J. Cryptol. (2014) 27: 358–395
DOI: 10.1007/s00145-013-9147-8

Journal of
CRYPTOLOGY

Computationally Secure Pattern Matching in the Presence of Malicious Adversaries

Carmit Hazay
Department of Engineering, Bar-Ilan University, Ramat-Gan, Israel
carmit.hazay@biu.ac.il

Tomas Toft
Department of Computer Science, Aarhus University, Aarhus, Denmark
ttoft@cs.au.dk

Communicated by Jonathan Katz.

Received 3 July 2012
Online publication 14 March 2013

Abstract. We propose a protocol for the problem of secure two-party pattern matching, where 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 (only) learn where his pattern occurs in Alice's text, while Alice learns nothing. Private pattern matching is an important problem that has many applications in the area of DNA search, computational biology and more. Our construction guarantees full simulation in the presence of malicious, polynomial-time adversaries (assuming the hardness of DDH assumption) and exhibits computation and communication costs of $O(n+m)$ group elements in a constant round complexity. This improves over previous work by Gennaro et al. (Public Key Cryptography, pp. 145–160, 2010) whose solution requires overhead of $O(nm)$ group elements and exponentiations in $O(m)$ rounds. In addition to the above, we propose a collection of protocols for important variations of the secure pattern matching problem that are significantly more efficient than the current state of art solutions: First, we deal with secure pattern matching with wildcards. In this variant the pattern may contain wildcards that match both 0 and 1. Our protocol requires $O(n+m)$ communication and $O(1)$ rounds using $O(nm)$ computation. Then we treat secure approximate pattern matching. In this variant the matches may be approximated, i.e., have Hamming distance less than some threshold, τ. Our protocol requires $O(n\tau)$ communication in $O(1)$ rounds using $O(nm)$ computation. Third, we have secure pattern matching with hidden pattern length. Here, the length, m, of Bob's pattern remains a secret. Our protocol requires $O(n+M)$ communication in $O(1)$ rounds using $O(n+M)$ computation, where M is an upper bound on m. Finally, we have secure pattern matching with hidden text length. Finally, in this variant the length, n, of Alice's text remains a secret. Our protocol requires $O(N+m)$ communication in $O(1)$ rounds using $O(N+m)$ computation, where N is an upper bound on n.

Key words. Pattern matching, Secure two-party computation, Simulation-based security, 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 [7,12,21,34] 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). This way of defining security is very appealing and has many important advantages; for example, protocols proven secure in this way remain secure under sequential modular composition [12]. We call this definition *simulation-based security* because protocols are proven secure by simulating a real execution while running in the ideal model.

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 [22,23,46]. These results apply in various settings, considering semi-honest and malicious adversaries. In the semi-honest setting corrupted parties follow the protocol instructions (but still try to gain additional private information), whereas, malicious players follow an arbitrary strategy. 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 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, while the malicious setting typically remained impractical (a notable exception is [4]).

In this paper we consider the following fundamental search problem: Alice holds a text $t \in \{0,1\}^*$ of length n and Bob is given a pattern (i.e., a 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 only 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 comparing two DNA strings; the following example is taken from [20]. 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. The importance of this application is further illustrated in [5].

In an insecure setting this problem can be solved with linear time complexity. Nevertheless, most of the existing solutions do not attempt to achieve any level of security (if at all); see [2,9,10,31,35,36] 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. This paper is an extended version of [26]. The primary new contribution of this version is the design of special purpose zero-knowledge proofs that enable to reduce the communication complexity of our protocols for pattern matching with wildcards and approximate pattern matching discussed below.

1.1. *Our Contribution*

We present secure solutions for the following problems in the plain model under the DDH hardness assumption with simulation-based security in the presence of malicious adversaries. The security proofs of our protocols can be easily extended to the UC framework [13] as well. Our constructions achieve efficiency that is a significant improvement on the current state of the art; see a concrete analysis below. Throughout this paper we measure computation by the *number of exponentiations* and *the number of group multiplications* (by default we mean the former), and communication by the *number of exchanged group elements* within the protocol. In more details:

- 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 other important variants of pattern matching which are described next.
- 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 $\star$ (or a 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 an insecure setting in $O(n+m)$ time [28]. In this paper, we develop a protocol that computes this functionality with $O(n+m)$ communication and $O(nm)$ computation costs. The core idea of our solution is to proceed as in the above solution 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 the pattern at those positions. Ensuring correct behavior requires further modification of the protocol; see Sect. 4 for the complete description of the protocol.
- SECURE APPROXIMATE PATTERN MATCHING. In this problem the goal is for Bob to find the text locations where the Hamming distance of each text substring and the pattern is less than some threshold $\tau \leq m$. This problem is an extension of pattern matching with don't cares due to the fact that Bob is able to learn *all* the matches within some error bound instead of learning the matches for specified error locations. An important application of this problem is secure face recognition [38]. The best algorithm for solving this problem in an insecure setting is the solution by Amir et al. [3] which introduces a solution in $O(n\sqrt{\tau \log \tau})$ time. We design a protocol for this problem with $O(n\tau)$ communication and $O(nm)$ computation costs. The main idea behind our construction is to have the parties securely compute the (encrypted) Hamming distance for each text position. See Sect. 5 for further details.

- SECURE PATTERN MATCHING WITH HIDDEN PATTERN/TEXT LENGTH. Finally, we consider two variants with an additional security requirement of hiding the input lengths using padding of a special character. For public upper bounds on the lengths, $M \geq m$ and $N \geq n$, the solutions for these problems require $O(n + M)$ communication and exponentiations, and $O(N + m)$ communication and exponentiations, respectively.

  Note that in the semi-honest setting the length of the pattern can be remained hidden by letting Bob run all the computations locally and then engage with Alice in a comparison phase. Nevertheless, this task is particularly challenging in the malicious setting due to correctness issues, and it is not clear how to efficiently enhance the security of the semi-honest protocol without leaking anything about m from the communication. An efficient analogue solution for hiding the text length is not known—not even for the semi-honest setting. Therefore, using padding is currently the best alternative that enables to obtain some level of privacy regarding the pattern/text lengths, even if the padding must be large enough to hide these lengths.

  We point out to a recent work by Chase and Visconti that studies the feasibility of size-hiding (database) commitments [16], proposes a construction based on universal arguments. Although this construction is viewed as purely theoretical and precludes practical implementations it illustrates the difficulty in designing cryptographic primitives that hide the input length.

1.2. *Overview of Our Approach*

Our approach for computing private pattern matching follows by having the parties jointly (and securely) transform their inputs from binary representation into elements of $\mathbb{Z}_q$, which they can later compare. More explicitly, the parties break their inputs into bits and encrypt each bit separately. Next, they map every m consecutive encryptions of bits into a single encryption. That is, for every m encrypted bits $a_1, \ldots, a_m$ the parties compute the encryption of $\sum_{i=1}^{m} 2^{i-1} a_i$, relying on the additively homomorphism of the encryption scheme. Importantly, the parties exploit the fact that every two consecutive substrings $\bar{t}_i, \bar{t}_{i+1}$ of the text (starting in positions i and $i + 1$, respectively) overlap with $m - 1$ positions. Therefore, computing the encoding of $\bar{t}_{i+1}$ from the encoding of $\bar{t}_i$ can be obtained by subtracting from the latter the bit t_i, dividing the result by 2 and finally adding $2^{m-1} t_{i+m}$. This reduces the problem to comparing two elements of $\mathbb{Z}_{2^m}$ (embedded into $\mathbb{Z}_q$). Thus upon computing the encoding, the parties complete the protocol by comparing the encoding of p against every encoding of a substring of length m in the text, so that only Bob learns whether there is a much or not.

1.3. *Prior Work*

Secure Pattern Matching To the best of our knowledge, the first work that considered pattern matching in the context of secure computation was [43], which solves pattern matching using a secure version of oblivious automata evaluation for implementing the KMP algorithm [31] in the semi-honest setting. The KMP algorithm is a well known technique that requires $O(n)$ time complexity 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. The overall costs of [43] are $O(nm)$ exponentiations and bandwidth. Several followup improvements have been suggested based on this work, e.g., [19,33]. These works reduce the round complexity and the number of exponentiations but still maintain security in the semi-honest setting; we provide a comparison with these works below.

This problem was also studied by Hazay and Lindell in [24] which used a different approach of oblivious pseudorandom function (PRF) evaluation. Their protocol achieves only a weaker notion of security called one-sided simulatability that does not guarantee full simulation for both corruption cases. A more recent construction that achieves full simulation in the malicious setting was developed by Gennaro et al. [20]. This work implements the KMP algorithm in the malicious setting using $O(m)$ rounds and $O(nm)$ exponentiations and bandwidth.

Finally, a recent paper by Katz and Malka [30] presents a secure solution for a generalized pattern matching problem of text processing. Here the party that holds the pattern has some additional information y, and its goal is to learn a function of the text and y with respect to the text locations where the pattern matches. Katz and Malka show how to use 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 its length n). Their costs are dominated by the size of the circuit times the number of occurrences u (as u circuits are being transferred). They therefore need to assume some common knowledge of a threshold on the number of occurrences.

Variants of Pattern Matching To the best of our knowledge, the first work that addresses a variant of secure pattern matching is the work by Jarrous and Pinkas [29], which solves the Hamming distance problem for two equal length strings against semi-honest adversaries (which is relevant in the context of approximate pattern matching). The costs of their protocol are inflated by a statistical parameter s for running a subprotocol of the oblivious polynomial evaluation functionality. This implies $O(nm)$ exponentiations and groups elements.

Another work by Vergnaud [45] studies the problems of approximate pattern matching and pattern matching with wildcards in the presence of malicious adversaries by taking a different approach of Fast Fourier Transform (FFT). The paper implements the well known technique by Fischer and Paterson [18] in a distributed setting using convolutions and FFT, where the inputs are viewed as coefficients of two polynomials for which their product is computed using FFT (for each text alignment). The paper presents protocols that exhibit $O(n)$ communication and $O(n \log m)$ computational costs in the semi-honest and malicious settings (but does not provide a complete proof in the malicious setting).

Finally, a very recent paper Baron by et al. [6] studies the problem of pattern matching with wildcards in a more general sense of non-binary alphabet, implementing a different algorithm based on linear algebra formulation and additive homomorphic encryption. Their protocol requires $O(m + n)$ communication complexity and $O(nm)$ computational complexity.

Table 1. Comparisons with semi-honest constructions.

	Symmetric	Asymmetric	Communication
[46]	$O(nm)$	$O(m)$ $O(\kappa)$ using extended OT	$O(nm)$ symmetric encryptions $O(m)$ group elements
[19,33]	$O(nm)$	$O(n)$ $O(\kappa)$ using extended OT	$O(nm)$ symmetric encryptions $O(m)$ group elements
This work	$2n$	$8n$	$6n$ group elements

1.4. *Efficiency*

Secure Pattern Matching We measure the efficiency of our protocol by comparisons against generic secure two-party protocols, as well as protocols designed for this specific task. The most common technique for designing secure protocols in the two-party setting is Yao's garbling technique for Boolean circuits [46]. The current best known circuit that computes the pattern matching functionality requires $O(nm)$ gates, since the circuit compares the pattern against every text location. (As noted by [20], a circuit that implements the KMP algorithm requires $O(nm \log m)$ gates). It is an open problem whether better circuits can be constructed.

In the semi-honest setting, Yao's technique induces a protocol that uses $O(nm)$ symmetric key operations and $O(m)$ exponentiations that can be made independent of the input length (where the latter is obtained by employing the ideas of extended oblivious transfer (OT) [27], but also requires an additional assumption on the hash function). The works by [19,33] present specific protocols that require $O(nm)$ symmetric key operations (due to the automaton size) and $O(n)$ exponentiations, which can also be made independent of n using extended OT.

