds). The length, m, of Bob's pattern is secret ($O(nm)$ communication in $O(1)$ rounds). The length, n, of Alice's text is secret ($O(n+m)$ communication in $O(1)$ rounds).

Key words: Pattern matching, secure two-party computation, full simulation, malicious adversary.

1 Introduction

In the setting of secure two-party computation, two parties with private inputs wish to jointly compute some function of their inputs while preserving certain security properties like privacy, correctness and more. The standard definition [GL91,Bea92,MR91,Can00] formalizes security by comparing the execution of such protocol to an "ideal execution" where a trusted third party computes the function for the parties. 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. Then, a real protocol is said to be secure if no adversary can do more harm in a real protocol execution than in an ideal one (where by definition no harm can be done).

Secure two-party computation has been extensively studied, and it has been demonstrated that any polynomial-time two-party computation can be generically compiled into a secure function evaluation protocol with polynomial complexity [Yao86,GMW87,Gol04]. These results apply in various settings, (considering semi-honest and malicious adversaries). However, more often than not, the

resulting protocols are inefficient for practical uses (in part because they are general and so do not utilize any specific properties of the protocol problem at hand) and hence attention has been given to constructing efficient protocols for specific functions. This approach has proved quite successful for the semi-honest setting (see, e.g., [LP02,AMP04,FNP04,KS05,TPKC07]), while the malicious setting remained impractical (a notable exception is [AMP04]).

In this paper we consider the following classic search problem: Alice holds a text $t \in \{0,1\}^*$ of length n and Bob is given a pattern (i.e., search word) $p \in \{0,1\}^*$ of length m, where the sizes of t and p are mutually known. The goal is for Bob to learn all the locations in the text that match the pattern, while Alice learns nothing about the pattern. This problem has been widely studied for decades due to its potential applications for text retrieval, music retrieval, computational biology, data mining, network security, and many more. The most known application in the context of privacy is in compering two DNA strings; our example is taken from [GHS10]. Consider the 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 he is working on, nor trust the hospital to perform the search correctly.

Although most of the existing solutions are highly practical they fail to achieve any level of security (if at all); see [Blo70,KMP77,BM77,ACR99,NM07] for just a few examples. In this work, we focus our attention on the secure computation of the basic pattern matching problem and several important variants of it.

Our Contribution. We achieve efficiency that is a significant improvement on the current state of the art for the following problems:

- Secure Pattern Matching. We develop an efficient, constant rounds protocol for this problem that requires $O(n+m)$ exponentiations and bandwidth of $O(n+m)$ group elements. Our protocol lays the foundations for the following constructions.
- Secure Pattern Matching with Wildcards. This problem is a known variant of the classic problem where Bob (who holds the pattern) introduces a new "don't care" character to its alphabet, denoted by $\star$ (wildcard). The goal is for Bob to learn all the locations in the text that match the pattern, where $\star$ matches any character in the text. This problem has been widely looked at by researchers with the aim of generalizing the basic searching model to searching with errors. This variant is known as pattern matching with don't cares and can be solved in $O(n+m)$ time [IR07]. In this paper, we develop a protocol that computes this functionality with $O(nm)$ costs.
- Secure Approximate Pattern Matching. In this problem the goal is for Bob to find the locations where the Hamming distance of the (text) substrings and the pattern is less than some threshold $\tau \leq m$. We design a protocol for this problem with $O(mn)$ costs.

– Secure Pattern Matching where the length of the pattern or the text remains hidden. Finally, we consider two variants with an additional security requirement of hiding the input length. Solutions for these problems can be achieved in $O(nm)$ time.

Our protocols are based on ElGamal encryption and are proven secure in the plain model under the standard DDH assumption and achieve full simulation in the presence of malicious adversaries.

Prior Work. To the best of our knowledge, the first who considered pattern matching in the context of secure computation were [TPKC07] who considered a secure version of oblivious automata evaluation to achieve secure pattern matching. Their protocol implements the KMP algorithm [KMP77] in the semi honest setting. Loosely speaking, the KMP algorithm works in $O(n)$ time and searches for occurrences of the pattern within the text by employing the observation that when a mismatch occurs, the pattern embodies sufficient information to determine where the next match could begin. Their costs are linear in the input length.

This problem was also studied by Hazay and Lindell in [HL08] who used oblivious pseudorandom function (PRF) evaluation. However, their protocol achieves only a weaker notion of security called one-sided simulatability which does not guarantee full simulation for both corruption cases. The only construction to achieve full simulation in the malicious setting was developed by Gennaro et al. [GHS10]. They took a different approach to implement the KMP algorithm and described a protocol that runs in $O(m)$ rounds and requires $O(nm)$ exponentiations and bandwidth.

Finally, a recent paper by Katz and Malka [KM10] presents a secure solution for a generalized pattern matching problem, denoted text processing. Namely, the party who holds the pattern has some additional information y and his goal is to learn a function of the text and y, for the text locations where the pattern matches. They 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 $|t|$). Their costs are dominated by the size of the circuit times the number of occurrences u (as P_1 sends u such circuits). Nevertheless, they assume a common input of some threshold on the number of occurrences.

To the best of our knowledge, the only work which addresses one of the above variants is the work by Jarrous and Pinkas [JP09]. In this work, the authors solve the hamming distance problem for two equal length strings against malicious adversaries. Their protocol requires a committed oblivious transfer for each bit. Moreover, the costs of their protocol are inflated by a statistical parameter s for running a subprotocol for the oblivious polynomial evaluation functionality (namely, the protocol requires $O(d \cdot s)$ exponentiations, where d is the degree of the polynomial, i.e., the input length). Finally, their protocol utilizes the Paillier encryption scheme and thus requires an RSA modulus with unknown

factorization. Our protocol, on the other hand, takes a different approach and requires linear costs, for the case of equal length strings.

Efficiency. In addition to prior work, we compare our protocols to the generic garbling-technique by Yao (formally proved by Lindell and Pinkas) [LP07] for secure computation of any functionality in the two-party setting. Recall that Yao's protocol uses a Boolean circuit that computes the function, and its computational complexity is linear in the size of the circuit. Note that computing the pattern matching functionality would require a circuit of size $O(nm)$, as the circuit will compare every pattern against every text location (As noted by [GHS10], a circuit that implements the functionality for oblivious automata evaluation would require $O(mn \log m)$ gates, thus the KMP technique does not contribute to efficiency here). Consequently, our protocol for the basic pattern matching functionality is more efficient than Yao's construction even in the presence of semi-honest adversaries; this is also the case for other circuit based approaches.

Organization of this paper. We first present the underlying primitives in Section 2. The following sections then contain our protocols. The basic protocol is presented in Section 3. This is then extended, first with wildcards in the pattern (Section 4) followed by approximate matching (Section 5). Finally, the paper concludes with the protocols which hide the pattern and texts lengths (Sections 6 and 7).

