Open access • Journal Article • DOI:10.3233/JCS-130481

5PM: Secure pattern matching — Source link

Joshua W. Baron, Karim El Defrawy, Kirill Minkovich, Rafail Ostrovsky ...+1 more authors

Institutions: HRL Laboratories, University of California, Los Angeles

Published on: 01 Sep 2013 - Journal of Computer Security (IOS Press)

Topics: Pattern matching, Substring, Secure two-party computation, Hamming distance and Communication complexity

5PM: Secure Pattern Matching[⋆]

Joshua Baron,[2] Karim El Defrawy,[2] Kirill Minkovich,[2] Rafail Ostrovsky,[1] and Eric Tressler[2]

[1] Departments of Mathematics and Computer Science, UCLA, Los Angeles, CA, USA 90095
[2] Information and System Sciences Laboratory, HRL Laboratories, LLC, Malibu, CA, USA, 90265
{jwbaron,kmeldefrawy,kminkovich,eptressler}@hrl.com, rafail@cs.ucla.edu

Abstract. In this paper we consider the problem of secure pattern matching that allows single-character wildcards and substring matching in the malicious (stand-alone) setting. Our protocol, called 5PM, is executed between two parties: Server, holding a text of length n, and Client, holding a pattern of length m to be matched against the text, where our notion of matching is more general and includes non-binary alphabets, non-binary Hamming distance and non-binary substring matching.

5PM is the first secure expressive pattern matching protocol designed to optimize round complexity by carefully specifying the entire protocol round by round. In the malicious model, 5PM requires $O((m+n)k^2)$ bandwidth and $O(m+n)$ encryptions, where m is the pattern length and n is the text length. Further, 5PM can hide pattern size with no asymptotic additional costs in either computation or bandwidth. Finally, 5PM requires only two rounds of communication in the honest-but-curious model and eight rounds in the malicious model. Our techniques reduce pattern matching and generalized Hamming distance problems to a novel linear algebra formulation that allows for generic solutions based on any additively homomorphic encryption. We believe our efficient algebraic techniques are of independent interest.

1 Introduction

Pattern matching is fundamental to computer science. It is used in many areas, including text processing, database search [1], networking and security applications [2] and recently in the context of bioinformatics and DNA analysis [3,4,5]. It is a problem that has been extensively studied, resulting in several efficient (although insecure) techniques to solve its many variations, e.g., [6,7,8,9]. The most common interpretation of the pattern matching problem is the following: given a finite alphabet Σ, a text $T \in \Sigma^n$ and a pattern $p \in \Sigma^m$, the *exact pattern matching decision problem* requires one to decide whether or not a pattern appears in the text. The *exact pattern matching search problem* requires finding all indices i of T (if any) where p occurs as a substring starting at position i. If we denote by T_i the ith character of T, the output should be the set of matching positions $MP \coloneqq \{i \mid p \text{ matches } T \text{ beginning at } T_i\}$. The following generalizations of the exact matching problem are often encountered, where the output in all cases is the set MP:

- *Pattern matching with single-character wildcards*[1]*:* There is a special character "$*$" $\notin \Sigma$ that matches any single-character of the alphabet, where $p \in \{\Sigma \cup \{*\}\}^m$ and $T \in \Sigma^n$. Using such

[⋆] This work was done while the first author was at UCLA. The work of the first and fourth author is supported in part by NSF grants CCF-0916574, IIS-1065276, CCF-1016540, CNS-1118126, CNS-1136174, and by US-Israel BSF grant 2008411. It was also supported by the OKAWA Foundation Research Award, IBM Faculty Research Award, Xerox Faculty Research Award, B. John Garrick Foundation Award, Teradata Research Award and Lockheed-Martin Corporation Research Award. The material contained herein is also based upon work supported by the Defense Advanced Research Projects Agency through the U.S. Office of Naval Research under Contract N00014-11-1-0392. The views expressed are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. The authors would like to thank Jonathan Katz, Sky Faber and Matt Cheung for helpful discussions and comments.

[1] Such wildcards are also called "do not cares" and "mismatches" in the literature.

Paper	NB Hamming Distance	Exact Matching	Wildcard Matching	NB Substring Matching	Security
[13]	No	Yes	No	No	HBC/M
[14]	Yes*	Yes	Yes	Yes*	HBC/M
[15]	Yes	No**	No**	No**	HBC
5PM	Yes	Yes	Yes	Yes	HBC/M

Table 1. Comparison of previous protocol functionality, NB=non-binary HBC=honest but curious, M=malicious, *=using unary encoding and additional tools, **=can be extended

a "wildcard" character allows one pattern to be specified that could match several sequences of characters. For example the pattern "$TA*$" would match any of the following character sequence in a text[2]: TAA, TAC, TAG, and TAT.

- *Substring pattern matching:* Fix some $l \leq m$; a match for p is found whenever there exists in T an m-length string that differs in l characters from p (i.e., has Hamming distance l from p). For example, the pattern "TAC" has $m = 3$. If $l = 1$, then any of the following words will match: $*AC$, $T*C$, or $TA*$; note that this is an example of non-binary substring matching.

A secure version of pattern matching has many applications. For example, secure pattern matching can help secure databases that contain medical information such as DNA records, while still allowing one to perform pattern matching operations on such data. The need for privacy-preserving DNA matching has been highlighted in recent papers [10,11,12]. In addition to the case of DNA matching, where substring matching may be particularly useful, Hamming distance-based approximate matching has also been demonstrated in the case of secure facial recognition [3]. We note that both of these settings require computation over non-binary alphabets.

1.1 Our Contributions

This paper presents **5**ecure **P**attern **M**atching (or 5PM), a new protocol for arbitrary alphabets that addresses, in addition to exact matching, more expressive search queries including single-character wildcards and substring pattern matching, and also provides the ability to hide pattern length.

5PM has communication complexity sublinear in circuit size (as opposed to general MPC, which has communication complexity linear in circuit size) to securely compute non-binary substring matching in the malicious model. In addition, our extension of Hamming distance computation to substring matching has minimal overhead; our protocol makes a single computation pass per text element, even for multiple Hamming distance values, and therefore is able to securely compute non-binary substring matching efficiently (see Table 1 for a comparison of protocol functionality and Tables 2 and 3 for a comparison of protocol overhead).

5PM performs exact, single-character wildcards, and substring pattern matching in the honest-but-curious and malicious (static corruption) models. Our malicious model protocol requires $O((m+n)k^2)$ bandwidth complexity. Further, our protocol can be specified to require two (one-way) rounds of communication in the semi-honest model and eight (one-way) rounds of communication in the malicious model.

We construct our protocols by reducing the problems of Hamming distance and pattern matching, including single-character wildcards and substring matching, to a sequence of linear operations.

[2] Here and throughout, we use the DNA alphabet ($\Sigma = \{A, C, G, T\}$) for examples.

Paper	Encryptions	Exponentiations	Multiplications	Bandwidth	Rds
[16]	$O(mn)$	$O(mn)$	$O(mn)$	$O(mnk^2)$	$O(1)$
[14]	$O(n+m)$	$O(n \log m)$	$O(nm)$	$O((n+m)k^2)$	$O(1)$
5PM	$O(n+m)$	$O(nm)$	$O(nm)$	$O((n+m)k^2)$	8

Table 2. Detailed comparison with [14] and [16] for single-character wildcards and substring matching in malicious model with text length=n, pattern length=m, security parameter=k, rounds=Rds.

Paper	Encryptions	Exponentiations	Multiplications	Bandwidth	Rds
[15]	$O(n+m)$	$O(nm)$	$O(nm)$	$O((nm)k)$	$O(1)$
5PM	$O(n+m)$	$O(n+m)$	$O(nm)$	$O((n+m)k)$	2

Table 3. Detailed comparison with [15] for non-binary substring matching in HBC model with text length=n, pattern length=m, security parameter=k, rounds=Rds.

We then rely on the observation that these linear operations, such as the inner products and matrix multiplication, can be efficiently computed in the malicious model using additively homomorphic encryption schemes.

The security requirements (informally) dictate that the party holding the text learns nothing except the upper bound on the length of the pattern, while the party holding the pattern only learns either a binary (yes/no) answer for the decision problem or the matching positions (if any), and nothing else.

1.2 Comparison to Previous Work

Exact Matching. In the exact pattern matching setting, the algorithm of Freedman, Ishai, Pinkas and Reingold [13] achieves polylogarithmic overhead in m and n and polynomial overhead in security parameters in the honest-but-curious setting. Using efficient arguments [17,18] with the modern probabilistically checkable proofs (PCPs) of proximity [19], one can extend (at least asymptotically) their results to the malicious (static corruption) model. However, the protocol in [13] works only for exact matching and does not address more general problems, including single-character wildcards and substring matching, which are the main focus of our work. Other protocols that address secure exact matching (and not wildcard or substring matching) are [12,20,21,22,23,11]; of these, only [22] obtains (full) security in the malicious setting. We note that [23] is more efficient than [13], but only in the random oracle model; here, we are interested in standard security models.

Single-Character Wildcards and Substring Matching. Recently, Vergnaud [14] built on the work of Hazay and Toft [16] to construct an efficient secure pattern matching scheme for wildcard matching and substring matching (requiring t runs over the preliminary matching result to search for t different Hamming distance values, which is also required by 5PM) in the malicious adversary model. More specifically, [14,16] take advantage of the fact that $(p_i - t_i)^2$ equals 0 if binary values p_i and t_i are equal and 1 if they are not equal; therefore, binary Hamming distance can essentially be computed by counting the number of 1s in a particular polynomial-based computation. However, when p_i and t_i are non-binary, it is unknown how to obtain 0 when p_i and t_i equal, and 1 (or some other fixed value) when they are not equal using oblivious polynomial evaluations.

However, non-binary elements can be computed by unary encoding; that is, an element $\alpha \in \Sigma$ can be encoded as an element $\alpha' \in \{0,1\}^{|\Sigma|}$ with all 0s except for a single 1 in the place representing

α (lexicographically). There are two subtleties of such an approach. The first is that if $\alpha \neq \beta$, then α' and β' will have Hamming distance 2 instead of 1; the second is, in the malicious case, zero knowledge proofs are needed to demonstrate that α' is well formed.

[14] requires $O(m+n)$ encryptions, $O(n \log m)$ exponentiations, $O(nm)$ multiplications (of encrypted elements), and $O(n+m)$ bandwidth, all in a constant number of rounds. By contrast, 5PM has the same overhead except for $O(nm)$ exponentiations (see Table 2). However, our work is of interest for several reasons. The first is that we have implemented our protocol and believe it to be more efficient (additional work is needed on this front). The second is that our techniques are of independent interest and may be extended to additional functionalities. Finally, the protocol presented here is fully specified; by contrast, additional work is needed to transform the work of [14] into a protocol that can support non-binary alphabets for substring matching or to calculate Hamming distance in the malicious case.

Non-binary Hamming Distance. Jarrous and Pinkas [15] gave the first construction of a secure protocol for computing non-binary Hamming distances. In order to count the non-binary mismatches, they leverage 1-out-of-2 oblivious transfers. 5PM can also compute non-binary Hamming distance even when the text and pattern have the same length (and where the output is not blinded to only reveal whether or not a pattern match occurred). We note that [15] can be used to implement exact and substring matching with additional tools to blind Hamming distance output (for instance, see [14]). [15], to compare two strings of length n, requires $O(n)$ 1-out-of-2 OTs, $O(n)$ multiplications of encryptions and $O(nk)$ bandwidth, while 5PM requires $O(n)$ exponentiations (which require less computation than OTs), $O(n^2)$ multiplications, and $O(nk)$ bandwidth. The advantage of 5PM over [15] is twofold: the first is that 5PM is proven secure in the malicious model while [15] is not; the second is that 5PM, in both the honest-but-curious and malicious models, amortizes well in the substring matching setting, while [15] does not amortize because it cannot reuse OT outputs to compute substring matching (see Table 3).

Other Techniques. In the most general case, secure exact, approximate and single-character wildcards pattern matching is an instance of general secure two-party computation techniques (for instance, [24,25,26,27]). All of these schemes have bandwidth and computational complexity at best linear in the circuit size. For instance, a naive implementation of Yao [24] requires bandwidth $O(mn)$ in the security parameter. In contrast, we aim for a protocol where circuit size is $O(mn)$, yet we achieve communication complexity of $O(m+n)$.

Finally, we observe that with the construction of fully homomorphic encryption (FHE) schemes [28], the following "folklore" construction can be executed for any pattern matching algorithm: Client encrypts its pattern using an FHE scheme and sends it to Server. Server applies the appropriate pattern matching circuit to the encrypted pattern (where the circuit output is a *yes/no* indicating whether a match exists or not), and sends the FHE circuit output to Client. Client decrypts to obtain the answer. Such a scheme requires $O(m)$ bandwidth, but since FHE schemes are not yet practical, we view the 5PM protocol outlined here as an efficient and practical solution to secure pattern matching with single-character wildcards and substring matching.

2 Preliminaries

The rationale behind our secure 5PM protocol is based on a modification of an insecure pattern matching algorithm (IPM) [29] that can perform exact matching, exact matching with single-character wildcards, and substring matching within the same algorithm. In Section 3.1, we show how our modified algorithm can be reduced to basic linear operations whose secure and efficient evaluation allows us to obtain our 5PM protocol.

2.1 Insecure Pattern Matching (IPM) Algorithm

To illustrate how our modified algorithm works, we begin by describing how it performs exact matching; we then show how it handles single-character wildcards and substring matching.

2.1.1 Exact Matching. IPM involves the following steps:

a. *Inputs*: An alphabet Σ, a text $T \in \Sigma^n$ and a pattern $p \in \Sigma^m$.

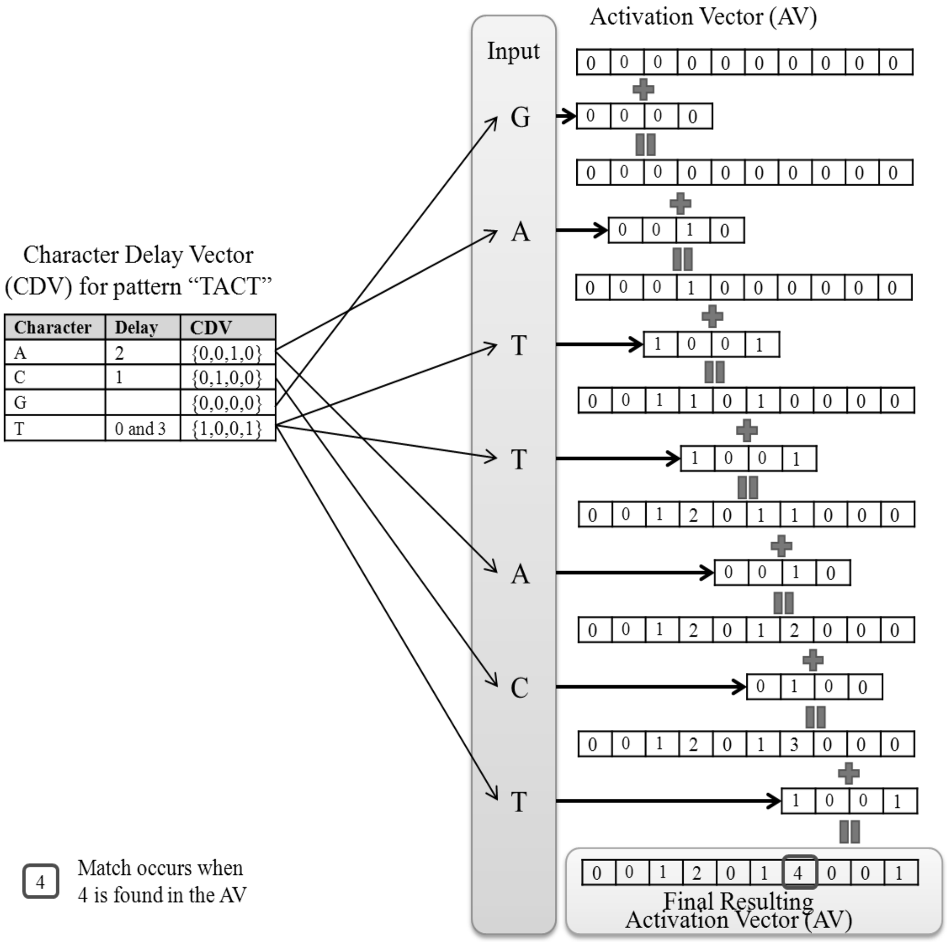
b. *Initialization*: For each character in Σ, the algorithm constructs a vector, here termed a **Character Delay Vector** (CDV), of length equal to the pattern length, m. These vectors are initialized with zeros. For example, if the pattern is: "$TACT$" over $\Sigma = \{A, C, G, T\}$, then the CDVs will be initialized to: $CDV(A) = [0, 0, 0, 0]$, $CDV(C) = [0, 0, 0, 0]$, $CDV(G) = [0, 0, 0, 0]$ and $CDV(T) = [0, 0, 0, 0]$.

c. *Pattern preprocessing*: For each pattern character p_i ($i \in \{1, ..., m\}$), a delay value, $d^r_{p_i}$, is computed to be the number of characters from p_i to the end of the pattern, i.e., $d^r_{p_i} = m - i$ for the rth occurrence of p_i in p. The $d^r_{p_i}$th position of $CDV(p_i)$ is set to 1. For example, the CDVs of "$TACT$" would be:
$CDV(A) = [0, 0, 1, 0]$ because $d^1_A = 4 - 2 = 2$
$CDV(C) = [0, 1, 0, 0]$ because $d^1_C = 4 - 3 = 1$
$CDV(G) = [0, 0, 0, 0]$ because $G \notin p$
$CDV(T) = [1, 0, 0, 1]$ because $d^1_T = 4 - 4 = 0$ and $d^2_T = 4 - 1 = 3$

d. *Matching pass and comparison with pattern length*: A vector of length n called the **Activation Vector (**AV**)** is constructed, and its elements are initialized with zeros. For each input text character T_j, $CDV(T_j)$ is added element-wise to the AV from position j to position $\min(n, j+m-1)$. To determine if there was a pattern match in the text, after these operations the algorithm checks (when $j \geq m$) if $AV_j = m$. If so, then the match started at position $j - m + 1$. The value $j - m + 1$ is added to the set of matching positions (MP). *Note that* $n - AV_j$ *is the non-binary Hamming distance of the pattern and the text starting at position* $j - m + 1$.

The intuition behind the algorithm is that when an input text character matches a character in the pattern, the algorithm *optimistically* assumes that the following characters will correspond to the rest of the pattern characters. It then adds a 1 at the position in the activation vector several steps ahead, where it would expect the pattern to end (if the character appears in multiple positions in the pattern, it adds a 1 to all the corresponding positions where the pattern might end). If all subsequent characters are indeed characters in the pattern, then at the position where a pattern would end the number of added 1s will sum up to the pattern length; otherwise the sum will be strictly less than the pattern length. This algorithm does not incur false positives and always indicates when (and where) a pattern occurs if it exists, as shown in [29].

Insecure Pattern Matching (IPM) Example. The pattern to be matched is "*TACT*" and the text is "*GATTACT...* ." The first step is to construct the *CDV* using the delays for the characters of the pattern "*TACT*." Delays for "*T*" will be 3 and 0, for "*A*" they will be 2, and for "*C*" they will be 1. These delays are then converted to *CDV*s as shown in Figure 1. The activation vector will be initialized to all zeros. The characters of the text are then considered one at a time. For each input text character ($T[j]$) at position j in text, the following steps have to be taken: (1) retrieve $CDV[T[j]]$; (2) add elements of $CDV[T[j]]$ to elements of activation vector from position j to $j+m-1$; and (3) check if $AV[j]$ is equal to the pattern length ($|TACT| = m = 4$).



Character Delay Vector (CDV) for pattern "TACT"

Character	Delay	CDV
A	2	{0,0,1,0}
C	1	{0,1,0,0}
G		{0,0,0,0}
T	0 and 3	{1,0,0,1}

Fig. 1. Example of IPM's operation

2.1.2 Single-Character Wildcards, Pattern Hiding and Substring Matching. Single-character wildcards can be handled in IPM by representing a single-character wildcard with a special character,"$*$" which is not in the text alphabet. When "$*$" is encountered in the pattern preprocessing phase it is ignored; i.e., no 1s are added to any *CDV*. Additionally, at the last step when elements of the AV are searched in the comparison phase, the threshold value being used for comparison will be $m - l$ instead of m, where l is the number of occurrences of "$*$" in the pattern. The intuition behind single-character wildcards is that by reducing the threshold for each wildcard, the algorithm implicitly skips matching that position in the text, allowing that position of the pattern to correspond to any character. This operation does not incur any false positives for the same reason that the exact matching IPM algorithm does not: there, for each pattern p, there is only one encoding into CDVs and only one sequence of adding CDVs as one moves along the text that could add up to m. The same reasoning holds when "$*$" is present in p (except that the sequence adds to $m - l$).

We note that by using single-character wildcards, one can always hide pattern length by setting p' as the concatenation of p and a string of $n - m$ wildcards, $*^{n-m}$, and using p' to execute pattern matching for p.

Substring matching, or matching text substrings of Hamming distance $m - l$ from the pattern, is handled similarly to single-character wildcards; the threshold value being compared against in the AV is decreased to $m - l$. For further details, we refer the reader to [29].

2.2 Preliminary Cryptographic Tools

This section outlines preliminary cryptographic tools required for our protocols. For $x, y \in \mathbb{Z}_q^n$, we define the inner product of x and y over $\mathbb{Z}_q$, denoted $\langle x, y \rangle$, as $\sum x_i y_i \bmod q$.

Additively Homomorphic Encryption: We make use of additively homomorphic semantically secure encryption schemes. For concreteness, in the rest of this paper we concentrate on the additively homomorphic ElGamal encryption scheme whose security depends on the Decisional Diffie-Hellman (DDH) computational hardness assumption. An additively homomorphic ElGamal encryption scheme [30] is instantiated by choosing a group of appropriate prime order q, $\mathbb{G}_q$, with generator g, and setting the secret key to be $x \in \mathbb{Z}_q$ and the public key to be $(g, h = g^x)$. To encrypt a message m one chooses a uniformly random $r \in \mathbb{Z}_q$ and computes $(g^r, g^m h^r)$. To decrypt a pair (α, β), one computes $\log_g \frac{\beta}{\alpha^x}$. It is important to note that for additive ElGamal the decryptor has to both decrypt and also compute a discrete logarithm to discover the message. However, our scheme only requires a determination of whether an encrypted value is of a 0 or not, which can be accomplished without computing logarithms.

Threshold Encryption: The malicious model version of 5PM requires an additively homomorphic, semantically secure, threshold encryption scheme [31]. While we use threshold ElGamal, in practice, any scheme is acceptable if it satisfies the required properties and supports the needed zero-knowledge arguments. Threshold ElGamal, in the two-party case, can be informally defined as follows [32]: party P_1 has share x_1 and party P_2 has share x_2. The parties jointly set the secret key to be $x = x_1 + x_2$ (this can be performed without revealing x_1 and x_2, see subprotocol π_{encr} in Section 3.3). Without loss of generality, P_1 partially decrypts (α, β) by sending $(\alpha, \frac{\beta}{\alpha^{x_1}})$ to P_2, who fully decrypts (α, β) by computing $\frac{\beta}{\alpha^{x_1}\alpha^{x_2}} = \frac{\beta}{\alpha^{x_1+x_2}}$. We denote the partial decryption algorithm for party P_i as D_{P_i}.

Commitment Schemes: For the malicious model protocol, we will make use of perfectly hiding, computationally binding commitment schemes (for further discussion, see [33]). The Pedersen commitment scheme [34] is a well-known example of such a commitment scheme: for a multiplicative group of prime order q, $\mathbb{G}_q$ and for fixed generators $g, h \in \mathbb{G}_q$, commitment to message s using randomness r is $g^s h^r = comm(g, h, r, s)$.

Zero-Knowledge Arguments of Knowledge: In order to construct a protocol that guarantees that each party behaves properly even in the malicious setting, we use efficient interactive zero-knowledge arguments of knowledge (ZK-AoKs). For further details, see Section 4.