On the other hand, our protocol for the semi-honest setting requires $8n$ exponentiations and n group multiplications, where at first Alice forwards Bob the encryptions of her encoding for each substring of length m, Bob then computes the difference with his encoding and finally the parties rerandomize the outcome and decrypt it. A summary of these comparisons is presented in Table 1.

In the malicious setting, the state of the art generic implementation is a recent protocol by Lindell and Pinkas [32] that relies on the garbling technique of Yao. Due to enforcing correct behavior the overhead of their protocol is inflated by a statistical parameter $s = 132$. Therefore, the constants of such a protocol when realizing pattern matching are relatively high and dominated by $5.66sm + 39m + 10s + 6$ exponentiations and $6.5snm$ symmetric key operations. Moreover the communication complexity is at least $7sm + 22n + 7s + 5$ group elements and $4snm$ symmetric ciphertexts. For large databases this bandwidth as well as the number of symmetric operations introduce huge overheads. The only work that proves full simulation in the malicious setting was developed by Gennaro et al. [20]. This protocol runs in $O(m)$ rounds and requires $O(nm)$ exponentiations and bandwidth due to rerandomization of the automaton for each iteration. Thus, even asymptotically their protocols achieves worse overhead than our protocol (which also leads to higher constants since the [20] protocol uses zero-knowledge proofs in each step). On the other hand, our protocol induces $38n + 6m$

Table 2. Comparisons with malicious constructions.

	Symmetric	Asymmetric	Communication
[32]	$6.5snm$	$5.66sm + 39m + 10s + 6$	$7sm + 22n + 7s + 5$ group elements & $4snm$ symmetric ciphertexts
[20]	$O(nm)$	$O(nm)$	$O(nm)$ group elements
This Work	$nm + 19n + 10$	$38n + 6m$	$26n + 6m + 14$ group elements

exponentiations and $nm + 19n + 10$ group multiplications. The main advantage of our protocol is regarding the communication complexity. A summary of these comparisons is presented in Table 2.

Variants of Pattern Matching Generic protocols achieve the same overhead as in the case of computing the standard pattern matching problem since circuit size is $O(nm)$ gates. Moreover, the protocols by Vergnaud [45] compute approximate pattern matching and pattern matching with don't cares with better computational overhead than our protocols. The solution for the former problem introduces $O(n(\log m + \tau))$ computation (in comparison to $O(nm)$ exponentiations in our protocol). The solution for the latter problem introduces $O(n \log m)$ computational overhead (in comparison to $O(nm)$ exponentiations in our protocol). Finally, the work of [6] studies pattern matching with wildcards in the malicious setting and achieves similar costs to our protocols but for larger alphabets.

1.5. *A Roadmap*

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

2. Preliminaries and Tools

Throughout the paper, we denote the security parameter by κ. A probabilistic machine is said to run in *polynomial-time* (PPT) if it runs in time that is polynomial in the security parameter κ and its input. 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 $a \in \{0,1\}^*$

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

2.1. *Hardness Assumptions*

Our constructions rely on the following hardness assumption.

Definition 1 (DDH). We say that *the decisional Diffie–Hellman (DDH) problem is hard relative to the group* $\mathbb{G}_q$ if for all PPT $\mathcal{A}$ there exists a negligible function negl such that

$$\left|\Pr\left[\mathcal{A}\left(\mathbb{G}_q, q, g, g^x, g^y, g^z\right) = 1\right] - \Pr\left[\mathcal{A}\left(\mathbb{G}_q, q, g, g^x, g^y, g^{xy}\right) = 1\right]\right| \leq \mathsf{negl}(\kappa),$$

where $\mathbb{G}_q$ has order q and the probabilities are taken over the choices of g generating $\mathbb{G}_q$ and $x, y, z \in \mathbb{Z}_q$.

2.2. *Σ-Protocols*

Definition 2 (Σ-protocol). A protocol π is a Σ-protocol for relation R if it is a 3-round public-coin protocol and the following requirements hold:

- COMPLETENESS: If P and V follow the protocol on input x and private input w to P where $(x, w) \in R$, then V always accepts.
- SPECIAL SOUNDNESS: There exists a polynomial-time algorithm A that given any x and any pair of accepting transcripts (a, e, z), (a, e', z') on input x, where $e \neq e'$, outputs w such that $(x, w) \in R$.
- SPECIAL HONEST-VERIFIER ZERO KNOWLEDGE: There exists a PPT algorithm M such that

$$\left\{\left\langle P(x, w), V(x, e)\right\rangle\right\}_{(x,w)\in R, e\in\{0,1\}^*} \equiv \left\{M(x, e)\right\}_{x\in L_R, e\in\{0,1\}^*},$$

  where L_R is the language of relation R, $M(x, e)$ denotes the output of M upon input x and e, and $\langle P(x, w), V(x, e)\rangle$ denotes the output transcript of an execution between P and V, where P has input (x, w), V has input x, and V's random tape (determining its query) equals e.

2.3. *Public Key Encryption Schemes*

We begin by specifying the definitions of public key encryption, semantic security and homomorphic encryption.

Definition 3 (PKE). We say that $\Pi = (G, E, D)$ is a *public-key encryption scheme* if G, E, D are polynomial-time algorithms specified as follows:

- G, given a security parameter κ (in unary), outputs keys (pk, sk), where pk is a public key and sk is a secret key. We denote this by $(pk, sk) \leftarrow G(1^\kappa)$.
- E, given the public key pk and a plaintext message m, outputs a ciphertext c encrypting m. We denote this by $c \leftarrow E_{pk}(m)$; and when emphasizing the randomness r used for encryption, we denote this by $c \leftarrow E_{pk}(m; r)$.
- D, given the public key pk, secret key sk and a ciphertext c, outputs a plaintext message m s.t. there exists randomness r for which $c = E_{pk}(m; r)$ (or $\perp$ if no such message exists). We denote this by $m \leftarrow D_{pk,sk}(c)$.

For a public key encryption scheme $\Pi = (G, E, D)$ and a non-uniform adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, we consider the following *Semantic security game*:

$$(pk, sk) \leftarrow G\left(1^{\kappa}\right).$$
$$(m_0, m_1, history) \leftarrow \mathcal{A}_1(pk), \text{ s.t. } |m_0| = |m_1|.$$
$$c \leftarrow E_{pk}(m_b), \text{ where } b \leftarrow \{0, 1\}.$$
$$b' \leftarrow \mathcal{A}_2(c, history).$$
$$\mathcal{A} \text{ wins if } b' = b.$$

Denote by $\mathsf{Adv}_{\Pi,\mathcal{A}}(\kappa)$ the probability that $\mathcal{A}$ wins the semantic security game.

Definition 4 (Semantic security). A public key encryption scheme $\Pi = (G, E, D)$ is *semantically secure*, if for every non-uniform adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ there exists a negligible function negl such that

$$\mathsf{Adv}_{\Pi,\mathcal{A}}(\kappa) \leq \frac{1}{2} + \mathsf{negl}(\kappa).$$

An important tool that we exploit in our construction is *homomorphic* encryption over an additive group as defined below.

Definition 5 (Homomorphic PKE). A public key encryption scheme (G, E, D) is additively homomorphic if for all n and all (pk, sk) output by $G(1^{\kappa})$, it is possible to define groups $\mathbb{M}$, $\mathbb{C}$ such that

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

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

  where the group operations are carried out in $\mathbb{C}$ and $\mathbb{M}$, respectively, and the encryptions of m_1 and $m_1 + m_2$ use independent randomness.

Any additive homomorphic scheme supports the multiplication of a ciphertext by a scalar by computing multiple additions.

2.4. *The ElGamal PKE*

At the core of our proposed protocols lies the additively homomorphic variation of ElGamal PKE [17]. Essentially, we use the framework of Brandt [11] with minor variations. Formally, ElGamal PKE is a semantically secure public key encryption scheme assuming the hardness of the decisional Diffie–Helmann problem (DDH). We describe the plain scheme here; the distributed version is presented below. Let $\mathbb{G}_q$ be a group of prime order q in which DDH is hard (we assume that multiplication and testing group membership can be performed efficiently). Then the public key is a tuple

$pk = \langle \mathbb{G}_q, q, g, h \rangle$ and the corresponding secret key is $sk = s$, s.t. $g^s = h$. Encryption is performed by choosing $r \in_R \mathbb{Z}_q$ and computing $E_{pk}(m; r) = \langle g^r, h^r \cdot g^m \rangle$. Decryption of a ciphertext $C = \langle \alpha, \beta \rangle$ is performed by computing $g^m = \beta \cdot \alpha^{-s}$ and then finding m by running an exhaustive search. This variant of encrypting in the exponent suffices for our purposes as we do not require full decryption, but just the ability to distinguish between $m = 0$ and $m \neq 0$. Note that this variant of ElGamal meets Definition 5 for $\mathbb{M} = \mathbb{Z}_q$ and $\mathbb{C} = \mathbb{G}_q^2$. We present the computation of the parties with respect to the ciphertext space componentwise. Namely, we write C^r to denote $\langle \alpha^r, \beta^r \rangle$ and C/C' for $\langle \alpha/\alpha', \beta/\beta' \rangle$, for ciphertexts $C = \langle \alpha, \beta \rangle$ and $C' = \langle \alpha', \beta' \rangle$, and $r \in \mathbb{Z}_q$.

2.4.1. *Distributed ElGamal PKE*

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 decrypted plaintext. Note that a public key and an additive sharing of the corresponding secret key is easily generated [40]. Namely, 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 $h = h_1 h_2$ and set $pk = \langle \mathbb{G}_q, q, g, h \rangle$. Clearly, the secret key $s = s_1 + s_2$ associated with this public key is shared amongst the parties. In order to ensure correct behavior, the parties must prove knowledge of their s_i by running on (g, h_i) the zero-knowledge proof π_{DL}, specified in Sect. 2.5. We denote this key generation protocol by π_{KeyGen} which is correlated with the functionality $\mathcal{F}_{\mathrm{KeyGen}}(1^\kappa, 1^\kappa) = ((pk, sk_1), (pk, sk_2))$.

To decrypt a ciphertext $C = \langle \alpha, \beta \rangle$, each party P_i raises α to the power of its share, sends the outcome α_i to the other party and then proves this was done correctly using π_{DL}. Both parties then output $\beta/(\alpha_1 \alpha_2)$. We denote this protocol by π_{Dec}. This protocol allows a variation where only one party obtains the decrypted result. Another variation of π_{Dec} allows a party, say P_1, to learn whether a ciphertext $C = \langle \alpha, \beta \rangle$ encrypts g^0 or not, but nothing more. This can be carried out as follows. 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_{NZ}, defined below, 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 $\mathcal{F}_{\mathrm{ZK}}^{\mathcal{R}_{\mathrm{NZ}}}$. We denote this protocol by π_{Dec0} and the associated ideal functionality by $\mathcal{F}_{\mathrm{Dec0}}$.

2.5. *Zero-Knowledge Proofs for $\mathbb{G}_q$ and ElGamal PKE*

To prevent malicious behavior, the parties must demonstrate that they are well-behaved. To achieve this, our protocols utilize zero-knowledge proofs of knowledge. All our proofs are Σ-protocols which show knowledge of a witness that some statement is true (i.e., belong to some relation $\mathcal{R}$). A generic efficient technique that enables to transform any Σ-protocol into a zero-knowledge proof (of knowledge) can be found in [25]. This transformation requires additionally five (six) exponentiations.

2.5.1. *Zero-Knowledge Proofs with Constant Overhead*

1. π_{DL}, for demonstrating the knowledge of a solution x to a discrete logarithm problem [41].

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

2. π_{EqDL}, for demonstrating equality of two discrete logarithms [15].