2 Preliminaries and Tools

Throughout the paper, we denote the security parameter by κ. A function $\mu(\cdot)$ is *negligible* in κ (or simply *negligible*) if for every polynomial $p(\cdot)$ there exists a value K such that $\mu(\kappa) < \frac{1}{p(\kappa)}$ for all $\kappa > K$; i.e., $\mu(\kappa) = \kappa^{-\omega(1)}$. Let $X = \{X(\kappa, a)\}_{\kappa \in \mathbb{N}, a \in \{0,1\}^*}$ and $Y = \{Y(\kappa, a)\}_{\kappa \in \mathbb{N}, a \in \{0,1\}^*}$ 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 for every $\kappa \in \mathbb{N}$ and $a \in \{0,1\}^*$

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

2.1 The ElGamal Encryption Scheme

At the core of the proposed protocols lies the additively homomorphic variation of ElGamal encryption – $E_{pk}(m, r) = \langle g^r, h^r g^m \rangle$ with distributed decryption over a group $\mathbb{G}_q$ in which DDH is hard, [ElG85]. Essentially, we use the framework of Brandt [Bra05] with minor variations. We present the computation of the parties with respect to the ciphertext space, in particular, we write C^r meaning $\langle \alpha^r, \beta^r \rangle$ and C/C' meaning $\langle \alpha/\alpha', \beta/\beta' \rangle$ for ciphertexts $C = \langle \alpha, \beta \rangle$ and $\mathbb{C}' = \langle \alpha', \beta' \rangle$, and $r \in \mathbb{Z}_q$.

2.2 Zero-knowledge Proofs for $\mathbb{G}_q$ and ElGamal Encryption

To prevent malicious behaviour, the parties must demonstrate that they are well-behaved. To achieve this, our protocols utilize zero-knowledge proofs of knowledge. All of them are Σ-protocols (with constant communication complexity) which show knowledge of a witness that some statement is true (belong to a relation, $\mathcal{R}$) about one or more elements of $\mathbb{G}_q$. The Σ-protocols can be made secure against malicious verifiers using standard techniques; we denote the associated ideal functionalities for these protocols, $\mathcal{F}_{\text{ZK}}^{\mathcal{R}_{\text{DL}}}$, $\mathcal{F}_{\text{ZK}}^{\mathcal{R}_{\text{EqDL}}}$, $\mathcal{F}_{\text{ZK}}^{\mathcal{R}_{\text{isBit}}}$, $\mathcal{F}_{\text{ZK}}^{\mathcal{R}_{\text{mult}}}$, $\mathcal{F}_{\text{ZK}}^{\mathcal{R}_{\text{perm}}}$, and $\mathcal{F}_{\text{ZK}}^{\mathcal{R}_{\text{nze}}}$.

π_{DL}, due to Schnorr, allows the prover to demonstrate knowledge of the solution, x, to a discrete logarithm problem, [Sch89].

$$\mathcal{R}_{\text{DL}} = \{((\mathbb{G}_q, q, g, h), x) \mid h = g^x\}$$

π_{EqDL}, due to Chaum and Pedersen, demonstrates equality of two discrete logarithm problems (as well as knowledge of the solution), [CP93].

$$\mathcal{R}_{\text{EqDL}} = \{((\mathbb{G}_q, q, g_1, g_2, h_1, h_2), x) \mid h_1 = g_1^x \wedge h_2 = g_2^x\}$$

Phrased differently, π_{EqDL} demonstrates that a quadruple forms a Diffie-Hellman tuple or, equivalently, that a ciphertext is an encryption of 0.

π_{isBit} demonstrates that for ciphertext C, either C or $C\left\langle 1, g^{-1}\right\rangle$ is an encryption of 0, i.e. that it is an encryption of either 0 or 1. This can be obtained directly from π_{EqDL} using the compound proof of Cramer et al. [CGS97].

$$\mathcal{R}_{\text{isBit}} = \left\{((\mathbb{G}_q, q, g, h, \alpha, \beta), (b, r)) \mid (\alpha, \beta) = (g^r, h^r \cdot g^b) \wedge b \in \{0, 1\}\right\}$$

π_{mult}, due to Abe et al., demonstrates that a party, the prover P has performed a multiplication under the encryption correctly [ACF02]. I.e. given ciphertext C, P, knowing f, has computed $C_f = E_{pk}(f, r_f)$ and $C_\pi = C^f \cdot E_{pk}(0, r_\pi)$; clearly the plaintext of C_π is the product of the other plaintexts.

$$\mathcal{R}_{\text{mult}} = \left\{((sk, C, C_f, C_\pi), (f, r_f, r_\pi)) \text{ s.t. } \begin{array}{l} C_f = \left\langle g^{r_f}, h^{r_f} \cdot g^f \right\rangle \wedge \\ C_\pi = C^f \cdot \langle g^{r_\pi}, h^{r_\pi} \rangle \end{array} \right\}$$

π_{perm} allows a prover to demonstrate that a set of encryptions, $\{C_i\}_i$, is a permutation and rerandomization of the another, $\{C'_i\}_i$ – i.e. that their plaintexts are equal. Any protocol will do, Groth's solution [Gro03] is one possibility.

$$\mathcal{R}_{\text{perm}} = \left\{((sk, \{C_i\}_i, \{C'_i\}_i), (\pi, \{r_i\}_i)) \text{ s.t. } \langle \alpha'_i, \beta'_i \rangle = \left\langle \alpha_{\pi(i)} g^{r_i}, \beta_{\pi(i)} h^{r_i} \right\rangle\right\}$$

π_{nze} demonstrates that the prover has obtained ciphertext C' from C, by raising C to a non-zero exponent and rerandomizing, i.e. $C' = C^R \cdot E_{pk}(0, r)$. The tricky part when constructing a proof of knowledge for the relation,

$$\mathcal{R}_{\text{nze}} = \left\{((sk, \alpha, \beta, \alpha', \beta'), (R, r)) \text{ s.t. } \langle \alpha', \beta' \rangle = \left\langle \alpha^R g^r, \beta^R h^r \right\rangle \wedge R \neq 0\right\},$$

is to show that $R \neq 0$. To do this, the prover, P, picks $R' \in_R \mathbb{Z}_q^*$, supplies the verifier with additional ciphertexts, $C_R = E_{pk}(R, r_R)$, $C_{R'} = E_{pk}(R', r_{R'})$, and $C_\pi = E_{pk}(RR', r_\pi)$, and executes π_{mult} twice: on (C, C_R, C') and $(C_R, C_{R'}, C_\pi)$. The prover then sends RR' to the verifier and demonstrates it is the plaintext of C_π using π_{EqDL}. Finally, the verifier checks that the RR' is non-zero.