2.3 Computing Linear Operations Using Additively Homomorphic Encryption Schemes.

Our secure pattern-matching protocol relies on the following observations about linear operations and additively homomorphic encryption schemes. In what follows, let E be the encryption algorithm for an additively homomorphic encryption scheme for key pair (pk, sk). Suppose the plaintext group G can be expressed as $\mathbb{Z}_n$ for some $n \in \mathbb{N}$; in particular, G is a ring. Let $M_{a,b}(G)$ denote the set of matrices of size $a \times b$ with entries in G.

2.3.1 Matrix Multiplication. Consider two matrices, A and B, where $A \in M_{k,l}(G)$ and $B \in M_{l,m}(G)$. Suppose that P_1 possesses pk, $E_{pk}(A)$, the entry-wise encryption of A, and also the unencrypted matrix B. Then P_1 can compute $E_{pk}(A \cdot B)$, the encryption of the multiplication of A and B under the same pk. Such an operation is possible because one can obtain an encryption of the inner product over G of an unencrypted vector $(x_1, ..., x_m)$ with an encrypted vector $(E(y_1), ..., E(y_m))$ by computing $\mathit{\Pi} E(y_i)^{x_i} = E(\sum x_i y_i))$.

2.3.2 Matrix Operators. Consider a matrix $A \in M_{k,l}(G)$. One can construct a $k \times (k + l - 1)$ matrix A' by initializing A' as a matrix with all 0s and then, for each row $1 \leq i \leq k$, setting $(A'(i, i), ..., A'(i, i + l - 1)) = (A(i, 1), ..., A(i, l))$. We denote such a function by $A' \leftarrow \mathit{Stretch}(A)$, and note that since this function is a linear operator, it can be computed using matrix multiplication. We observe that for any encryption scheme E, $E(\mathit{Stretch}(A)) = \mathit{Stretch}(E(A))$, when E is applied to each entry in A.

Consider a matrix $A \in M_{k,l}(G)$. We denote $\mathit{Cut}(A, j)$ as the matrix $A' \in M_{k,l-2j+2}$ such that for $1 \leq a \leq k$, $1 \leq b \leq l - 2j + 2$, $A'(a, b) = A(a, b + j - 1)$. In particular, such a function outputs the middle $l - 2j + 2$ columns of $M_{k,l}$. We note that Cut is a simple projection operator and is also computable by matrix multiplication. We observe that for any encryption scheme E, $E(\mathit{Cut}(A, j)) = \mathit{Cut}(E(A), j)$

Finally, consider a matrix $A \in M_{k,l}(G)$. We denote by $\mathit{ColSum}(A)$ the function that takes as input A and outputs a $1 \times l$ vector whose ith entry is the sum of all entries in the i column of A. In particular, $\mathit{ColSum}(A) = [1....1] \cdot A$. We observe that for any additively homomorphic encryption scheme E, $\mathit{ColSum}(E(A)) = E(\mathit{ColSum}(A))$.

Since we will be composing these functions, a shorthand for their composition will be convenient. For matrices $A \in M_{k,l}(G)$ and $B \in M_{l,m}(G)$, we denote the composition function $\mathit{ColSum}(\mathit{Cut}(\mathit{Stretch}(A \cdot B), j))$ by $PM_{5PM}(A, B, j)$.

2.3.3 Searching an Encrypted Vector, π_{VFind}. Suppose party P_1 possesses (pk, sk) for an additively homomorphic encryption algorithm E, and a single value $m \in G$ and P_2 possesses a vector of l distinct encryptions $E_{pk}(vec)$, where $vec = (x_1, \ldots, x_l) \in G^l$. Then P_1 can determine if $E(vec)$ contains an encryption of m while learning nothing else about vec, while P_2 cannot learn m, through the following protocol π_{VFind}:

(a) P_1 computes $E(-m)$ from $-m$. P_1 sends $E(-m)$ to P_2.

(b) P_2 computes $E(vec')$ by multiplying (via the group operation of the ciphertext space) $E(-m)$ to each encrypted entry in $E(vec)$. Note that an entry in $E(vec')$ will be an encryption of 0 if and only if one of the encryptions of $E(vec)$ was an encryption of m. P_2 computes $E(vec^r)$ from $E(vec')$ by exponentiating each encrypted entry of $E(vec')$ by an (independent) random exponent. P_2 sends $E(vec^r)$ to P_1.

(c) P_1 decrypts $E(vec^r)$ to obtain vec^r; if a 0 exists at position i, the ith position of $E(vec)$ is $E(m)$.

Note that if P_2 wishes to hide the position of $E(m)$ from P_1, P_2 could randomly permute the positions of $E(vec^r)$ and send the permuted vector to P_1.

2.3.4 Efficiently Determining Equality of Two Matrices, π_{VecEQ}. Suppose parties P_1 and P_2 have agreed upon an additively homomorphic threshold encryption scheme E_{th}. Further, suppose P_1 and P_2, possess encrypted matrices $E_{th}(A) \in M_{k,l}(G')$ and $E_{th}(B) \in M_{k,l}(G')$, respectively, where the message space G' is the group $\mathbb{Z}_q$, for a prime q. Let D_{P_i} denote the partial decryption algorithm of party P_i. P_1 and P_2 wish to determine if their encrypted matrices are equal without exchanging their decryptions. They can do so by hashing their encrypted matrices to a single group element and exchanging the outcome of the hashes. More specifically, an affine hash function $\mathbb{Z}_q^{kl} \to \mathbb{Z}_q$ can be specified by letting P_1 and P_2 jointly compute a uniformly random pair $(a, b) \in \mathbb{Z}_q^{kl} \times \mathbb{Z}_q$ using standard commitment techniques and setting the hash to $hf(x) = \langle x, a\rangle + b$, where $\langle \cdot, \cdot \rangle$ is the inner product over $\mathbb{Z}_q$ (here, we consider the matrices as kl-length strings). Note that such a hash function can be computed on encrypted strings because the encryption scheme is additively homomorphic. Denote by *comm* a (perfectly hiding, computationally binding) commitment scheme; in practice we use Pedersen commitments [34]. We denote the following subprotocol by π_{VecEQ}:

(a) P_1 selects $(a_1, b_1) \in \mathbb{Z}_q^{kl} \times \mathbb{Z}_q$ uniformly at random and computes $E_{th}(b_1)$. P_1 computes and sends $comm(a_1), comm(E_{th}(b_1)), comm(E_{th}(A))$ to P_2.

(b) P_2 selects $(a_2, b_2) \in \mathbb{Z}_q^{kl} \times \mathbb{Z}_q$ uniformly at random and computes $E_{th}(b_2)$. P_2 sends $a_2, E_{th}(b_2), E_{th}(B)$ to P_1.

(c) P_1 sets $a = a_1 + a_2$, $E_{th}(b) = E_{th}(b_1 + b_2)$ and computes $z_1 = E_{th}(\langle a, A\rangle + b)$, $z_2 = E_{th}(\langle a, B\rangle + b)$. P_1 decommits to a_1, $E_{th}(b_1)$ and $E_{th}(A)$ to P_2 and sends $D_{P_1}(z_1)$, $D_{P_1}(z_2)$ to P_2.

(d) P_2 aborts if it does not accept the decommitments, else P_2 sets $a = a_1 + a_2$, $E_{th}(b) = E_{th}(b_1 + b_2)$ and computes $z_1 = E_{th}(\langle a, A\rangle + b)$, $z_2 = E_{th}(\langle a, B\rangle + b)$. P_2 sends $D_{P_2}(z_1)$, $D_{P_2}(z_2)$, $D_{P_2}(D_{P_1}(z_1))$, and $D_{P_2}(D_{P_1}(z_2))$ to P_1.

(e) P_1 aborts if $D_{P_2}(D_{P_1}(z_1)) \neq D_{P_2}(D_{P_1}(z_2))$; otherwise P_1 sends $D_{P_1}(D_{P_2}(z_1))$ and $D_{P_1}(D_{P_2}(z_2))$ to P_2.

(f) P_2 aborts if $D_{P_1}(D_{P_2}(z_1)) \neq D_{P_1}(D_{P_2}(z_2))$.

The bandwidth complexity of π_{VecEQ} is dominated by the size of $E_{th}(A)$ (and $E_{th}(B)$). Only with probability $1/q$ will the decryptions equal each other when $A \neq B$ because the hash function is chosen uniformly at random. In the malicious case, arguments of consistency for correct partial decryptions will also be needed.

3 5PM Protocol

This section uses the above observations and cryptographic tools to construct the secure pattern-matching protocol (5PM). We develop π^H_{5PM} for the honest-but-curious adversary model and π^M_{5PM} for the malicious (static corruption) adversary model.

Notation	Description	Section
π^{H}_{5PM}	Pattern matching algorithm secure in HBC adversary model	3.2
π^{M}_{5PM}	Pattern matching algorithm secure in malicious adversary model	3.3
Key	Key generation algorithm for homomorphic encryption scheme	2.2
E	Homomorphic encryption algorithm	2.2
D	Decryption algorithm for encryption scheme E	2.2
D_{P_i}	Partial decryption algorithm for party P_i using E for threshold encryption	2.2
$Stretch(A)$	Function from n by m matrices to n by $n+m-1$ matrices that “stretches” rows of A	2.3.2
$Cut(A, n)$	Function that outputs the first n columns of matrix A	2.3.2
$ColSum(A)$	Function that outputs the sums of the columns of matrix A	2.3.2
$PM_{5PM}(A, B, n)$	Composition function $ColSum(Cut(Stretch(A \cdot B), n))$	2.3.2
π_{VFind}	Two-party protocol that determines if encryption of P_1's value exists in P_2's encrypted vector	2.3.3
Gen_{CDV}	Algorithm with input of a pattern $p \in (\Sigma \cup \{*\})^m$ outputs $\|\Sigma\| \times m$ matrix M_{CDV}	3.1
Gen_T	Algorithm that on input of a text $T \in \Sigma^n$ outputs the $n \times \|\Sigma\|$ matrix M_T	3.1
π_{VecEQ}	Two-party protocol that determines equality of two encrypted vectors (π^{M}_{5PM})	2.3.4
A_{rel}	Arguments of consistency require for malicious protocol (π^{M}_{5PM})	3.3.2

Table 4. Notation used for 5PM protocols

3.1 Converting IPM to Linear Operations.

For a fixed alphabet Σ, a text $T \in \Sigma^n$, and pattern $p \in (\Sigma \cup \{*\})^m$, IPM can be represented in terms of linear operations described in Section 2.3 as follows:

a. The text T can be transformed into an $n \times |\Sigma|$ matrix, M_T. The transformation is performed by applying a unary encoding of alphabet characters to T, i.e., $M_T(i, T_i) = 1, \forall i \in \{1, ..., n\}$; all other entries in M_T are 0. We denote the algorithm that computes M_T from T as $M_T \leftarrow Gen_{M_T}(T)$.

b. The CDVs of alphabet characters can be grouped into a $|\Sigma| \times m$ matrix, M_{CDV}. This step is equivalent to constructing CDVs for alphabet characters (steps b and c in Section 2.1.1).We denote the algorithm that compute M_{CDV} from p as $M_{CDV} \leftarrow Gen_{M_{CDV}}(p)$.

c. Multiply M_T by M_{CDV} to obtain an $n \times m$ matrix $M_{T(CDV)}$ that represents T row-wise in terms of CDVs, where the ith row is $CDV(T_i)$. In reality, since M_T and M_{CDV} are 0/1 matrices, multiplication is more computationally expensive than necessary, and vectors can simply be selected (as shown in the IPM description in Section 2.1).

d. Compute $\overline{M}_{T(CDV)} = Stretch(M_{T(CDV)})$. This transformation, jointly with the previous step, constructs a matrix of CDVs where the ith row contains only $CDV(T_i)$, which starts in the ith position in the ith row (sets up step d in Section 2.1.1).

e. Compute $AV = ColSum(Cut(\overline{M}_{T(CDV)}, m))$ to obtain the final activation vector AV of length $n - m + 1$. Entries in AV are checked to see if any are equal to the threshold value m, or $m - l$ for single-character wildcards or substring matching (completes step d in Section 2.1.1).

A key observation is that if only one of M_T and M_{CDV} are encrypted, an encrypted activation vector, $E(AV)$, can be obtained by both parties, as shown in Sections 2.3.1 and 2.3.2.

3.2 Honest-Cut-Curious (HBC) 5PM Protocol

We begin by describing the intuition behind required modifications to secure IPM in the HBC adversary model. We then describe details of the HBC protocol, π_{5PM}^{H}.

3.2.1 Protocol Intuition. Let E be an additively homomorphic encryption scheme. When Client sends Server $E(M_{CDV})$, by the reasoning of Sections 2.3 and 3.1, Server can compute $E(AV)$, an encrypted activation vector, using only M_T and $E(M_{CDV})$. This is because the pattern matching operation can be reduced to a sequence of linear operations (namely matrix multiplication and the functions *Stretch*, *Cut*, and *ColSum*). Since Client sends only $E(M_{CDV})$ and $E(m-l)$, Server learns nothing about Client's pattern due to semantic security of the encryption scheme.

Next, Client, for pattern-matching threshold m (or $m-l$ in the single-character wildcards/substring matching case), executes π_{VFind} specified in Section 2.3.3, where Client uses $E(AV)$ to discover whether (and where) a pattern exists. Because of the security of π_{VFind}, Server does not learn m and Client learns nothing about $E(AV)$, other than whether or not (and where, if the pattern-matching locations are not hidden by Server) an encryption of m exists in $E(AV)$. In practice, Client sends $E(m)$ in the same (first) round as $E(M_{CDV})$, and Server's response to π_{VFind} occurs in the second round, concluding execution of the secure pattern-matching protocol.

Client		**Server**
Input: $p \in (\Sigma \cup \{*\})^m$		**Input:** $T \in \Sigma^n$
Initialization: 1) $(pk, sk) \leftarrow Key(1^k)$ 2) $M_{CDV} \leftarrow Gen_{CDV}(p)$ 3) $E(M_{CDV}) \leftarrow M_{CDV}$		
$E(-m+l) \leftarrow -m+l$	$\xrightarrow{4) E(M_{CDV});E(-m+l);pk_C}$	**Activation Vector Formation:** 5) $M_T \leftarrow Gen_T(T)$; $E(AV_S) \leftarrow PM_{5PM}(M_T, E(M_{CDV}), m)$ 6) $E(AV_S^r) \leftarrow \pi_{VFind}(E(AV_S, E(-m+l)))$ 7) Optional: Permute $E(AV_S^r)$
Decrypting and Determining Matches: 8) $MP = \{i \mid D(E(AV_S^r[i])) = 0\}$	$\xleftarrow{6) E(AV_S^r)}$	
Output: $MP = \{j \mid T_j ... T_{j+m-1} = p\}$		**Output:** Nothing

Table 5. Overview of 5PM protocol for HBC adversary model, π_{5PM}^{H}. See Table 4 for notation.

3.2.2 π_{5PM}^{H} Protocol Specification. Recall that over a specified alphabet Σ, Server holds text $T \in \Sigma^n$ and Client holds a pattern $p \in (\Sigma \cup \{*\})^m$. The output of Server is an encrypted activation vector $E(AV)$ of length n. We refer the reader to Sections 3.1 and 2.3.2 for the notation used here. The protocol operation is as follows:

a. Client computes $(sk, pk) \leftarrow Key(1^k)$ using the key generation algorithm of an additively homomorphic encryption scheme, E.
b. Client computes $M_{CDV} \leftarrow Gen_{CDV}(p)$. In the case where Client wishes to hide the length of p, Client computes M_{CDV} for the pattern p' equal to the concatenation of p with $*^{n-m}$.

c. Client encrypts M_{CDV} entry-wise using public key pk to obtain $E(M_{CDV})$.
d. Client sends $E(M_{CDV})$ and pk to Server. In addition, Client sends $E(-m)$ (or $E(-m+l)$ in the single-character wildcards or substring matching cases).
e. Server computes $M_T \leftarrow Gen_T(T)$. Server computes $E(AV) = E(PM_{5PM}(M_T, M_{CDV}, m))$, which is computed as specified in Sections 2.3.1 and 2.3.2.
f. Server executes round 2 of π_{VFind} (see Section 2.3.3) using $E(-m)$ and $E(AV)$. Server sends output of the subprotocol, denoted $E(AV_S^r)$, to Client.
g. Optional: Per π_{VFind}, Server randomly permutes $E(AV_S^r)$ to hide possible pattern match locations.
h. Client executes round 3 of π_{VFind} using $E(AV_S^r)$ to determine results of the pattern matching.

We note that π_{5PM}^H can perform substring matching for multiple substring lengths (such as a Hamming distance bound) simultaneously by sending multiple $E(m-l)$ values at step 6 in the specification shown in Table 5. Then, for each value of l, Server constructs a distinct $E(AV)$ and sends Client a distinct corresponding $E(AV_S^r)$ indicating matching locations for that l value. In particular, π_{5PM}^H does not require multiple independent protocol executions to compute substring matching for a range of substring length values. In addition, π_{5PM}^H can simply compute the Hamming distance of the pattern with each consecutive m positions of the text by simply not executing π_{VFind} and sending the output of the protocol at step 5, and Client can decrypt to obtain all of the Hamming distance values between the pattern and the text.

Theorem 1. *Given an additively homomorphic semantically secure encryption scheme over a prime-order cyclic group (Key, E, D), π_{5PM}^H is secure in the HBC model.*

See Section 7 for a detailed security proof.

3.3 Malicious Model 5PM Protocol

In this section we explain how to modify π_{5PM}^H to obtain a protocol, π_{5PM}^M, which is secure in the malicious (static corruption) model. We describe an instantiation of π_{5PM}^M based on additively homomorphic threshold ElGamal encryption (see Section 2.2) for concreteness; generalization to other encryption schemes follows provided they have efficient Σ protocols for the statements required here. First, we explain the intuition behind π_{5PM}^M. Second, we give interactive zero-knowledge consistency arguments that will be required. Finally, we divide π_{5PM}^M into six subprotocols and describe their construction and how they are combined into the final protocol π_{5PM}^M. In the interest of clarity and space, we leave the exact protocol specification and security proof to Sections 6 and 7, respectively. Note that this protocol, as shown in Section 2.1.2, can be modified to both hide pattern length (by using, for pattern p, the pattern p' equal to to p concatenated with $*^{n-m}$) and also to match against multiple substring values without multiple executions of the entire protocol (i.e., by sending multiple $E(m-l)$ values and computing a new activation vector for each value).

3.3.1 Protocol Intuition. The eight round protocol for the malicious model, π_{5PM}^M, consists of the following six subprotocols:

(a) π_{encr}: initializes an additively homomorphic threshold encryption scheme.
(b) $\pi_{S,AV}$: allows Server to construct an encrypted activation vector for Client's encrypted pattern and Server's text.
(c) $\pi_{C,AV}$: allows Client to construct an encrypted activation vector for Client's pattern and Server's encrypted text.

(d) π_{vec}: allows Client and Server to verify that their activation vectors are equal without revealing them.

(e) π_{rand}: allows Server to send an encryption of its randomized activation vector to Client.

(f) π_{ans}: shows to Client where the pattern matches the text (if at all).

The intuition behind constructing π^M_{5PM} is as follows: In π^H_{5PM}, only Server performs the computation to obtain the activation vector, AV. In the malicious setting, Client has to verify that Server correctly computed AV. Since Server performs $O(nm)$ multiplications when computing AV in π^H_{5PM}, requiring a zero-knowledge argument for each multiplication therefore would require bandwidth of at least $O(nm)$. Such overhead is unacceptable if bandwidth $O(n+m)$ is desired.

We utilize a more bandwidth-efficient approach to ensure that a malicious Server has computed the correct AV: in π^M_{5PM}, both Client and Server perform secure pattern matching independently using the function PM_{5PM} where one of M_{CDV} and M_T are encrypted, and then compare their results. Each party computes an AV in parallel (see subprotocols $\pi_{C,AV}$ and $\pi_{S,AV}$, respectively, in Section 3.3.3) using an additively homomorphic threshold encryption scheme (instantiated using subprotocol π_{encr} in Section 3.3.3). To ensure that no cheating has occurred, Client and Server then check that each other's AV was computed correctly. Therefore, proving that Server has behaved honestly is reduced to proving that Client and Server have obtained the same result from matching p against T. To efficiently perform a comparison of encrypted AVs, Client and Server check that their encrypted AVs are equal using subprotocol π_{VecEQ} described in Section 2.3.4 (in addition to some zero-knowledge arguments to demonstrate well-formedness). Only if hashed AV values match will Server provide Client with its decrypted (and blinded) AV (using the subprotocols π_{rand} and π_{ans} in Section 3.3.3). The comparison subprotocol is denoted by π_{vec} in Section 3.3.3.

Throughout, both Client and Server will have to use various arguments of consistency outlined in Section 3.3.2 to prove that they have not deviated from the protocol.

There is one additional technical difficulty that we have to overcome: In order to prove security we must provide simulators that simulate transcripts when interacting with adversarial parties (see Section 7 for security definitions and simulator constructions). When constructing the simulator for Client's view, the simulator receives the actual answer that it must provide to Client from the ideal functionality only at the last moment (if Client does not abort). Thus, the simulator must provide a final answer which is not consistent with the previous interactions, while the real Server must be unable to do so. To achieve this, we demonstrate that the simulator can extract the knowledge of the exponent of some h^* specified by Client during the first subprotocol (π_{encr}); then, the final subprotocol (π_{ans}) uses a zero-knowledge argument of knowledge that demonstrates that either the final randomized AV is correct *or* that Server knows the discrete logarithm of h^*. Since a real Server cannot extract the discrete logarithm of h^*, but the simulator can by construction, this allows the simulator to reveal the correct randomized AV even when it is inconsistent with the previous outputs of the conversation. We stress that we do not use NP-reductions but rather build highly efficient protocols to fit our needs.

3.3.2 Zero-Knowledge Arguments of Knowledge (ZK-AoKs) of Consistency. We first describe five required interactive arguments which we rely on to prove statements required for the π^M_{5PM} protocol. They are designed for use with the specified threshold ElGamal encryption scheme (Section 2.2). We apply a standard construction outlined in Section 4 of this paper to transform three-move arguments of knowledge and construct five-move ZK arguments of knowledge π_{DL}, π_{isBit}, π_{eqDL} and π_{fin}, respectively. All ZK-AoKs are executed between a prover P and a verifier V

in five moves; we note that either Client or Server may execute the arguments of consistency as P while the other party will then execute as V. π_{DL} is the only ZK-AoK used on its own in π^{M}_{5PM}; it proves knowledge of a discrete logarithm of a public $h = g^x$. π_{isBIT} is a ZK-AoK that proves that an encryption is either of 0 or of 1; π_{eqDL} is a ZK-AoK that proves that two discrete logarithms are equal; and π_{fin} is a ZK-AoK that proves that *either* two discrete logarithms are equal *or* that P knows the discrete logarithm of a public $h = g^x$. The five required interactive arguments are:

(a) $\mathbf{A_{M01}}$*, an AoK of Consistency for Matrix Formation 0/1:* P, for an $l \times u$ matrix of encryptions, $E(M)$, proves to V that each column of $E(M)$ contains encryptions of 0 and at most one 1.

(b) $\mathbf{A_{M1}}$*, an AoK of Consistency for Matrix Formation* 0/1-1*:* P, for an $l \times u$ matrix of encryptions, $E(M)$, proves to V that each row of $E(M)$ contains encryptions of 0 and *exactly* one 1.

(c) $\mathbf{A_{PD}}$*, an AoK of Consistency for Partial Decryption:* P, for a vector of l encryptions, (x_i, y_i) and a vector of their l partial decryptions (x'_i, y'_i), proves to V that the partial decryptions are correctly constructed.

(d) $\mathbf{A_{Rand}}$*, an AoK of Consistency for Randomization:* P, for a vector of l encryptions (x_i, y_i) and a vector of their exponentiations, $(x_i^{r_i}, y_i^{r_i})$, proves to V that P knows r_i for each i.

(e) $\mathbf{A_{FD}}$*, an AoK of Consistency for Final Decryption:* P, for a vector of l encryptions (x_i, y_i), their partial decryptions (x'_i, y'_i), and some g^w, proves to V that either P has computed all the partial decryptions correctly *or* that it possesses the discrete logarithm w of g^w.