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

   Phrased differently, π_{EqDL} demonstrates that a quadruple forms a Diffie–Hellman tuple or, equivalently, that an ElGamal ciphertext is an encryption of 0, where g_1, g_2 is part of the public key and $\langle h_1, h_2\rangle$ is the computed ciphertext; see Sect. 2.4 for the complete details of ElGamal.
3. π_{isBit}, for demonstrating that a ciphertext $C = \langle \alpha, \beta\rangle$ is either an encryption of 0 or 1. This can be obtained directly from π_{EqDL} using the compound proof of Cramer et al. [14].

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

4. π_{Mult}, for demonstrating that a ciphertext encrypts the product of two encrypted plaintexts [1]. Namely, given a ciphertext C the prover proves the knowledge of a plaintext f and randomness r_f, r_π such that $C_f = E_{pk}(f; r_f)$ and $C_\pi = C^f \cdot E_{pk}(0; r_\pi)$, where exponentiation is computed componentwise.

$$\mathcal{R}_{\text{Mult}} = \left\{\left((\mathbb{G}_q, q, g, h, C, C_f, C_\pi), (f, r_f, r_\pi)\right) \text{ s.t. } C_f = \left\langle g^{r_f}, h^{r_f} \cdot g^f\right\rangle \wedge C_\pi = C^f \cdot \left\langle g^{r_\pi}, h^{r_\pi}\right\rangle\right\}.$$

5. π_{NZ}, for demonstrating that a ciphertext C' can be computed from $C = \langle \alpha, \beta\rangle$ by raising C (componentwise) to a non-zero exponent and rerandomizing it, i.e. $C' = C^R \cdot E_{pk}(0; r) = \langle \alpha', \beta'\rangle$.

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

   The challenging part when constructing a proof for this relation is to show that $R \neq 0$. To do this, the prover 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: once on (C, C_R, C') and once on $(C_R, C_{R'}, C_\pi)$. 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 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. Running 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 distributed in $\mathbb{Z}_q^*$, π_{NZ} is zero-knowledge.

2.5.2. *Additional Zero-Knowledge Proofs*

1. π_{Perm}, for demonstrating that a set of ciphertexts $\{C_i\}_i$ is a random permutation and rerandomization of another set, $\{C'_i\}_i$. A number of potential proofs exist in the literature; the most recent solution by Bayer and Groth [8] obtains sublinear communication, whereas the amount of the prover's work in quasilinear. Other works, such as [44], require linear communication/computation complexity.

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

2. $\pi_{\star\text{-proof}}$, for demonstrating the correctness with respect to the following relation, defined in two phases. Looking ahead, this proof is used within Protocol $\mathcal{F}_{\text{PM-}\star}$ for secure pattern matching with wildcards. Specifically, let $\mathbb{G}_q$ be a group of prime order and let $\mathbb{G}_q^2$ be the ciphertext domain for the associated, additively homomorphic ElGamal encryption scheme with encryption function $E_{pk}(\cdot;\cdot)$. Let $T_1, \ldots, T_n \in {\mathbb{G}_q}^2$ be a collection of encryptions. Then, for $j \in \{1, \ldots, n-m+1\}$ define first a function $\phi_j : ({\mathbb{Z}_q}^m \times \mathbb{Z}_q) \mapsto {\mathbb{G}_q}^2$ by

$$\phi_j\left(\{w_i\}_{i=1}^m, r_j\right) = \left(\prod_{i=1}^{m} (T_{i+j})^{w_i \cdot 2^{i-1}}\right) \cdot E_{pk}(0; r_j).$$

That is, the output is the rerandomization of an encryption of Alice's substring (which holds the text), starting at position j with wildcard positions replaced by 0. Next, define a function $\phi_{T_1,\ldots,T_n} : ({\mathbb{Z}_q}^m \times {\mathbb{Z}_q}^m \times {\mathbb{Z}_q}^{n-m+1}) \mapsto ({\mathbb{G}_q}^2)^m \times ({\mathbb{G}_q}^2)^{n-m+1}$ as follows:

$$\phi_{T_1,\ldots,T_n}\left(\{w_i\}_{i=1}^m, \{r_{w_i}\}_{i=1}^m, \{r_j\}_{j=1}^{n-m+1}\right) = \begin{pmatrix} \{E_{pk}(w_i; r_{w_i})\}_{i=1}^m, \\ \{\phi_j(\{w_i\}_{i=1}^m, r_j)\}_{j=1}^{n-m+1} \end{pmatrix}. \quad (1)$$

I.e., $\phi_{T_1,\ldots,T_n}$ consists of m encryptions of values w_i with randomness r_{w_i} as well as $n-m+1$ rerandomized encryptions, each computed from m pairs, (w_i, T_{i+j}) as defined by ϕ_j. Therefore, the set ciphertexts encrypting the text and $\{w_i\}_i$ is the statement and the set of plaintexts and randomness is the witness. A detailed protocol as well as a complete proof can be found in Appendix A. Our proof introduces communication complexity $O(n+m)$ which is linear in the inputs lengths, and computation cost $O(nm)$.

3. $\pi_{\text{H-proof}}$, for demonstrating correctness with respect to the following relation, also defined in two phases. Looking ahead, this proof is used within Protocol π_{APM} for approximate pattern matching. The goal is for Alice to verify that the Hamming distances have been correctly computed, i.e., that Bob correctly performed his part of the computation between the substrings of the text, t, and his pattern, p. For $j = 1, \ldots, n-m+1$ let

$${H_{T_1,\ldots,T_n}}^{(j)} : \mathbb{Z}_q^m \times \mathbb{Z}_q \mapsto {\mathbb{G}_q}^2$$

be defined as

$$H_{T_1,\ldots,T_n}{}^{(j)}\left(\{p_i\}_{i=1}^m, r_j\right) = \left(\prod_{i=1}^{m}(T_{j+i-1})^{-2p_i} \cdot E_{pk}(1;0)^{p_i}\right) \cdot E_{pk}(0;r_j).$$

Now define

$$H_{T_1,\ldots,T_n} : {\mathbb{Z}_q}^m \times {\mathbb{Z}_q}^{n-m+1} \times {\mathbb{Z}_q}^m \mapsto \left({\mathbb{G}_q}^2\right)^{n-m+1} \times \left({\mathbb{G}_q}^2\right)^m$$

as

$$\begin{aligned} &H_{T_1,\ldots,T_n}\left(\{p_i\}_{i=1}^m; \{r_{p_i}\}_{i=1}^m; \{r_j\}_{j=1}^{n-m+1}\right) \\ &\quad = \left(\left\{H_{T_1,\ldots,T_n}{}^{(j)}\left(\{p_i\}_{i=1}^m, r_j\right)\right\}_{j=1}^{n-m+1}, \left\{E_{pk}(p_i; r_{p_i})\right\}_{i=1}^m\right). \end{aligned}$$

A detailed protocol as well as a complete proof can be found in Appendix B. Our proof introduces communication complexity $O(n+m)$ which is linear in the inputs lengths, and computation cost $O(nm)$.

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 to compute the functionality $\mathcal{F}_{\mathrm{PM}}$ defined by

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

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 [10,31] when no level of security is required. We examine a secure version for this problem where Alice, who holds the text, does not gain any information about the pattern from the protocol execution, whereas Bob, who holds the pattern, 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 a 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. More specifically, the parties compute $O(n+m)$ exponentiations and exchange $O(n+m)$ group elements.

Here and below, we have the parties jointly (and securely) transform their inputs from binary representation into elements of $\mathbb{Z}_q$ (we assume that $m < \log_2 q$; larger pattern-lengths can be accommodated by encoding the pattern and substrings of the text into multiple values; see Sect. 3.1 for further details), 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 $\mathbb{Z}_{2^m}$ (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 π_{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.
  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,
  $$P \leftarrow \prod_{i=1}^{m} P_i^{2^{i-1}} \tag{2}$$
  using the homomorphic property of ElGamal PKE.
  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,
  $$\bar{T}_j \leftarrow \prod_{i=j}^{j+m-1} T_i^{2^{i-j}}. \tag{3}$$
  5. For every $\bar{T}_j$, $j = 1, \ldots, n - m + 1$, both parties compute
  $$\Delta_j \leftarrow \bar{T}_j \cdot P^{-1}. \tag{4}$$
  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. (2) (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 PKE,

$$P = \prod_{i=1}^{m} P_i^{2^{i-1}} = E_{pk}\left(\sum_{i=1}^{m} 2^{i-1} p_i; \sum_{i=1}^{m} 2^{i-1} r_{p_i}\right). \tag{5}$$

Note that P is obtained deterministically from the P_i, hence both Alice and Bob hold the same fixed encryption. Similarly, in Eq. (3) 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. (4) 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

$$\begin{aligned} \Delta_j &= \bar{T}_j \cdot P^{-1} \\ &= E_{pk}(\bar{t}_j - p; r_{\bar{t}_j} - r_p). \end{aligned}$$

At this point, it simply remains for Bob to securely determine which of the Δ_j are encryptions of zero, as

$$\delta_j = 0 \quad \Longleftrightarrow \quad \bar{t}_j = p.$$

Security of π_{PM} We are now ready to prove the following theorem:

Theorem 6 (Main). *Assume that the DDH assumption holds in* $\mathbb{G}_q$*, 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 $\mathcal{F}_{\text{ZK}}^{\mathcal{R}_{\text{isBit}}}$.

Alice is Corrupted Recalling that 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). Formally, let $\mathcal{A}$ denote an adversary controlling Alice then construct a simulator $\mathcal{S}$ as follows:

1. $\mathcal{S}$ is given a text t of length n, an integer m 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 $s_A, s_B \in \mathbb{Z}_q$ and hands $\mathcal{A}$ its share s_A and the public key $\langle \mathbb{G}_q, q, g, h = g^{s_A + s_B} \rangle$.

3. Next, $\mathcal{S}$ sends m encryptions of 0 and emulates $\mathcal{F}_{\text{ZK}}^{\mathcal{R}_{\text{isBit}}}$ by sending 1.
4. $\mathcal{S}$ receives from $\mathcal{A}$ n encryptions and the witness for the trusted party for π_{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.
5. Otherwise, $\mathcal{S}$ defines t according to the witness for π_{isBit} and records it.
6. $\mathcal{S}$ and $\mathcal{A}$ compute P, $\{\bar{T}_j\}_j$ and $\{\Delta_j\}_j$ as in the hybrid execution. Then, $\mathcal{S}$ emulates $\mathcal{F}_{\text{Dec0}}$ accepting if the ideal functionality would accept as well.
7. 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 sends (t, m) to the trusted party and outputs whatever $\mathcal{A}$ does.

Clearly, $\mathcal{S}$ runs in probabilistic polynomial time. We prove now that the joint output distribution is computationally indistinguishable in both executions. To see that $\mathcal{A}$'s view is computationally indistinguishable, note first that the only difference between the executions is with respect to the encryptions that assemble p, i.e., the bits encryptions of the pattern (as $\mathcal{S}$ sends encryptions of zero).

We prove that $\mathcal{A}$ cannot distinguish the simulated and hybrid views via a reduction to the semantic security of ElGamal (cf. Definition 4). More formally, assume there exists a distinguisher D for these executions, we construct a distinguisher D_E as follows. Upon receiving a public key pk and auxiliary input p, D_E engages in an execution of π_{KeyGen} with $\mathcal{A}$ and sends it (s_A, pk) where $s_A \in_R \mathbb{Z}_q$. D_E continues emulating the role of Bob as $\mathcal{S}$ does except for Step 2 of the protocol where it needs to send the encryptions of $p_1, \ldots, p_m$. In this step D_E outputs two sets of plaintexts: (i) $p_1, \ldots, p_m$ and (ii) $0, \ldots, 0$. We denote by $\tilde{P}_1, \ldots, \tilde{P}_m$ the set of encryptions it receives back. D_E hands $\mathcal{A}$ this set and completes the run as $\mathcal{S}$ does. Finally, it invokes D on $\mathcal{A}$'s output and outputs whatever D outputs. Note that at no point in the reduction, will D_E need to use the actual plaintexts that correspond to the challenge ciphertexts. Moreover, if D_E is given the encryptions of p then the adversary's view is distributed as in the hybrid execution. Similarly, if it receives encryptions of zeros, then the adversary's view is as in the simulation with $\mathcal{S}$.