The executions of π_{mult} demonstrate that C' has been obtained from C through exponentiation, and that the plaintext of C_π depends on R. π_{EqDL} and the final check ensures that $RR' \neq 0$ implying that so is R. Hence the protocol demonstrates that C' has been obtained correctly. Further, since the verifier receives only ciphertexts along with RR' – which is uniformly random due to R' – π_{nze} is zero-knowledge.

2.3 Distributed ElGamal Encryption

In a distributed scheme, the parties hold shares of the secret key so that the combined key remains a secret. In order to decrypt, each party uses its share to generate an intermediate computation which are eventually combined into the decryption.

Note that the Diffie-Hellman key exchange [DH76] can be used for generating a public key and an additive sharing of the corresponding secret key [Ped91]. The parties first agree on $\mathbb{G}_q$ and g. Then, each party P_i picks $s_i \in_R \mathbb{Z}_q$ and sends $h_i = g^{s_i}$ to the other. Finally, the parties compute $h = h_1 \cdot h_2$ and set $pk = \langle \mathbb{G}_q, q, g, h \rangle$. Clearly the secret key associated with this public key is $s = s_1 + s_2$. In order to ensure correct behavior, the parties must prove knowledge of their s_i by running π_{DL} on (g, h_i). We denote this protocol by π_{KeyGen} which is correlated with the functionality $\mathcal{F}_{\text{KeyGen}}(1^\kappa, 1^\kappa) = ((pk, sk_1), (pk, sk_2))$.

To decrypt a ciphertext $C = \langle \alpha, \beta \rangle$, the parties raise α to the power of their shares, send these to each other, and prove this was done correctly. Both then output $\beta/(\alpha_1\alpha_2)$. We denote this protocol by π_{Dec}. Note that this protocol allows variation where only one party obtains the decrypted result.

Our final primitive is a variation of π_{Dec} where P_1 learns whether the ciphertext m of the input $C = \langle \alpha, \beta \rangle$ is zero, but nothing more. P_2 first raises C to a random, non-zero power, rerandomizes the result, and sends it to P_1. The parties then execute π_{nze} to let P_1 verify P_2's behavior. They then decrypt the final ciphertext towards P_1, who concludes that $m = 0$ iff the masked plaintext was 0. Simulation is trivial given access to $\mathcal{F}_{\text{ZK}}^{\mathcal{R}_{\text{nze}}}$. We denote this protocol π_{dec0} and the associated ideal functionality $\mathcal{F}_{\text{dec0}}$

3 The Basic, Linear Solution

In this section we present our solution for the classic pattern matching problem. Initially, Alice holds an n-bit string t, while Bob holds an m-bit pattern, p and the parties wish the compute the functionality $\mathcal{F}_{\text{PM}}$ defined by,

$$((p, n), (t, m)) \mapsto \begin{cases} (\{j \mid \bar{t}_j = p\}_{j=1}^{n-m+1}, \lambda) & \text{if } |p| = m \text{ and } |t| = n \\ (\lambda, \lambda) & \text{otherwise} \end{cases}$$

where λ is an empty string and $\bar{t}_j$ is the substring of length m that begins at the jth position in t. This problem has been widely studied for decades due to its potential applications and can be solved in linear time complexity [KMP77,BM77], when no level of security is required. We examine a secure version for this problem where Alice does not gain any information about the pattern from the protocol execution, whereas Bob does not learn anything but the matched text locations. In our setting, the parties share no information (except for the input length), though it is assumed that they are connected by an authenticated communication channel, and that the inputs are over the binary alphabet. Extending this to larger alphabets is discussed below. Our protocol exhibits overall linear communication and computation costs and achieves full simulation in the presence of malicious adversaries.

Here and below, we have the parties jointly (and securely) transform their input from binary representation into elements of $\mathbb{Z}_q$ (we assume that $m < \log_2 q$; larger pattern-lengths can be accommodated, e.g. by increasing the plaintext space.), while exploiting the fact that every two consecutive substrings of the text are closely related. Informally, both parties break their inputs into bits and encrypt each bit separately. Next, the parties map every m consecutive encryptions of bits into a single encryption that denotes an m-character for which its binary representation is assembled from these m bits. Thus, the problem is reduced to comparing two elements of $\mathbb{Z}_m$ (embedded into $\mathbb{Z}_q$). The crux of our protocol is to efficiently compute this mapping.

We are now ready to give a detailed description of our construction.

Protocol $\pi_{\mathbf{PM}}$

- **Inputs:** The input of *Alice* is a binary string t of length n and an integer m, whereas the input of *Bob* is a binary string p of length m and an integer n. The parties share a security parameter 1^κ as well.
- **The protocol:**

  1. Alice and Bob run protocol $\pi_{\text{KeyGen}}(1^\kappa, 1^\kappa)$ to generate a public key $pk = \langle \mathbb{G}_q, q, g, h \rangle$, and the respective shares s_A and s_B of the secret key sk of Alice and Bob.
  2. Bob sends encryptions $P_i = E_{pk}(p_i; r_{p_i})$, $i = 1, \ldots, m$, of his m-bit pattern, p, to Alice. Further, for each encryption the parties run the zero-knowledge proof of knowledge π_{isBit}, allowing Alice to verify that the plaintext of P_i is a bit known to Bob, i.e. that he has provided a bit-string of length m. Both parties then compute an encryption of Bob's pattern,

  $$P \leftarrow \prod_{i=1}^{m} P_i^{2^{i-1}} \tag{1}$$

  using the homomorphic property of ElGamal encryption.
  3. Alice sends encryptions, $T_j = E_{pk}(t_j; r_{t_j})$ $j = 1, \ldots, n$, of the bits t_j of her n-bit text, t, to Bob. Further, for each encryption the parties run π_{isBit}, allowing Bob to verify that the plaintext of T_j is a bit known to Alice, i.e. that she has indeed provided the encryption of a bit-string of length n that she knows.

4. Let $\bar{t}_j$ be the m-bit substring of Alice's text t, starting at position $j = 1, \ldots, n-m+1$. For each such string both parties compute an encryption of that string,
$$\bar{T}_j \leftarrow \prod_{i=j}^{j+m-1} T_i^{2^{i-j}}. \tag{2}$$
5. For every $\bar{T}_j$, $j = 1, \ldots, n-m+1$, both parties compute
$$\Delta_j \leftarrow \bar{T}_j \cdot P^{-1}. \tag{3}$$
6. For every Δ_j $j = 1, \ldots, n-m+1$, Alice and Bob reveal to Bob whether its plaintext δ_j is zero by running π_{dec0}. Bob then outputs j if this is the case.