3.3.3 $\boldsymbol{\pi^{M}_{5PM}}$ Protocol Outline. We provide the details of π^{M}_{5PM} by describing individual subprotocols that constitute it, π_{encr}, $\pi_{S,AV}$, $\pi_{C,AV}$, π_{vec}, π_{rand} and π_{ans}. These subprotocols use the interactive arguments described in Section 3.3.2 to prove various statements of consistency. We denote *comm*(s) as the (perfectly hiding, computationally binding) commitment of s, which, using Pedersen commitments [34], is $g^s h^r = \mathit{comm}(g, h, r, s)$. For the exact protocol specification of π^{M}_{5PM}, including precisely how the subprotocols are interleaved so that π^{M}_{5PM} requires only eight rounds, see the Section 6.2; we will, however, mention here during which global rounds (1 through 8) in which these subprotocols occur.

We remark that in our construction of ZK arguments of knowledge from Σ protocols, whenever a ZK subprotocol is required, the first two rounds of the five-round protocol can be completed in parallel at the very beginning of the overall protocol π^{M}_{5PM}. Such "preprocessing" does not affect security. Further, knowledge extraction used in the security proofs is not affected by this preprocessing.

$\pi_{\mathbf{encr}}$ is a two =party protocol executed between Client and Server that initializes an additively homomorphic threshold encryption scheme (e.g., ElGamal) and also sets up an independent "trapdoor" s^* alluded to in Section 3.3.1 and required for the simulator in the security proof. In the ElGamal case, for simplicity, we assume that Client and Server have already agreed on appropriate prime q such that $\log q = O(k)$, $\mathbb{G}_q$ and $g \in \mathbb{G}_q$. This subprotocol begins at the first global round and ends at global round 6. Client chooses its secret key s_C and trapdoor s^*, and sets $h_1 \leftarrow g^{s_C}$, $h^* \leftarrow g^{s^*}$. Client sends h_1, h^* to Server. Client executes two parallel instantiations of π_{DL} proving knowledge of the discrete logs of h_1 and h^* (i.e., s_C and s^*). Then, Server chooses its secret key s_S, sets $h_2 \leftarrow g^{s_S}$, and sends h_2 to Client and executes π_{DL} proving knowledge of the discrete logarithm of h_2 (i.e., s_S). Both parties set the public key to be $h = h_1 h_2 = g^{s_C + s_S}$.

$\pi_{\mathbf{C,AV}}$ is a two-party protocol executed between Client and Server which outputs to Client an encrypted activation vector $E(AV_C)$ corresponding to matching Client's p against Server's T.

This subprotocol starts at global round 2 and ends at global round 6. First, Server constructs $M_T \leftarrow Gen_{M_T}(T)$ as specified in Section 3.1. Then, Server encrypts M_T and sends $E(M_T)$ to Client. Server also executes, for $E(M_T)$, A_{M1} to prove that $E(M_T)$ is formatted correctly (namely, that each row of $E(M_T)$ has one encryption of a 1 per row and encryptions of 0 everywhere else. Therefore, each row of $E(M_T)$ corresponds to the encoding of exactly one element of the alphabet Σ). Client then obtains $E(AV_C)$ by computing $E(PM_{5PM}(M_T, M_{CDV}, m))$ (see Section 2.3.2) and then multiplying each encryption by $E(-p_t)$ (where p_t is the pattern-matching threshold), observing that the function PM_{5PM} can be computed using encrypted $E(M_T)$.

$\pi_{\mathbf{S,AV}}$ is a two-party protocol executed between Client and Server which outputs to Server an encrypted activation vector corresponding to matching Client's p against Server's T. This subprotocol starts at global round 3 and ends at global round 5, with ZK preprocessing occurring during global rounds 1 and 2. Client encrypts M_{CDV} and p_t, and sends $E(M_{CDV})$ and $E(p_t)$ to Server. Client also executes A_{M01} to prove that $E(M_{CDV})$ is formatted correctly (namely, $E(M_{CDV})$ consists of at most one encryption of 1 per column and consists of encryptions of 0 everywhere else. This therefore ensures that there is at most one character delay value per distance). Server computes $E(AV_S)$ by computing $E(PM_{5PM}(M_T, M_{CDV}, m))$ and then multiplying each encryption by $E(-p_t)$ (this slightly differs from Server's actions during π^H_{5PM} since the consistency proof of π_{vec} must also include subtraction of the pattern-matching threshold p_t).

$\pi_{\mathbf{vec}}$ is a two-party protocol executed between Client and Server that outputs to each party whether their respective encrypted activation vectors are equal (without revealing their values). This subprotocol begins at global round 3 and ends at global round 8, with ZK preprocessing occurring during global rounds 1, 2 and 3. Client computes $E(AV'_C)$ by multiplying each element of AV_C with an encryption of 0; Server computes $E(AV'_S)$ from $E(AV_S)$ similarly. Client and Server execute π_{VecEQ} (see Section 2.3.4) where Client has input $E(AV'_C)$ and Server has input $E(AV'_S)$. In addition, whenever a party sends the other a partial decryption, they execute A_{PD} to prove that the execution is well formed. Note that the probability that π_{VecEQ} will complete without abort for unequal vectors AV_S and AV_C is negligible $(\frac{1}{q})$.

$\pi_{\mathbf{rand}}$ is a two-party protocol executed between Client and Server that outputs to Client an encrypted vector $E(AV^r_S)$ that contains randomizations of the values in non-matching (non-zero) positions in $E(AV'_S)$. This subprotocol starts at global round 6 and ends at global round 8, with ZK preprocessing occurring during global rounds 2 and 3. Server computes $E(AV^r_S)$ from $E(AV'_S)$ by exponentiating each encryption in $E(AV'_S)$ by a random value. Server sends $E(AV^r_S)$ to Client and executes A_{rand} to prove that $E(AV^r_S)$ was obtained correctly from $E(AV'_S)$.

$\pi_{\mathbf{ans}}$ is a two-party protocol executed between Client and Server that outputs to Client the randomization, AV^r_S, of Server's activation vector AV_S. Note that AV^r_S will have a 0 wherever there is a match; every non-matching entry will contain a random element. Client is assumed to already know $E(AV^r_S)$. This subprotocol starts at global round 6 and ends at global round 8, with ZK preprocessing occurring during global rounds 2 and 3. We present a slightly modified version of the actual subprotocol used because in practice this protocol must be rearranged slightly to keep π^M_{5PM} at 8 rounds (see Section 6.2 for details). Server sends $D_S(E(AV^r_S))$ to Client and executes A_{FD} to prove that either $D_S(E(AV^r_S))$ was obtained correctly or that Server knows s^* (for h^* sent by Client in the first round of π_{encr}). Client aborts if it does not accept A_{FD} and otherwise obtains AV^r_S by computing $D_C(D_S(E(AV^r_S)))$.

3.3.4 Protocol Efficiency and Security. Overall the bandwidth of π^M_{5PM} is dominated by the $O(m|\Sigma|)$ encrypted values that Client sends to Server in $\pi_{S,AV}$ and $O(n|\Sigma|)$ encrypted values that Server sends to Client in $\pi_{C,AV}$ and π_{ans}. Since alphabet size, $|\Sigma|$, is constant, we obtain the desired bandwidth, including the ZK protocols, of $O((m+n)k^2)$ for security parameter k and total number of encryptions of $O(m+n)$. In particular, when Client hides pattern size, the corresponding pattern will have length n, and therefore the bandwidth complexity is $O(nk^2)$. Computational complexity for Client is dominated by the subprotocol $\pi_{C,AV}$, where Client performs $O(mn)$ exponentiations of encrypted elements, and computational complexity for Server is dominated by subprotocols $\pi_{S,AV}$, where Server performs $O(mn)$ multiplications of encrypted elements, and π_{vec} and π_{ans}, where $O(nk)$ exponentiations are needed for the ZK protocols.

Theorem 2. *Assuming that the Decisional Diffie-Hellman (DDH) problem is hard, π^M_{5PM} is secure in the malicious (static corruption) model.*

See Section 7 for a detailed security proof.

4 Converting Σ protocols to Zero-Knowledge Arguments of Knowledge (ZK-AoKs)

Here, we describe the construction used to convert a Σ protocol into an efficient zero-knowledge argument of knowledge. We first provide the necessary definitions. We then give a construction of an efficient extractable equivocable commitment scheme. We finally use this scheme to construct a zero-knowledge argument of knowledge from a Σ protocol for the same relation. We note that an algorithm is expected PPT if it is a probabilistic algorithm that runs in expected polynomial time.

4.1 Definitions

Let R be a binary relation where for all $(x, w) \in R$, $|w| \in poly(|x|)$. w is called the *witness* for x. Consider an interactive argument consisting of a pair of PPT algorithms (P, V) (thought of as probabilistic next-message functions). x is known to both P and V, while w is only known to P. Informally, P proves to V that there is a w such that $(x, w) \in R$. We consider interactive protocols that have the following specification:

a. P sends message a, $|a| \in poly(|x|)$.
b. V selects message $e \in \{0,1\}^{poly(|x|)}$ uniformly at random and sends e to P. We denote e as the *challenge*.
c. P sends a reply $z \in \{0,1\}^{poly(|x|)}$.

Note that this interaction is public coin for V. Based on the tuple (also called a *conversation*) (a, e, z), V either *accepts* or *rejects*. For any x, we call a conversation (a, e, z) that V accepts an *accepting conversation*.

Definition 1. *A 3-move interactive protocol $\Pi = (P, V)$ of the above specification is said to be a Σ protocol for a relation R if it satisfies the following properties:*

a. Completeness: *On common input x, if the honest prover P has as private input w such that $(x, w) \in R$, then honest V always accepts.*

b. Special Soundness: *For any common input x and any pair of accepting conversations (a, e, z) and (a, e', z') for x where $e \neq e'$, there exists a w that can be computed in polynomial time such that $(x, w) \in R$.*

c. Special Honest-Verifier Zero-Knowledge (SHVZK): *There exists a PPT M that on input x and a properly formatted e outputs an accepting conversation of the form (a, e, z) with the same probability distribution (over e) as conversations between honest P and honest V.*

Definition 2. *An interactive protocol for a relation R consisting of a pair of PPT algorithms (P, V) is an* argument of knowledge *with knowledge error κ if the following properties are satisfied:*

a. Completeness: *On common input x, if the honest prover P has as private input w such that $(x, w) \in R$, then honest V always accepts.*

b. Knowledge Soundness: *There exists a (expected) PPT E called the* knowledge extractor *which, given input x and oracle (black-box) access to P, attempts to compute w such that $(x, w) \in R$. For any prover P^*, let $\epsilon(x)$ be the probability that V accepts on input x. Then there exists a constant c such that whenever $\epsilon(x) > \kappa(x)$, E will output a correct w in expected time at most $\frac{|x|^c}{\epsilon(x)-\kappa(x)}$ where an individual oracle call to P^* is considered as one step.*

κ can be thought of as the probability that V can be convinced that there exists a w such that $(x, w) \in R$ even if such a pair does not exist.

Lemma 3 ([35]) *Let Π be a Σ protocol for relation R where the challenge e is drawn uniformly at random from $\{0, 1\}^t$. Then Π is a proof of knowledge with knowledge error 2^{-t}.*

Remark 1. Lemma 3 holds because Definition 4.1 includes the special soundness property. Σ protocols that only have standard soundness will not always satisfy the lemma.

Definition 3. *For any binary relation R, an interactive protocol consisting of a pair of PPT algorithms (P, V) is a* zero-knowledge argument *if it satisfies the following properties:*

a. Completeness: *On common input x, if the honest prover P has as private input w satisfying $(x, w) \in R$, then honest V always accepts.*

b. Soundness: *For all x such that there does not exist a w with $(x, w) \in R$, V will only accept with negligible probability.*

c. Zero-Knowledge: *For all PPT V^*, there is a PPT simulator M with oracle access to V^* such that, given input x and V^*'s auxiliary input, V^*'s view of its interaction with real P is computationally indistinguishable from V^*'s view of its interaction with M.*

We demonstrate that any Σ protocol for a binary relation R can be converted into a ZK argument of knowledge for R. We first construct an extractable equivocable commitment scheme and use this scheme together with the Σ protocol specification for the ZK-AoK construction.

4.2 Extractable Equivocable Commitment Schemes

To construct a ZK-AoK from a Σ protocol, an efficient *extractable equivocable commitment scheme* will be required. Such a scheme is an interactive protocol between a PPT committer C and a PPT receiver R consisting of three functions: $EComSet$ instantiates the commitment scheme, com computes the commitment, and $EComVer$ verifies that decommitment is valid. More specifically, R, for a security parameter k, computes $(pk, t) \leftarrow EComSet(1^k)$ and sends pk to C. C computes

$c \leftarrow com(s, r, pk)$ for message s and randomness r and sends c to R as its commitment to m. To decommit, C sends (r, m) to R. R computes $\{0,1\} \leftarrow EComVer(m, r, c, pk)$, *accepts* if 1 is output and *rejects* otherwise.

Definition 4. *A* computationally binding equivocable commitment scheme *is a pair of PPT algorithms (R, C) that interact as above and satisfy the following properties.*

a. Statistically Hiding: *For pk correctly constructed and any messages s and s', the distributions of $com(s, r, pk)$ and $com(s', r', pk)$ are statistically indistinguishable over the choice of random input (e.g., r and r').*

b. Computationally Binding: *For any PPT algorithm C running in expected time polynomial in k, the probability that C on input pk can compute a tuple (s, r, s', r') such that $com(s, r, pk) = commit(s', r', pk)$ with $s \neq s'$ is negligible in k.*

c. Equivocable: *There is a PPT algorithm S that, on inputs t, pk, any commitment c and any accepting decommitment (s, r) to c, can construct for any valid s' an r' such that $c = com(s', r', pk)$.*

An equivocable commitment scheme is extractable *if there is a PPT algorithm E that, upon oracle access to R, is able to obtain a trapdoor t in expected polynomial time.*

We now give a construction of an equivocable commitment scheme, *EP*, based on Pedersen commitments [34]. We assume that the receiver R and committer C have already agreed on a prime order group $\mathbb{G}_q$ and generator $g \in \mathbb{G}_q$. The committer C has a message s. For $b \in \{0,1\}$, we denote $\bar{b} = 1 - b$.

EP - Commitment:

EP-1: R chooses for $1 \leq i \leq k$ and $0 \leq j \leq 1$, $x_{1,0}, x_{1,1}, \dots, x_{k,0}, x_{k,1} \in \mathbb{Z}_q$ uniformly and independently at random and sets $h_{i,j} \leftarrow g^{x_{i,j}}$. R sends $(h_{1,0}, h_{1,1}, ..., h_{k,0}, h_{k,1})$ to C.

EP-2: C chooses $e \in \{0,1\}^k$. C chooses $r_1, ...r_k \in \mathbb{Z}_q$ uniformly and independently at random. C sets $c = h_{1,e_1}^{r_1} \cdot ... \cdot h_{k,e_k}^{r_k} \cdot g^s \leftarrow com_{\mathbb{G}_q, g, h_{1,e_1}, .., h_{k,e_k}}(s, r_1, ..r_k)$. C sends e and c to R.

EP-3: R sends $(x_{1,\bar{e}_1}, ..., x_{k,\bar{e}_k})$ to C.
C checks that for $1 \leq i \leq k$, $h_{i,\bar{e}_i} = g^{x_{i,\bar{e}_i}}$. If not, C aborts.

EP - Decommitment

EP-4: C sends $s, r_1, ..., r_k$.

EP-5: R verifies that $c = h_{1,e_1}^{r_1} \cdot ... \cdot h_{k,e_k}^{r_k} \cdot g^s$ and aborts if equality does not hold.

The above *EP* protocol has bandwidth complexity $O(k^2)$ and computational complexity $O(k^2 \log^2 k)$.

Just like Pedersen commitments, this commitment scheme is statistically hiding and computationally binding.

We show that knowledge of *any* discrete logarithm $x_{i,j}$ from the public key together with a valid decommitment would allow S to open commitments to any value. Indeed, let the public key be $(h_{1,e_1}, ..., h_{k,e_k})$, where $h_{i,e_i} = g^{x_i}$. Without loss of generality, suppose S obtained x_1 and also obtained $c, s, r_1, ..., r_k$ such that $c = com(s, r_1, ..., r_k, pk)$. Let s' be any message. S sets $r'_i = r_i$ for $2 \leq i \leq k$ and sets $r'_1 = \frac{s + x_1 r_1 - s'}{x_1} \mod q$. Then $com(s', r'_1, r_2, ..., r_k) = c = com(s, r_1, ..., r_k)$.

To demonstrate that this scheme is extractable, for any R we construct a simulator M_R. M_R runs EP honestly with R through **EP-3**, then rewinds to **EP-2** and sends a new $e' \in \{0,1\}^k$ to R. Since $e' \neq e$, M_R obtains some discrete logarithm $x_{i,j}$ of the public key $(h_{1,e_1}, ..., h_{k,e_k})$ and therefore can decommit to any s'. We require many possible trapdoors because the probability that R can both complete the *EP* protocol (namely the step **EP-3**) and not know or abort when asked for the discrete logarithm of any of the $h_{i,j}$s is roughly the same probability that M_R will fail to extract a trapdoor since knowledge of the discrete logarithms occurs at **EP-3**; therefore, we require many trapdoors to ensure that the probability that R sends an invalid response at **EP-3** is negligible in the security parameter.

4.3 Construction of a ZK-AoK from Σ Protocols

We give a construction for how to transform a three-move Σ argument of knowledge Σ_{rel} for a binary relation R_{rel} into a five-move ZK argument of knowledge π_{rel} for R_{rel} using the extractable equivocable commitment scheme EP described in Section 4.2. Recall from Section 4.1 that we denote the transcript for a Σ protocol as (a, e, z), where e is chosen uniformly at random by V. We assume that V and P have already agreed on a multiplicative group $\mathbb{G}_q$ with prime order q and generator $g \in \mathbb{G}_q$. P $(x, w) \in R_{rel}$; V possesses x. We denote the following construction by Σ-ZK-AoK, which consists of the following steps:

rel-1: P executes **EP-1** acting as receiver.
rel-2: V selects e according to the second message of Σ_{rel}. V executes **EP-2** using e as the value being committed.
rel-3: P computes a according to the first message specification of Σ_{rel}. P executes **EP-3** and also sends a to V.
rel-4: V executes **EP-4**, opening e.
rel-5: P executes **EP-5** and aborts if P does not accept. P computes z according to the third message specification of Σ_{rel} in response to a, e and x. P sends z to V.
rel-6: V verifies (a, e, z) according to Σ_{rel}.

π_{rel} has bandwidth complexity $O(k^2)$ and computational complexity $O(k^2 \log^2 k)$ in addition to that of Σ_{rel}.

Lemma 1. *If Σ_{rel} is a Σ-protocol, then π_{rel} is a ZK argument of knowledge.*

Proof: Completeness. Completeness follows from the completeness of Σ_{rel}.

Soundness. To demonstrate soundness, assume that there exists an x such that there is no w with $(x, w) \in R_{rel}$ and yet V accepts π_{rel} with non-negligible probability. Let (a, e, z) be the transcript for Σ_{rel} contained within π_{rel}. Then it follows that Σ_{rel} has a verifier V that accepts a transcript with non-negligible probability for the same x. This implies that there are at least two distinct challenges e and e' such that P can produce accepting transcripts (a, e, z) and (a, e', z') for Σ_{rel} within π_{rel} (in fact, there must be a non-negligible number of such challenges). However, by special soundness of Σ_{rel}, a w can be computed in polynomial time from these transcripts such that $(x, w) \in R_{rel}$. But such a w does not exist, which leads to a contradiction.

To demonstrate that π_{rel} is zero-knowledge, for any verifier V^*, we describe the simulator M_V. M_V acts as an honest prover for steps **rel-1** through **rel-4**. At step **rel-4**, M_V receives V^*'s challenge e. M_V then, by the SHVZK property of Σ_{rel}, computes (a, z) such that (a, e, z) is an accepting transcript for x. Note that in particular, the fact that Σ protocols are *special* honest verifier zero knowledge is important, as it implies the ability to construct correct transcripts for arbitrary (pre-selected) distributions of verifier messages. M_V rewinds to step **rel-3** where it sends a (as well as executes **EP-3**) and executes the rest of π_{rel} honestly. In particular, M_V sends z at step **rel-5**. V^*'s view of its interaction with P is indistinguishable from its view of its interaction with M_V because V^* cannot affect the distribution of its challenges based on P's messages since V^*

commits to its challenge (in a perfectly binding fashion) before it receives the first message of Σ_{rel}. Since the distribution of e is not affected by initial messages, M_V's transcript of Σ_{rel} within π_{rel} is computationally indistinguishable from P's output for Σ_{rel} by the special honest-verifier zero-knowledge (SHVZK) property of Σ_{rel}. Computational indistinguishability of the whole transcript follows.

To show the existence of a knowledge extractor, E_P, for each P, let $E_{rel,P}$ be the knowledge extractor for Σ_{rel} and let S_P be the trapdoor extractor for the commitment scheme EP. E_P then runs ZK-AoK using S as a subprotocol to extract the trapdoor for EP. E_P then rewinds to **rel-4**, after P has already instantiated the commitment scheme and sent its initial message a for Σ_{rel}, and changes its challenge for Σ_{rel} according to the specification of $E_{rel,P}$. Note that E_P will have to decommit to multiple challenges for Σ_{rel} at step **rel-4** in order to execute $E_{rel,P}$ as a subprotocol. However, since E_P possesses the trapdoor for EP and EP is equivocable, E_P can decommit (e.g., construct messages for **EP-4**) to whatever challenge $E_{rel,P}$ specifies. Since S can extract the trapdoor in polynomial time, and E_{rel_P} can extract the witness for Σ_{rel} in (expected) polynomial time, E_P can extract the witness for π_{rel} in (expected) polynomial time.

Remark 2. We note that above, the zero-knowledge simulator M_V was able to interact with V without actually knowing the witness w for x. This is because of the simulation soundness of Σ_{rel}; namely, since a simulator can produce accepting transcripts only by seeing the Verifier's challenge there (and without seeing w), M_V can produce proper transcripts for π_{rel} without ever knowing w. Such a property is called *simulation soundness* and will be useful for a security reduction needed for π^M_{5PM} (see Section 7.3.3).

5 Required Σ Protocols

In this section we outline specific Σ protocols needed for the malicious model version of $5PM$, π^M_{5PM}. These three-move protocols are executed between a PPT prover (P) and a PPT verifier (V) and are used to construct zero-knowledge arguments of knowledge using the transformation in Section 4.3. For each Σ protocol, we first describe the relation demonstrated by P, and then we describe the three protocol messages exchanged between P and V. We note that with the exception of Σ_{fin}, the security of each of the following Σ protocols is proven in the places in which they are cited; the security of Σ_{fin} follows since it is a standard example of an OR Σ protocol of two Σ protocols already shown here (for more, see [35]).

1. Σ_{DL}, Proving Knowledge of Discrete Logs [36]: For g and $h = g^x$, P demonstrates knowledge of witness x. The relation R_{DL} is $((\mathbb{G}_q, q, g, h), x) \in R_{DL}$ if $h = g^x$. The Σ protocol steps are:

- Σ_{DL-1}: P chooses $r \in \mathbb{Z}_q$ and sets $a \leftarrow g^r$. P sends a to V.
- Σ_{DL-2}: V chooses a challenge $c \in \mathbb{Z}_q$ and sends c to P.
- Σ_{DL-3}: P sets $z \leftarrow r + cx$ and sends z to V.
- Σ_{DL-4}: V checks that $g^z = ah^c$ and aborts if not.

2. Σ_{eqDL}, Proving Equality of Discrete Logs [30]: For $g, h, x, y \in \mathbb{G}_q$, P demonstrates knowledge of a witness w such that $x = g^w$ and $y = h^w$. The relation R_{eqDL} is $((\mathbb{G}_q, q, g_1, g_2, h_1, h_2), w) \in R_{eqDL}$ if $h_1 = g_1^w$ *and* $h_2 = g_2^w$. The Σ protocol steps are:

- Σ_{eqDL-1}: P chooses $r \in \mathbb{Z}_q$ and sets $(a, b) \leftarrow (g^r, h^r)$. P sends (a, b) to V.

