SWiM: Secure Wildcard Pattern Matching From OT Extension*

Vladimir Kolesnikov† Mike Rosulek‡ Ni Trieu‡

March 24, 2019

Abstract

Suppose a server holds a long text string and a receiver holds a short pattern string. Secure pattern matching allows the receiver to learn the locations in the long text where the pattern appears, while leaking nothing else to either party besides the length of their inputs. In this work we consider secure *wildcard* pattern matching (WPM), where the receiver's pattern is allowed to contain wildcards that match to any character.

We present SWiM, a simple and fast protocol for WPM that is heavily based on oblivious transfer (OT) extension. As such, the protocol requires only a small constant number of public-key operations and otherwise uses only very fast symmetric-key primitives. SWiM is secure against semi-honest adversaries. We implemented a prototype of our protocol to demonstrate its practicality. We can perform WPM on a DNA text (4-character alphabet) of length 10^5 and pattern of length 10^3 in just over 2 seconds, which is over two orders of magnitude faster than the state-of-the-art scheme of Baron et al. (SCN 2012).

1 Introduction

Secure two-party computation allows mutually untrusted parties to perform a computation on their private inputs without revealing any additional information except for the result itself. Over the last few years, secure two-party computation has been extensively studied and has become practical for a variety of applications [MNPS04, KS08, ZRE15, GLMY16, KNR+17, WRK17] . Two adversarial models are usually considered. In the semi-honest model, the adversary is assumed to follow the protocol, while trying to learn information from the protocol transcript. In the malicious model, the adversary can follow an arbitrary polynomial-time strategy. We consider the semi-honest model in this work.

Pattern matching is a basic problem in secure computation. It has been extensively studied in the past decade, e.g., [HL08, BEDM+12, DF13, DCFT13, FHV13, YSK+13, HT14, CS15, YSK+14, WJW+15, WZX17]. Pattern matching is frequently used in text processing, database search [GHS10, CS15], network security [NN10], DNA analysis [OPJM10], and other practical algorithms. The most commonly considered variant of secure pattern matching, which we will call exact PM, is the setting where a server with input a text $x \in \Sigma^n$ (over some alphabet Σ) interacts with a receiver with input a pattern $p \in \Sigma^m$ (for $m < n$). The receiver learns where the pattern occurs as a substring of the server's text without revealing any additional information. There

*Full version of a paper published in Financial Cryptography and Data Security 2018. The first author was supported by Office of Naval Research (ONR) contract number N00014-14-C-0113. The second and third authors were supported by NSF awards #1149647 and #1617197.

†Georgia Institute of Technology, kolesnikov@gatech.edu

‡Oregon State University, {rosulekm,trieun}@eecs.oregonstate.edu

are several important variants of pattern matching, including approximate pattern matching and outsourced pattern matching, which we discuss in Section 1.2.

In this work, we focus on secure pattern matching with *wildcards*, which we will call WPM. In this variant, the receiver's pattern can include wildcard characters that can match any character in the data, hence $p \in (\Sigma \cup \{\star\})^m$. With wildcards, the security requirements are more demanding: the server should not learn which positions of p contain wildcards, and in the case of a match the receiver should not learn the text character that matches a wildcard character in the pattern (unless this could be inferred from the presence or absence of an overlapping match).

Allowing wildcards in a pattern matching functionality has been well studied in the absence of a security requirement [CH02, CWZ+06, CLI07, CEPR09, SOF10, Tha11, BGVV14, BI14, SSSS15, AWY15], and is motivated by the goal of providing the facility of searching with errors/unknowns. Privacy issues arise in searching on sensitive data and secure pattern matching with wildcards has applications, e.g., in computational genetics and DNA analysis. Indeed, consider the case of a hospital or biomedical research center holding patient genomic data, and a researcher holding a specific cancer marker sequence with some errors. The researcher wishes to know the frequency and positions of the gene occurrences in the database. Due to the genome's highly sensitive nature, the hospital is required keep genomic data private, while the researcher needs to protect specific genome sequence he is working on. The abundance of WPM applications, such as privacy-preserving DNA matching described above, is our main motivation for improving the state-of-the-art in secure wildcard pattern matching.

1.1 Pattern Matching with Wildcards

In this section, we discuss directions and related work that achieves, or can be naturally used to achieve, the WPM functionality in the semi-honest setting.

Circuit based. Generic secure computation protocols [Yao86, GMW87], allowing evaluation of arbitrary functions, have seen tremendous performance improvements in the last decade. Modern garbled circuit (GC) protocols evaluate two million AND gates per second on a 1Gbps LAN. Several garbled circuits for Pattern Matching and its variants were studied in [JKS08, KM10]. The best protocol using this technique were proposed by Katz and Malka [KM10]. The authors showed how to modify Yao's garbled circuit to solve Pattern Matching where the size of circuit is linear in the (*a priori* upper bound on the) number of occurrences of the pattern in text. While it is possible to extend circuit-based protocol [KM10] to allow wildcards, it would still require a bound on the number of matches to be provided *a priori* for the circuit construction. When such bound is high or simply unknown, their protocol suffers corresponding performance penalty. The work [KM10] does not provide implementation or experimental results.