It remains to show that the honest Bob outputs the same set of indices with overwhelming probability in both executions. This follows directly from the correctness argument above. In particular, assuming that Alice indeed completes the execution honestly (which is indeed the case due to the zero-knowledge proofs), the protocol correctly computes the matching text locations. This concludes the case that Alice is corrupted.

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 similarly 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 $\langle \mathbb{G}_q, q, g, h = g^{s_A + s_B} \rangle$.
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 the trusted party. Let Z be the set of returned indexes.
5. Next, $\mathcal{S}$ sends n fresh encryptions of 0 and emulates $\mathcal{F}_{\text{ZK}}^{\mathcal{R}_{\text{isBit}}}$ by sending 1.
6. Finally, $\mathcal{S}$ and $\mathcal{A}$ compute P, $\{\bar{T}_j\}_j$ and $\{\Delta_j\}_j$ as in the hybrid execution. Then $\mathcal{S}$ emulates $\mathcal{F}_{\text{Dec0}}$ by sending an output as specified by Z rather than by the encrypted "result", $\{\Delta_j\}_j$. Namely, $\mathcal{S}$ "decrypts" Δ_j into zero if and only if $j \in Z$.
7. 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 semantic security of ElGamal. Recall that the key difference between the executions is that the encryptions of Alice's text are replaced by encryptions of 0's, which implies that the result given to $\mathcal{A}$ in Step 6 of the simulation may not match the actual plaintexts.

Formally, assume there exists a distinguisher D for the simulated and hybrid protocol views. We may then construct a distinguisher D_E breaking the semantic security of ElGamal PKE as follows. Upon receiving a public key pk and auxiliary input t, D_E emulates $\mathcal{F}_{\text{KeyGen}}$ by sending (s_B, pk) to $\mathcal{A}$ where $s_B \in_R \mathbb{Z}_q$. Note that this perfectly matches $\mathcal{A}$'s view in both protocol and simulation. D_E continues emulating the role of Alice as $\mathcal{S}$ does except for Step 5 of the simulation. Instead of simulating Alice's input, D_E outputs two sets of plaintexts: (i) $(t_1, \ldots, t_n)$ and, (ii) $(0 \ldots, 0)$. 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 simulated run. Finally, D_E invokes D on $\mathcal{A}$'s output and outputs whatever D outputs.

If D successfully distinguishes between a simulated view and a view of the hybrid protocol, then D_E distinguishes between encryptions of the t_i's and encryptions of 0's. For case (i), i.e., if D_E received encryptions of $t_1, \ldots, t_n$, $\mathcal{A}$'s view is identical to the view when executing the hybrid protocol, since except for the interaction with $\mathcal{F}_{\text{KeyGen}}$, $\mathcal{F}_{\text{ZK}}^{\mathcal{R}_{\text{isBit}}}$, and $\mathcal{F}_{\text{Dec0}}$, Alice's only action is to send her encrypted input. For case (ii), D_E sends n encryptions of 0 to $\mathcal{A}$, hence in this case D_E's behavior exactly matches that of $\mathcal{S}$. □

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 a constant number of exponentiations. Regarding computational complexity, it is clear that except for Step 4 at most $O(m+n)$ exponentiations are required. Note first that Eq. (3) can be implemented using the *square and multiply* technique. Namely, for every $j = 1, \ldots, n-m+1$, $\bar{T}_j$ is computed by $(\cdots((T_{j+m-1})^2 \cdot T_{j+m-2})^2 \cdot T_{j+m-3} \cdots)^2 \cdot T_j$. 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 by a constant factor) can be easily shown. That is, 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. This enables to complete the transformation from binary representation in constant time per text location. We comment that 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 additional constant number of exponentiations.) Finally, note that our protocol utilizes 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. This also means that the length of the pattern must be bounded by 160 bits. For applications that require longer patterns we propose a different approach; see Sect. 3.1.

3.1. *Variations*

The following variations can be handled similarly to the classic problem of pattern matching.

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, $\{p^{(i)}\}_i, \{\bar{t}_j^{(i)}\}_i$ for $i \in [\log_2 s^m / \log_2 q]$. Namely, the number of blocks of length $\log q$ that are required to "cover" $\log s^m$; denote this value by ρ. 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 exponent r_i, rerandomizes them and sends them to Bob (proving that everything was done correctly). The parties then execute 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. The overhead of this approach is dominated by repeating the basic linear solution ρ times for each text location. As now, the parties compare ρ blocks each time rather than just one. Hence, communication/computation complexities are multiplied by ρ.

Hiding Matched 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. (4). 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 to compute the functionality $\mathcal{F}_{\text{PM}-\star}$ defined by

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

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 [28]. 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,

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

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 positions.
5. Alice supplies her input as in Step 3 of Protocol π_{PM} in Sect. 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 m-bit substring of t starting at position $j = 1, \ldots, n - m + 1$, Bob computes an encryption

$$\bar{T}'_j \leftarrow \left(\prod_{i=1}^{m} \left((T_{j+i-1})^{w_i} \right)^{2^{i-1}} \right) \cdot E_{pk}(0; r_j).$$

He sends these to Alice, and they run $\pi_{\star\text{-proof}}$ on the tuple consisting of the encryptions of Alice's input and Bob's w_i, as well as the $\bar{T}'_j$. This allows Alice to verify that Bob correctly computed encryptions of her substrings with her input replaced by 0 at Bob's wildcard positions.

7. The protocol concludes as Protocol π_{PM} does. Namely, 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's 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 for $i = 1, \ldots, m$, 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 $\pi_{\star\text{-proof}}$ ensure his correct behavior. Formal simulation is analogous to that in Sect. 3. We state the following theorem:

Theorem 7 (Wildcards). *Assume that the DDH assumption holds in $\mathbb{G}_q$, 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 send $\Theta(n + m)$ encryptions to Alice, as well as an invocation of $\pi_{\star\text{-proof}}$. Hence, due to the latter communication complexity is $O(n + m)$ and round complexity

remains constant, while computation is increased to $O(nm)$ multiplications and exponentiations. We remark that dropping the ZK-proofs results in a passively secure variant requiring only $O(n+m)$ exponentiations since the computation of $\bar{T}'_j$ in Step 6 can be implemented similarly to *square and multiply*.

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

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

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 Sect. 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. [3] which introduced a solution in time $O(n\sqrt{\tau \log \tau})$. Our solution achieves $O(nm)$ computation and $O(n\tau)$ 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 distance h_j for each position j, using the homomorphic properties of ElGamal PKE (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 π_{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 m-bit substring of t starting at position $j = 1, \ldots, n - m + 1$, Bob computes an encryption

$$H'_j \leftarrow \prod_{i=1}^{m} (T_{j+i-1})^{-2p_i} \cdot E_{pk}(1;0)^{p_i} \tag{6}$$

and rerandomizes it. He then sends all these to Alice and demonstrates that they have been correctly computed by executing $\pi_{\text{h-proof}}$ on the encryptions P_i of the p_i and the H'_j.

6. For every m-bit substring, $\bar{t}_j$ of t starting at position $j = 1, \ldots, n - m + 1$, both parties locally compute encryptions of the Hamming distance between $\bar{t}_j$ and p,

$$H_j \leftarrow H'_j \cdot \left(\prod_{i=1}^{m} T_{j+i-1} \right). \tag{7}$$

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

$$\Delta_{j,k} \leftarrow H_j \cdot \langle 1, g^{-k} \rangle. \tag{8}$$

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 $\{\Delta_{j,k}\}_k$,

$$\left(\Delta'_{j,0}, \ldots, \Delta'_{j,\tau-1}\right) \leftarrow \pi_j(\Delta_{j,0}, \ldots, \Delta_{j,\tau-1}),$$

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 $((\Delta_{j,0}, \ldots, \Delta_{j,\tau-1}), (\Delta''_{j,0}, \ldots, \Delta''_{j,\tau-1}))$ 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 Eq. (7) are the desired Hamming distances. It is straightforward to verify that if the H'_j have been correctly computed, the p_i are bits, and the T_j are encryptions of bits, then the encryption

$$H'_j \cdot \left(\prod_{i=1}^{m} T_{j+i-1} \right) = \prod_{i=1}^{m} (T_{j+i-1})^{1-2p_i} \cdot E_{pk}(1;0)^{p_i}$$

contains the Hamming distance between the string $p \in \{0,1\}^m$ and the encrypted substring of length m starting at position j. The expression

$$(T_{j+i-1})^{1-2p_i} \cdot E_{pk}(1;0)^{p_i}$$

simply negates the encrypted bit, T_{j+i-1}, if p_i is set, i.e., computes an encryption of $t_{j+i-1} \oplus p_i$. Further, as multiplying ciphertexts computes the encrypted sum of the plaintexts and $m < q$, then clearly the overall result is the number of differing bits—in other words, the Hamming distance.

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. (8), 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. Both are easily simulated, hence we state the following theorem:

Theorem 8 (Approximate). *Assume that the DDH assumption holds in $\mathbb{G}_q$, 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, and the permutations and decryptions needed to compare the Hamming distances to τ, Steps 8 and 9. The former requires $O(m+n)$ communication, but $O(nm)$ multiplications and exponentiations. The latter requires both $O(n\tau)$ communication, multiplications and exponentiations. As $\tau \leq m$ this implies $O(n\tau)$ communication and $O(mn)$ computation overall. Round complexity is constant as in the previous solutions. We remark that dropping the ZK-proofs results in a more efficient, passively secure variant, since the computational complexity of Steps 5 and 6 is reduced to $O(nm)$ multiplications and $O(n+m)$ exponentiations.

5.1. *A Variation—Using Paillier Encryption*

The approximate pattern matching protocol is our most costly construction in terms of communication, as $O(n\tau)$ elements are exchanged between the parties. This was due to implementing the comparison between Hamming distance and threshold using τ equality tests. We now propose an alternative to the above scheme, and note that it requires $o(n\sqrt{\tau \log \tau})$ communication, i.e., exchange fewer elements than any "naive", secure implementation based on [3] would.

Our protocol could equally well be constructed using Paillier encryption, [39]. The drawbacks include a *significantly less efficient* key generation as well as larger ciphertexts due to basing security on factoring rather than discrete logarithms. However, comparison (greater-than) becomes much more efficient requiring communication complexity of $O(\text{loglog}\,\tau \cdot (\text{logloglog}\,\tau + k))$ where k is a security or correctness parameter, [42]. This implies an overall communication complexity of $O(n \cdot \text{loglog}\,\tau\,(\text{logloglog}\,\tau + k))$.[1]

[1] This construction increases the round-complexity to $O(\text{loglog}\,\tau)$; for constant round complexity $O(n \cdot \sqrt{\log \tau}(\text{loglog}\,\tau + k))$ elements will be exchanged.

We remark that in practice, it may be preferable to avoid statistical security/correctness; with present knowledge this requires $O(\log \tau)$ elements to be exchanged, e.g., by adapting the protocol of Nishide and Ohta, [37]. Despite an overall worse asymptotic behavior of $O(n \log \tau)$, avoiding the factor of k improves efficiency for "small" τ.

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

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

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 $\pi_{\text{PM-}\star}$. The main idea is to have Bob construct a pattern p' of length M by padding p with $M - m$ wildcards. Though not completely correct, intuitively, executing $\pi_{\text{PM-}\star}$ on input $((p', n), (t, M))$ 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 p'. To eliminate these issues, Alice's text must be extended, while Bob must demonstrate that his wildcards are correctly placed. In detail, our construction is the following.