- Σ_{eqDL-2}: V chooses $c \in \mathbb{Z}_q$ and sends c to P.
- Σ_{eqDL-3}: P sets $z \leftarrow r + wc$ and sends z to V.
- Σ_{eqDL-4}: V checks that $g^z = ax^c$ and that $h^z = by^c$ and aborts if not.

3. Σ_{isBit}, Proving Encryption of 0 or 1 [37]: P demonstrates that it possesses an ElGamal encryption of $m \in \{0,1\}$ with generators g, public key h and randomness r (recall that encryption of m is of the form $(g^r, g^m h^r) = (x,y)$). P proves that either $\log_g x = \log_h y$ or $\log_g x = \log_h y/g$. The relation R_{isBit} is $((\mathbb{G}_q, q, g, h, \alpha, \beta), (b, r)) \in R_{isBit}$ if $(\alpha, \beta) = (g^r, g^b h^r)$ *and* $b \in \{0,1\}$. The Σ protocol steps are:

- $\Sigma_{isBit-1}$:
  - If $m = 1$: P chooses $r_1, d_1, w_2 \in \mathbb{Z}_q$ and sets $a_1 \leftarrow h^{r_1}(gh^r)^{-d_1}$, $a_2 \leftarrow h^{w_2}$.
  - If $m = 0$: P chooses $w_1, r_2, d_2 \in \mathbb{Z}_q$ and sets $a_1 \leftarrow h^{w_1}$, $a_2 \leftarrow h^{r_2}(h^r g^{-1})^{-d_2}$.

  P sends (a_1, a_2) to V.
- $\Sigma_{isBit-2}$: V chooses $c \in \mathbb{Z}_q$ and sends c to P.
- $\Sigma_{isBit-3}$:
  - If $m = 1$: P sets $d_2 \leftarrow c - d_1$, $r_2 \leftarrow w_2 + rd_2$.
  - If $m = 0$: P sets $d_1 \leftarrow c - d_2$, $r_1 \leftarrow w_1 + rd_1$.

  P sends (d_1, d_2, r_1, r_2) to V.
- $\Sigma_{isBit-4}$: V verifies that $c = d_1 + d_2$, $h^{r_1} = a_1(g^m h^r)^{d_1}$ and $h^{r_2} = a_2(g^{m-1}h^r)^{d_2}$ and aborts if not.

4. Σ_{fin}, Proving Equality of Discrete Logarithmss Or Knowledge of Discrete Logarithm: P demonstrates for $g, h, g_1, h_1, g_2, h_2 \in \mathbb{G}_q$ that *either* it knows the value $\log_{g_1} h_1 = \log_{g_2} h_2$ *or* he knows x such that $h = g^x$ given g. The relation R_{fin} is $((\mathbb{G}_q, q, g_1, g_2, h_1, h_2, g, h), (\alpha, x)) \in R_{fin}$ if either (DLE): $h_1 = g_1^\alpha$ and $h_2 = g_2^\alpha$ *or* (DL:) $h = g^x$. The Σ protocol steps are:

- Σ_{fin-1}:
  - If DLE: P chooses $r_1, d_1, w_2 \in \mathbb{Z}_q$ and sets $a_1 \leftarrow g^{r_1}h^{-d_1}$, $a_2 \leftarrow g_1^{w_2}$ and $a_3 \leftarrow g_2^{w_2}$.
  - If DL: P chooses $w_1, r_2, d_2 \in \mathbb{Z}_q$ and sets $a_1 \leftarrow g^{w_1}$, $a_2 \leftarrow g_1^{r_2}h_1^{-d_2}$ and $a_3 \leftarrow g_2^{r_2}h_2^{-d_2}$.

  P sends $(g_1, h_1, g_2, h_2, g, h, a_1, a_2, a_3)$ to V.
- Σ_{fin-2}: V choses $c \in \mathbb{Z}_q$ and sends c to P.
- Σ_{fin-3}:
  - If DLE: P sets $d_2 \leftarrow c - d_1$, $r_2 \leftarrow w_2 + \alpha d_2$.
  - If DL: P sets $d_1 \leftarrow c - d_2$, $r_1 \leftarrow w_1 + xd_1$.

  P sends (d_1, d_2, r_1, r_2) to V.
- Σ_{fin-4}: V verifies that $d_1 + d_2 = c$, $g^{r_1} = a_1 h^{d_1}$, $g_1^{r_2} = a_2 h_1^{d_2}$, and $g_2^{r_2} = a_3 h_2^{d_2}$ and aborts if not.

6 Detailed π_{5PM}^M Specification

We provide here the detailed protocol specification of the malicious model version of $5PM$, π_{5PM}^M. First, we must specify the various zero-knowledge arguments of consistency that are required.

6.1 Arguments of Knowledge of Consistency

We first describe five required interactive arguments which we rely on to prove statements required in the π_{5PM}^M protocol. They are designed for use with the specified threshold ElGamal encryption scheme (Section 2.2). We apply the Σ-ZK-AoK construction outlined in Section 4 to transform the three-move arguments of knowledge outlined in Section 5 to construct the five-move ZK arguments of knowledge π_{DL}, π_{isBit}, π_{eqDL} and π_{fin}. All arguments are executed between a prover P and a verifier V. π_{DL} is the only ZK-AoK used on its own in π_{5PM}^{M}; it proves knowledge of a discrete logarithm of a public $h = g^x$. π_{isBIT} is a ZK-AoK that proves that an encryption is either of 0 or of 1; π_{eqDL} is a ZK-AoK that proves that two discrete logarithms are equal; and π_{fin} is a ZK-AoK that proves that *either* two discrete logarithms are equal *or* that P knows the discrete logarithm of a public $h = g^x$. The five required interactive arguments are:

$\mathbf{A_{M01}}$*, an AoK of Consistency for Matrix Formation 0/1:* In this interactive argument, P sends an $l \times u$ matrix of encryptions, $E(M)$. P demonstrates to V that each column in $E(M)$ contains at most one encryption of 1, and the rest of the encryptions are of 0. We assume that P has sent $E(M)$ to V. We denote by A_{M01} the five-move interactive argument where P proves to V using $(l+1)u$ parallel instantiations of π_{isBit} that each entry of $E(M)$ is an encryption of either 0 or 1 and that each column-wise product of $E(M)$ is an encryption of either 0 or 1. If V accepts the argument A_{M01}, then it accepts that each column is made up of entries that are either 0 or 1 and sum up to 0 or 1; therefore, each column contains encryptions of 0 and at most one 1.

$\mathbf{A_{M1}}$*, an AoK of Consistency for Matrix Formation* 0/1-1*:* Similar to the above interactive argument, P sends an $l \times u$ matrix of encryptions, $E(M)$. P demonstrates to V that (unlike the above argument) each row in $E(M)$ contains *exactly* one encryption of 1 and the rest are encryptions of 0. To prove that an encryption (x, y) is of 1, P sends $y' = y/g$ and uses π_{DL} to prove that $\log_g x = \log_h y'$. V then can see that (x, y) is an encryption of 1 only if $y/y' = g$. We assume that P has sent $E(M)$ to V. We denote by A_{M1} the five-move interactive argument where P sends (x_i, y_i') for each of the row-wise products, (x_i, y_i), of $E(M)$ and then P proves to V using $l \cdot u$ instantiations of π_{isBit} that each entry of $E(M)$ is an encryption of a 0 or a 1 and proves to V using u instantiations of π_{DL} that each row-wise product of $E(M)$ is an encryption of a 1. If V accepts the argument A_{M1}, then it accepts that each row is made up of entries that are either 0 or 1 and sum to 1; therefore, each row contains encryptions of 0 and exactly one 1.

$\mathbf{A_{PD}}$*, an AoK of Consistency for Partial Decryption:* In this interactive argument, P possesses and sends a vector of l encryptions (x_i, y_i) and a vector of their l partial decryptions $(x_i, y_i/x_i^{s_P})$, where s_P is P's private key. P demonstrates to V that he has computed the partial decryptions correctly. We assume that P has already sent the vector of l encryptions and l partial decryptions and that V already knows g^{s_P}. We denote by A_{PD} the five-move interactive argument where P sends, for each i, $x_i^{s_P}$, and proves to V using l parallel instantiations of π_{eqDL}, that $\log_g g^{s_P} = \log_{x_i} x_i^{s_P}$.

$\mathbf{A_{Rand}}$*, an AoK of Consistency for Randomization:* In this interactive argument, P possesses and sends a vector of l encryptions (x_i, y_i) and a vector of their randomizations, $(x_i^{r_i}, y_i^{r_i})$, to V and demonstrates knowledge of r_i for each i. P proves, using π_{eqDL}, that $\log_{x_i} x_i^{r_i} = \log_{y_i} y_i^{r_I}$ for each i. We assume that P has already sent the l encryptions and l randomizations. We denote by A_{Rand} the five-move interactive argument where P proves to V, using l parallel instantiations of π_{eqDL}, that each of the l randomizations is formatted correctly.

$\mathbf{A_{FD}}$, *an AoK of Consistency for Final Decryption:* In this interactive argument, P possesses and sends a vector of l encryptions (x_i, y_i), their partial decryptions $(x_i, y_i/x_i^{s_P})$ as well as g^w to V and demonstrates that either P has computed their partial decryptions $(x_i, y_i/x_i^{s_P})$ correctly *or* that he possesses the discrete logarithm w of g^w. We denote by A_{FD} the five-move interactive argument where P proves, using l parallel instantiations of π_{fin}, that either the l encryptions (x_i, y_i) has been partially decrypted correctly or that P knows the discrete logarithm of g^w.

6.2 π_{5PM}^M Protocol Specification

The eight round protocol for the malicious model, π_{5PM}^M, consists of the following six subprotocols:

(a) π_{encr}: initializes an additively homomorphic threshold encryption scheme.
(b) $\pi_{S,AV}$: allows Server to construct an encrypted activation vector for Client's encrypted pattern and Server's text.
(c) $\pi_{C,AV}$: allows Client to also construct an encrypted activation vector for Client's pattern and Server's encrypted text.
(d) π_{vec}: allows Client and Server to verify that their activation vectors are equal without revealing them.
(e) π_{rand}: allows Server to send an encryption of its randomized activation vector to Client.
(f) π_{ans}: demonstrates to Client where the pattern matches the text (if at all).

In what follows, we describe π_{5PM}^M by specifying in detail the individual subprotocols that are required and specifying for each subprotocol where each round of the subprotocol occurs in the overall (global) rounds of π_{5PM}^M. Table 6 contains the notation used to describe the subprotocols in Tables 7 to 12. The required subprotocols utilize the interactive arguments described in Section 6.1 to prove various statements; these arguments are all five-move protocols between a prover (P) and a verifier (V), where, for instance, $A_{M1}^{P,i}$ and $A_{M1}^{V,j}$ denote the ith and jth messages sent by P and V, respectively, in interactive argument A_{M1}. We denote $comm(s)$ as shorthand for the commitment of s which, using Pedersen commitments [34], is $g^s h^r = comm(g, h, r, s)$.

We remark that in our construction of ZK arguments of knowledge from Σ protocols, whenever a ZK subprotocol is required, the first two rounds of the five-round protocol can be completed in parallel at the very beginning of the overall protocol π_{5PM}^M. Such "preprocessing" will not affect security since these rounds do not involve any Σ protocol-related information from P, and as long as V commits to his Σ protocol challenge prior to seeing P's first message of the underlying Σ protocol (see Section 4 for details of the ZK constructions and Section 7 for a proof that preprocessing does not affect security).

p = Pattern of length m	T = Text of length n
AV_i = Activation vector of party i	AV_S^r = Randomized activation vector
AV_i' = Blinded activation vector	M_{CDV}= Matrix encoding of p in terms of Σ
sk_C = Client's secret key	sk_S = Server's secret key
p_t = Pattern match threshold	$\langle , \rangle$ = Inner product over $\mathbb{G}_q$
$\pi_{rel}^{i,j}(x)$ = Party i's jth message of π_{rel} for x	$E()$ = Additively homomorphic encryption
$D_i()$ = Partial decryption by party i	h = Threshold public key
s^* = Simulator trapdoor	M_T = Matrix encoding of T in terms of Σ
$A_{rel}^{i,j}(x)$ =Player i's jth message of A_{rel} for x	

Table 6. Notation used in subprotocols in Tables 7 through 12

Global Round	**Client**	**Messages**	**Server**
1	$s_C, s^* \in \mathbb{Z}_q$, $h_1 \leftarrow g^{s_C}$, $h^* \leftarrow g^{s^*}$	$\xrightarrow{h_1, h^*, \pi_{DL}^{P,1}(h_1), \pi_{DL}^{P,1}(h^*)}$	
2		$\xleftarrow{h_2, \pi_{DL}^{P,1}(h_2), \pi_{DL}^{V,1}(h_1), \pi_{DL}^{V,1}(h^*)}$	$s_S \in \mathbb{Z}_q$, $h_2 \leftarrow g^{s_S}$, $h = h_1 h_2$
3	$h = h_1 h_2$	$\xrightarrow{\pi_{DL}^{P,2}(h_1), \pi_{DL}^{P,2}(h^*), \pi_{DL}^{V,1}(h_2)}$	
4		$\xleftarrow{\pi_{DL}^{P,2}(h_2), \pi_{DL}^{V,2}(h_1), \pi_{DL}^{V,2}(h^*)}$	
5		$\xrightarrow{\pi_{DL}^{P,3}(h_1), \pi_{DL}^{P,3}(h^*), \pi_{DL}^{V,2}(h_2)}$	
6		$\xleftarrow{\pi_{DL}^{P,3}(h_2)}$	

Table 7. Subprotocol π_{encr}

$\pi_{\mathbf{encr}}$, shown in Table 7, is a two-party protocol for Client and Server that initializes a threshold ElGamal encryption scheme. For simplicity, we assume that Client and Server have already agreed on $\mathbb{G}_q$ and $g \in \mathbb{G}_q$. Client input is $\mathbb{G}_q, g \in G_q$. Server input is $\mathbb{G}_q, g \in G_q$. Output to Client is $h = g^{s_C} g^{s_S}$. Output to Server is $h = g^{s_C} g^{s_S}$, $h^* = g^{s^*}$. This subprotocol begins at the first global round and ends at global round 6.

- At global round 1 Client chooses $s_C, s^* \in \mathbb{Z}_q$ and sets $h_1 \leftarrow g^{s_C}$, $h^* \leftarrow g^{s^*}$. Client sends h_1, h^* to Server. Client sends the Server two parallel instantiations of $\pi_{DL}^{P,1}$ proving knowledge of the discrete logs of h_1 and h^* (e.g., of s_C and s^*). The last message of Client's instantiations of π_{DL} is exchanged at global round 5.
- At global round 2 Server chooses $s_S \in \mathbb{Z}_q$ and sets $h_2 \leftarrow g^{s_S}$. Server sends h_2 to Client, as well as $\pi_{DL}^{P,1}$, proving knowledge of the discrete logarithm of h_2 (e.g., of s_S). The last message of Server's π_{DL} is sent at global round 6. Both parties set the public key to be $h = h_1 h_2$.

$\pi_{\mathbf{C},\mathbf{AV}}$, shown in Table 8, is a two-party protocol for Client and Server which outputs to Client an encrypted activation vector corresponding to matching Client's p against Server's T. Client input is pattern p, threshold p_t. Server input is text T and M_T, which is the $|\varSigma| \times n$ matrix encoding T in terms of $\varSigma$ (see Section 3.1). Server receives no output. Output to Client is $E(AV_C)$, an encrypted activation vector corresponding to matching p against T. This subprotocol starts at global round 2 and ends at global round 6.

- At global round 2 Server sends $E(M_T)$ to Client. Server also sends, for $E(M_T)$, $A_{M1}^{P,1}$ to prove that $E(M_T)$ is formatted correctly. The last message of Server's A_{M1} is exchanged at global round 6.
- During global round 3, Client computes $E(AV_C)$ from $E(M_T)$, M_{CDV} for p, and p_t by first computing $M_{CDV} \cdot E(M_T)$. This can be performed by recognizing that one can obtain an encryption of the inner product over $\mathbb{Z}_q$ of an unencrypted vector $(x_1, ..., x_m)$ with an encrypted vector $(E(y_1), ..., E(y_m))$ by computing $\varPi E(y_i)^{x_i} = E(\sum x_i y_i)$ (see Section 2.3.2).

$\pi_{\mathbf{S},\mathbf{AV}}$, shown in Table 9, is a two-party protocol for Client and Server which outputs to Server an encrypted activation vector corresponding to matching Client's p against Server's T.

Global Round	Client	Messages	Server
2		$\xleftarrow{E(M_T),A^{P,1}_{M1}(E(M_T))}$	$M_T \leftarrow T$, $E(M_T) \leftarrow M_T$
3	$E(AV_C) \leftarrow M_{CDV}, p_t, E(M_T)$	$\xrightarrow{A^{V,1}_{M1}(E(M_T))}$	
4		$\xleftarrow{A^{P,2}_{M1}(E(M_T))}$	
5		$\xrightarrow{A^{V,2}_{M1}(E(M_T))}$	
6		$\xleftarrow{A^{P,3}_{M1}(E(M_T))}$	

Table 8. Subprotocol $\pi_{C,AV}$

Global Round	Client	Messages	Server
1		$\xrightarrow{A^{P,1}_{M01}(E(M_{CDV}))}$	
2		$\xleftarrow{A^{V,1}_{M01}(E(M_{CDV}))}$	
3	$E(M_{CDV}) \leftarrow CDV$, $E(p_t) \leftarrow p_t$	$\xrightarrow{E(M_{CDV}),A^{P,2}_{M01}(E(M_{CDV}))}$	
4		$\xleftarrow{A^{V,2}_{M01}(E(M_{CDV}))}$	$E(AV_S) \leftarrow T, E(M_{CDV}), E(p_t)$
5		$\xrightarrow{A^{P,3}_{M01}(E(M_{CDV}))}$	

Table 9. Subprotocol $\pi_{S,AV}$

Client input is pattern p, M_{CDV} for p, and p_t, the matching threshold. Server input is T. Client receives no output. Output to Server is $E(AV_S)$, an encrypted activation vector corresponding to matching p against T. This subprotocol starts at global round 3 and ends at global round 5, with ZK preprocessing occurring during global rounds 1 and 2.

- At global round 3 Client sends $E(M_{CDV})$ and $E(p_t)$ to Server. Client also sends $A^{P,2}_{M01}$ to prove that $E(M_{CDV})$ is formatted correctly, where $A^{P,1}_{M01}$ and $A^{V,1}_{M01}$ occur during global rounds 1 and 2, respectively. The last message of Client's A_{M01} is sent at global round 5.
- During global round 4, using $E(M_{CDV})$, T and $E(p_t)$, Server computes $E(AV_S)$ (see step 5 in Section 3.2.2).

$\pi_{\mathbf{vec}}$, shown in Table 10, is a two-party protocol for Client and Server that outputs to each party the results of whether their respective encrypted activation vectors are equal (without revealing their values). Client input is $E(AV_C)$, $E(AV'_C)$ which is constructed from $E(AV_C)$ by multiplying each element by an encryption of 0. Server input is $E(AV_S)$, $E(AV'_S)$ which is constructed from $E(AV_S)$ by multiplying each element by an encryption of 0. Output to both Client and Server is results of whether $AV_C = AV_S$ or not. This subprotocol begins at global round 3 and ends at global round 8, with ZK preprocessing occurring during global rounds 1, 2 and 3. $\langle \cdot, \cdot \rangle$ denotes the inner product over $\mathbb{Z}_q$.

- At global round 3 Client chooses $r_1 \in \mathbb{Z}_q^{n-m+1}$ and $r'_1 \in \mathbb{Z}_q$. Client computes $E(AV'_C)$ by multiplying each element of AV_C with an encryption of 0, thus blinding the ciphertext. Client generates $comm(E(AV'_C))$, $comm(r_1)$, and $comm(E(r'_1))$ and sends them to Server.

– At global round 4 Server chooses $r_2 \in \mathbb{Z}_q^{n-m+1}$ and $r'_2 \in \mathbb{Z}_q$. Server computes $E(AV'_S)$ by multiplying each element of AV_S by an encryption of 0, thus blinding the ciphertext. Server sends $r_2, E(r'_2), E(AV'_S)$ to Client.

– At global round 5 Client sets $r = r_1 + r_2$ and $E(r') = E(r'_1 + r'_2)$. Client computes $z_1 = E(\langle AV_C, r\rangle + r')$ and $z_2 = E(\langle AV_S, r\rangle + r')$. Client opens the commitments of $E(AV'_C)$, r_1, and $E(r'_1)$ to Server. Client sends z_1 and z_2 to Server. Client computes partial decryptions $D_C(z_1)$, $D_C(z_2)$ and sends $D_C(z_1)$, $D_C(z_2)$ to Server, as well as $A_{PD}^{P,2}$, to prove that the partial decryptions $D_C(z_1)$, $D_C(z_2)$ are computed correctly, whereas messages $A_{PD}^{P,1}$ and $A_{PD}^{V,1}$ are sent during global rounds 1 and 2, respectively. Execution of Client's A_{PD} continues until global round 7.

– At global round 6 Server obtains z_1, z_2 from $D_C(z_1)$ and $D_C(z_2)$. Server aborts if $z_1 \neq z_2$. Server sets $r = r_1 + r_2$ and $E(r') = E(r'_1 + r'_2)$. Server computes $z_1 = E(\langle AV_C, r\rangle + r')$ and $z_2 = E(\langle AV_S, r\rangle + r')$. Server computes partial decryptions $D_S(z_1)$, $D_S(z_2)$ and sends $D_S(z_1)$ and $D_S(z_2)$ to Client, as well as $A_{PD}^{P,2}$, to prove that the partial decryptions $D_S(z_1)$, $D_S(z_2)$ are computed correctly, whereas $A_{PD}^{P,1}$ and $A_{PD}^{V,1}$ are sent during global rounds 2 and 3, respectively. Execution of Server's A_{PD} continues until global round 8.

– At global round 7 Client obtains z_1, z_2 from $D_S(z_1)$ and $D_S(z_2)$. Client aborts if $z_1 \neq z_2$.

Since r and r' are uniform, the probability that z_1 and z_2 have equal decryptions for unequal vectors is negligible ($\frac{1}{q}$).

$\pi_{\mathbf{rand}}$, shown in Table 11, is a two-party protocol for Client and Server that outputs to Client an encrypted vector $E(AV_S^r)$ that contains randomizations of the values in non-matching (non-zero) positions in $E(AV'_S)$. Client input is nothing. Server input is $E(AV'_S)$. Output to Client is $E(AV_S^r)$. Server receives no output. Client is assumed to already know $E(AV'_S)$. This subprotocol starts at global round 6 and ends at global round 8, with ZK preprocessing occurring during global rounds 2 and 3.

– At global round 6 Server computes $E(AV_S^r)$ from $E(AV'_S)$ by exponentiating each encryption in $E(AV'_S)$ by a random value. Server sends $E(AV_S^r)$ to Client and sends $A_{rand}^{P,2}$ to prove that $E(AV_S^r)$ was obtained correctly from $E(AV'_S)$, where $A_{rand}^{P,1}$ and $A_{rand}^{V,1}$ are sent during global rounds 2 and 3, respectively . The last message of Server's A_{rand} is exchanged at global round 8.

$\pi_{\mathbf{ans}}$, shown in Table 12, is a two-party protocol for Client and Server that outputs to Client the randomization, AV_S^r, of Server's activation vector AV_S. Client input is none. Server input is $E(AV_S^r)$. Server receives no output. Output for Client is AV_S^r. Client is assumed to already know $E(AV_S^r)$. This subprotocol starts at global round 6 and ends at global round 8, with ZK preprocessing occurring during global rounds 2 and 3.

– At global round 6 Server sends $comm(D_S(E(AV_S^r)))$. Server also sends the message $comm(A_{FD}^{P,2})$, where A_{FD} is the argument to prove that either $D_S(E(AV_S^r))$ was obtained correctly or that Server knows s^* (for h^* sent by Client in the first global round during π_{encr}), whereas $A_{FD}^{P,1}$ and $A_{FD}^{V,1}$ are sent during global rounds 2 and 3, respectively.