CORRECTNESS OF π_{PM}. Before turning to our proof, we explain the intuition and demonstrate that protocol π_{PM} correctly determines which substrings of the text t match the pattern p. Recall that the value P that is computed in Eq. (1) (Step 2) is an encryption of Bob's pattern, $p = \sum_{i=1}^{m} 2^{i-1} p_i$. This follows from the homomorphic property of ElGamal encryption,
$$P = \prod_{i=1}^{m} P_i^{2^{i-1}} = E_{pk}\left(\sum_{i=1}^{m} 2^{i-1} p_i, \sum_{i=1}^{m} 2^{i-1} r_{p_i}\right). \tag{4}$$
Note that P is obtained deterministically from the P_i, hence both Alice and Bob hold the same fixed encryption. Similarly, in Eq. (2) computed in Step 4, the parties compute encryptions of the substrings of length m of Alice's text,
$$\bar{t}_j = \sum_{i=j}^{j+m-1} 2^{i-j} t_i,$$
see a detailed discussion in the complexity paragraph regarding the efficiency of this step. As with P, the parties hold the same, fixed encryptions (with randomness $r_{\bar{t}_j} = \sum_{i=j}^{j+m-1} 2^{i-j} r_{t_i}$). The encryption Δ_j computed by Eq. (3) is an encryption of $\delta_j = \bar{t}_j - p$, i.e. the ($\mathbb{Z}_q$) difference between the substring of the text starting at position j and the pattern.
$$\begin{aligned} \Delta_j &= \bar{T}_j \cdot P^{-1} \\ &= E_{pk}\left(\bar{t}_j - p, r_{\bar{t}_j} - r_p\right) \end{aligned}$$
At this point, it simply remains for Bob to securely determine which of the Δ_j are encryptions of zero, as
$$\delta_j = 0 \Leftrightarrow \bar{t}_j = p.$$

SECURITY OF π_{PM}. We are now ready to prove the following theorem,

Theorem 1 (linear pattern matching): *Assume that π_{KeyGen}, π_{dec0} and π_{isBit} are as described in Section 2 and that (G, E, D) is the ElGamal scheme. Then π_{PM} securely computes $\mathcal{F}_{\text{PM}}$ in the presence of malicious adversaries.*

Proof. We separately prove security in the case that Alice is corrupted and the case that Bob is corrupted. Our proof is in a hybrid model where a trusted party computes the ideal functionalities $\mathcal{F}_{\text{KeyGen}}$, $\mathcal{F}_{\text{dec0}}$ and $\mathcal{F}_{\text{ZK}}^{\mathcal{R}_{\text{isBit}}}$.

Bob is corrupted. Let $\mathcal{A}$ denote an adversary controlling Bob. In this case we need to prove that Bob does not learn anything but the matching text locations. We construct a simulator $\mathcal{S}$ as follows,

1. $\mathcal{S}$ is given a pattern p of length m, an integer n and $\mathcal{A}$'s auxiliary input and invokes $\mathcal{A}$ on these values.
2. $\mathcal{S}$ emulates the trusted party for π_{KeyGen} as follows. It first chooses two random elements $s_A, s_B \in \mathbb{G}_q$ and hands $\mathcal{A}$, its share s_B and the public key $\langle \mathbb{G}_q, q, g, h = g^{s_A \cdot s_B} \rangle$.
3. $\mathcal{S}$ receives from $\mathcal{A}$, m encryptions and $\mathcal{A}$'s input for the trusted party for $\mathcal{F}_{\text{ZK}}^{\mathcal{R}_{\text{isBit}}}$. If the conditions for which the functionality outputs 1 are not met, $\mathcal{S}$ aborts by sending $\perp$ to the trusted party for $\mathcal{F}_{\text{PM}}$ and outputs whatever $\mathcal{A}$ outputs.
4. Otherwise, $\mathcal{S}$ defines P according to the witness for π_{isBit} and sends it to its trusted party. Let Z be the set of returned indices.
5. $\mathcal{S}$ defines a text t' that is consistent with Z. That is, for every $j \in Z$, $\mathcal{S}$ defines the substring $t'_j = p_1, \ldots, t'_{j+m-1} = p_m$. For the remaining indices $\mathcal{S}$ uses the bit one. ($\mathcal{S}$ verifies that the only matches in t' indeed correspond to the indices from set Z). $\mathcal{S}$ completes the execution as the honest Alice would on input t'.
6. If at any point $\mathcal{A}$ sends an invalid message $\mathcal{S}$ aborts, sending $\perp$ to the trusted party for $\mathcal{F}_{\text{PM}}$. Otherwise, it outputs whatever $\mathcal{A}$ does.

It is immediate to see that $\mathcal{S}$ runs in probabilistic polynomial time. We prove next that the adversary's views are computational indistinguishable via a reduction to the security of ElGamal. Recalling that the only difference within these views is with respect to the text locations that do not match the pattern, (as $\mathcal{S}$ uses the bit one instead of the actual bit value from t) we reduce the ability to distinguish these views to the ability to distinguish the encryptions of the real text against the simulated one for these locations.

Assume there exists a distinguisher D for these executions, we construct a distinguisher D_E breaking the semantic security of ElGamal encryption as follows. Upon receiving a public key pk and auxiliary input t, D_E engages in an execution of π_{KeyGen} with $\mathcal{A}$ and sends it (s_B, pk) where $s_B \in_R \mathbb{Z}_q$. D_E continues emulating the role of Alice as $\mathcal{S}$ does except for Step 3 where it needs to send the encryptions of $t_1, \ldots, t_n$. In this step D_E outputs two sets of plaintexts: (i) $t_1, \ldots, t_n$ and, (ii) $t'_1, \ldots, t'_n$. We denote by $\tilde{T}_1, \ldots, \tilde{T}_n$ the set of encryptions it receives back. D_E hands $\mathcal{A}$ this set and completes the run as follows. In Step 6 D_E replaces Δ_j with an encryption of zero if and only if $j \in Z$. Otherwise, D_E sends an encryption of a random value in $\mathbb{Z}_q^*$. Clearly, this step is computed differently than in both the hybrid and simulated executions. Nevertheless, we claim that the distributions on the encryptions are identical. This is due to the fact that for every matched text location the masking result equals zero, and for every non-matching text location the masking result equals a random element of $\mathbb{Z}_q^*$. Hence, the adversary's views are identical.

Finally, D_E invokes D on $\mathcal{A}$'s output and outputs whatever D outputs. Note that if D_E is given the encryptions of t then the adversary's view is distributed

as in the hybrid execution. Moreover, if it receives an encryption of t', then the adversary's view is as in the simulation with $\mathcal{S}$.

Alice is corrupted. Since Alice does not receive any output from the execution, we only need to prove that privacy is preserved, and that Bob's output cannot be affected (except with negligible probability). The proof follows the outlines of the former case. Therefore, due to space considerations we omit the details here.