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

- *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 $\{0, 1\}$ of length $m \leq M$ 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. Bob constructs a pattern p' of length M by padding p with $M - m$ zeros. 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 *ternary*,

$$P' \leftarrow \prod_{i=1}^{m} {P'_i}^{3^{i-1}}.$$

3. For each position $i = 1, \ldots, M$ of p', Bob computes a bit denoting if this position is padding

$$w_i \leftarrow \begin{cases} 0 & \text{if } i > m, \\ 1 & \text{otherwise.} \end{cases}$$

He encrypts these and sends the result to Alice,

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

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 if Bob claims some position is padding ($w_i = 0$) then the associated p'_i is also 0.
5. For each $i = 1, \ldots, M-1$, Bob and Alice run π_{isBit} on W_i / W_{i+1}. This demonstrates to Alice that a 1 never follows a 0 in the w_i, i.e., that $w_1, \ldots, w_M$ is monotonically non-increasing. Hence zeros (signifying padding) occur at the end.
6. Alice supplies her input as in Step 3 of Protocol π_{PM} in Sect. 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.
7. Alice and Bob pad Alice's encrypted text with $M - 1$ default encryptions of 2, $T_j = \langle 1, g^2 \rangle$ for $j \in \{n+1, n+2, n+M-1\}$.
8. For every M-bit substring of the padded t starting at position $j = 1, \ldots, n$, Bob computes an encryption

$$\bar{T}'_j \leftarrow \left(\prod_{i=1}^{M} \left((T_{j+i-1})^{w_i} \right)^{3^{i-1}} \right) \cdot E_{pk}(0; r_j).$$

He sends these to Alice, and they run $\pi_{\star\text{-proof}}$ on the tuple consisting of the encryptions of Alice's padded input and Bob's w_i, as well as the $\bar{T}'_j$. This allows Alice to verify that Bob correctly computed encryptions of her substrings in ternary with her input replaced by 0 at Bob's padding positions.[2]

9. The protocol concludes as above: For each of the $\bar{T}'_j$ where $j = 1, \ldots, n$, 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.

Correctness is straightforward: Alice pad her text with the character 2, which will match Bob's padding but not his binary pattern. This explains the need for ternary representation rather than binary representation in Steps 2 and 8. Specifically, any character, including 2, will match the padding characters of p' since it is replaced by 0 in the

[2] $\pi_{\star\text{-proof}}$ specified in Appendix A deals with binary representation. Modifying it into ternary representation is straightforward.

computation of the encrypted substring $\bar{T}'_j$, in Step 8. Thus, if Bob behaves honestly and supplies a correct input, then the matches are correctly output.

Moreover, due to the use of zero-knowledge proofs, malicious parties cannot deviate, i.e., they are forced to behave as an honest party would. In particular, Alice verifies that Bob's "padding vector" $w_1, \ldots, w_M$, is not malformed in Steps 4 and 5. All padding of p' is 0 and padding is added only at the end of p, such that the 1 character never follows the 0 character in the padding portion. Finally, Bob cannot use non-binary inputs due to the execution of π_{isBit}. Hence a malicious Bob is reduced to supplying an input that an honest Bob could supply implying that the correct matches are found. The security argument for a malicious Alice follows similarly.

The communication complexity of $\pi_{\text{PM-hpl}}$ is $O(n+M)$, whereas the computation is $O(nM)$ multiplications due to the computation of Step 8. The analysis is analogous to the one for $\pi_{\text{PM-}\star}$; the main differences are Bob's demonstration that the padding occurs at the end, Step 5, and the extension of Alice's text to one of length $n+M-1$, Step 7, which clearly is linear in $n+M$. We conclude with the following theorem,

Theorem 9 (Pattern length hiding). *Assume that the DDH assumption holds in* $\mathbb{G}_q$, *then* $\pi_{\text{PM-hpl}}$ *securely computes* $\mathcal{F}_{\text{PM-hpl}}$ *in the presence of malicious adversaries.*

Adding a Lower Bound on m Allowing Bob to input arbitrary patterns of length at most M may not be acceptable. In particular using a single-bit pattern in $\pi_{\text{PM-hpl}}$ reveals all of Alice's text, and if an honest Bob is allowed this action, then so is a malicious one. This "attack" can be prevented by adding a lower bound, μ on Bob's pattern length. This can be enforced by setting $W_1, \ldots, W_\mu$ to default encryptions of 1, $\langle 1, g \rangle$, in the above protocol.

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 to compute the functionality $\mathcal{F}_{\text{PM-htl}}$,

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

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

The core idea of the solution is to extend the alphabet with an additional character and have Alice pad her text with $N-n$ occurrences of this. Overall, the protocol is similar to π_{PM}; moreover, Alice is forced to behave honestly using a similar construction to the one ensuring Bob's honesty in $\pi_{\text{PM-hpl}}$ above. The whole construction is as follows:

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

- *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.
2. As in the basic solution, 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 execute π_{isBit}, allowing Alice to verify that Bob has provided a bit-string of length m. Both parties then compute an encryption of Bob's pattern,

$$P \leftarrow \prod_{i=1}^{m} P_i^{3^{i-1}}.$$

   Note that contrary to the basic solution, the *binary* pattern is encoded in *ternary* to allow an additional symbol, 2.
3. Initially Alice pads her text with 1's; we denote the padded text t'. She then sends encryptions, $T'_j = E_{pk}(t'_j; r_{t'_j})$, $j = 1, \ldots, N$, of the bits of this N-bit input, to Bob. Further, for each of the N encryptions, the parties execute π_{isBit}, allowing Bob to verify that Alice has indeed provided the encryption of a known N-bit string.
4. Then, for $j = 1, \ldots, N$ Alice computes

$$d_j \leftarrow \begin{cases} 1 & \text{if } j > n, \\ 0 & \text{otherwise.} \end{cases}$$

   These bits represent Alice's padding, and encryptions of them, $D_j = E_{pk}(d_j; r_{d_j})$ $j = 1, \ldots, N$, are then sent to Bob. Alice then proves that they indeed contain bits by running π_{isBit}, and she further demonstrates that $d_1, \ldots, d_N$ is monotonically non-decreasing. Similarly to Bob's proof in Step 5 of $\pi_{\text{PM-hpl}}$, running π_{isBit} on D_{j+1}/D_j demonstrates that all padding occurs at the end of t'.
5. Next, Alice and Bob run π_{isBit} on T'_j/D_j for $j = 1, \ldots, N$. This demonstrates to Bob that whenever d_j is set, then so is t'_j, hence Alice's padding contains only 1's.
6. For every m-bit substring of the padded text t', starting at position $j = 1, \ldots, N - m + 1$, both parties compute an encryption of that string with any padding replaced by 2's:

$$\bar{T}'_j \leftarrow \prod_{i=j}^{j+m-1} \left(T'_i \cdot D_i\right)^{3^{i-j}}.$$

7. As Step 5 of π_{PM}, for every $\bar{T}'_j$, $j = 1, \ldots, N - m + 1$ the parties compute
$$\Delta_j \leftarrow \bar{T}'_j \cdot P^{-1}.$$
8. For every $j = 1, \ldots, N - m + 1$ Alice and Bob run π_{Dec0} on Δ_j; Bob outputs j iff $\delta_j = 0$.

Correctness of $\pi_{\text{PM-htl}}$ is easily verified. The honest Alice sets the $N - n$ rightmost d_j and t'_j to 1. Therefore, the $\bar{T}'_j$ computed in Step 6 consists of an m-character substring of t' in ternary, where any 1's from padding has been replaced by $1 + 1 = 2$. Bob's pattern is similarly computed in ternary, implying that Δ_j contains 0 iff the pattern matches.

Regarding security, Bob's behavior is essentially the same as in π_{PM}; hence the proof of security is analogous. Regarding Alice, note that even if she is malicious, she is forced to provide a well-formed text and denotation of padding due to the zero-knowledge proofs of knowledge. In Step 4 she demonstrates that the $d = d_1, \ldots, d_N$ consists of a string of 0's followed by a string of 1's. (This is equivalent to saying that all padding occurs at the end.) Then in Step 5 she demonstrates that she indeed padded t with 1's. In other words, an honest Alice could have supplied the same input. Formally, simulating the view is analogous to the basic case.

Complexity is similar to the basic protocol and only $O(N + m)$ encryptions change hands, hence only this many zero-knowledge proofs of knowledge are needed as well. Analogously to the computation of the $\bar{T}_j$ in π_{PM}, computing $\bar{T}'_j$ in Step 6 naïvely requires $O(Nm)$ multiplications. Again, it is possible to reduce this to linear at the cost of increasing the number of exponentiations by a constant factor. Thus, both communication and computation complexities are linear while the required number of rounds is constant.

Theorem 10 (Text length hiding). *Assume that the DDH assumption holds in $\mathbb{G}_q$, then $\pi_{\text{PM-htl}}$ securely computes $\mathcal{F}_{\text{PM-htl}}$ in the presence of malicious adversaries.*

Appendix A. Σ-Protocol $\pi_{\star\text{-proof}}$

In this section we provide a Σ-protocol $\pi_{\star\text{-proof}}$ used within Protocol $\pi_{\text{PM-}\star}$, Step 6. The detailed protocol is seen in Fig. A.1 and demonstrates knowledge of a preimage of $\phi_{T_1,\ldots,T_n}$ for the value $(\{C_i\}_{i=1}^m, \{\bar{T}'_j\}_{j=1}^{n-m+1})$, i.e., demonstrates knowledge of the plaintexts and randomness of the m first encryptions and—more importantly in the present work—demonstrates that the final $n - m + 1$ encryptions are computed correctly from the encryptions T_j and values, w_i. Hence, this protocol allows Alice to verify that Bob has correctly replaced her encrypted bits (i.e., T_j) with encryptions of 0 (i.e., $(T_j)^0$—a default encryption of 0) at the wildcard positions of the encrypted substrings, denoted by $\bar{T}'_j$.

Theorem 11. *$\pi_{\star\text{-proof}}$ is a Σ-protocol.*

Proof. We show correctness, special soundness and special honest verifier zero-knowledge.

Fig. A.1. Protocol $\pi_{*\text{-proof}}$.