– At global round 7 Client sends $A_{FD}^{V,2}$ to Server (our AoKs are public coin so $A_{FD}^{V,2}$ is not determined by$A_{FD}^{P,2}$).

Global Round	Client	Messages	Server
1		$\xrightarrow{A_{PD}^{P,1}(D_C(z_1)),A_{PD}^{P,1}(D_C(z_2))}$	
2		$\xleftarrow{A_{PD}^{P,1}(D_S(z_1)),A_{PD}^{P,1}(D_S(z_2)),A_{PD}^{V,1}(D_C(z_1)),A_{PD}^{V,1}(D_C(z_2))}$	
3	$r_1 \in \mathbb{Z}_q^{n-m+1}$, $r'_1 \in \mathbb{Z}_q$, $E(r'_1) \leftarrow r'_1$, $E(AV'_C) \leftarrow E(AV_C)$	$\xrightarrow[A_{PD}^{V,1}(D_S(z_1)),A_{PD}^{V,1}(D_S(z_2))]{comm(E(AV'_C)),comm(r_1),comm(E(r'_1))}$	
4		$\xleftarrow{E(AV'_S),r_2,E(r'_2)}$	$r_2 \in \mathbb{Z}_q^{n-m+1}$, $r'_2 \in \mathbb{Z}_q$, $E(r'_2) \leftarrow r'_2$, $E(AV'_S) \leftarrow E(AV_S)$
5	$r \leftarrow r_1 + r_2$, $E(r') \leftarrow E(r'_1 + r'_2)$ $z_1 \leftarrow E(\langle AV_C, r\rangle + r')$, $z_2 \leftarrow E(\langle AV_S, r\rangle + r')$ $D_C(z_1) \leftarrow z_1$, $D_C(z_2) \leftarrow z_2$	$\xrightarrow[z_1,z_2,D_C(z_1),D_C(z_2),A_{PD}^{P,2}(D_C(z_1)),A_{PD}^{P,2}(D_C(z_2))]{decom(E(AV'_C)),decom(r_1),decom(E(r'_1))}$	
6		$\xleftarrow[A_{PD}^{V,2}(D_C(z_1)),A_{PD}^{V,2}(D_C(z_2))]{z_1,z_2,D_S(z_1),D_S(z_2),A_{PD}^{P,2}(D_S(z_1)),A_{PD}^{P,2}(D_S(z_2))}$	$D_S(D_C(z_1)) \overset{?}{=} D_S(D_C(z_2))$, $r \leftarrow r_1 + r_2$ $z_1 \leftarrow E(\langle AV_C, r\rangle + r')$, $z_2 \leftarrow E(\langle AV_S, r\rangle + r')$ $D_S(z_1) \leftarrow z_1$, $D_S(z_2) \leftarrow z_2$
7	$D_C(D_S(z_1)) \overset{?}{=} D_C(D_S(z_2))$	$\xrightarrow{A_{PD}^{P,3}(D_C(z_1)),A_{PD}^{P,3}(D_C(z_2)),A_{PD}^{V,2}(D_S(z_1)),A_{PD}^{V,2}(D_S(z_2))}$	
8		$\xleftarrow{A_{PD}^{P,3}(D_S(z_1)),A_{PD}^{P,3}(D_S(z_2))}$	

Table 10. Subprotocol π_{vec}

Global Round	Client	Messages	Server
2		$\xleftarrow{A^{P,1}_{rand}(E(AV^r_S))}$	
3		$\xrightarrow{A^{V,1}_{rand}(E(AV^r_S))}$	
6		$\xleftarrow{E(AV^r_S),A^{P,2}_{rand}(E(AV^r_S))}$	$E(AV^r_S) \leftarrow E(AV'_S)$
7		$\xrightarrow{A^{V,2}_{rand}(E(AV^r_S))}$	
8		$\xleftarrow{A^{P,3}_{rand}(E(AV^r_S))}$	

Table 11. Subprotocol π_{rand}

Global Round	Client	Messages	Server
2		$\xleftarrow{A^{P,1}_{FD}(D_S(E(AV^r_S)))}$	
3		$\xrightarrow{A^{V,1}_{FD}(D_S(E(AV^r_S)))}$	
6		$\xleftarrow[comm(A^{P,2}_{FD}(D_S(E(AV^r_S))))]{comm(D_S(E(AV^r_S))),}$	$D_S(E(AV^r_S)) \leftarrow E(AV^r_S)$
7		$\xrightarrow{A^{V,2}_{FD}(D_S(E(AV^r_S)))}$	
8	$AV^r_S \leftarrow D_C(D_S(E(AV^r_S)))$	$\xleftarrow[decom(A^{P,2}_{FD}(D_S(E(AV^r_S))))]{decom(D_S(E(AV^r_S))),A^{P,3}_{FD}(D_S(E(AV^r_S)))}$	

Table 12. Subprotocol π_{ans}

- At global round 8 Server opens commitments of $A^{P,2}_{FD}$, $D_S(E(AV^r_S))$ to Client. Server sends $A^{P,3}_{FD}$ to Client.
- Client aborts if it does not accept the argument A_{FD}.

7 Security Analysis

Here we define and prove security for π^H_{5PM} and π^M_{5PM}.

7.1 Adversarial Model

Let $\mathcal{F}_f$ be a functionality for two parties P_1 and P_2 where P_1 inputs x_1, P_2 inputs x_2, P_1 obtains $f(x_1, x_2)$, and P_2 obtains nothing. A protocol π_f that securely computes f can be defined as an interactive two-party protocol. We refer the reader to [33,38] for further discussion of the definitions given here.

Execution of π^A_f in the real world: Let A denote the adversary model: H for honest but curious and M for malicious (static corruption). Both parties are assumed to be probabilistic (expected) polynomial time (PPT) algorithms. In particular, we denote by $\bar{P} = (P_1, P_2)$ a pair of PPT algorithms that execute π^A_f, where at most one of the parties is adversarial (or corrupted). Such a pair $\bar{P}$ is called *admissible*. Let r_i be P_i's internal randomness, x_i be P_i's private input and y_i be P_i's auxiliary input. Let $P_1 \leftrightarrow P_2$ be the transcript of the public interactions between parties P_1 and P_2. Note that parties can be defined via their next message functions; see, for example, [39].

In the honest-but-curious (HBC) adversary model, before the protocol begins, the adversary can choose to corrupt one party for the duration of the entire protocol; that party may not deviate from the protocol specification of π_f^A. In the (static corruption) malicious adversary model, before the protocol begins, the adversary can choose to corrupt one party for the duration of the entire protocol; this party may deviate arbitrarily from the protocol specification of π_f^M. In particular, the corrupted party may choose to abort and to not complete the protocol at all. Denote by x_c, y_c, r_c the internal input, randomness and auxiliary input of a corrupted party, respectively (if there is no corrupted party, then this sequence is the empty sequence). Denote $\bar{x} = (x_1, x_2)$, $\bar{y} = (y_1, y_2)$, $\bar{r} = (r_1, r_2)$. We denote $REAL_{\bar{P}}^{\pi_f^A}(\bar{x}, \bar{y}, \bar{r})$ as $(x_c, r_c, y_c, P_1 \leftrightarrow P_2)$.

Execution of π_f^A in the ideal world: In the ideal world setting, an admissible pair of two PPT parties $\bar{P}' = (P_1', P_2')$ interact with a trusted ideal functionality to jointly compute the function f specified by π_f^A. At any point, a dishonest party may send abort rather than send what it is supposed to.

Ideal functionality $\mathcal{F}_f^H$ for the honest-but-curious (HBC) model: P_2' sends its input, x_2, to the ideal functionality. The ideal functionality sends the size of x_2, $|x_2|$, to P_1'. P_1' sends its input, x_1, to the ideal functionality. The ideal functionality sends the size of x_1, $|x_1|$, to P_2'. The ideal functionality provides the correct value of $f(x_1, x_2)$ to P_1'. An honest party must output what was output to it by the ideal functionality (in particular, if P_2' is honest, it outputs nothing); a dishonest party may output what it wishes. We denote by $IDEAL_{\bar{P}'}^{\pi_f}(\bar{x}, \bar{y}, \bar{r})$ as the pair of public outputs of P_1' and P_2'.

Ideal functionality $\mathcal{F}_f^M$ for the malicious static corruption model: P_2' sends its input, x_2, to the ideal functionality. The ideal functionality sends the size of x_2, $|x_2|$, to P_1'. P_1' sends its input, x_1, to the ideal functionality. The ideal functionality sends the size of x_1, $|x_1|$, to P_2'. P_2' sends "proceed" to the ideal functionality (note that an adversarial party can choose to abort this procedure at any time). The ideal functionality provides the correct value of $f(x_1, x_2)$ to P_1'. An honest party must output what was output to it by the ideal functionality (in particular, if P_2' is honest, it outputs nothing); a dishonest party may output what it wishes. We denote $IDEAL_{\bar{P}'}^{\pi_f^A}(\bar{x}, \bar{y}, \bar{r})$ as the pair of public outputs of P_1' and P_2'.

Using the standard ideal/real formulation, we obtain the following definitions of security.

Definition 5. *π_f^H securely realizes $\mathcal{F}_f^H$ in the honest-but-curious (static corruption) model if, for every admissible PPT pair $\bar{P}$ in the real world, there exists an admissible PPT pair $\bar{P}'$ in the ideal world such that $REAL_{\bar{P}}^{\pi_f^H}(\bar{x}, \bar{y}, \bar{r})$ and $IDEAL_{\bar{P}'}^{\pi_f^H}(\bar{x}, \bar{y}, \bar{r})$ are computationally indistinguishable, where the distributions are over the uniformly random, independent choices of private input $\bar{x}$, randomness $\bar{r}$ and auxiliary input $\bar{y}$.*

Definition 6. *π_f^M securely realizes $\mathcal{F}_f^M$ in the malicious (static corruption) model if, for every admissible PPT pair $\bar{P}$ in the real world, there exists an admissible PPT pair $\bar{P}'$ in the ideal world such that $REAL_{\bar{P}}^{\pi_f^M}(\bar{x}, \bar{y}, \bar{r})$ and $IDEAL_{\bar{P}'}^{\pi_f^M}(\bar{x}, \bar{y}, \bar{r})$ are computationally indistinguishable, where the distributions are over the uniformly random, independent choices of private input $\bar{x}$, randomness $\bar{r}$ and auxiliary input $\bar{y}$.*

Simulation in the ideal world: In practice, for security to hold in the HBC (respectively malicious) adversary model, for each corrupted party P_i' in the real world, there exists a PPT simulator

$\mathcal{S}_{P'_i}$ in the ideal world with oracle access to P'_i such that $REAL_{\bar{P}}^{\pi_f}(\bar{x}, \bar{y}, \bar{r})$ and $IDEAL_{\bar{P}'}^{\pi_f}(\bar{x}, \bar{y}, \bar{r})$ are computationally indistinguishable, where the other (honest) party of $\bar{P}$ and $\bar{P}'$ acts honestly. In particular, $\mathcal{S}_{P'_i}$, when it is allowed to output per the ideal-world specification, will attempt to output a transcript that is computationally indistinguishable from P'_i's view of the transcript in the real world without knowing the private input of the real-world honest party.

7.2 Simulator Constructions and Security for π_{5PM}^{H}

We provide, for each admissible pair in the real world, an admissible pair in the ideal world such that $REAL_{\bar{P}}^{\pi_{5PM}^{H}}(\bar{x}, \bar{y}, \bar{r})$ and $IDEAL_{\bar{P}'}^{\pi_{5PM}^{H}}(\bar{x}, \bar{y}, \bar{r})$ are computationally indistinguishable.

7.2.1 Simulator Specification for the Adversarial Server, $\mathcal{S}_S$, for π_{5PM}^{H}. Without loss of generality, consider the case where the matching locations are not hidden. Consider the admissible pair $\bar{P}$ = (Client, Server) in the real world. We construct $\mathcal{S}_S$ for an admissible pair $\bar{P}'$ =(Client, $\mathcal{S}_S$) in the ideal world (where Client behaves honestly in both cases) such that $REAL_{\bar{P}}^{\pi_{5PM}^{H}}(\bar{x}, \bar{y}, \bar{r})$ and $IDEAL_{\bar{P}'}^{\pi_{5PM}^{H}}(\bar{x}, \bar{y}, \bar{r})$ are computationally indistinguishable. Note that $\mathcal{S}_S$ has oracle access to real-world Server.

Initial Interactions With the Ideal Functionality: We assume that the encryption scheme (Key, E, D) is fixed. Server, upon oracle call to $\mathcal{S}_S$, sends its text T to $\mathcal{S}_S$, who forwards it on to the ideal functionality. Once the ideal functionality reveals the length that the pattern should be, $\mathcal{S}_S$ sets the pattern $p_t = p^*$ to be all 1s (we state this without loss of generality; in the case that $1 \notin \Sigma$, an arbitrary $a \in \Sigma$ is chosen and p_t is set to be all a's) of the right length (any arbitrary vector can be used here, but without loss of generality we use all 1s). The following is what $\mathcal{S}_S$ outputs at the output phase of the ideal-world specification.

(a) $\mathcal{S}_S$ computes $(sk_C, pk_C) \leftarrow Key(1^k)$. Using p_t, $\mathcal{S}_S$ constructs $M_{CDV} \leftarrow Gen_{CDV}(p_t)$ and encrypts it to obtain $E(M_{CDV})$. $\mathcal{S}_S$ sends $E(M_{CDV})$ and pk_C to Server. In addition, $\mathcal{S}_S$ sends $E(-m)$, where $m = |p_t|$ (or $E(-m + l)$ in the single-character wildcard or substring cases, where l is the threshold).

(b) Server considers each character in the text T_i $(1 \le i \le n)$ and retrieves the corresponding row of $E(M_{CDV})$. This is the step that corresponds to multiplying $M_T \cdot M_{CDV}$ in Section 3.1. The resulting vectors are multiplied with the encrypted activation vector $E(AV)$ element by element in positions $i, ..., i+m-1$ of the AV. This is the step corresponding to transforming $M_{T(M_{CDV})}$ to $\overline{M}_{T(M_{CDV})}$ and then performing the multiplication $[1...1]^n \cdot \overline{M}_{T(M_{CDV})}$ to get the final AV. Server then multiplies $E(-m)$ $(= E(m)^{-1})$ to each of the entries in AV and exponentiates each entry by a randomly chosen number to blind entries in $E(AV)$. We call the randomized activation vector $E(AV_S^r)$. Server sends $E(AV_S^r)$ to $\mathcal{S}_S$.

(c) $\mathcal{S}_S$ aborts the ideal functionality before Client outputs its pattern-matching results obtained from the ideal functionality.

7.2.2 Simulator Specification for the Adversarial Client, $\boldsymbol{\mathcal{S}_C}$, for $\boldsymbol{\pi^H_{5PM}}$. Consider the admissible pair $\bar{P}$ = (Client, Server) in the real world. We construct $\mathcal{S}_S$ for an admissible pair $\bar{P'}$ =($\mathcal{S}_C$, Server) in the ideal world (where Server behaves honestly in both cases) such that $REAL^{\pi^H_{5PM}}_{\bar{P}}(\bar{x},\bar{y},\bar{r})$ and $IDEAL^{\pi^H_{5PM}}_{\bar{P'}}(\bar{x},\bar{y},\bar{r})$ are computationally indistinguishable. Note that $\mathcal{S}_S$ has oracle access to real-world Client.

Initial Interactions with the Ideal Functionality: We assume that the encryption scheme (Key, E, D) is fixed. Upon oracle call to Client, Client sends pattern p to $\mathcal{S}_C$, who forwards it on to the ideal functionality. Once the ideal functionality reveals the length that the text should be, $\mathcal{S}_C$ sets the text $T = T^*$ to be all 1s (we note that, as above, if $1 \notin \Sigma$, T^* can be set to all a's for any $a \in \Sigma$) of the right length (any arbitrary vector can be used here, but without loss of generality we use all 1s).

(a) Client computes $(sk_C, pk_C) \leftarrow Key(1^k)$. Using, p, Client constructs $M_{CDV} \leftarrow Gen_{CDV}(p)$ and encrypts it to obtain $E(M_{CDV})$. Client sends $E(M_{CDV})$ and pk_C to $\mathcal{S}_C$. In addition, Client sends $E(-m)$, where $m = |p|$ (or $E(-m + l)$ in the single-character wildcard or substring cases, where l is the threshold).

(b) The ideal functionality sends $\mathcal{S}_C$ the pattern-matching results with the correct (e.g., real world) pattern and text (e.g., all positions where the pattern should match). $\mathcal{S}_C$ constructs a new vector of encryptions, $E(AV_{IF})$, by using Client's public key to encrypt an $(m+n-1)$-length vector with 0s where the pattern should match, and random elements elsewhere. $\mathcal{S}_C$ sends $E(AV_{IF})$ to Client.

7.2.3 Security of $\boldsymbol{\pi^H_{5PM}}$. We prove that π^H_{5PM} securely realizes $\mathcal{F}^H_{5PM}$ in the honest-but-curious (static corruption) model by demonstrating the computational indistinguishability of $REAL^{\pi^M_{5PM}}_{\bar{P}}(\bar{x},\bar{y},\bar{r})$ and $IDEAL^{\pi^M_{5PM}}_{\bar{P'}}(\bar{x},\bar{y},\bar{r})$.

Theorem 4. *Given an additively homomorphic semantically secure encryption scheme over prime-order cyclic groups (Key, E, D), π^H_{5PM} securely realizes $\mathcal{F}^H_{5PM}$ in the honest-but-curious (static corruption) model.*

Proof (Theorem 4). We demonstrate that the two simulators $\mathcal{S}_S$ and $\mathcal{S}_C$ output transcripts such that $REAL^{\pi^H_{5PM}}_{\bar{P}}(\bar{x},\bar{y},\bar{r})$ and $IDEAL^{\pi^H_{5PM}}_{\bar{P'}}(\bar{x},\bar{y},\bar{r})$ are computationally indistinguishable.

Case 1: Adversarial Server. The transcript in the real world is the transcript $(pk_C, T, E(M_{CDV})_p, E(-m))$, where by $E(M_{CDV})_p$ we mean the encrypted M_{CDV} matrix constructed from p. Note then that $E(M_{CDV})_p$ is the encrypted M_{CDV} matrix constructed from p_t. The view in the ideal world is $(pk_C, T, E(M_{CDV})_{p_t}, E(m))$, where p_t is a string of m 1s. Suppose a distinguisher D can distinguish the distributions of the real and ideal transcripts with non-negligible probability. In particular, D distinguishes $(E(M_{CDV})_p)$ from $(E(M_{CDV})_{p_t})$ with non-negligible probability given pk_C and T. In particular, let a distinguisher D have as input pk_C and T. Define the distribution

$$X_i = (E(M_{CDV})_{p,1}, ..., E(M_{CDV})_{p,k}, E(M_{CDV})_{p_t,k+1}, ..., E(M_{CDV})_{p_t,m|\Sigma|}),$$

where $E(M_{CDV})_{p_t,i}$ is the ith encrypted element in $E(M_{CDV})$ constructed from the pattern p_t (where the matrix here is thought of as a string). By a hybrid argument, for some $0 \leq k \leq$

$m|\varSigma|$, given pk_C and T, D can distinguish X_k from X_{k+1} in polynomial time with non-negligible probability. But this violates the semantic security of E since D only has the public key of E and T, both of which are independent of p/p_t.

More precisely, let the hybrid experiment H_i, $0 \leq i \leq m|\varSigma|$ be such that the first i encryptions sent by the simulator come from $M_{CDV_{p_t}}$ while the remaining encryptions come from M_{CDV_p}. Note that H_0 is the distribution of the real Client while $H_{m|\varSigma|}$ is the distribution of the simulator in the ideal world. Suppose that H_i is computationally indistinguishable from H_{i+1} for some i. Then we reduce security to the semantic security of ElGamal encryption. Namely, we consider a player P_{enc} who encrypts and another R who receives. P_{enc} is given, as auxiliary inputs, the p and T such that H_i and H_{i+1} are computationally distinguishable with non-negligible probability via a distinguisher D. P_{enc} sends R the public key, which R uses to internally execute H_i and H_{i+1} using p_t, p and T. At the $i+1$ encryption, R sends P_{enc} the two plain texts from $M_{CDV_{p_t}}$ and M_{CDV_p} used for the $i+1$ encryption in the respective hybrids; P_{enc} sends back the encryptions in a randomized order. P_{enc} continues the internal execution. Note that these distributions are identical to H_i and H_{i+1}. If H_i and H_{i+1} are computationally distinguishable with non-negligible probability using D, then R, using D, can distinguish the two encryptions with non-negligible probability, which implies that ElGamal encryption is not semantically secure; this a contradiction.

Case 2: Adversarial Client.

The transcript in the real world is the transcript (sk_C, p, m, AV_S^r), while the view in the ideal world is the transcript (sk_C, p, m, AV_{IF}^r). Indistinguishability here is statistical; namely, by construction, AV_S^r and AV_{IF}^r have zeros in the same places, so that their non-zero locations contain elements chosen uniformly (and independently) at random since the group has prime order. This implies statistical indistinguishability of the transcripts (in particular, in this case, security does not rely on the semantic security of the encryption scheme at all, other than to hide the matching result from an eavesdropper).

7.3 Simulator Constructions and Security for π_{5PM}^M

We provide, for each admissible pair in the real world, an admissible pair in the ideal world such that $REAL_{\bar{P}}^{\pi_{5PM}^M}(\bar{x}, \bar{y}, \bar{r})$ and $IDEAL_{\bar{P'}}^{\pi_{5PM}^M}(\bar{x}, \bar{y}, \bar{r})$ are computationally indistinguishable.

In what follows, we assume that if a message is incorrectly formatted, the simulator will simply abort; since this might occur at every individual interaction, we omit it for simplicity.

7.3.1 Simulator Specification for an Adversarial Server, $\mathcal{S}_S$, for π_{5PM}^M. We describe how an adversarial Server interacts with $\mathcal{S}_S$ for π_{5PM}^M. In particular, consider the admissible pair $\bar{P}$ = (Client, Server) in the real world. We construct $\mathcal{S}_S$ for an admissible pair $\bar{P'}$ =(Client, $\mathcal{S}_S$) in the ideal world (where Client behaves honestly in both cases) such that $REAL_{\bar{P}}^{\pi_{5PM}^M}(\bar{x}, \bar{y}, \bar{r})$ and $IDEAL_{\bar{P'}}^{\pi_{5PM}^M}(\bar{x}, \bar{y}, \bar{r})$ are computationally indistinguishable. Note that $\mathcal{S}_S$ will have oracle access to Server.

We list the subprotocols of π_{5PM}^M and describe how an adversarial Server interacts with $\mathcal{S}_S$. Recall that π_{encr} is a protocol to instantiate a threshold ElGamal encryption scheme, $\pi_{C,AV}$ is for Client to compute an activation vector, $\pi_{S,AV}$ is for Server to compute an activation vector, π_{vec} is for the parties to determine that their activation vectors are equal, π_{rand} is for Server to send

Client an encrypted vector that only reveals matching locations upon decryption, and π_{ans} is for Server to partially decrypt that encrypted vector. We refer the reader to Section 3.3.3 and Section 6.2 for protocol details.

Initial Interactions With the Ideal Functionality: We may view the following interaction with the ideal functionality as occurring during the execution of $\pi_{C,AV}$ between $\mathcal{S}_S$ and Server. Upon oracle call, Server reveals the length of its text to $\mathcal{S}_S$ (it will do so in the protocol when it sends $E(M_T)$ during $\pi_{C,AV}$, since this matrix is of size $|T| \times |\Sigma|$). For the purpose of ideal functionality interaction, $\mathcal{S}_S$ will set the text T' to be all 1s (here and throughout this specification, this is without loss of generality assuming $1 \in \Sigma$; else another character may be used) with length the same as Server's text, and send T' to the ideal functionality. The ideal functionality will return the length of Client's pattern. $\mathcal{S}_S$ sets the pattern $p_t = p^*$ to be all 1s of the length of the pattern (any arbitrary vector can be used here, but without loss of generality we use all 1s). $\mathcal{S}_S$ will then abort the ideal functionality so that (ideal world) Client does not output anything in the ideal functionality. The following is what $\mathcal{S}_S$ will output for $IDEAL^{\pi_{5PM}^M}$ (explaining how $\mathcal{S}_S$ makes oracle calls to Server).