COMPLEXITY OF π_{PM}. The round complexity is constant, as the key generation process and the zero knowledge proofs run in constant rounds. Further, the number of group elements exchanged is bounded by $O(n+m)$, as there are $n-m+1$ substrings of length m and each zero-knowledge proof requires constant number of group elements.

Regarding computational complexity, it is clear that except for Step 4 at most $O(m+n)$ exponentiations are required. Note first that Eq. (2) can be implemented using the *square and multiply* technique. Namely, for every $j = 1,\ldots,n-m+1$, $\bar{T}_j$ is computed by $(\ldots((T_j)^2 \cdot T_{j+1})^2 \cdot T_{j+2}\ldots)^2 \cdot T_{j+m-1}$.

This requires $O(m)$ multiplications for each text location, which amounts to total $O(nm)$ multiplications for the entire text. Reducing the number of multiplications into $O(n)$ (on the expense of increasing the number of exponentiations) can be easily shown. Loosely speaking, in addition to sending an encryption of 0 or 1 for each text location, Alice sends an encryption of 0 or 2^m, respectively, and proves consistency. From practical point of view, it may be much more efficient to compute $O(m)$ multiplications for each location, than proving this consistency (even though it only requires a constant number of exponentiations.)

Finally, note that our protocols utilize ElGamal encryption which can be implemented over an elliptic curve group. This may reduce the modulus value dramatically, as now only 160 bits are typically needed for the size of the key.

3.1 Variations

NON-BINARY ALPHABETS Alphabets of larger size, s, can be handled by encoding the characters as elements of $\mathbb{Z}_s$ and using s-ary rather than binary notation for the $\bar{T}_j$ and P. Proving in ZK that an encryption contains a valid character is straightforward, e.g. it can be provided in binary (which of course requires $O(\log s)$ encryptions).

LONG PATTERNS When the pattern length, m, (or the alphabet size, s) is large, requiring $q > s^m$ may not be acceptable. This can be avoided by encoding the pattern p and substrings $\bar{t}_j$ into multiple $\mathbb{Z}_q$ values, $\{p^{(i)}\}_i, \{\bar{t}_j^{(i)}\}_i$. Having computed encryptions $\{\Delta_i\}_i$ of the differences $\{\delta_i = p^{(i)} - \bar{t}_j^{(i)}\}_i$, Alice raises each encryption to a random, non-zero exponents r_i, rerandomizes them and sends them to Bob (and proves that everything was done correctly). The parties then executes π_{dec0} on the product of these encryptions and Bob reports a match

if a 0 is found. Note that the plaintext of this product is $\sum_i r_i \cdot \delta_i$. Thus, if the pattern matches, all $\delta_i = 0$ implying that this is an encryption of 0. If one or more $\delta_i \neq 0$, then the probability of this being an encryption of 0 is negligible.

HIDING MATCH LOCATIONS It may be required that Bob only learns the number of matches and not the actual locations of the hits. One example is determining *how frequently* some gene occurs rather than *where* it occurs in some DNA sequence. This is easily achieved by simply having Alice pick a uniformly random permutation and permute (and rerandomize) the Δ_j of Eq. (3). The encryptions are sent to Bob, and π_{perm} is executed, allowing him to verify Alice's behavior. Finally, π_{dec0} is run and Bob outputs the number of encryptions of 0 received.

Correctness is immediate: An encryption of 0 still signals that a match occurred. However, due to the random permutation that Alice applies, the locations are shuffled, implying that Bob does not learn the actual matches.

4 Secure Pattern Matching with Wildcards

The first variant of the classical pattern matching problem allows Bob to place wildcards, denoted by $\star$, in his pattern; these should match both 0 and 1. More formally, the parties wish the compute the functionality $\mathcal{F}_{\text{PM}-\star}$ defined by,

$$((p,n),(t,m)) \mapsto \begin{cases} (\{j \mid \bar{t}_j \overset{\star}{\equiv} p\}_{j=1}^{n-m+1}, \lambda) & \text{if } |p| = m \text{ and } |t| = n \\ (\lambda, \lambda) & \text{otherwise} \end{cases}$$

where $\bar{t}_j$ is the substring of length m that begins at the jth position of t and $\overset{\star}{\equiv}$ is defined as "equal except with respect to $\star$-positions." This problem has been widely looked at by researchers with the aim to generalize the basic searching model to searching with errors. This variant is known as *pattern matching with don't cares* and can be solved in $O(n+m)$ time [IR07]. The secure version of this problem guarantees that Alice will not be able to trace the locations of the don't cares in addition to the security requirement introduced for the basic problem.

The core idea of the solution is to proceed as in the standard one with two exceptions: Bob must supply the wildcard positions in encrypted form, and the substrings of Alice's text must be modified to ensure that they will match (i.e. equal) the pattern at those positions. Achieving correctness and ensuring correct behavior requires substantial modification of the protocol. Intuitively, for every m-bit substring $\bar{t}_j$ of t, Bob replaces Alice's value by 0 at the wildcard positions resulting in a string $\bar{t}'_j$, see Step 6 below. Similarly, a pattern p' is obtained from p by replacing the wildcards by 0. Clearly this ensures that the bits of $\bar{t}'_j$ and p' are equal at all wildcard positions. Thus, $\bar{t}'_j = p'$ precisely when $\bar{t}_j$ equals p at all non-wildcard positions.

Protocol $\pi_{\text{PM}-\star}$

- **Inputs:** The input of *Alice* is a binary string t of length n and an integer m, whereas the input of *Bob* is a string p over the alphabet $\{0,1,\star\}$ of length m and an integer n. The parties share a security parameter 1^κ as well.

– **The protocol:**

1. Alice and Bob run protocol $\pi_{\text{KeyGen}}(1^\kappa, 1^\kappa)$ to generate a public key $pk = \langle \mathbb{G}_q, q, g, h \rangle$, and the respective shares s_A and s_B of the secret key sk.
2. For each position $i = 1, \ldots, m$, Bob first replaces $\star$ by 0

$$p'_i \leftarrow \begin{cases} 1 & \text{if } p_i = 1 \\ 0 & \text{otherwise} \end{cases}.$$

He then sends encryptions $P'_i = E_{pk}(p'_i; r_{p'_i})$ for $i = 1, \ldots, m$ to Alice, and for each one they execute π_{isBit}. Finally, both parties compute an encryption of Bob's "pattern" in binary,

$$P' \leftarrow \prod_{i=1}^{m} {P'_i}^{2^{i-1}}.$$

3. For each position $i = 1, \ldots, m$ of Bob's pattern, he computes a bit denoting the occurrences of a $\star$,

$$w_i \leftarrow \begin{cases} 0 & \text{if } p_i = \star \\ 1 & \text{otherwise} \end{cases}.$$