Prover:		*Verifier:*
Witness	*Public Statement*	
$\{w_i\}_{i=1}^m \in \mathbb{Z}_q^m$	$T_1, \ldots T_n \in (\mathbb{G}_q^2)^n$;	
$\{r_{w_i}\}_{i=1}^m \in \mathbb{Z}_q^m$	$\{C_i\}_{i=1}^m, \{\bar{T}'_j\}_{j=1}^{n-m+1}$	
$\{r_j\}_{j=1}^{n-m+1} \in \mathbb{Z}_q^{n-m+1}$	$\in (\mathbb{G}_q^2)^m \times (\mathbb{G}_q^2)^{n-m+1}$	
s.t.		
$\phi_{T_1,\ldots,T_n}(\{w_i\}_{i=1}^m; \{r_{w_i}\}_{i=1}^m; \{r_j\}_{j=1}^{n-m+1})$		
$= (\{C_i\}_{i=1}^m, \{\bar{T}'_j\}_{j=1}^{n-m+1})$		
Uniformly generate:		
$\{\tilde{w}_i\}_{i=1}^m \in_R \mathbb{Z}_q^m$		
$\{r_{\tilde{w}_i}\}_{i=1}^m \in_R \mathbb{Z}_q^m$		
$\{\tilde{r_j}\}_{j=1}^{n-m+1} \in_R \mathbb{Z}_q^{n-m+1}$		
$A \leftarrow \phi_{T_1,\ldots,T_n}(\{\tilde{w}_i\}_{i=1}^m;$		
$\{r_{\tilde{w}_i}\}_{i=1}^m; \{\tilde{r_j}\}_{j=1}^{n-m+1})$		
	$A = (\{C_i^{(A)}\}_{i=1}^m, \{\bar{T}_j^{(A)}\}_{j=1}^{n-m+1})$ $\longrightarrow$	
		Uniformly generate: $E \in_R \mathbb{Z}_q$
	$\longleftarrow$ E	
For $i = 1, \ldots, m$		
$w_i^{(Z)} \leftarrow w_i \cdot E + \tilde{w}_i \bmod q$		
$r_{w_i}^{(Z)} \leftarrow r_{w_i} \cdot E + r_{\tilde{w}_i} \bmod q$		
For $j = 1, \ldots, n-m+1$		
$r_j^{(Z)} \leftarrow r_j \cdot E + \tilde{r_j} \bmod q$		
	$Z = (\{w_i^{(Z)}\}_{i=1}^m; \{r_{w_i}^{(Z)}\}_{i=1}^m; \{r_j^{(Z)}\}_{j=1}^{n-m+1})$ $\longrightarrow$	
		Verification: accept if
		$\{E_{pk}(w_i^{(Z)}; r_{w_i}^{(Z)})\}_{i=1}^m$
		$= \{(C_i)^E \cdot C_i^{(A)}\}_{i=1}^m$
		and
		$\{\phi_j(\{w_i^{(Z)}\}_{i=1}^m, r_j^{(Z)})\}_{j=1}^{n-m+1}$
		$= \{(\bar{T}'_j)^E \cdot \bar{T}_j^{(A)}\}_{j=1}^{n-m+1}$

Correctness An honest verifier always accepts when interacting with an honest prover. This is clear, as for $i \in \{1, \ldots, m\}$

$$\begin{aligned} E_{pk}\big(w_i^{(Z)}; r_{w_i}^{(Z)}\big) &= \big\langle g^{r_{w_i}^{(Z)}}, h^{r_{w_i}^{(Z)}} \cdot g^{w_i^{(Z)}} \big\rangle \\ &= \big\langle g^{r_{w_i} \cdot E + r_{\tilde{w}_i} \bmod q}, h^{r_{w_i} \cdot E + r_{\tilde{w}_i} \bmod q} \cdot g^{w_i \cdot E + \tilde{w}_i \bmod q} \big\rangle \\ &= \big(\big\langle g^{r_{w_i}}, h^{r_{w_i}} \cdot g^{w_i} \big\rangle\big)^E \cdot \big(\big\langle g^{\tilde{w}_i}, h^{\tilde{w}_i} \cdot g^{\tilde{w}_i} \big\rangle\big) \\ &= (C_i)^E \cdot C_i^{(A)} \end{aligned}$$

as well as for $j \in \{1, \ldots, n - m + 1\}$

$$\begin{aligned} &\phi_j\big(\big\{w_i^{(Z)}\big\}_{i=1}^m, r_j^{(Z)}\big) \\ &= \left(\prod_{i=1}^m (T_{i+j})^{w_i^{(Z)} \cdot 2^{i-1}}\right) \cdot E_{pk}\big(0; r_j^{(Z)}\big) \\ &= \left(\prod_{i=1}^m (T_{i+j})^{w_i \cdot E + \tilde{w}_i \bmod q \cdot 2^{i-1}}\right) \cdot E_{pk}(0; r_j \cdot E + \tilde{r_j} \bmod q) \\ &= \left(\prod_{i=1}^m (T_{i+j})^{w_i \cdot 2^{i-1}}\right)^E \cdot \left(\prod_{i=1}^m (T_{i+j})^{\tilde{w}_i \cdot 2^{i-1}}\right) \cdot E_{pk}(0; r_j)^E \cdot E_{pk}(0; \tilde{r_j}) \\ &= \left(\prod_{i=1}^m (T_{i+j})^{w_i \cdot 2^{i-1}} \cdot E_{pk}(0; r_j)\right)^E \cdot \left(\prod_{i=1}^m (T_{i+j})^{\tilde{w}_i \cdot 2^{i-1}} E_{pk}(0; \tilde{r_j})\right) \\ &= \big(\bar{T}_j'\big)^E \cdot \bar{T}_j^{(A)}. \end{aligned}$$

Special Soundness To prove special soundness, it must be shown that given two accepting executions (A, E, Z) and (A, E', Z') with the same commitment, A, there exists an algorithm to efficiently compute a witness, i.e., an algorithm to efficiently compute

$$\big(\{\hat{w}_i\}_{i=1}^m, \{r_{\hat{w}_i}\}_{i=1}^m, \{\hat{r}_j\}_{j=1}^{n-m+1}\big) \in \mathbb{Z}_q^m \times \mathbb{Z}_q^m \times \mathbb{Z}_q^{n-m+1},$$

such that

$$\phi_{T_1, \ldots, T_n}\big(\{\hat{w}_i\}_{i=1}^m, \{r_{\hat{w}_i}\}_{i=1}^m, \{\hat{r}_j\}_{j=1}^{n-m+1}\big) = \big(\{C_i\}_{i=1}^m, \big\{\bar{T}_j'\big\}_{j=1}^{n-m+1}\big).$$

First let

$$Z = \big(\big\{w_i^{(Z)}\big\}_{i=1}^m; \big\{r_{w_i}^{(Z)}\big\}_{i=1}^m; \big\{r_j^{(Z)}\big\}_{j=1}^{n-m+1}\big)$$

and

$$Z' = \big(\big\{w_i^{(Z')'}\big\}_{i=1}^m; \big\{r_{w_i}^{(Z')'}\big\}_{i=1}^m; \big\{r_j^{(Z')'}\big\}_{j=1}^{n-m+1}\big).$$

Since $E - E' \not\equiv 0 \bmod q$ it is invertible; letting $\delta = (E - E')^{-1} \bmod q$, we may compute

$$\begin{aligned}
\{\hat{w}_i\}_{i=1}^m &\leftarrow \left\{\left(z_i^{(w)} - z_i^{(w)'}\right) \cdot \delta\right\}_{i=1}^m \\
\{r_{\hat{w}_i}\}_{i=1}^m &\leftarrow \left\{\left(r_{w_i}^{(Z)} - r_{w_i}^{(Z')}\right) \cdot \delta\right\}_{i=1}^m \\
\{\hat{r}_j\}_{j=1}^{n-m+1} &\leftarrow \left\{\left(r_j^{(Z)} - r_j^{(Z')}\right) \cdot \delta\right\}_{j=1}^{n-m+1}.
\end{aligned}$$

Verifying that this is a witness is straightforward:

$$\begin{aligned}
E_{pk}(\hat{w}_i; r_{\hat{w}_i}) &= E_{pk}\left(\left(w_i^{(Z)} - w_i^{(Z')}\right) \cdot \delta; \left(r_{w_i}^{(Z)} - r_{w_i}^{(Z')}\right) \cdot \delta\right) \\
&= \left(E_{pk}\left(w_i^{(Z)}; r_{w_i}^{(Z)}\right) \cdot E_{pk}\left(w_i^{(Z')}; r_{w_i}^{(Z')}\right)^{-1}\right)^{\delta} \\
&= \left((C_i)^E \cdot C_i^{(A)} \cdot \left((C_i)^{E'} \cdot C_i^{(A)}\right)^{-1}\right)^{(E-E')^{-1} \bmod q} \\
&= C_i
\end{aligned}$$

$$\begin{aligned}
\phi_j\left(\{\hat{w}_i\}_{i=1}^m; \hat{r}_j\right) &= \phi_j\left(\left\{\left(w_i^{(Z)} - w_i^{(Z')}\right) \cdot \delta\right\}_{i=1}^m; \left(r_j^{(Z)} - r_j^{(Z')}\right) \cdot \delta\right) \\
&= \left(\prod_{i=1}^m (T_{i+j})^{(w_i^{(Z)} - w_i^{(Z')}) \cdot \delta \cdot 2^{i-1}}\right) \cdot E_{pk}\left(0; \left(r_j^{(Z)} - r_j^{(Z')}\right) \cdot \delta\right) \\
&= \left(\left(\prod_{i=1}^m (T_{i+j})^{(w_i^{(Z)} - w_i^{(Z')}) \cdot 2^{i-1}}\right) \cdot E_{pk}\left(0; r_j^{(Z)} - r_j^{(Z')}\right)\right)^{\delta} \\
&= \left(\left(\prod_{i=1}^m (T_{i+j})^{(w_i^{(Z)}) \cdot 2^{i-1}}\right) \cdot E_{pk}\left(0; r_j^{(Z)}\right)\right. \\
&\quad \left. \times \left(\left(\prod_{i=1}^m (T_{i+j})^{(w_i^{(Z')}) \cdot 2^{i-1}}\right) \cdot E_{pk}\left(0; r_j^{(Z')}\right)\right)^{-1}\right)^{\delta} \\
&= \left(\phi_j\left(\left\{w_i^{(Z)}\right\}_{i=1}^m, r_j^{(Z)}\right) \cdot \left(\phi_j\left(\left\{w_i^{(Z')}\right\}_{i=1}^m, r_j^{(Z')}\right)\right)^{-1}\right)^{\delta} \\
&= \left(\left(\bar{T}_j'\right)^E \cdot \bar{T}_j^{(A)} \cdot \left(\left(\bar{T}_j'\right)^{E'} \cdot \bar{T}_j^{(A)}\right)^{-1}\right)^{((E-E')^{-1} \bmod q)} \\
&= \bar{T}_j'
\end{aligned}$$

Special Honest Verifier Zero-Knowledge Given challenge, E, the simulator picks

$$Z = \left(\left\{w_i^{(Z)}\right\}_{i=1}^m; \left\{r_{w_i}^{(Z)}\right\}_{i=1}^m; \left\{r_j^{(Z)}\right\}_{j=1}^{n-m+1}\right) \in_R \mathbb{Z}_q^m \times \mathbb{Z}_q^m \times \mathbb{Z}_q^{n-m+1}.$$

From this an appropriate $A = (\{C_i^{(A)}\}_{i=1}^m; \{\bar{T}_j^{(A)}\}_{j=1}^{n-m+1})$ is computed as

$$\left\{C_i^{(A)}\right\}_{i=1}^m \leftarrow \left\{C_i^{-E} \cdot E_{pk}\left(w_i^{(Z)}; r_{w_i}^{(Z)}\right)\right\}_{i=1}^m$$

and

$$\left\{\bar{T}_j^{(A)}\right\}_{j=1}^{n-m+1} \leftarrow \left\{\left(\bar{T}_j'\right)^{-E} \cdot \phi_j\left(\left\{w_i^{(Z)}\right\}_{i=1}^m, r_j^{(Z)}\right)\right\}_{j=1}^{n-m+1}.$$

Multiplying the former ones by $(C_i)^E$ and the latter ones by $(\bar{T}_j')^E$ clearly results in $\phi(Z)$, hence, Z is exactly the reply that an honest prover would send—since $\phi(Z)$ consists of m "fresh" encryptions, no other possibilities exist for the $w_i^{(Z)}$ and $r_{w_i}^{(Z)}$. Further, once the $w_i^{(Z)}$ become fixed, the evaluations of the ϕ_j's are simply rerandomizations, hence no other options exist for the $r_j^{(Z)}$ exist either. □

Complexity Communication complexity of $\pi_{\star\text{-proof}}$ is clearly $O(m+n)$. The prover sends something in the image of $\phi_{T_1,\ldots,T_n}$ as well as a preimage, Z, and both are linear in m and n. The verifier on the other hand sends only a single $\mathbb{Z}_q$ element. Regarding computation, the most expensive step is the evaluation of $\phi_{T_1,\ldots,T_n}$, which both parties must do. This requires computing $O(m)$ encryptions—the C_i—as well as $n-m+1$ evaluations of functions ϕ_j. It is immediate to see that the former requires $O(m)$ multiplications and exponentiations. The latter on the other hand is more expensive. Each ϕ_j consist of the rerandomization of the product of m exponentiations. Since there are $n-m+1$ of these, overall $O(nm)$ multiplications and exponentiations are needed.