$\pi_{\mathbf{encr}}$:

This subprotocol begins at the first global round and ends at global round 6.

(a) $\mathcal{S}_S$ chooses $s_C, s^* \in \mathbb{Z}_q$ and sets $h_1 \leftarrow g^{s_C}$, $h^* \leftarrow g^{s^*}$. $\mathcal{S}_S$ sends h_1, h^* to Server. $\mathcal{S}_S$ sends the Server two parallel instantiations of $\pi_{DL}^{P,1}$ for h_1 and h^* with witnesses s_C and s^*. This continues through global round 5.

(b) Server sends h_2 (normally equal to g^{s_S}) to $\mathcal{S}_S$ as well as $\pi_{DL}^{P,1}$ for h_2 with witness s_S. Protocol π_{DL} continues through global round 6. Both parties set $h = h_1 h_2$ as the public key (with secret key $s = s_C + s_S$).

$\pi_{\mathbf{C,AV}}$:

This subprotocol starts at global round 2 and ends at global round 6.

(a) Server sends $E(M_T)$ to $\mathcal{S}_S$. Server also sends, for $E(M_T)$, $A_{M1}^{P,1}$ to demonstrate that $E(M_T)$ is formatted correctly . A_{M1} continues through global round 6.

(b) During global round 3, $\mathcal{S}_S$ computes $E(AV_C)$ from $E(M_T)$, p_t and p as specified in π_{5PM}^M.

$\pi_{\mathbf{S,AV}}$:

This subprotocol starts at global round 3 and ends at global round 5, with ZK preprocessing occurring during global rounds 1 and 2.

(a) $\mathcal{S}_S$ sends $E(M_{CDV})$, constructed using p_t, and $E(|p_t|)$ to Server. $\mathcal{S}_S$ also sends $A_{M01}^{P,2}$ to demonstrate that $E(M_{CDV})$ is formed correctly, where $A_{M01}^{P,1}$ and $A_{M01}^{V,1}$ occur during global rounds 1 and 2, respectively. A_{M01} continues until global round 5.

$\pi_{\mathbf{vec}}$:

This subprotocol begins at global round 3 and ends at global round 8, with ZK preprocessing occurring during global rounds 1, 2 and 3. $\langle \cdot, \cdot \rangle$ denotes the inner product over $\mathbb{G}_q$.

(a) At global round 3 $\mathcal{S}_S$ chooses $r_1 \in \mathbb{G}_q^{n-m+1}$ and $r'_1 \in \mathbb{G}_q$. $\mathcal{S}_S$ computes $E(AV'_C)$ by multiplying each element of AV_C with an encryption of 0 thus blinding the ciphertext. $\mathcal{S}_S$ generates $comm(E(AV'_C))$, $comm(r_1)$, and $comm(E(r'_1))$ and sends them to Server.

(b) Server sends $r_2, E(r'_2), E(AV'_S)$ to $\mathcal{S}_S$.

(c) At global round 5, $\mathcal{S}_S$ sets $r = r_1 + r_2$ and $E(r') = E(r'_1 + r'_2)$. $\mathcal{S}_S$ computes $z_1 = E(\langle AV_C, r\rangle + r')$ and $z_2 = E(\langle AV_S, r\rangle + r')$. $\mathcal{S}_S$ opens the commitments of $E(AV'_C)$, r_1, and $E(r'_1)$ to Server. $\mathcal{S}_S$ sends z_1 and z_2 to Server. $\mathcal{S}_S$ computes partial decryptions $D_C(z_1)$, $D_C(z_2)$ and sends $D_C(z_1)$, $D_C(z_2)$ to Server as well as $A_{PD}^{P,2}$ to prove that the partial decryptions $D_C(z_1)$, $D_C(z_2)$ are computed correctly, where messages $A_{PD}^{P,1}$ and $A_{PD}^{V,1}$ are sent during global rounds 1 and 2, respectively. Execution of $\mathcal{S}_S$'s A_{PD} continues until global round 7.

(d) At global round 6, Server sends (for independently obtained z_1 and z_2), $D_S(z_1)$ and $D_S(z_2)$ to $\mathcal{S}_S$ as well as $A_{PD}^{P,2}$ to prove that the partial decryptions $D_S(z_1)$, $D_S(z_2)$ are computed correctly, where $A_{PD}^{P,1}$ and $A_{PD}^{V,1}$ are sent during global rounds 2 and 3, respectively. Execution of Server's A_{PD} continues until global round 8.

(e) At global round 7 $\mathcal{S}_S$ obtains z_1, z_2 from $D_S(z_1)$ and $D_S(z_2)$. $\mathcal{S}_S$ aborts if $z_1 \neq z_2$.

$\mathcal{S}_S$ aborts if the decryptions of z_1 and z_2 do not equal each other.

$\pi_{\mathbf{rand}}$:

This subprotocol starts at global round 6 and ends at global round 8, with ZK preprocessing occurring during global rounds 2 and 3.

(a) Server sends $E(AV_S^r)$ to $\mathcal{S}_S$ and sends $A_{rand}^{P,2}$ that $E(AV_S^r)$ was obtained correctly from $E(AV'_S)$, where $A_{rand}^{P,1}$ and $A_{rand}^{V,1}$ are sent during global rounds 2 and 3, respectively. This A_{rand} continues until global round 8.

$\pi_{\mathbf{ans}}$:

This subprotocol starts at global round 6 and continues until global round 8.

(a) At global round 6, Server sends $comm(D_S(E(AV_S^r)))$. Server also sends the message $comm(A_{FD}^{P,2})$, where A_{FD} is the argument to prove that either $D_S(E(AV_S^r))$ was obtained correctly or that Server knows s^* (for h^* sent by $\mathcal{S}_S$ in the first global round during π_{encr}), where $A_{FD}^{P,1}$ and $A_{FD}^{V,1}$ are sent during global rounds 2 and 3, respectively.

(b) At global round 7, $\mathcal{S}_S$ sends $A_{FD}^{V,2}$ to Server (our AoKs are public coin so $A_{FD}^{V,2}$ is not determined by $A_{FD}^{P,2}$).

(c) At global round 8, Server opens commitments of $A_{FD}^{P,2}$, $D_S(E(AV_S^r))$ to $\mathcal{S}_S$. Server sends $A_{FD}^{P,3}$ to $\mathcal{S}_S$.

(d) $\mathcal{S}_S$ aborts if it does not accept the argument A_{FD}.

7.3.2 Simulator Specification for an Adversarial Client, $\mathcal{S}_C$, for π_{5PM}^M. Consider the admissible pair $\bar{P}$ = (Client, Server) in the real world. We construct $\mathcal{S}_S$ for an admissible pair $\bar{P'}$ =($\mathcal{S}_C$, Server) in the ideal world (where Server behaves honestly in both cases) such that $REAL_{\bar{P}}^{\pi_{5PM}^M}(\bar{x}, \bar{y}, \bar{r})$ and $IDEAL_{\bar{P'}}^{\pi_{5PM}^M}(\bar{x}, \bar{y}, \bar{r})$ are computationally indistinguishable. Note that $\mathcal{S}_S$ has oracle access to real-world Client.

We list the subprotocols of π_{5PM}^{M} and describe how Client interacts with $\mathcal{S}_C$. Unlike the case for $\mathcal{S}_S$, $\mathcal{S}_C$ must interact with the ideal functionality in the final rounds of the protocol because Client receives an output from π_{5PM}^{M}; thus $\mathcal{S}_C$ must retrieve this output in the final steps of the simulation. Recall that π_{encr} is a protocol to instantiate a threshold ElGamal encryption scheme, $\pi_{C,AV}$ is for Client to compute an activation vector, $\pi_{S,AV}$ is for Server to compute an activation vector, π_{vec} is for the parties to determine that their activation vectors are equal, π_{rand} is for Server to send Client an encrypted vector that only reveals matching locations upon decryption, and π_{ans} is for Server to partially decrypt that encrypted vector. We refer the reader to Section 3.3.3 and Section 6.2 for details.

Initial Interactions With the Ideal Functionality: Once the ideal functionality reveals to $\mathcal{S}_C$ the length that the text should be, $\mathcal{S}_C$ sets the text $T = T^*$ to be all 1s of the right length (any arbitrary vector can be used here, but without loss of generality we use all 1s where $1 \in \Sigma$ and some other fixed character in Σ otherwise).

$\pi_{\mathbf{encr}}$:

This subprotocol begins at the first global round and ends at global round 6.

(a) Client sends h_1, h^* to $\mathcal{S}_C$ (where in the honest execution, $h_1 = g^{s_C}$ and $h^* = g^{s^*}$). Client sends $\mathcal{S}_C$ two parallel instantiations of $\pi_{DL}^{P,1}$ for h_1 and h^* with witnesses s_C and s^*. This continues through global round 5.

(b) $\mathcal{S}_C$ chooses $s_S \in \mathbb{Z}_q$ and sets $h_2 \leftarrow g^{s_S}$. $\mathcal{S}_C$ sends h_2 to Client as well as $\pi_{DL}^{P,1}$ for h_2 with witness s_S. Protocol π_{DL} continues through global round 6. Both parties set $h = h_1 h_2$ as the public key (with secret key $s = s_C + s_S$).

(c) Using as a subprotocol the extractor E, which guaranteed since π_{DL} is an argument of knowledge, $\mathcal{S}_C$ rewinds from global round 5 to global round 2 and interacts with Client until global round 5 and then rewinds again until it extracts s^* from Client, at which point it rewinds once more and executes π_{DL} per the usual specification. $\mathcal{S}_C$ only uses E for its responses related to the proof of knowledge π_{DL} for s^*; only messages relating to π_{DL} for s^* are affected from one rewinding to the next. $\mathcal{S}_C$, using E, is guaranteed to succeed in extracting s^* in (expected) polynomial time since π_{DL} is an argument of knowledge.

(d) Using as a subprotocol the extractor E, which is guaranteed since π_{DL} is an argument of knowledge, $\mathcal{S}_C$ rewinds from global round 5 to global round 2 and interacts with Client until global round 5 and then rewinds again until it extracts s_C from Client and then rewinds again until it extracts s^* from Client, at which point it rewinds once more and executes π_{DL} per the usual specification. $\mathcal{S}_C$ only uses E for its responses related to the proof of knowledge π_{DL} for s_C; only messages relating to π_{DL} for s_C are affected from one rewinding to the next. $\mathcal{S}_C$, using E, is guaranteed to succeed in extracting s_C in (expected) polynomial time since π_{DL} is an argument of knowledge. Note now that $\mathcal{S}_C$ can decrypt encryptions computed by Client because $\mathcal{S}_C$ possesses both s_S and s_C.

$\pi_{\mathbf{C,AV}}$:

This subprotocol starts at global round 2 and ends at global round 6.

(a) $\mathcal{S}_C$ sends $E(M_T)$ to Client. $\mathcal{S}_C$ also sends, for $E(M_T)$, $A_{M1}^{P,1}$ that $E(M_T)$ is formatted correctly. $A_{M,1}$ continues through global round 6.

$\pi_{\mathbf{S,AV}}$:

This subprotocol starts at global round 3 and ends at global round 5, with ZK preprocessing occurring during global rounds 1 and 2.

(a) Client sends $E(M_{CDV})$ and $E(|p|)$ to $\mathcal{S}_C$. Client also sends $A_{M01}^{P,2}$ that $E(M_{CDV})$ is formed correctly, where $A_{M01}^{P,1}$ and $A_{M01}^{V,1}$ occur during global rounds 1 and 2, respectively. A_{M01} continues until global round 5.

(b) During global round 4, using $E(M_{CDV})$, T and $E(|p|)$, $\mathcal{S}_C$ computes $E(AV_S)$ as specified in π_{5PM}^{H}.

Further Interaction With the Ideal Functionality: $\mathcal{S}_C$ can decrypt encryptions computed by Client because it extracted Client's secret key s_C during π_{encr}. During $\pi_{S,AV}$, $\mathcal{S}_C$ obtains $E(M_{CDV})$. Therefore, $\mathcal{S}_C$ obtains the pattern p that Client is trying to have matched (which may be different than the pattern Client output to the real-world transcript). $\mathcal{S}_C$ resets the ideal functionality, now using this p. The output for the ideal functionality to $\mathcal{S}_C$ will be the correct matching of the pattern with the text that real Server is using. See π_{ans}, where $\mathcal{S}_C$ uses this ideal functionality output.

$\pi_{\mathbf{vec}}$:

This subprotocol begins at global round 3 and ends at global round 8, with ZK preprocessing occurring during global rounds 1, 2 and 3. $\langle \cdot, \cdot \rangle$ denotes the inner product over $\mathbb{G}_q$.

(a) At global round 3, Client chooses $r_1 \in \mathbb{G}_q^{n-m+1}$ and $r_1' \in \mathbb{G}_q$. Client computes $E(AV_C')$. Client generates $comm(E(AV_C'))$, $comm(r_1)$, and $comm(E(r_1'))$ and sends them to $\mathcal{S}_C$.

(b) At global round 4, $\mathcal{S}_C$ chooses $r_2 \in \mathbb{G}_q^{n-m+1}$ and $r_2' \in \mathbb{G}_q$. $\mathcal{S}_C$ computes $E(AV_S')$ by multiplying each element of AV_S by an encryption of 0, thus blinding the ciphertext. $\mathcal{S}_C$ sends $r_2, E(r_2'), E(AV_S')$ to Client.

(c) At global round 5, Client opens the commitments of $E(AV_C')$, r_1, and $E(r_1')$ to $\mathcal{S}_C$. Client sends elements z_1 and z_2 to $\mathcal{S}_C$. Client sends $D_C(z_1)$, $D_C(z_2)$ to $\mathcal{S}_C$ as well as $A_{PD}^{P,2}$ to prove that the partial decryptions $D_C(z_1)$, $D_C(z_2)$ are computed correctly, where messages $A_{PD}^{P,1}$ and $A_{PD}^{V,1}$ are sent during global rounds 1 and 2, respectively. Execution of Client's A_{PD} continues until global round 7.

(d) At global round 6, $\mathcal{S}_C$ obtains z_1, z_2 from $D_C(z_1)$ and $D_C(z_2)$. $\mathcal{S}_C$ aborts if $z_1 \neq z_2$. $\mathcal{S}_C$ sets $r = r_1 + r_2$ and $E(r') = E(r_1' + r_2')$. $\mathcal{S}_C$ computes $z_1 = E(\langle AV_C, r\rangle + r')$ and $z_2 = E(\langle AV_S, r\rangle + r')$. $\mathcal{S}_C$ computes partial decryptions $D_S(z_1)$, $D_S(z_2)$ and sends $D_S(z_1)$ and $D_S(z_2)$ to Client as well as $A_{PD}^{P,2}$ to prove that the partial decryptions $D_S(z_1)$, $D_S(z_2)$ are computed correctly, where $A_{PD}^{P,1}$ and $A_{PD}^{V,1}$ are sent during global rounds 2 and 3, respectively. Execution of $\mathcal{S}_C$'s A_{PD} continues until global round 8.

$\pi_{\mathbf{rand}}$:

This subprotocol starts at global round 6 and ends at global round 8, with ZK preprocessing occurring during global rounds 2 and 3.

(a) $\mathcal{S}_C$ computes $E(AV_S^r)$ from $E(AV_S')$ by exponentiating each encryption in $E(AV_S')$ by a random exponent. $\mathcal{S}_C$ sends $E(AV_S^r)$ to Client and sends $A_{rand}^{P,2}$ that $E(AV_S^r)$ was obtained correctly from $E(AV_S')$, where $A_{rand}^{P,1}$ and $A_{rand}^{V,1}$ are sent during global rounds 2 and 3, respectively. This A_{rand} continues until global round 8.

$\boldsymbol{\pi_{\mathbf{ans}}}$:

This subprotocol starts at global round 6 and continues until global round 8.

(a) At this stage (global round 6), the ideal functionality sends $\mathcal{S}_C$ the correct output that Client should receive (e.g., the locations in a string of length $|AV_S|$ that tells Client where matches occur (see "Further interactions with the ideal functionality" after $\pi_{S,AV}$ above). Denote AV_{IF} as the string that Client should receive per the ideal functionality. $\mathcal{S}_C$ computes $E(AV_{IF}^r)$ from AV_{IF} by setting the non-matching locations of AV_{IF} to be random elements and then encrypting the vector. $\mathcal{S}_C$ computes and sends $comm(D_S(E(AV_{IF}^r)))$ to Client. $\mathcal{S}_C$ also sends the message $comm(A_{FD}^{P,2})$, where A_{FD} is the argument to prove that either $D_S(E(AV_{IF}^r))$ was obtained correctly or that $\mathcal{S}_C$ knows s^* (for h^* sent by Client in the first global round during π_{encr}), where $A_{FD}^{P,1}$ and $A_{FD}^{V,1}$ are sent during global rounds 2 and 3, respectively. Note that here, $\mathcal{S}_C$ uses the witness for knowledge of s^*.

(b) At global round 7, Client sends $A_{FD}^{V,2}$ to $\mathcal{S}_C$ (our AoKs are public coin so $A_{FD}^{V,2}$ is not determined by $A_{FD}^{P,2}$).

(c) At global round 8, $\mathcal{S}_C$ opens commitments of $A_{FD}^{P,2}$, $D_S(E(AV_{IF}^r))$ to Client. $\mathcal{S}_C$ sends $A_{FD}^{P,3}$ to Client.

7.3.3 Security of $\boldsymbol{\pi_{5PM}^M}$.

We prove that π_{5PM}^M securely realizes $\mathcal{F}_{5PM}^M$ in the malicious (static corruption) model.

Theorem 5. *Assuming the Decisional Diffie Hellman (DDH) problem is hard, π_{5PM}^M securely realizes $\mathcal{F}_{5PM}^M$ in the malicious (static corruption) model.*

Proof (Theorem 5). We proceed to prove Theorem 5 by considering two cases: the first where the Client is corrupted and the second where the Server is corrupted.

Case 1: Client is Corrupted.

We prove security by examining a sequence of experiments. Note that we assume here that the subprotocols of π_{5PM}^M are run *consecutively* instead of interleaved in order to simplify the proof. We will argue why interleaving does not affect our reasoning afterwards. We assume that Client has pattern p and Server has text T.

Intuition. The intuition for the proof is as follows: As long as $\mathcal{S}_C$ completes the last proof π_{fin} according to specification, $\mathcal{S}_C$ can use any text he wishes. What is required is a sequence of hybrid arguments that begins with the real Server's actual text and concludes with $\mathcal{S}_C$ using a dummy text T^* (namely 1^n). However, for a security reduction to the semantic security of ElGamal, the zero-knowledge arguments executed during the reduction must be executed without plain-text witnesses. Therefore, what first must be undertaken is that zero-knowledge proofs must be executed by zero-knowledge simulation in order to be executed irrespective of (plain text) witnesses, then the text T is shifted to T^*, then, per specification of $\mathcal{S}_C$, the zero-knowledge arguments are then executed using plain text witnesses again. Formal arguments follow.

$\mathbf{H}_0$**.** In this experiment Client and Server interact as in the real world.

$\mathbf{H}_{1,1}$**.** This experiment is identical to H_0 except that $\mathcal{S}_C$ executes π_{enc} with the Client and extracts s^* from the Client execution of π_{DL} for knowledge of s^*. Computational indistinguishability holds because the distribution for the outputs for π_{DL} are not affected from H_0 to $H_{1,1}$, as $\mathcal{S}_C$, after extraction, executes π_{DL} as verifier per specification of π_{DL}.

$\mathbf{H}_{1,2}$**.** This experiment is identical to H_0 except that $\mathcal{S}_C$ executes π_{enc} with the Client and extracts s_C from the Client execution of π_{DL} for knowledge of s_C. Computational indistinguishability holds because the distribution for the outputs for π_{DL} are not affected from $H_{1,1}$ to $H_{1,2}$, as $\mathcal{S}_C$, after extraction, executes π_{DL} as verifier per specification of π_{DL}.

$\mathbf{H}_2$**.** This experiment is identical to $H_{1,2}$ except that for the proof of knowledge π_{DL} of the Server-side secret key s_S, $\mathcal{S}_C$ simulates the transcript rather than actually using the witness s_S.

We prove that H_2 is computationally indistinguishable from $H_{1,2}$ by reducing to the computational indistinguishability of ZK transcripts. Assume that there existed a pattern p, Server text T, Client strategy and Server/$\mathcal{S}_C$ such that a distinguisher D could distinguish $H_{1,2}$ from H_2 in polynomial time with non-negligible probability. Then let P_{ZK} and V_{ZK} be prover and verifier for the zero knowledge argument π_{DL} for the g^{s_S}. Let V_{ZK} have as auxiliary inputs p and T. P_{ZK} also has p and T as auxiliary inputs, as well as s_S; note that these auxiliary inputs may be given to P_{ZK} after V_{ZK} has internally executed the corresponding component of π^M_{5PM}.

V_{ZK} executes π^M_{5PM} internally with input T for Server and input p for Client, and executes the protocol as H_1 and H_2 specify. However, for the proof π_{DL} for g^{s_S}, V_{ZK} interacts with P_{ZK} as follows. For one of the two executions for π_{DL}, P_{ZK} executes π_{DL} with witness s_S; for the other execution, P_{ZK} uses the simulator $\mathcal{S}_{ZK}$ (guaranteed to exist by definition of zero knowledge) to construct a valid transcript without knowledge of the plain text/witness. V_{ZK} responds to these two interactions per the output of the internally executed Client. Once the two interactions are done, V_{ZK} completes the internal execution of π^M_{5PM}. Note that the views of V_{ZK} are identical to the views of the interactions specified by H_1 and H_2. Therefore, the zero knowledge distinguisher D_{ZK} distinguishes the two cases of V_{ZK}'s interaction with non-negligible probability by running D internally, which will distinguish the two views of the ZK execution with non-negligible probability, which is a contradiction.

$\mathbf{H}^i_3$**.** This experiment is identical to H_2 except that for each of the first i executions of π_{isBit} for M_T in $\pi_{C,AV}$, $0 \leq i \leq n|\Sigma|$, $\mathcal{S}_C$ produces a valid transcript for the ZK argument without using the witness for the encryption. We note that the encryptions that $\mathcal{S}_C$ uses are unchanged from H_1; only the ZK transcripts for π_{isBit} in $\pi_{C,AV}$ are affected. Note that H_2 is identical to H^0_3.

We prove that H^i_3 is computationally indistinguishable from H^{i+1}_3 by reducing to the computational indistinguishability of ZK transcripts. Assume that there existed a pattern p, Server text T, Client strategy and Server/$\mathcal{S}_C$ such that a distinguisher D could distinguish H^i_3 from H^{i+1}_3 in polynomial time with non-negligible probability. Then let P_{ZK} and V_{ZK} be prover and verifier for the zero knowledge argument π_{DL} for the ith encryption of M_T. Let V_{ZK} have as auxiliary inputs p and T. P_{ZK} also has p and T as auxiliary inputs, as well as the public key and the randomness used to encrypt the $i+1$ encryption of M_T; note that these auxiliary inputs may be given to P_{ZK} after V_{ZK} has internally executed the corresponding component of π^M_{5PM}.

V_{ZK} executes π^M_{5PM} internally with input T for Server and input p for Client, and executes the protocol as H_3^i and H_3^{i+1} specify. However, for the $i+1$ encryption of M_T, V_{ZK} interacts with P_{ZK} as follows. For one of the two executions of π_{isBit} for the $i+1$ encryption of M_T, P_{ZK} executes π_{isBit} with private input the witness for the encryption to run π_{DL}; for the other execution, P_{ZK} uses the simulator $\mathcal{S}_{ZK}$ (guaranteed to exist by definition of zero knowledge) to construct a valid transcript without knowledge of the plain text/witness. V_{ZK} responds to these two interactions per the output of the internally executed Client. Once the two interactions are done, V_{ZK} completes the internal execution of π^M_{5PM}. Note that the views of V_{ZK} are identical to the views of the interactions specified by H_3^i and H_3^{i+1}. Therefore, the zero-knowledge distinguisher D_{ZK} distinguishes the two cases of V_{ZK}'s interaction with non-negligible probability by running D internally, which will distinguish the two views of the ZK execution with non-negligible probability, which is a contradiction.