He then encrypts these and sends the result to Alice,

$$W_i \leftarrow E_{pk}\left(w_i, r_{w_i}\right),$$

and the two run π_{isBit} for each one.
4. For each $i = 1, \ldots, m$, Bob and Alice run π_{isBit} on W_i / P'_i. This demonstrates to Alice that if p'_i is set, then so is w_i, i.e. that only 0's occur at wildcard position.
5. Alice supplies her input as in Step 3 of Protocol π_{PM} in Section 3. She sends encryptions, $T_j = E_{pk}(t_j; r_{t_j})$ $j = 1, \ldots, n$, of the bits of t to Bob. Then the parties run π_{isBit} for each of the encryptions.
6. For every entry $i = 1, \ldots, m$ of every m-bit substring of t starting at position $j = 1, \ldots, n - m + 1$, Bob computes an encryption

$$\hat{T}_{j,i} \leftarrow \left(T_{j+i-1}\right)^{w_i} \cdot E_{pk}\left(0, r_{j,i}\right).$$

He sends these to Alice, and they run π_{mult} on each triple $\left(T_{j+i-1}, W_i, \hat{T}_{j,i}\right)$, allowing Alice to verify that Bob has correctly multiplied the plaintexts of the W_i onto the T_{j+i-1}. Both parties then compute encryptions of the modified substrings of Alice's text

$$\bar{T}'_j \leftarrow \prod_{i=1}^{m} \left(\hat{T}_{j,i}\right)^{2^{i-1}}.$$

7. The protocol concludes as Protocol π_{PM} does. For each of the $\bar{T}'_j$ where $j = 1, \ldots, n - m + 1$, the parties compute

$$\Delta_j \leftarrow \bar{T}'_j \cdot {P'}^{-1},$$

and run π_{dec0}. This reveals to Bob which of plaintexts δ_j are 0. For each $\delta_j = 0$ he concludes that the pattern matched and outputs j.

To see that the protocol does not introduce new opportunities for malicious behavior, first note that Alice specification is essentially as in the basic protocol π_{PM}. Regarding Bob, the proofs of correct behavior limit him to supplying an input that an honest Bob could have supplied as well. Bob's input, p'_i $i = 1, \ldots, m$, is first shown to be a bit string, Step 2. The invocations of π_{isBit} of Step 3 then ensure that so is the "wildcard string." Finally, in Step 4 it is verified that for each wildcard p_i of p, $p'_i = 0$. In other words, there is a valid input where the honest Bob would send encryptions of the values that the malicious Bob can use. The only remaining option for a malicious Bob is in Step 6, however, the invocations of π_{mult} ensure his correct behavior. Formal simulation is analogous to that in Section 3. We state the following theorem:

Theorem 2 (wildcards): *Assume that π_{KeyGen}, π_{dec0}, π_{isBit}, and π_{mult} are as described in Section 2 and that (G, E, D) is the ElGamal scheme. Then $\pi_{\text{PM}-\star}$ securely computes $\mathcal{F}_{\text{PM}-\star}$ in the presence of malicious adversaries.*

Regarding complexity, clearly the most costly part of the protocol is Step 6 which requires Bob to sends $\Theta(nm)$ encryptions, $\hat{T}_{j,i}$ to Alice, as well as an invocation of π_{mult} for each of them. Hence, communication and computation complexity is increased to $O(nm)$, while round complexity remains constant.

5 Secure Approximate Matching

The second variation considered is approximate pattern matching: Alice holds an n-bit string t, while Bob holds an m-bit pattern p. The parties wish to determine approximate matches – strings with Hamming distance less than some threshold $\tau \leq m$. This is captured by the functionality $\mathcal{F}_{\text{APM}}$ defined by,

$$((p, n, \tau), (t, m, \tau')) \mapsto \begin{cases} (\{j \mid \delta_H(\bar{t}_j, p) < \tau\}_{j=1}^{n-m+1}, \lambda) & \text{if } |p| = m \geq \tau = \tau' \\ & \text{and } |t| = n \\ (\lambda, \lambda) & \text{otherwise} \end{cases}$$

where δ_H denotes Hamming distance and $\bar{t}_j$ is the substring of length m that begins at the jth position in t. We assume that the parties share some threshold $\tau \in \mathbb{N}$. Note that this problem is an extension of pattern matching with don't cares problem introduced in Section 4. Bob is able to learn *all* the matches within some error bound instead of learning the matches for specified error locations.

Two of the most important applications of approximate pattern matching are spell checking and matching DNA sequences. The most recent algorithm for solving this problem without considering privacy is by Amir et al. [ALP00] which introduced a solution in time $O(n\sqrt{\tau \log \tau})$. Our solution achieves $O(nm)$ computation and communication complexity.

The main idea behind the construction is to have the parties securely supply their inputs in binary as above. Then, to determine the matches, the parties first compute the (encrypted) Hamming distances h_j using the homomorphic properties of ElGamal encryption (Steps 5 and 6). They then check whether

$h_j = k$ for each $k < \tau$. To avoid leaking information, these results are permuted before the final decryption.