Appendix B. Σ-Protocol $\pi_{\text{H-proof}}$

In this section we provide a Σ-protocol $\pi_{\text{H-proof}}$ used within Protocol π_{APM}, Step 5. The detailed protocol is seen in Fig. B.1 and demonstrates knowledge of a preimage of $H_{T_1,\ldots,T_n}$ for the value

$$\left(\{H_j\}_{j=1}^{n-m+1}, \{P_i\}_{i=1}^m\right) \in \left(\mathbb{G}_q{}^2\right)^{n-m+1} \times \left(\mathbb{G}_q{}^2\right)^m,$$

i.e., demonstrates knowledge of the p_i and the randomness used in the computation of the ciphertexts. Most importantly, this demonstrates to the verifier that the H_j were correctly computed from $\{T_j\}_{j=1}^n$. Hence, this protocol allows Alice to verify that Bob has correctly computed his contribution to the Hamming distance computation based on both his and her encrypted input.

Theorem 11. *$\pi_{\text{H-proof}}$ is a Σ-protocol.*

Proof. We show correctness, special soundness and special honest verifier zero-knowledge.

Fig. B.1. Protocol $\pi_{\text{H-proof}}$.

Prover:		*Verifier:*
Witness $\{p_i\}_{i=1}^m \in \mathbb{Z}_q^m$ $\{r_{p_i}\}_{i=1}^m \in \mathbb{Z}_q^m$ $\{r_j\}_{j=1}^{n-m+1} \in \mathbb{Z}_q^{n-m+1}$ s.t. $H_{T_1,\ldots,T_n}(\{p_i\}_{i=1}^m; \{r_{p_i}\}_{i=1}^m; \{r_j\}_{j=1}^{n-m+1})$ $= (\{H_j\}_{j=1}^{n-m+1}, \{P_i\}_{i=1}^m)$		*Public Statement* $T_1, \ldots T_n \in (\mathbb{G}_q^2)^n$; $\{H_j\}_{j=1}^{n-m+1}, \{P_i\}_{i=1}^m$ $\in (\mathbb{G}_q^2)^{n-m+1} \times (\mathbb{G}_q^2)^m$	
Uniformly generate: $\{\tilde{p_i}\}_{i=1}^m \in_R \mathbb{Z}_q^m$ $\{r_{\tilde{p}_i}\}_{i=1}^m \in_R \mathbb{Z}_q^m$ $\{\tilde{r_j}\}_{j=1}^{n-m+1} \in_R \mathbb{Z}_q^{n-m+1}$ $A \leftarrow H_{T_1,\ldots,T_n}(\{\tilde{p_i}\}_{i=1}^m;$ $\{r_{\tilde{p}_i}\}_{i=1}^m; \{\tilde{r_j}\}_{j=1}^{n-m+1})$	$\xrightarrow{A = (\{H_j^{(A)}\}_{j=1}^{n-m+1}, \{P_i^{(A)}\}_{i=1}^m)}$	
		Uniformly generate: $E \in_R \mathbb{Z}_q$
	$\xleftarrow{E}$	
For $i = 1, \ldots, m$ $p_i^{(Z)} \leftarrow p_i \cdot E + \tilde{p_i} \bmod q$ $r_{p_i}^{(Z)} \leftarrow r_{p_i} \cdot E + r_{\tilde{p}_i} \bmod q$ For $j = 1, \ldots, n-m+1$ $r_j^{(Z)} \leftarrow r_j \cdot E + \tilde{r_j} \bmod q$	$\xrightarrow{Z = (\{p_i^{(Z)}\}_{i=1}^m; \{r_{p_i}^{(Z)}\}_{i=1}^m; \{r_j^{(Z)}\}_{j=1}^{n-m+1})}$	
		Verification: accept if $\{E_{pk}(p_i^{(Z)}; r_{p_i}^{(Z)})\}_{i=1}^m$ $= \{(P_i)^E \cdot P_i^{(A)}\}_{i=1}^m$ and $\{H_{T_1,\ldots,T_n}^{(j)}(\{p_i^{(Z)}\}_{i=1}^m, r_j^{(Z)})\}_{j=1}^{n-m+1}$ $= \{(H_j)^E \cdot H_j^{(A)}\}_{j=1}^{n-m+1}$

Correctness An honest verifier always accepts when interacting with an honest prover. This is clear, as for $i \in \{1, \ldots, m\}$

$$\begin{aligned} E_{pk}\big(p_i^{(Z)}; r_{p_i}^{(Z)}\big) &= \big\langle g^{r_{p_i}^{(Z)}}, h^{r_{p_i}^{(Z)}} \cdot g^{p_i^{(Z)}} \big\rangle \\ &= \big\langle g^{r_{p_i} \cdot E + r_{\tilde{p}_i} \bmod q}, h^{r_{p_i} \cdot E + r_{\tilde{p}_i} \bmod q} \cdot g^{p_i \cdot E + \tilde{p}_i \bmod q} \big\rangle \\ &= \big\langle g^{r_{p_i}}, h^{r_{p_i}} \cdot g^{p_i} \big\rangle^E \cdot \big\langle g^{r_{\tilde{p}_i}}, h^{r_{\tilde{p}_i}} \cdot g^{\tilde{p}_i} \big\rangle \\ &= (P_i)^E \cdot P_i^{(A)}, \end{aligned}$$

while for $j \in \{1, \ldots, n - m + 1\}$

$$\begin{aligned} & H^{(j)}_{T_1,\ldots,T_n}\big(\{p_i^{(Z)}\}_{i=1}^m, r_j^{(Z)}\big) \\ &= \left(\prod_{i=1}^m (T_{i+j-1})^{-2p_i^{(Z)}} \cdot E_{pk}(1;0)^{p_i^{(Z)}}\right) \cdot E_{pk}\big(0; r_j^{(Z)}\big) \\ &= \left(\prod_{i=1}^m (T_{i+j-1})^{-2(p_i E + \tilde{p}_i \bmod q)} E_{pk}(1;0)^{p_i E + \tilde{p}_i \bmod q}\right) \\ &\quad \times E_{pk}(0; r_j \cdot E + \tilde{r_j} \bmod q) \\ &= \left(\prod_{i=1}^m \big((T_{i+j-1})^{-2p_i} E_{pk}(1;0)^{p_i}\big)^E \cdot \big((T_{i+j-1})^{-2\tilde{p}_i} E_{pk}(1;0)^{\tilde{p}_i}\big)\right) \cdot E_{pk}(0; r_j)^E \\ &\quad \times E_{pk}(0; \tilde{r_j}) \\ &= \left(\left(\prod_{i=1}^m (T_{i+j-1})^{-2p_i} E_{pk}(1;0)^{p_i}\right) \cdot E_{pk}(0; r_j)\right)^E \\ &\quad \times \left(\prod_{i=1}^m \big((T_{i+j-1})^{-2\tilde{p}_i} E_{pk}(1;0)^{\tilde{p}_i}\big) \cdot E_{pk}(0; \tilde{r_j})\right) \\ &= (H_j)^E \cdot H_j^{(A)}. \end{aligned}$$

Special Soundness To prove special soundness, it must be shown that given two accepting executions (A, E, Z) and (A, E', Z') with the same commitment, A, there exists an algorithm to efficiently compute a witness, i.e., an algorithm to efficiently compute

$$\big(\{\hat{p}_i\}_{i=1}^m, \{r_{\hat{p}_i}\}_{i=1}^m, \{\hat{r}_j\}_{j=1}^{n-m+1}\big) \in \mathbb{Z}_q^m \times \mathbb{Z}_q^m \times \mathbb{Z}_q^{n-m+1},$$

such that

$$H_{T_1,\ldots,T_n}\big(\{\hat{p}_i\}_{i=1}^m, \{r_{\hat{p}_i}\}_{i=1}^m, \{\hat{r}_j\}_{j=1}^{n-m+1}\big) = \big(\{H_j\}_{j=1}^{n-m+1}, \{P_i\}_{i=1}^m\big).$$

Letting

$$Z = \left(\left\{p_i^{(Z)}\right\}_{i=1}^{m}; \left\{r_{p_i}^{(Z)}\right\}_{i=1}^{m}; \left\{r_j^{(Z)}\right\}_{j=1}^{n-m+1}\right)$$

and

$$Z' = \left(\left\{p_i^{(Z')}\right\}_{i=1}^{m}; \left\{r_{p_i}^{(Z')}\right\}_{i=1}^{m}; \left\{r_j^{(Z')}\right\}_{j=1}^{n-m+1}\right),$$

and noting that $E \neq E'$ implies that $E - E'$ is invertible modulo q, the algorithm initially computes $\delta = (E - E')^{-1} \bmod q$. It then proceeds to compute a preimage:

$$\begin{aligned}
\{\hat{p}_i\}_{i=1}^{m} &\leftarrow \left\{\left(p_i^{(Z)} - p_i^{(Z')'}\right) \cdot \delta\right\}_{i=1}^{m}, \\
\{r_{\hat{p}_i}\}_{i=1}^{m} &\leftarrow \left\{\left(r_{p_i}^{(Z)} - r_{p_i}^{(Z')}\right) \cdot \delta\right\}_{i=1}^{m}, \\
\{\hat{r}_j\}_{j=1}^{n-m+1} &\leftarrow \left\{\left(r_j^{(Z)} - r_j^{(Z')}\right) \cdot \delta\right\}_{j=1}^{n-m+1}.
\end{aligned}$$

Verifying that this is a witness for $\{H_j\}_{j=1}^{n-m+1}, \{P_i\}_{i=1}^{m}$ is straightforward:

$$\begin{aligned}
E_{pk}(\hat{p}_i; r_{\hat{p}_i}) &= E_{pk}\left(\left(p_i^{(Z)} - p_i^{(Z')}\right) \cdot \delta; \left(r_{p_i}^{(Z)} - r_{p_i}^{(Z')}\right) \cdot \delta\right) \\
&= \left(E_{pk}\left(p_i^{(Z)}; r_{p_i}^{(Z)}\right) \cdot E_{pk}\left(p_i^{(Z')}; r_{p_i}^{(Z')}\right)^{-1}\right)^{\delta} \\
&= \left((P_i)^{E} \cdot P_i^{(A)} \cdot \left((P_i)^{E'} \cdot P_i^{(A)}\right)^{-1}\right)^{(E-E')^{-1} \bmod q} \\
&= P_i,
\end{aligned}$$

$$\begin{aligned}
&H^{(j)}_{T_1,\ldots,T_n}\left(\{\hat{p}_i\}_{i=1}^{m}; \hat{r}_j\right) \\
&= \left(\prod_{i=1}^{m} (T_{j+i-1})^{-2\hat{p}_i} \cdot E_{pk}(1;0)^{\hat{p}_i}\right) \cdot E_{pk}(0; \hat{r}_j) \\
&= \left(\prod_{i=1}^{m} (T_{j+i-1})^{-2((p_i^{(Z)} - p_i^{(Z')}) \cdot \delta)} \cdot E_{pk}(1;0)^{(p_i^{(Z)} - p_i^{(Z')}) \cdot \delta}\right) \cdot E_{pk}\left(0; \left(z_j^{(r)} - z_j^{(r)'}\right) \cdot \delta\right) \\
&= \left(\prod_{i=1}^{m} \left((T_{j+i-1})^{-2p_i^{(Z)}} \cdot E_{pk}(1;0)^{p_i^{(Z)}}\right) E_{pk}\left(0; r_j^{(Z)}\right)\right)^{\delta} \\
&\quad \times \left(\left(\prod_{i=1}^{m} \left((T_{j+i-1})^{-2p_i^{(Z')}} \cdot E_{pk}(1;0)^{p_i^{(Z')}}\right) E_{pk}\left(0; r_j^{(Z')}\right)\right)^{-1}\right)^{\delta} \\
&= \left((H_j)^{E} \cdot H_j^{(A)} \cdot \left((H_j)^{E'} \cdot H_j^{(A)}\right)^{-1}\right)^{(E-E')^{-1} \bmod q} \\
&= H_j.
\end{aligned}$$