$\mathbf{H}_4^i$**.** This experiment is identical to $H_2^{n|\Sigma|}$ except that for each of the first i executions of π_{DL} for $\pi_{C,AV}$ (namely, that the sum of each row of M_T is a 1), $0 \leq i \leq n$, $\mathcal{S}_C$ produces a valid transcript for the ZK argument without using the witness for the encryption, which is a product of the encryptions in the ith column of $E(M_T)$. Note that H_4^0 is identical to $H_3^{n|\Sigma|}$

We prove that H_4^i is computationally indistinguishable from H_4^{i+1} by reducing to the computational indistinguishability of ZK transcripts. Assume that there existed a pattern p, Server text T, Client strategy and Server such that a distinguisher D could distinguish H_4^i from H_4^{i+1} in polynomial time with non-negligible probability. Then let P_{ZK} and V_{ZK} be prover and verifier for the zero knowledge argument π_{DL} for the product of the ith row of $E(M_T)$. We let V_{ZK} have as auxiliary inputs p and T. P_{ZK} also has p and T as auxiliary inputs, as well as the public key and the randomness used to encrypt the $i+1$ row of encryptions of M_T; note that these auxiliary inputs may be given to P_{ZK} after V_{ZK} has internally executed the corresponding component of π^M_{5PM}.

V_{ZK} executes π^M_{5PM} internally with input T for Server and input p for Client, and executes the protocol as H_4^i and H_4^{i+1} specify. However, for the product of the $i+1$ row of $E(M_T)$, V_{ZK} interacts with P_{ZK} as follows. For one of the two executions, P_{ZK} uses the witness for the encryption to run π_{DL}, and for the other, P_{ZK} uses the simulator $\mathcal{S}_{ZK}$ (guaranteed to exist by definition of zero knowledge) to construct a valid transcript without knowledge of the plain text. V_{ZK} responds to these two interactions per the output of the internally executed Client. Once the two interactions are done, then V_{ZK} completes the internal execution of π^M_{5PM}. Note that the views of V_{ZK} are identical to the view of the interactions specified by H_4^i and H_4^{i+1}. Therefore, the zero knowledge distinguisher D_{ZK} distinguishes the two cases of V_{ZK}'s interaction with non-negligible probability by running D internally, which will distinguish the two views with non-negligible probability, which is a contradiction.

$\mathbf{H_5^i}$. This experiment is identical to H_3^n except that for each of the first i executions of π_{eqDL} executed as part of A_{PD} for correct server-side partial decryption of z_i during π_{vec} is executed without using the witness for partial decryption and instead is simulated ($0 \leq i \leq 2$, where the 0 case corresponds to H_4^n).

We prove that H_5^i is computationally indistinguishable from H_5^{i+1} by reducing to the computational indistinguishability of ZK transcripts. Assume that there existed a pattern p, Server text T, Client strategy and Server such that a distinguisher D could distinguish H_5^i from H_5^{i+1} in polynomial time with non-negligible probability. Then let P_{ZK} and V_{ZK} be prover and verifier for the zero knowledge argument π_{eqDL}. We let V_{ZK} have as auxiliary inputs p and T. P_{ZK} also has p and T as auxiliary inputs, Server-side secret key s_S, the decryption of z_i and the randomness used for the encryption; note that these auxiliary inputs may be given to P_{ZK} after V_{ZK} has internally executed the corresponding component of π_{5PM}^M.

V_{ZK} executes π_{5PM}^M internally with input T for Server and input p for Client, and executes the protocol as H_5^i and H_5^{i+1} specify. However, for the proof of partial decryption by the Server for z_{i+1} in π_{vec}, V_{ZK} interacts with P_{ZK} as follows. For one of the two executions, P_{ZK} uses the witness s_S for the decryption to run π_{eqDL}, and for the other, P_{ZK} uses the simulator $\mathcal{S}_{ZK}$ (guaranteed to exist by definition of zero knowledge) to construct a valid transcript without knowledge of the secret key. V_{ZK} responds to these two interactions per the output of the internally executed Client. Once the two interactions are done, then V_{ZK} completes the internal execution of π_{5PM}^M. Note that the views of V_{ZK} are indistinguishable from the view of the interactions specified by H_5^i and H_5^{i+1}. Therefore, the zero knowledge distinguisher D_{ZK} distinguishes the two cases of V_{ZK}'s interaction with non-negligible probability by running D internally, which will distinguish the two views with non-negligible probability, which is a contradiction.

$\mathbf{H_6^i}$. This experiment is identical to H_5^n except that for each of the first i executions of π_{fin} during A_{FD}, $0 \leq i \leq n-m+1$, for the proof of partial decryption of $E(AV_S^r)$, Server (or $\mathcal{S}_C$) uses the witness for s^* to complete the execution. Note that use of different witnesses is computationally indistinguishable for ZK arguments (since the executions with a different witness are themselves indistinguishable from the execution by the simulator). Note that H_6^0 is identical to H_5^n.

We prove that H_6^i is computationally indistinguishable from H_6^{i+1} by reducing it to the computational indistinguishability of ZK transcripts. Assume that there existed a pattern p, Server text T, Client strategy and Server such that a distinguisher D could distinguish H_6^i from H_6^{i+1} in polynomial time with non-negligible probability. Then let P_{ZK} and V_{ZK} be prover and verifier for the zero knowledge argument π_{fin} for the ith encryption of $E(AV_S^r)$. We let V_{ZK} have as auxiliary inputs p and T. P_{ZK} also has p and T as auxiliary inputs, as well as the $i+1$ element of $E(AV_S^r)$, the corresponding Server-side partial decryption and the Server-side private key. Finally, P_{ZK} also has s^* as auxiliary input; note that these auxiliary inputs may be given to P_{ZK} after V_{ZK} has internally executed the corresponding component of π_{5PM}^M.

V_{ZK} executes π_{5PM}^M internally with input T for Server and input p for Client, and executes the protocol as H_6^i and H_6^{i+1} specify. However, for $i+1$ encryption of $E(AV_S^r)$, V_{ZK} interacts with P_{ZK} as follows. For one of the two executions, P_{ZK} uses the witness for the partial decryption of the $i+1$ element of $E(AV_S^r)$ to execute π_{fin} with V_{ZK}, and for the other, P_{ZK} uses the witness s^* to execute π_{fin}. V_{ZK} responds to these two interactions per the output of the internally executed Client. Once the two interactions are done, then V_{ZK} completes the internal execution of π_{5PM}^M. Note that the views of V_{ZK} are identical to the view of the interactions specified by H_6^i and H_6^{i+1}. Therefore, the zero knowledge distinguisher D_{ZK} distinguishes the two cases of V_{ZK}'s interaction with non-negligible probability by running D internally, which will distinguish the two views with non-negligible probability, which a contradiction.

$\mathbf{H_7^i}$. This experiment is identical to H_6^{n-m+1} except that the first i entries of AV_{IF}^r, $0 \leq i \leq n-m+1$, are used instead of the corresponding entries of AV_S^r. Note that AV_{IF}^r and AV_S^r, are, by construction, distributed identically over the choice of their respective randomized entries (e.g., in the non-matching locations), and that both contain zeros in exactly the same places. Note that by the construction of H_6^n, the proof of partial decryption π_{fin} for $D_S(E(AV_{IF}^r))$ uses witness s^* rather than the witness for decryption. Therefore H_7^i is computationally (indeed, statistically) indistinguishable from H_7^{i+1} for any i because of the identical distributions of AV_{IF}^r and AV_S^r.

$\mathbf{H_8^i}$. This experiment is identical to H_7^{n-m+1} except that the first i encryptions of $E(M_T)$, $0 \leq i \leq n|\Sigma|$, are drawn from $E(M_{T'})$, which corresponds to $T' = 1^n$. Note that the final output of π_{5PM}^M is AV_{IF}^r, so the protocol output does not change. Note also that H_8^0 is identical to H_5^n.

We prove that H_8^i is computationally indistinguishable from H_8^{i+1} by reducing indistinguishability to the semantic security of ElGamal encryption (that is to say, assuming the hardness of Decisional Diffie Hellman); in particular, we will use a reduction to non-threshold (e.g., standard) ElGamal encryption. Assume that there existed a pattern p, Server text T, Client strategy and Server/$\mathcal{S}_C$ such that a distinguisher D could distinguish H_8^i from H_8^{i+1} in polynomial time with non-negligible probability. Then let S_{enc} be the computer of ElGamal encryptions and R_{enc} be the receiver. R_{enc} is given as auxiliary inputs p and T. S_{enc} also has p and T as auxiliary inputs, as well as the public key and the randomness used to encrypt the $i+1$ encryption of $M_T/M_{T'}$; note that these auxiliary inputs may be given to P_{ZK} after V_{ZK} has internally executed the corresponding component of π_{5PM}^M.

R_{enc} executes π_{5PM}^M internally with input T/T' for Server and input p for Client. However, when Server must publish g^{s_S}, R_{enc} queries S_{enc}, who sends g^{s_S}; this is the publishing of the public ElGamal key. Note that R_{enc} must select s_C independently of s_S or it will not be able to create a transcript computationally indistinguishable from the π_{5PM}^M transcript; because of the binding and hiding property of Pedersen commitments, real Client and Server are required to do the same. R_{enc}, as Server, sends S_{enc} and completes the ZK proof without the witness s_S. Since this is already being accomplished per the above hybrid specification, H_2, R_{enc} is acting according to specification.

For each encryption of M_T, except for the i and $i+1$ entries, R_{enc} sends the unencrypted value to S_{enc}, who sends back the corresponding encryptions. R_{enc} encrypts with the Client's s_C (which it obtains by running the knowledge extractor; as in hybrid H_1, this does not affect transcript indistinguishability), and uses this final encryption as the Server-side encryption; this final encryption corresponds to encryption with the secret key $s_C + s_S$.

Note that in all cases, the ZK proofs of the well-formedness of the encryptions are simulated rather than using the actual witnesses per hybrid experiments $H_2^{n|\Sigma|}$ and H_3^n.

For the i and $i+1$ encryptions, R_{enc} sends both to S_{enc}, who sends back encryptions of each without identifying which encryption corresponds to which plaintext. R_{enc} completes the internal execution of π_{5PM}^M. Note that witnesses for partial decryption by the Server do not use s_S per construction of the previous hybrids (H_3 and H_4). We see that at the end of π_{vec}, Client and Server must jointly compute decryptions of z_1 and z_2. Here, we note that R_{enc} will simply use the same (randomly chosen) "secret key" s_{fake} and use it for Server-side decryption of z_1 and z_2; if the actual decryptions of z_1 and z_2 equal each other, than so will these new "decryptions". Further, by the uniformly random distribution of the hash functions used to compute z_1 and z_2, the transcripts will be identical to the hybrid experiment executions. Also, Server does not use the witness for partial decryption due to hybrid H_5^2.

Finally, for the final decryption of AV^r_{IF}, R_{enc} can simply send the Server's partial decryption, $D_S(E(AV^r_{IF}))$, as the partial *encryption* of AV^r_{IF} using the Client's public key (note that the Server-side can extract this rather than having R_{enc} simply use it by virtue of executing Client internally). Again, the witness for partial decryption isn't used because by hybrid H_6, the witness s^* is already being used.

We note that by the above reasoning, the views of R_{enc} are distributed identically to the views of an adversarial Client for H^i_8 and H^{i+1}_8. Therefore, if the distinguisher D_{enc} executes the hybrid distinguisher D internally, then D_{enc} will be able to distinguish the i and $i+1$ encryptions of M_T with non-negligible probability, which a contradiction.

$\mathbf{H^i_9}$. This experiment is identical to $H^{n|\Sigma|}_8$ except that for each of the first i executions of π_{eqDL} executed as part of A_{PD} for correct server-side partial decryption of z_i during π_{vec} ($0 \leq i \leq 2$, where the 0 case corresponds to H^n_4), Server uses the secret key s_S rather than simulating the transcript.

That the hybrids H^i_9 and H^{i+1}_9 are computationally indistinguishable is essentially the same argument as for hybrids H^i_5 and H^{i+1}_5 (except moving in the "reverse" direction).

$\mathbf{H^i_{10}}$. This experiment is identical to H^2_9 except that for the first i executions of π_{DL} in $\pi_{C,AV}$, $0 \leq i \leq n$, $\mathcal{S}_C$ uses the witness for the encryption, which is a product of the encryptions in the ith row of $E(M_{T'})$. Note that H^0_{10} is identical to H^2_9.

That hybrids H^i_{10} and H^{i+1}_{10} are computationally indistinguishable is essentially the same argument as for hybrids H^i_3 and H^{i+1}_3 (except moving in the "reverse" direction).

$\mathbf{H^i_{11}}$. This experiment is identical to H^n_{10} except that for each of the first i executions of π_{isBit} in $\pi_{C,AV}$, $0 \leq i \leq n|\Sigma|$, $\mathcal{S}_C$ uses the witness for the encryption. Note that H^0_{11} is identical to H^n_{10}.

That hybrids H^i_{11} and H^{i+1}_{11} are computationally indistinguishable is essentially the same argument as for hybrids H^i_3 and H^{i+1}_3 (which was the same process but in reverse).

$\mathbf{H_{12}}$. This experiment is identical to $H^{n|\Sigma|}_{11}$ except that $\mathcal{S}_C$ uses the witness s_S for π_{DL} in π_{vec}. Note that H_{12} is the distribution of view of output of the simulator $\mathcal{S}_C$.

That hybrids H_{12} and $H^{n|\Sigma|}_{11}$ are computationally indistinguishable is the essentially the same argument as for hybrids $H_{1,2}$ and H_2 above.

Interleaving the ZK Arguments. The ZK arguments used in π^M_{5PM} are all modified Σ protocols. The first three global rounds of π^M_{5PM} force the respective ZK provers to instantiate all needed (equivocable) commitment schemes while forcing the respective ZK verifiers to commit to all challenges for the subsequent Σ protocols. Thus, the verifiers have *less* power than in the sequential composition case (recall that ZK arguments are closed under sequential composition), since in the sequential composition case, the verifiers could change their challenges based on previous ZK iterations while here they cannot. Therefore, zero knowledge is not affected. Soundness is also not affected by interleaving since a dishonest prover must break soundness by distinguishing the verifier's (public coin) committed challenge to the underlying Σ protocol. However, the prover cannot distinguish multiple committed challenges as long as separate randomness was used for each commitment, yielding soundness of the ZK arguments even when they are interleaved as in π_{5PM}^{M}.

Interleaving Subprotocols. We now demonstrate that interleaving the subprotocols into the final π_{5PM}^{M} does not affect transcript indistinguishability. We do so by considering the interleaving subprotocols and demonstrating that each new subprotocol's transcript is not affected by interleaving. We can consider this sequential case because it is already demonstrated (above) that if each subprotocol is executed sequentially without interleaving, then the protocol is secure in the malicious model.

We denote by A_{rel}^{S} and A_{rel}^{C} (also by, for instance, π_{rel}^{S}) the zero-knowledge building blocks outlined in Section 6.1 where Server and Client, separately, act as prover. As shorthand, we will write $A_{rel}^{S}(x)$ as the argument A_{rel} where S is the prover and x is the common input.

π_{encr} starts at global round 1. The preprocessing steps for all required ZK protocols occur during π_{encr}; their outputs here are indistinguishable in the real versus ideal settings by the fact that interleaved zero-knowledge protocols still yield indistinguishable transcripts. While $\mathcal{S}_C$ must rewind to extract s^* and s_C, this does not change the distribution of $\mathcal{S}_C$'s output to Client's view.

$\pi_{C,AV}$ starts at global round 2, after the global public key has been determined. Indistinguishability is not affected since π_{encr} is a protocol for setting up the threshold encryption scheme while $\pi_{C,AV}$ is a protocol for proper pattern formation (e.g., of $E(M_{CDV})$). Since these protocols deal with independent inputs (other than $\pi_{C,AV}$, which needs the existence of the threshold encryption scheme), interleaving them does not affect transcript indistinguishability.

$\pi_{S,AV}$ starts at global round 3. The only remaining outputs from π_{encr} and $\pi_{C,AV}$ are the remaining zero-knowledge argument outputs of $\pi_{DL}^{C}(h_1)$, $\pi_{DL}^{C}(h^*)$, $\pi_{DL}^{S}(h_2)$, and $A_{M1}^{S}(E(M_T))$, which are independent of the non ZK-related outputs of $\pi_{S,AV}$ and therefore do not affect indistinguishability. Interleaving the ZK-related outputs of $A_{M01}^{C}(E(M_{CDV}))$ with remaining ZK outputs of previous subprotocols does not affect their indistinguishability (since they could be considered as auxiliary inputs in the ZK security proof).

π_{vec} starts at global round 4. The only remaining outputs from the previous subprotocols are from the ZK arguments $\pi_{DL}^{C}(h_1)$, $\pi_{DL}^{C}(h^*)$, $\pi_{DL}^{S}(h_2)$, $A_{M1}^{S}(E(M_T))$, and $A_{M01}^{C}(E(M_{CDV}))$, whose interleaving do not affect indistinguishability since the non-ZK outputs of π_{vec} are chosen independently of previous ZK outputs, and interleaving ZK arguments does not affect indistinguishability.

π_{rand} starts at global round 6, at the same time that $E(AV_S')$ is introduced. Since π_{rand} only involves reformatting $E(AV_S')$, namely by introducing $E(AV_S^r)$ and its associated ZK outputs, indistinguishability is not affected.

π_{ans} starts at global round 6; the ZK transcript for A_{final} does not reveal any information by the indistinguishability of ZK transcripts and the final decryption is only revealed (by decommitment) at the very last round. Distinguishing the transcript here would imply violating the commitment scheme (e.g., Pedersen commitments, which are perfectly hiding).

This completes the proof for the indistinguishability of an adversarial Client's views.

Case 2: Server is Corrupted.

We prove security by examining a sequence of experiments. Note that we assume here that the subprotocols of π_{5PM}^{M} are run *consecutively* instead of interleaved in order to simplify the proof. We will argue why interleaving does not meaningfully affect the proof afterwards. We assume that

Client has pattern p and Server has text T.

Intuition. The intuition for the proof is as follows: In contrast to the above case of a corrupted Client, here, Client can use any encryption he wants as long as the zero-knowledge proofs hold, and the views will still be computationally indistinguishable. However, for a security reduction to the semantic security of ElGamal encryption, the zero-knowledge arguments executed during the reduction must be executed without plain-text witnesses. Therefore, what first must be undertaken is that zero-knowledge proofs must be executed by zero-knowledge simulation in order to be executed irrespective of (plain text) witnesses; then the text p is shifted to p^*. Next, per specification of $\mathcal{S}_S$, the zero-knowledge arguments are executed using plain text witnesses again. Formal arguments follow.

$\mathbf{H}_0$. In this experiment Client and Server interact as in the real world.

$\mathbf{H}_1$. This experiment is identical to H_0 except that for the proof of knowledge π_{DL} of the Client-side secret key s_C, $\mathcal{S}_S$ simulates the transcript rather than actually using the witness s_C.

We prove that H_1 is computationally indistinguishable from H_1 by reducing indistinguishability to the computational indistinguishability of ZK transcripts. Assume that there existed a pattern p, Server text T, Server strategy and Client/$\mathcal{S}_S$ such that a distinguisher D could distinguish H_0 from H_1 in polynomial time with non-negligible probability. Then let P_{ZK} and V_{ZK} be prover and verifier for the zero-knowledge argument π_{DL} for the g^{s_C}. Let V_{ZK} have as auxiliary inputs p and T. P_{ZK} also has p and T as auxiliary inputs, as well as s_S; note that these auxiliary inputs may be given to P_{ZK} after V_{ZK} has internally executed the corresponding component of π_{5PM}^{M}.

V_{ZK} executes π_{5PM}^{M} internally with input T for Server and input p for Client, and executes the protocol as H_0 and H_1 specify. However, during the proof π_{DL} for g^{s_C}, V_{ZK} interacts with P_{ZK} as follows. For one of the two executions of π_{DL}, P_{ZK} executes π_{DL} with witness s_C; for the other execution, P_{ZK} uses the simulator $\mathcal{S}_{ZK}$ (guaranteed to exist by definition of zero knowledge) to construct a valid transcript without knowledge of the plain text/witness. V_{ZK} responds to these two interactions per the output of the internally executed Server. Once the two interactions are done, V_{ZK} completes the internal execution of π_{5PM}^{M}. Note that the views of V_{ZK} are identical to the view of the interactions specified by H_0 and H_1. Therefore, the zero-knowledge distinguisher D_{ZK} distinguishes the two cases of V_{ZK}'s interaction with non-negligible probability by running D internally, which will distinguish the two views of the ZK execution with non-negligible probability, which is a contradiction.

$\mathbf{H}_2^i$. This experiment is identical to H_1 except that for each of the first i executions of π_{isBit} for M_{CDV} in $\pi_{S,AV}$, $0 \leq i \leq m|\Sigma|$, $\mathcal{S}_S$ produces a valid transcript for the ZK argument without using the witness for the encryption. We note that the encryptions that $\mathcal{S}_S$ uses are unchanged from H_0; only the ZK transcripts for π_{isBit} in $\pi_{S,AV}$ are affected. Note that H_1 is identical to H_2^0.

We prove that H_2^i is computationally indistinguishable from H_2^{i+1} by reducing indistinguishability to the computational indistinguishability of ZK transcripts. Assume that there existed a pattern p, Server text T, Server strategy and Client/$\mathcal{S}_S$ such that a distinguisher D could distinguish H_2^i from H_2^{i+1} in polynomial time with non-negligible probability. Then let P_{ZK} and V_{ZK} be prover and verifier for the zero-knowledge argument π_{DL} for the ith encryption of M_{CDV}. Let V_{ZK} have as auxiliary inputs p and T. P_{ZK} also has p and T as auxiliary inputs, as well as the public key and the randomness used to encrypt the $i+1$ encryption of M_{CDV}; note that these auxiliary inputs may be given to P_{ZK} after V_{ZK} has internally executed the corresponding component of π_{5PM}^{M}.

V_{ZK} executes π_{5PM}^{M} internally with input T for Server and input p for Client, and executes the protocol as H_2^i and H_2^{i+1} specify. However, for the $i+1$ encryption of M_{CDV} in $\pi_{S,AV}$, V_{ZK} interacts with P_{ZK} as follows. For one of the two executions of π_{isBit} for the $i+1$ encryption of M_{CDV}, P_{ZK} executes π_{isBit} with private input the witness for the encryption to run π_{DL}; for the other execution, P_{ZK} uses the simulator $\mathcal{S}_{ZK}$ (guaranteed to exist by definition of zero knowledge) to construct a valid transcript without knowledge of the plain text/witness. V_{ZK} responds to these two interactions per the output of the internally executed Server. Once the two interactions are done, V_{ZK} completes the internal execution of π_{5PM}^{M}. Note that the views of V_{ZK} are identical to the view of the interactions specified by H_2^i and H_2^{i+1}. Therefore, the zero-knowledge distinguisher D_{ZK} distinguishes the two cases of V_{ZK}'s interaction with non-negligible probability by running D internally, which will distinguish the two views of the ZK execution with non-negligible probability, which is a contradiction.

$\mathbf{H}_3^i$**.** This experiment is identical to $H_2^{m|\Sigma|}$ except that for each of the first i executions of π_{isBit} for $\pi_{S,AV}$ (namely, that the sum of each column of M_{CDV} is 0 or 1), $0 \le i \le m$, $\mathcal{S}_S$ produces a valid transcript for the ZK argument without using the witness for the encryption, which is a product of the encryptions in the ith column of $E(M_{CDV})$. Note that H_3^0 is identical to $H_2^{m|\Sigma|}$.