Protocol $\pi_{\mathbf{APM}}$

- **Inputs:** The input of *Alice* is a binary string t of length n, an integer m and a threshold τ', whereas the input of *Bob* is a binary string p of length m, an integer n and a threshold τ. The parties share a security parameter 1^κ as well.
- **The protocol:**
  1. Alice and Bob run protocol $\pi_{\text{KeyGen}}(1^\kappa, 1^\kappa)$ to generate a public key $pk = \langle \mathbb{G}_q, q, g, h \rangle$, and the respective shares s_A and s_B of the secret key sk.
  2. Alice sends Bob τ' and the parties continue if $\tau = \tau'$.
  3. As in the basic solution, Bob first sends encryptions $P_i = E_{pk}(p_i; r_{p_i})$ $i = 1, \ldots, m$, of the bits of his m-bit pattern, p, to Alice. They then run π_{isBit} for each one.
  4. Alice similarly provides encryptions, $T_j = E_{pk}(t_j; r_{t_j})$ $j = 1, \ldots, n$ of her input as in π_{PM}; for each one the parties execute π_{isBit}.
  5. For every entry $i = 1, \ldots, m$ of every m-bit substring of t starting at position $j = 1, \ldots, n - m + 1$, Bob computes an encryption
  $$\Pi_{j,i} \leftarrow T_{j+i-1}^{p_i} \cdot E_{pk}\left(0, r_{j,i}\right). \tag{5}$$
  He sends these to Alice, and for each triple $(T_{j+i-1}, P_i, \Pi_{j,i})$ the parties run π_{mult}. This allows Alice to verify that Bob has correctly multiplied the plaintexts of the P_i onto the T_{j+i-1}.
  6. For every entry $i = 1, \ldots, m$ of every m-bit substring of t starting at position $j = 1, \ldots, n - m + 1$, both parties compute encryptions $X_{j,i}$,
  $$X_{j,i} \leftarrow T_{j+i-1} \cdot P_i \cdot \Pi_{j,i}^{-2}.$$
  Note that as the plaintext of $\Pi_{j,i}$ is $p_i \cdot t_{j+i-1}$, the plaintext of $X_{j,i}$ is $p_i \oplus t_{j+i-1}$. For every $j = 1, \ldots, n - m + 1$ – i.e. for every substring – both parties compute
  $$H_j \leftarrow \prod_{i=1}^{m} X_{j,i}.$$
  7. For every $k = 0, \ldots, \tau - 1$ (i.e. for every Hamming distance which would be considered a match) and for every substring of length m starting at $j = 1, \ldots, n - m + 1$, both parties compute
  $$\Delta_{j,k} \leftarrow H_j \cdot \left\langle 1, g^{-k} \right\rangle. \tag{6}$$
  8. For every $j = 1, \ldots, n - m + 1$, Alice picks a uniformly random permutation $\pi_j : \mathbb{Z}_\tau \to \mathbb{Z}_\tau$ and applies π_j to the set $\{\Delta_{j,k}\}_k$,
  $$\left(\Delta'_{j,0}, \ldots, \Delta'_{j,\tau-1}\right) \leftarrow \pi_j\left(\Delta_{j,0}, \ldots, \Delta_{j,\tau-1}\right),$$
  rerandomizes all encryptions,
  $$\Delta''_{j,k} \leftarrow \Delta'_{j,k} \cdot E_{pk}\left(0, r'_{j,k}\right)$$
  for $j = 1, \ldots, n - m + 1$ and $k = 0, \ldots, \tau - 1$, and sends the $\Delta''_{j,k}$ to Bob. For every permutation, $j = 1, \ldots, n - m + 1$, the parties execute π_{perm} on $\left(\left(\Delta_{j,0}, \ldots, \Delta_{j,\tau-1}\right), \left(\Delta''_{j,0}, \ldots, \Delta''_{j,\tau-1}\right)\right)$ allowing Bob to verify that the plaintexts of the $\Delta''_{j,k}$ correspond to those of the $\Delta_{j,k}$ for all (fixed) j.

9. Finally, Alice and Bob execute π_{dec0} on each $\Delta''_{j,k}$ for $j = 1, \ldots, n-m+1$ and $k = 0, \ldots, \tau - 1$. This reveals to Bob which plaintexts $\delta_{j,k}$ are 0. He then outputs j iff this is the case for one of $\delta''_{j,0}, \ldots, \delta''_{j,\tau-1}$.

Correctness follows from the intuition: The plaintexts of the H_j from Equation (5) are the sum of the ones of the $X_{j,i}$ $i = 1, \ldots, m$. I.e. it is the number of differing bits of p and $\bar{t}_j$ – the Hamming distance – as the plaintext of $X_{j,i}$ is $t_{j+i-1} + p_i - 2 \cdot t_{j+i-1} \cdot p_i = t_{j+i-1} \oplus p_i$.

Each threshold test is performed using τ tests of equality, one for each possible value $k < \tau$, where each test simply subtracts the associated k from H_j under the encryption, Eq. (6), at which point the parties may mask and decrypt towards Bob. Note that the standard masking combined with the permutation of Step 8 ensures that for every potential match, Bob either receives τ uniformly random encryptions of random, non-zero values, or $\tau - 1$ such encryptions and a single encryption of zero. Hence we state the following theorem:

Theorem 3 (approximate): *Assume that π_{KeyGen}, π_{dec0} and π_{isBit}, and π_{mult} are as described in Section 2 and that (G, E, D) is the ElGamal scheme. Then π_{APM} securely computes $\mathcal{F}_{\text{APM}}$ in the presence of malicious adversaries.*

Regarding complexity, the most expensive steps are those associated with computing the Hamming distances, Steps 5 and 6, as there are $\Theta(nm)$ $\Pi_{j,i}$ and $X_{j,i}$. The concluding steps – computing, randomizing (permuting), and decrypting the $\Delta_{j,k}$ – require $\Theta(n\tau)$ work, however, as $\tau \leq m$ this is no more expensive. Hence overall communication and computation is $O(mn)$, while round complexity is constant as in the previous solutions.

6 Hiding the Pattern Length

Here Alice is not required to know the length m of Bob's pattern, only an upper bound $M \geq m$. Moreover, she will not learn any information about m. More formally, the parties wish to compute the functionality $\mathcal{F}_{\text{PM-hpl}}$ defined by,

$$((p, n), (t, M)) \mapsto \begin{cases} (\{j \mid \bar{t}_j = p\}_{j=1}^{n-m+1}, \lambda) & \text{if } |p| \leq M \text{ and } |t| = n \\ (\lambda, \lambda) & \text{otherwise} \end{cases}$$

where $\bar{t}_j$ is the substring of length m that begins at the jth position in t. A protocol $\pi_{\text{PM-hpl}}$ that realizes $\mathcal{F}_{\text{PM-hpl}}$ can be obtained through minor alterations of $\pi_{\text{PM-}\star}$. Due to space constraints we only sketch these, and postpone the detailed description and simulator proof to the full version of the paper.

The main idea is to have Bob construct a pattern $\hat{p}$ of length M by padding p with $M - m$ wildcards. Though not completely correct, intuitively, executing $\pi_{\text{PM-}\star}$ on input $((\hat{p}, n), (t, M))$ provides the desired result, as the wildcards ensure that the irrelevant postfixes of the $\bar{t}_j$ are "ignored." There are two reasons why this does not suffice. Firstly, the wildcards of $\pi_{\text{PM-}\star}$ mean *match any character*, however, matches must also be found when the wildcards occur *after* the

end of the text (where there are no characters). Secondly, a malicious Bob must not have full access to wildcard-usage – i.e. he must not be able to arbitrarily place wildcards, they must occur only at the end of $\hat{p}$.

– **Matching $\bar{t}_j$ when $j > n - M + 1$:** The solution to the former problem is completely straightforward: extend (pad) t with symbols that only match wildcards. Going into more detail, first let $N = n + M - 1$. The parties pad Alice's encrypted text, $T_1, \ldots, T_n$ with $M - 1$ default encryptions of 2,

$$T_{n+1} = \ldots = T_N = \langle 1, g^2 \rangle .$$

Then, rather than use a binary representation for the encryptions P' and $\bar{T}'_j$ (Steps 2 and 6 of $\pi_{\text{PM}-\star}$), we use a ternary representation

$$\hat{P} \leftarrow \prod_{i=1}^{M} (P'_i)^{3^{i-1}}, \qquad \bar{T}'_j \leftarrow \prod_{i=j}^{j+M-1} \hat{T}_{j,i}^{3^{i-j}}.$$

Intuitively, this works as we have simply extended our alphabet with an additional character, 2.