Special Honest Verifier Zero-Knowledge Given challenge, E, the simulator picks

$$Z = \left(\left\{p_i^{(Z)}\right\}_{i=1}^{m}; \left\{r_{p_i}^{(Z)}\right\}_{i=1}^{m}; \left\{r_j^{(Z)}\right\}_{j=1}^{n-m+1}\right) \in_R \mathbb{Z}_q^m \times \mathbb{Z}_q^m \times \mathbb{Z}_q^{n-m+1}$$

uniformly at random. From this an appropriate $A = (\{P_i^{(A)}\}_{i=1}^{m}; \{H_j^{(A)}\}_{j=1}^{n-m+1})$ is computed as:

$$\left\{P_i^{(A)}\right\}_{i=1}^{m} \leftarrow \left\{P_i^{-E} \cdot E_{pk}\left(p_i^{(Z)}; r_{p_i}^{(Z)}\right)\right\}_{i=1}^{m}$$

and

$$\left\{H_j^{(A)}\right\}_{j=1}^{n-m+1} \leftarrow \left\{(H_j)^{-E} \cdot H_{T_1,\ldots,T_n}^{(j)}\left(\left\{p_i^{(Z)}\right\}_{i=1}^{m}, r_j^{(Z)}\right)\right\}_{j=1}^{n-m+1}.$$

In a real protocol execution, $P_i^{(A)}$ is a fresh encryption of a uniformly random value; this is also the case here, due to the multiplication by $E_{pk}(p_i^{(Z)}; r_{p_i}^{(Z)})$.

Regarding the $H_j^{(A)}$, note that if all $T_k, k \in \{j, j+1, \ldots, j+m-1\}$, are encryptions of $(q+1)/2$, then evaluating $H_{T_1,\ldots,T_n}^{(j)}$ results in an encryption of 0. Thus, in this case, $H_j^{(A)}$ sent in the protocol execution is simply a uniformly random encryption of 0. This is the same for the simulation, assuming that H_j is indeed in the image of $H_{T_1,\ldots,T_n}^{(j)}$.

If at least one $t_k \neq (q+1)/2$, then evaluating $H_{T_1,\ldots,T_n}^{(j)}$ on a uniformly random input results in a uniformly random encryption of a uniformly random message. Since the p_i are uniformly random, then at least one

$$(T_{j+i-1})^{-2p_i} \cdot E_{pk}(1;0)^{p_i}$$

is an encryption of a uniformly random value, $p_i(1-2t_{j+i-1})$. The encryption is rerandomized with the multiplication by $E_{pk}(0; r_j)$. Thus in an honest protocol execution, $H_j^{(A)}$ is a uniformly random encryption of a uniformly random message; this is also the case in the simulation due to the multiplication by the uniformly random ${H_{T_1,\ldots,T_n}}^{(j)}(\{p_i^{(Z)}\}_{i=1}^{m}, r_j^{(Z)})$.

Finally, since there is only a single witness, Z is exactly the response that an honest prover would send on challenge E. To see this, note that the P_i are encryptions, hence they fix the values p_i and r_{p_i}. At this point only a single value is possible for each r_j, since the remainder of the ${H_{T_1,\ldots,T_n}}^{(j)}$-functions is simply rerandomization of fixed encryptions. The fact that these fixed encryptions are unknown to the verifier changes nothing. □

Complexity Communication complexity of $\pi_{\text{h-proof}}$ is clearly $O(m+n)$. The prover sends something in the image of $H_{T_1,\ldots,T_n}$ as well as a preimage, and both are linear in m and n, while the verifier sends a single $\mathbb{Z}_q$ element. Regarding computation, the most expensive step is the evaluation of $H_{T_1,\ldots,T_n}$, which both parties must do. This requires computing $O(m)$ encryptions—the P_i—as well as $n-m+1$ evaluations of functions $H_{T_1,\ldots,T_n}^{(j)}$. It is immediate to see that the former requires $O(m)$ multiplications and exponentiations. The latter on the other hand is more expensive. Each $H_{T_1,\ldots,T_n}^{(j)}$ consist of the rerandomization of the product of m exponentiations. Since there are $n-m+1$ of these, overall $O(nm)$ multiplications and exponentiations are needed.

References

[1] M. Abe, R. Cramer, S. Fehr, Non-interactive distributed-verifier proofs and proving relations among commitments, in *ASIACRYPT* (2002), pp. 206–223

[2] C. Allauzen, M. Crochemore, M. 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* (Springer, London, 1999), pp. 295–310

[3] A. Amir, M. Lewenstein, E. Porat, Faster algorithms for string matching with mismatches. In *SODA*, San Francisco, California (2000), pp. 794–803

[4] G. Aggarwal, N. Mishra, B. Pinkas, Secure computation of the k'th-ranked element, in *EUROCRYPT* (2004), pp. 40–55

[5] P. Baldi, R. Baronio, E. De Cristofaro, P. Gasti, G. Tsudik, Countering GATTACA: efficient and secure testing of fully-sequenced human genomes, in *ACM Conference on Computer and Communications Security* (2011), pp. 691–702

[6] J. Baron, K. El Defrawy, K. Minkovich, R. Ostrovsky, E. Tressler, 5pm: Secure pattern matching, in *SCN* (2012), pp. 222–240

[7] D. Beaver, Foundations of secure interactive computing. In *CRYPTO*, Springer, London (1992), pp. 377–391

[8] S. Bayer, J. Groth, Efficient zero-knowledge argument for correctness of a shuffle, in *EUROCRYPT* (2012), pp. 263–280

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

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

[11] F. Brandt, Efficient cryptographic protocol design based on distributed el gamal encryption, in *ICISC* (2005), pp. 32–47

[12] R. Canetti, Security and composition of multi-party cryptographic protocols. *J. Cryptol.* **13**, 143–202 (2000)

[13] R. Canetti, Universally composable security: A new paradigm for cryptographic protocols, in *FOCS* (2001), pp. 136–145

[14] R. Cramer, R. Gennaro, B. Schoenmakers, A secure and optimally efficient multi-authority election scheme, in *EUROCRYPT* (1997), pp. 103–118

[15] D. Chaum, T.P. Pedersen, Wallet databases with observers. In *CRYPTO'92: Proceedings of the 12th Annual International Cryptology Conference on Advances in Cryptology* (Springer, London, 1993). pp. 89–105

[16] M. Chase, I. Visconti, Secure database commitments and universal arguments of quasi knowledge, in *CRYPTO* (2012), pp. 236–254

[17] T. ElGamal, A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Trans. Inf. Theory* **31**(4), 469–472 (1985)

[18] M.J. Fischer, M.S. Paterson, String-matching and other products, in *Complexity of Computation*, SIAM-AMS, vol. 7 (1974), pp. 113–125

[19] K.B. Frikken, Practical private DNA string searching and matching through efficient oblivious automata evaluation, in *DBSec* (2009), pp. 81–94

[20] R. Gennaro, C. Hazay, J.S. Sorensen, Automata evaluation and text search protocols with simulation based security, in *Public Key Cryptography* (2010), pp. 145–160

[21] S. Goldwasser, L.A. Levin, Fair computation of general functions in presence of immoral majority, in *CRYPTO* (Springer, London, 1991), pp. 77–93

[22] O. Goldreich, S. Micali, A. Wigderson, How to play any mental game, in *STOC'87: Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing* (ACM, New York, 1987), pp. 218–229

[23] O. Goldreich, *Foundations of Cryptography: Volume 2, Basic Applications* (Cambridge University Press, New York, 2004)

[24] C. Hazay, Y. Lindell, Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries, in *TCC* (2008), pp. 155–175

[25] C. Hazay, Y. Lindell, *Efficient Secure Two-Party Protocols—Techniques and Constructions* (Springer, Berlin, 2010)

[26] C. Hazay, T. Toft, Computationally secure pattern matching in the presence of malicious adversaries, in *ASIACRYPT* (2010), pp. 195–212
[27] Y. Ishai, J. Kilian, K. Nissim, E. Petrank, Extending oblivious transfers efficiently, in *CRYPTO* (2003), pp. 145–161
[28] C.S. Iliopoulos, M. Sohel Rahman, Pattern matching algorithms with don't cares, in *SOFSEM* (2007), pp. 116–126
[29] A. Jarrous, B. Pinkas, Secure hamming distance based computation and its applications, in *ANCS*, vol. 5536 (2009), pp. 107–124
[30] J. Katz, L. Malka, Secure text processing with applications to private DNA matching, in *ACM Conference on Computer and Communications Security* (2010), pp. 485–492
[31] D.E. Knuth, J.H. Morris Jr., V.R. Pratt, Fast pattern matching in strings. *SIAM J. Comput.* **6**(2), 323–350 (1977)
[32] Y. Lindell, B. Pinkas, Secure two-party computation via cut-and-choose oblivious transfer, in *TCC* (2011), pp. 329–346
[33] P. Mohassel, S. Niksefat, S.S. Sadeghian, B. Sadeghiyan, An efficient protocol for oblivious DFA evaluation and applications, in *CT-RSA* (2012), pp. 398–415
[34] S. Micali, P. Rogaway, Secure computation (abstract), in *CRYPTO* (1991), pp. 392–404. This is preliminary version of unpublished 1992 manuscript
[35] G. Navarro, V. Mäkinen, Compressed full-text indexes. *ACM Comput. Surv.* **39**(1), 2 (2007)
[36] K.S. Namjoshi, G.J. Narlikar, Robust and fast pattern matching for intrusion detection, in *INFOCOM* (2010), pp. 740–748
[37] T. Nishide, K. Ohta, Multiparty computation for interval, equality, and comparison without bit-decomposition protocol, in *Public Key Cryptography* (2007), pp. 343–360
[38] M. Osadchy, B. Pinkas, A. Jarrous, B. Moskovich, Scifi—a system for secure face identification, in *IEEE Symposium on Security and Privacy* (2010), pp. 239–254
[39] P. Paillier, Public-key cryptosystems based on composite degree residuosity classes, in *EUROCRYPT* (1999), pp. 223–238
[40] T.P. Pedersen, Non-interactive and information-theoretic secure verifiable secret sharing, in *CRYPTO* (1991), pp. 129–140
[41] C.P. Schnorr, Efficient identification and signatures for smart cards, in *CRYPTO'89: Proceedings on Advances in Cryptology* (Springer, New York, 1989), pp. 239–252
[42] T. Toft, Sub-linear, secure comparison with two non-colluding parties, in *Public Key Cryptography* (2011), pp. 174–191
[43] J.R. Troncoso-Pastoriza, S. Katzenbeisser, M. Celik, Privacy preserving error resilient DNA searching through oblivious automata, in *CCS'07: Proceedings of the 14th ACM Conference on Computer and Communications Security* (ACM, New York, 2007), pp. 519–528
[44] B. Terelius, D. Wikström, Proofs of restricted shuffles, in *AFRICACRYPT* (2010), pp. 100–113
[45] D. Vergnaud, Efficient and secure generalized pattern matching via fast Fourier transform, in *AFRICACRYPT* (2011), pp. 41–58
[46] A.C.-C. Yao, How to generate and exchange secrets, in *SFCS'86: Proceedings of the 27th Annual Symposium on Foundations of Computer Science*, (IEEE Computer Society, Washington, 1986), pp. 162–167