We prove that H_3^i is computationally indistinguishable from H_3^{i+1} by reducing indistinguishability to the computational indistinguishability of ZK transcripts. Assume that there existed a pattern p, Server text T, Client/$\mathcal{S}_S$ and Server strategy such that a distinguisher D could distinguish H_3^i from H_3^{i+1} in polynomial time with non-negligible probability. Then let P_{ZK} and V_{ZK} be prover and verifier for the zero-knowledge argument π_{isBit} for the product of the $i+1$th column of $E(M_{CDV})$. We let V_{ZK} have as auxiliary inputs p and T. P_{ZK} also has p and T as auxiliary inputs, as well as the public key and the randomness used to encrypt the $i+1$ column of encryptions of M_{CDV}; note that these auxiliary inputs may be given to P_{ZK} after V_{ZK} has internally executed the corresponding component of π_{5PM}^{M}.

V_{ZK} executes π_{5PM}^{M} internally with input T for Server and input p for Client, and executes the protocol as H_3^i and H_3^{i+1} specify. However, for the product of the $i+1$ column of $E(M_{CDV})$, V_{ZK} interacts with P_{ZK} as follows. For one of the two executions, P_{ZK} uses the witness for the encryption to run π_{DL}, and for the other, P_{ZK} uses the simulator $\mathcal{S}_{ZK}$ (guaranteed to exist by definition of zero knowledge) to construct a valid transcript without knowledge of the plain text. V_{ZK} responds to these two interactions per the output of the internally executed Server. Once the two interactions are done, then V_{ZK} completes the internal execution of π_{5PM}^{M}. Note that the views of V_{ZK} are identical to the view of the interactions specified by H_3^i and H_3^{i+1}. Therefore, the zero knowledge distinguisher D_{ZK} distinguishes the two cases of V_{ZK}'s interaction with non-negligible probability by running D internally, which will distinguish the two views with non-negligible probability, which is a contradiction.

$\mathbf{H_4^i}$. This experiment is identical to H_3^m except that for each of the first i executions of π_{eqDL}, conducted as part of A_{PD} for correct client-side partial decryption of z_i during π_{vec}, is executed without using the witness for partial decryption, and instead is simulated (for $0 \le i \le 2$, where the 0 case corresponds to H_3^n).

We prove that H_4^i is computationally indistinguishable from H_4^{i+1} by reducing indistinguishability to the computational indistinguishability of ZK transcripts. Assume that there existed a pattern p, Server text T, Client strategy and Server such that a distinguisher D could distinguish H_4^i from H_4^{i+1} in polynomial time with non-negligible probability. Then let P_{ZK} and V_{ZK} be prover and verifier for the zero-knowledge argument π_{eqDL}. We let V_{ZK} have as auxiliary inputs p and T. P_{ZK} also has p and T as auxiliary inputs, Client-side secret key s_C, the decryption of z_i and the randomness used for the encryption. Note that these auxiliary inputs may be given to P_{ZK} after V_{ZK} has internally executed the corresponding component of π_{5PM}^M.

V_{ZK} executes π_{5PM}^M internally with input T for Server and input p for Client, and executes the protocol as H_4^i and H_4^{i+1} specify. However, for the proof of partial decryption by the Client for z_{i+1} in π_{vec}, V_{ZK} interacts with P_{ZK} as follows. For one of the two executions, P_{ZK} uses the witness s_C for the decryption to run π_{eqDL}, and for the other, P_{ZK} uses the simulator $\mathcal{S}_{ZK}$ (guaranteed to exist by definition of zero knowledge) to construct a valid transcript without knowledge of the secret key. V_{ZK} responds to these two interactions per the output of the internally executed Server. Once the two interactions are done, then V_{ZK} completes the internal execution of π_{5PM}^M. Note that the views of V_{ZK} are identical to the view of the interactions specified by H_4^i and H_4^{i+1}. Therefore, the zero-knowledge distinguisher D_{ZK} distinguishes the two cases of V_{ZK}'s interaction with non-negligible probability by running D internally, which will distinguish the two views with non-negligible probability, which is a contradiction.

$\mathbf{H_5^i}$. This experiment is identical to H_4^n except that the first i encryptions of $E(M_{CDV})$, $0 \leq i \leq m|\Sigma|$, correspond to $E(M_{CDV'})$ executed using $p' = 1^n$. Note that H_5^0 is identical to H_4^m.

We prove that H_5^i is computationally indistinguishable from H_5^{i+1} by reducing indistinguishability to the semantic security of ElGamal encryption (that is to say, assuming the hardness of Decisional Diffie Hellman). In particular, we will use a reduction to non-threshold (e.g., standard) ElGamal encryption. Assume that there existed a pattern p, Server text T, Server strategy and Client/$\mathcal{S}_S$ such that a distinguisher D could distinguish H_5^i from H_5^{i+1} in polynomial time with non-negligible probability. Then let S_{enc} be the computer of ElGamal encryptions and R_{enc} be the receiver. R_{enc} is given p and T as auxiliary inputs. S_{enc} also has p and T as auxiliary inputs, as well as the public key and the randomness used to encrypt the $i+1$ encryption of $M_{CDV}/M_{CDV'}$. Note that these auxiliary inputs may be given to P_{ZK} after V_{ZK} has internally executed the corresponding component of π_{5PM}^M.

R_{enc} executes π_{5PM}^M internally with input T for Server and input p/p' for Client. However, when Client must publish g^{s_C}, R_{enc} queries S_{enc}, who sends g^{s_C}; this is the publishing of the public ElGamal key. Note that R_{enc} must select s_S independently of s_C or she will not be able to create a transcript computationally indistinguishable from the π_{5PM}^M transcript; because of the binding and hiding property of Pedersen commitments, real Client and Server are required to do the same. R_{enc}, as Server, sends S_{enc} and completes the ZK proof without the witness s_C as per hybrid H_1. Since by an earlier hybrid experiment this is already being accomplished, R_{enc} is acting according to specification.

For each encryption of M_{CDV}, except for the i and $i+1$ entries, R_{enc} sends the unencrypted value to S_{enc}, who sends back the corresponding encryptions. R_{enc} encrypts with the Server's s_S (which it obtains by running the knowledge extractor; as in hybrid H_1, this does not affect transcript indistinguishability), and uses this final encryption as the Client-side encryption. This final encryption corresponds to encryption with the secret key $s_C + s_S$.

Note that in all cases, the ZK proofs of well-formedness of the encryptions are simulated rather than using the actual witnesses per hybrid experiments $H_1^{m|\Sigma|}$ and H_2^m.

For the i and $i+1$ encryptions, R_{enc} sends both to S_{enc}, who sends back encryptions of each without identifying which encryption corresponds to which plaintext. R_{enc} completes the internal execution of π_{5PM}^M. Note that witnesses for partial decryption by the Client do not use s_C per construction of the previous hybrids (H_1 and H_2). We see that at the end of π_{vec}, Client and Server must jointly compute decryptions of z_1 and z_2; here, we note that R_{enc} will simply use the same (randomly chosen) "secret key" s_{fake} and use it for Client-side decryption of z_1 and z_2. If the actual decryptions of z_1 and z_2 equal each other, than so will these new "decryptions"; by the uniformly random distribution of the hash functions used to compute z_1 and z_2, the transcripts will be identical to the hybrid experiment executions. Further, Server does not use the witness for partial decryption due to hybrid H_4^2.

Note that no protocol output is part of the view/distribution of the hybrid experiment here, so that the pattern-matching output changes are not a concern (the ideal functionality does not output to an adversarial Server). In particular, because the adversarial Server never learns the pattern-matching output, she can never output it out of turn at some point and skew the distribution of the hybrid experiments.

We note that by the above reasoning, the views of R_{enc} are computationally indistinguishable from the views of an adversarial Client for H_5^i and H_5^{i+1}. Therefore, if the distinguisher D_{enc} by executes the hybrid distinguisher D internally, then D_{enc} will be able to distinguish the i and $i+1$ encryptions of M_{CDV} with non-negligible probability, which is a contradiction.

$\mathbf{H_6^i}$. This experiment is identical to $H_5^{m|\Sigma|}$ except that for each of the first i executions of π_{eqDL} executed as part of A_{PD} for correct server-side partial decryption of z_i during π_{vec} ($0 \leq i \leq 2$, where the 0 case corresponds to H_3^n), Client uses the secret key s_C rather than simulating the transcript.

That the hybrid H_6^i and H_6^{i+1} are computationally indistinguishable is essentially the same argument as for hybrids H_4^i and H_4^{i+1} (except moving in the "reverse" direction).

$\mathbf{H_7^i}$. This experiment is identical to H_6^2 except that for each of the first i executions of π_{isBit} in $\pi_{S,AV}$, $1 \leq i \leq m$, $\mathcal{S}_C$ uses the witness for the encryption, which is a product of the encryptions in the ith row of $E(M_{CDV'})$. Note that H_7^0 is identical to H_6^2.

That hybrid H_7^i and H_7^{i+1} are computationally indistinguishable is essentially the same argument as for hybrids H_3^i and H_3^{i+1} (except moving in the "reverse" direction).

$\mathbf{H_8^i}$ This experiment is identical to H_7^m except that for each of the first i executions of π_{isBit} in $\pi_{S,AV}$, $0 \leq i \leq m|\Sigma|$, $\mathcal{S}_S$ uses the witness for the encryption. Note that H_8^0 is identical to H_7^m; further, $H_8^{m|\Sigma|}$ is the distribution of view of output of $\mathcal{S}_S$.

That hybrid H_8^i and H_8^{i+1} are computationally indistinguishable is essentially the same argument as for hybrids H_1^i and H_1^{i+1} (except moving in the "reverse" direction).

$\mathbf{H_9}$. This experiment is identical to $H_8^{m|\Sigma|}$ except that $\mathcal{S}_S$ uses the witness s_C for π_{DL} in π_{vec}. Note that H_9 is the distribution of view of output of $\mathcal{S}_S$.

That hybrids H_9 and $H_8^{n|\Sigma|}$ are computationally indistinguishable is the essentially the same argument as for hybrids H_1 and H_2 above (except moving in the "reverse" direction).

Interleaving the ZK Arguments. The argument for why interleaving ZK arguments does not break their security properties is explained in the proof-case of an adversarial Client (see Case 1 in this proof).

Interleaving Subprotocols. The reason why interleaving subprotocols does not affect indistinguishability is explained in the proof-case of an adversarial Client (see Case 1 in this proof).

This completes the proof for the indistinguishability of an adversarial Server's views.

Caveat for $\mathcal{S}_S$. In the case of an adversarial Server, a soundness-type argument is needed: if Server obtains s^* independently, it would be impossible to prove that Server had actually supplied the correct output for the functionality $\mathcal{F}_{5PM}^{M}$ during π_{ans} because of the witness hiding property of π_{fin}. To show that such an independent generation of s^* is impossible except with negligible probability, we assume by way of contradiction that Server had indeed generated s^* independently, and we let the simulator $\mathcal{S}_S$ adopt the following strategy:

(a) At global round 1, $\mathcal{S}_S$ sends Server h^* without knowing its discrete logarithm (which is s^*).

(b) Since Server must commit to its challenge, which occurs in global round 2 and which is decommitted in global round 4, $\mathcal{S}_S$ can rewind from global round 4 to global round 3 and change its output for π_{DL}, thus proving knowledge of the discrete logarithm of h^* so that the $\mathcal{S}_S$ provides a valid proof for h^* without actually knowing s^*. This is due to the fact that π_{DL} is constructed from a Σ protocol (Σ_{DL}), which is honest-verifier zero-knowledge as defined in Section 4.

(c) At global round 6, Server begins a proof of knowledge, A_{final}, that either its decryption is correct or that it knows s^*. In this case, Server uses its knowledge of s^*. $\mathcal{S}_S$ can extract this s^* using a subprotocol E_{fin} for π_{fin} guaranteed by definition of argument of knowledge:
   - In order to extract, $\mathcal{S}_S$ must be able to decommit to any message. In order to do so, $\mathcal{S}_S$, using the knowledge extractor E as a subprotocol, must rewind from global round 8 (where Server answers $\mathcal{S}_S$'s challenge to π_{final}) to the related outputs at global rounds 2 and 3, where the ZK preprocessing of A_{final} occurs, then interact with Server anew until round 8, and then rewind again to round 2, per E's specifications. Note that rewinding again and again back to round 2 does not affect either the extraction or Server's strategy, since Server's strategy is fixed at the outset of the overall protocol and s^*/h^* is used nowhere else in the protocol.

The success of this strategy implies that there exists an expected polynomial time algorithm, namely $\mathcal{S}_S$ running π_{5PM}^{M} against a Server, that is able to extract a discrete logarithm s^* in polynomial time. Therefore, the discrete logarithm problem is solvable in polynomial time, which is a contradiction. Accordingly, it must not be the case that Server can generate s^* independently, which completes the proof.

8 Detailed Performance Results of 5PM Implementation

Table 13 shows detailed performance results of the implementation of π_{5PM}^{H}. Our experiments were performed on an Intel dual quad-core 2.93GHz machine with 8GB of memory running Ubuntu Linux version 10.10. We used fast-decryption Paillier [40] from the Self-Certifying File System (SFS) library [41], and used alphabets of sizes 4 (DNA) and 36 (alphanumeric). Our implementation results in Table 13 show that on average, 95% of the total online runtime was spent in three components of the protocol, two at Server and one at Client. The first is searching the text at Server by adding CDVs, which correspond to pattern characters, to the activation vector; the second is blinding elements of the activation vector at the Server; the third is decrypting the blinded activation vector at Client. Key generation and CDV initialization times were not included in our results because they can be precomputed: CDVs are constructed from a series of encrypted 1s and 0s and therefore can be precomputed by encrypting a large number of 1s and 0s (independent of the pattern to be matched) and then constructing the CDVs according to the pattern without requiring any additional encryptions.

k	Σ	p	T	Search Time (sec)	Blind Time (sec)	Decrypt Time (sec)	Total Time (sec)
1024	36	1k	100000	373.95	5.34	32.01	411.77
1024	36	1k	10000	34.12	0.54	3.20	37.90
1024	36	1k	1000	0.00	0.06	0.32	0.39
1024	36	100	100000	39.14	5.56	31.99	77.16
1024	36	100	10000	3.76	0.54	3.20	7.55
1024	36	100	1000	0.34	0.06	0.32	0.73
1024	36	100	100	0.00	0.01	0.03	0.05
1024	36	10	100000	3.83	5.35	31.98	41.61
1024	36	10	10000	0.37	0.53	3.19	4.16
1024	36	10	1000	0.04	0.06	0.32	0.42
1024	36	10	100	0.00	0.00	0.04	0.05
1024	4	1k	100000	266.68	5.38	32.02	304.53
1024	4	1k	10000	25.35	0.55	3.20	29.15
1024	4	1k	1000	0.00	0.06	0.32	0.39
1024	4	100	100000	26.91	5.41	31.98	64.76
1024	4	100	10000	3.00	0.57	3.20	6.81
1024	4	100	1000	0.29	0.05	0.32	0.67
1024	4	100	100	0.00	0.01	0.03	0.05
1024	4	10	100000	2.70	5.39	31.89	40.43
1024	4	10	10000	0.29	0.54	3.18	4.08
1024	4	10	1000	0.03	0.06	0.32	0.42
1024	4	10	100	0.00	0.01	0.03	0.05

k	Σ	p	T	Search Time (sec)	Blind Time (sec)	Decrypt Time (sec)	Total Time (sec)
2048	36	1k	100000	1172.50	18.55	112.41	1304.83
2048	36	1k	10000	106.50	1.87	11.25	119.79
2048	36	1k	1000	0.00	0.21	1.13	1.40
2048	36	100	100000	118.26	18.64	112.80	251.07
2048	36	100	10000	11.73	1.87	11.25	25.01
2048	36	100	1000	1.05	0.21	1.12	2.44
2048	36	100	100	0.00	0.04	0.11	0.20
2048	36	10	100000	11.82	18.63	112.48	144.29
2048	36	10	10000	1.19	1.87	11.23	14.45
2048	36	10	1000	0.11	0.21	1.12	1.51
2048	36	10	100	0.01	0.04	0.12	0.21
2048	4	1k	100000	845.53	18.53	113.52	978.94
2048	4	1k	10000	79.00	1.88	11.24	92.29
2048	4	1k	1000	0.00	0.20	1.13	1.39
2048	4	100	100000	83.83	18.55	112.53	216.27
2048	4	100	10000	8.97	1.87	11.36	22.37
2048	4	100	1000	0.89	0.21	1.12	2.27
2048	4	100	100	0.00	0.04	0.11	0.20
2048	4	10	100000	8.39	18.56	112.19	140.52
2048	4	10	10000	0.90	1.87	11.23	14.18
2048	4	10	1000	0.10	0.21	1.13	1.50
2048	4	10	100	0.01	0.04	0.11	0.20

Table 13. Performance results for 1024 and 2048 bit key length in seconds for different settings of the alphabet Σ, the pattern p and the text T using fast-decryption Paillier [40].

References

1. Al-Khalifa, S., Jagadish, H.V., Patel, J.M., Wu, Y., Koudas, N., Srivastava, D.: Structural joins: A primitive for efficient xml query pattern matching. In: ICDE'02. (2002) 141–152
2. Namjoshi, K., Narlikar, G.: Robust and fast pattern matching for intrusion detection. In: INFOCOM'10, IEEE Press (2010) 740–748
3. Osadchy, M., Pinkas, B., Jarrous, A., Moskovich, B.: Scifi - a system for secure face identification. In: IEEE S&P '10, IEEE Computer Society (2010) 239–254

4. Tumeo, A., Villa, O.: Accelerating dna analysis applications on gpu clusters. In: SASP '10, IEEE Computer Society (2010) 71–76
5. Betel, D., Hogue, C.: Kangaroo - a pattern-matching program for biological sequences. BMC Bioinformatics **3** (2002) 20
6. Tsai, T.H.: Average case analysis of the Boyer-Moore algorithm. Random Struct. Algorithms **28** (2006) 481–498
7. Aho, A.V., Corasick, M.J.: Efficient string matching: an aid to bibliographic search. Commun. ACM **18** (1975) 333–340
8. Knuth, D.E., Jr., J.H.M., Pratt, V.R.: Fast pattern matching in strings. SIAM J. Comput. **6** (1977) 323–350
9. Karp, R.M., Rabin, M.O.: Efficient randomized pattern-matching algorithms. IBM J. Res. Dev. **31** (1987) 249–260
10. Baldi, P., Baronio, R., De Cristofaro, E., Gasti, P., Tsudik, G.: Countering gattaca: efficient and secure testing of fully-sequenced human genomes. In: CCS'11, ACM (2011) 691–702
11. Katz, J., Malka, L.: Secure text processing with applications to private DNA matching. In: CCS '10, ACM (2010) 485–492
12. Troncoso-Pastoriza, J.R., Katzenbeisser, S., Celik, M.: Privacy preserving error resilient DNA searching through oblivious automata. In: CCS'07, ACM (2007) 519–528
13. Freedman, M., Ishai, Y., Pinkas, B., Reingold, O.: Keyword search and oblivious pseudorandom functions. In Kilian, J., ed.: TCC'05. Volume 3378 of LNCS., Springer Berlin / Heidelberg (2005) 303–324
14. Vergnaud, D.: Efficient and secure generalized pattern matching via fast fourier transform. In Nitaj, A., Pointcheval, D., eds.: AFRICACRYPT '11. Volume 6737 of LNCS., Springer Berlin / Heidelberg (2011) 41–58
15. Jarrous, A., Pinkas, B.: Secure hamming distance based computation and its applications. In Abdalla, M., Pointcheval, D., Fouque, P.A., Vergnaud, D., eds.: ACNS'09. Volume 5536 of LNCS., (Springer Berlin / Heidelberg) 107–124
16. Hazay, C., Toft, T.: Computationally secure pattern matching in the presence of malicious adversaries. In Abe, M., ed.: ASIACRYPT'10. Volume 6477 of LNCS., Springer Berlin / Heidelberg (2010) 195–212
17. Kilian, J.: A note on efficient zero-knowledge proofs and arguments (extended abstract). In: STOC '92, ACM (1992) 723–732
18. Micali, S.: Cs proofs. In: FOCS'94, IEEE Computer Society (1994) 436–453
19. Ben-Sasson, E., Goldreich, O., Harsha, P., Sudan, M., Vadhan, S.P.: Robust PCPs of proximity, shorter PCPs, and applications to coding. SIAM J. Comput. **36** (2006) 889–974
20. Frikken, K.: Practical private dna string searching and matching through efficient oblivious automata evaluation. In Gudes, E., Vaidya, J., eds.: Data and Applications Security XXIII. Volume 5645 of LNCS., Springer Berlin / Heidelberg (2009) 81–94
21. Hazay, C., Lindell, Y.: Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries. In Canetti, R., ed.: TCC'08. Volume 4948 of LNCS., Springer Berlin / Heidelberg (2008) 155–175
22. Gennaro, R., Hazay, C., Sorensen, J.: Text search protocols with simulation based security. In Nguyen, P., Pointcheval, D., eds.: PKC'10. Volume 6056 of LNCS., Springer Berlin / Heidelberg (2010) 332–350
23. Mohassel, P., Niksefat, S., Sadeghian, S., Sadeghiyan, B.: An efficient protocol for oblivious dfa evaluation and applications (2012)
24. Yao, A.C.C.: How to generate and exchange secrets. In: FOCS'86, IEEE Computer Society (1986) 162–167
25. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game. In: STOC '87, ACM (1987) 218–229
26. Ishai, Y., Prabhakaran, M., Sahai, A.: Founding cryptography on oblivious transfer efficiently. In Wagner, D., ed.: CRYPTO'08. Volume 5157 of LNCS., Springer Berlin / Heidelberg (2008) 572–591
27. Damgrd, I., Orlandi, C.: Multiparty computation for dishonest majority: From passive to active security at low cost. In Rabin, T., ed.: CRYPTO'10. Volume 6223 of LNCS., Springer Berlin / Heidelberg (2010) 558–576
28. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: STOC '09, ACM (2009) 169–178
29. Hoffmann, H., Howard, M.D., Daily, M.J.: Fast pattern matching with time-delay neural networks. In: IJCNN'11. (2011) 2424–2429
30. Cramer, R., Gennaro, R., Schoenmakers, B.: A secure and optimally efficient multi-authority election scheme. In Fumy, W., ed.: EUROCRYPT'97. Volume 1233 of LNCS., Springer Berlin / Heidelberg (1997) 103–118
31. Desmedt, Y., Frankel, Y.: Threshold cryptosystems. In Brassard, G., ed.: CRYPTO'89. Volume 435 of LNCS., Springer Berlin / Heidelberg (1990) 307–315
32. Brandt, F.: Efficient cryptographic protocol design based on distributed el gamal encryption. In Won, D., Kim, S., eds.: ICISC'05. Volume 3935 of LNCS. Springer Berlin / Heidelberg (2006) 32–47

33. Goldreich, O.: Foundations of Cryptography: Basic Tools. Cambridge University Press, New York, NY, USA (2000)
34. Pedersen, T.: Non-interactive and information-theoretic secure verifiable secret sharing. In Feigenbaum, J., ed.: CRYPTO'91. Volume 576 of LNCS., Springer Berlin / Heidelberg (1992) 129–140
35. Damgård, I.: On Σ protocols. (`www.daimi.au.dk/~ivan/Sigma.pdf`)
36. Schnorr, C.P.: Efficient identification and signatures for smart cards. In: Proceedings of the 9th Annual International Cryptology Conference on Advances in Cryptology. CRYPTO '89, London, UK, UK, Springer-Verlag (1990) 239–252
37. Cramer, R., Damgrd, I., Schoenmakers, B.: Proofs of partial knowledge and simplified design of witness hiding protocols. In Desmedt, Y., ed.: CRYPTO'94. Volume 839 of LNCS., Springer Berlin / Heidelberg (1994) 174–187
38. Lindell, Y., Pinkas, B.: An efficient protocol for secure two-party computation in the presence of malicious adversaries. In: Proceedings of the 26th Annual International Conference on Advances in Cryptology. EUROCRYPT '07, Berlin, Heidelberg, Springer-Verlag (2007) 52–78
39. Katz, J., Ostrovsky, R.: Round-optimal secure two-party computation. In: CRYPTO 2004, Springer-Verlag (2004) 335–354
40. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In Stern, J., ed.: EUROCRYPT'99. Volume 1592 of LNCS., Springer Berlin / Heidelberg (1999) 223–238
41. : The self-certifying file system (SFS) library. (In: http://www.coralcdn.org/06wi-cs240d/lab/tools.html)