– **Ensuring a proper $\hat{p}$:** To prevent malicious behavior, Bob should demonstrate to Alice that $\hat{p}$ has been properly constructed, i.e. that all wildcards occur at the end of the pattern. This can be done by showing that $w_1, \ldots, w_M$ is monotonically non-increasing, i.e. that a 1 (non-wildcard) never follows a 0 (wildcard). Bob can demonstrate this fact by executing π_{isBit} on W_i/W_{i+1} for $i = 1, \ldots, M - 1$.

Complexity is equivalent to $\pi_{\text{PM}-\star}$. We conclude with the following theorem,

Theorem 4 (pattern length hiding): *Assume that π_{KeyGen}, π_{Dec}, π_{isBit}, and π_{mult} are as described in Section 2 and that (G, E, D) is the ElGamal scheme. Then $\pi_{\text{PM}-\text{hpl}}$ securely computes $\mathcal{F}_{\text{PM}-\text{hpl}}$ in the presence of malicious adversaries.*

7 Hiding the Text Length

The final variant does not require Bob to know the actual text length n, only an upper bound $N \geq n$. Moreover, he learns no information about n other than what can be inferred from the output. This property is desirable in applications where it is crucial to hide the size of the database as it gives away sensitive information. More formally, the parties wish the compute the functionality $\mathcal{F}_{\text{PM}-\text{htl}}$,

$$((p, N), (t, m)) \mapsto \begin{cases} (\{j \mid \bar{t}_j = p\}_{j=1}^{n-m+1}, \lambda) & \text{if } |p| = m \text{ and } |t| \leq N \\ (\lambda, \lambda) & \text{otherwise} \end{cases}$$

where $\bar{t}_j$ is the substring of length m that begins at the jth position in t.

Due to space constraints, we only sketch the solution. The core idea is to have Alice pad her text with $N - n$ 2s, and then demonstrate that any 2s occur at the end. The details of the solution are similar to those of $\pi_{\text{PM}-\text{hpl}}$ above.

Regarding complexity, it can be shown that only $O(N+m)$ encryptions change hands, hence only this many zero-knowledge proofs of knowledge are needed as well; i.e. communication and computation complexity are linear. The required number of rounds is constant.

Theorem 5 (text length hiding): *Assume that* π_{KeyGen}, π_{dec0} *and* π_{isBit} *are as described in Section 2 and that* (G, E, D) *is the ElGamal scheme. Then* $\pi_{\text{PM}-\text{htl}}$ *securely computes* $\mathcal{F}_{\text{PM}-\text{htl}}$ *in the presence of malicious adversaries.*

References

[ACF02] Masayuki Abe, Ronald Cramer, and Serge Fehr. Non-interactive distributed-verifier proofs and proving relations among commitments. In *ASIACRYPT*, pages 206–223, 2002.

[ACR99] 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.

[ALP00] Amihood Amir, Moshe Lewenstein, and Ely Porat. Faster algorithms for string matching with mismatches. In *SODA*, pages 794–803, San Francisco, California, USA, 2000.

[AMP04] Gagan Aggarwal, Nina Mishra, and Benny Pinkas. Secure computation of the k'th-ranked element. In *EUROCRYPT*, pages 40–55, 2004.

[Bea92] Donald Beaver. Foundations of secure interactive computing. In *CRYPTO*, pages 377–391, London, UK, 1992. Springer-Verlag.

[Blo70] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, 1970.

[BM77] Robert S. Boyer and J. Strother Moore. A fast string searching algorithm. *Commun. ACM*, 20(10):762–772, 1977.

[Bra05] Felix Brandt. Efficient cryptographic protocol design based on distributed el gamal encryption. In *ICISC*, pages 32–47, 2005.

[Can00] Ran Canetti. Security and composition of multi-party cryptographic protocols. *Journal of Cryptology*, 13:143–202, 2000.

[CGS97] Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers. A secure and optimally efficient multi-authority election scheme. In *EUROCRYPT*, pages 103–118, 1997.

[CP93] 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, 1993. Springer-Verlag.

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, November 1976.

[ElG85] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.

[FNP04] Michael J. Freedman, Kobbi Nissim, and Benny Pinkas. Efficient private matching and set-intersection. In *EUROCRYPT*, pages 1–19, 2004.

[GHS10] Rosario Gennaro, Carmit Hazay, and Jeffrey S. Sorensen. Automata evaluation and text search protocols with simulation based security. In *Public Key Cryptography*, pages 145–160, 2010.

[GL91] Shafi Goldwasser and Leonid A. Levin. Fair computation of general functions in presence of immoral majority. In *CRYPTO*, pages 77–93, London, UK, 1991. Springer-Verlag.

[GMW87] 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.

[Gol04] Oded Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge University Press, New York, NY, USA, 2004.

[Gro03] Jens Groth. A verifiable secret shuffle of homomorphic encryptions. In *Public Key Cryptography*, pages 145–160, 2003.

[HL08] Carmit Hazay and Yehuda Lindell. Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries. In *TCC*, pages 155–175, 2008.

[IR07] Costas S. Iliopoulos and M. Sohel Rahman. Pattern matching algorithms with don't cares. In *SOFSEM*, pages 116–126, 2007.

[JP09] Ayman Jarrous and Benny Pinkas. Secure hamming distance based computation and its applications. In *ANCS*, volume 5536, pages 107–124, 2009.

[KM10] Jonathan Katz and Lior Malka. Secure text processing with applications to private dna matching. In *To appear CCS*, 2010.

[KMP77] Donald E. Knuth, James H. Jr. Morris, and Vaughan R. Pratt. Fast pattern matching in strings. *SIAM J. Comput.*, 6(2):323–350, 1977.

[KS05] Lea Kissner and Dawn Xiaodong Song. Privacy-preserving set operations. In *CRYPTO*, pages 241–257, 2005.

[LP02] Yehuda Lindell and Benny Pinkas. Privacy preserving data mining. *Journal of Cryptology*, 15(3):177–206, 2002.

[LP07] 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.

[MR91] Silvio Micali and Phillip Rogaway. Secure computation (abstract). In *CRYPTO*, pages 392–404, 1991. This is preliminary version of unpublished 1992 manuscript.

[NM07] Gonzalo Navarro and Veli Mäkinen. Compressed full-text indexes. *ACM Comput. Surv.*, 39(1):2, 2007.

[Ped91] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO*, pages 129–140, 1991.

[Sch89] 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.

[TPKC07] 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.

[Yao86] 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.