Homomorphic encryption based. To our knowledge, Hazay and Toft [HT10] were the first to explicitly consider wildcard secure pattern matching. The core idea of their protocol is that the receiver provides the wildcard positions to the server in an encrypted form, and the substrings of the server's text are obliviously modified so as to match the pattern at those positions. Later, Vergnaud [Ver11] improved the work of [HT10] by employing Fast Fourier Transform. Both works rely on the fact that if a pattern bit p_i is equal to a text bit t_i, then $(t_i - p_i)^2$ equals 0, and otherwise it is equal to 1. The work of Vergnaud [Ver11] requires $O((n+m)\kappa^2)$ communication and $O(n \log m)$ computational cost in both semi-honest and malicious settings, where κ is computational security parameters. As [HT10, Ver11] do not provide the experimental results, we do not compare execution times with their work.

In 2012, Baron et al. [BEDM+12] proposed an efficient pattern matching protocol called 5PM, for 5ecure Pattern Matching. They consider both malicious and semi-honest models. 5PM works with character (non-binary) wildcards, and was the first to provide an accompanying implementation. The protocol is based on an insecure pattern matching algorithm proposed by Hoffmann [HHD11]. To obtain a secure pattern matching, 5PM modifies the algorithm [HHD11] to work with basic linear operations, which allowed instantiation with additive homomorphic encryption. 5PM requires $O(n\kappa)$ communication and $O(n+m)$ computational costs in semi-honest setting. In Section 5 we compare our performance to that of 5PM and report $2-499\times$ performance improvement even on medium-size instances.

Yasuda et al. [YSK+14] extend the exact pattern matching protocol of [YSK+13] to support wildcards. The security of [YSK+14] is based on the polynomial LWE assumption. Their scheme operates by blocks, limited by the lattice dimension; for larger inputs $\boldsymbol{x}$, inefficiency is introduced either by using a larger lattice, or by the difficulty and cost of handling boundaries of blocks. In [YSK+14], the authors do not present the performance comparison with 5PM protocol, but indirectly this can be calculated. Yasuda et al. [YSK+14] mention that their protocol only $4-5\times$ slower than the protocol [YSK+13], which does not allow wildcards. In addition, [YSK+13] estimated that their work is about $10\times$ faster than 5PM when using much stronger hardware than 5PM ([YSK+13] experiments were performed on Intel Xeon X3480 3.07 GHz machine with 16 GB RAM, while 5PM [BEDM+12] used Intel dual quad-core 2.93GHz machine with 8GB RAM). Putting all together, we conclude that [YSK+14] is approximately $2-2.5\times$ faster than 5PM. In contrast, our protocol is $2-499\times$ times faster than 5PM, while running on weak commodity hardware (same at 5PM, cf. Section 5.1); this translates into the corresponding improvement over [YSK+14] as well. Further, our approach is simpler and easier to implement.

We mention recent work of Saha and Koshiba [SK17], which improves on the work of [YSK+14] by proposing a new packing method that efficiently addresses continuous wildcards occurring in the pattern (e.g., pattern `10⋆⋆⋆⋆01⋆⋆⋆110` has $k=3$ sub-patterns: `10`, `01`, and `110`). The main idea of their packing method is to let the receiver break down the pattern into k sub-patterns and have the parties perform the traditional pattern matching on these patterns. This solution is about $k\times$ faster than previous work [YSK+14]. However, it reveals significant information about the pattern, especially for larger k.

1.2 Variants of Pattern Matching

For completeness, we briefly discuss work on several additional variants of secure pattern matching.

Exact pattern patching. To our knowledge, Troncoso-Pastoriza et al. [TPKC07] were the first to consider secure pattern matching. Their protocol is based on oblivious automaton evaluation. The protocol [TPKC07] requires $O(nm)$ communication and computational cost. Several follow-up works [Fri09, MNSS12] improved the computational cost and reduced the round complexity. Another line of work [HL08, GHS10] is based on oblivious pseudorandom functions (OPRF), and obtains security in the malicious setting using $O(nm)$ communication and computational cost with $O(m)$ rounds. De Cristofaro et al. [DCFT13] consider a secure and efficient pattern matching protocol which hides the length of the pattern.

Approximate/fuzzy pattern matching. The functionality of this problem is to find the text positions matches approximately (rather than exactly). This problem can be solved by determining whether the Hamming distance between each text substring and the pattern is less than a threshold

Protocol	Computation		Communication		Rounds		Security
	Online	Offline	Online	Offline	Online	Offline	Model
[Ver11]	$\mathcal{O}(n \log m)$		$\mathcal{O}((m+n)\kappa^2)$		$O(1)$		semi-honest & malicious
[BEDM+12]	$\mathcal{O}(mn)$		$\mathcal{O}((m+n)\kappa^2)$		8		malicious
	$\mathcal{O}(m+n)$		$\mathcal{O}(n\kappa)$		2		semi-honest
Ours	0	$\mathcal{O}(\kappa)$	$\mathcal{O}(m+(\lambda+\kappa)n)$	$\mathcal{O}(nm)$	2	2	semi-honest

Table 1: Communication (bits) and computation (number of exponentiations) complexities of WPM protocols, where n is length of text, m is length of pattern; and λ and κ are the statistical and computational security parameters, respectively. $\lambda = 40$ and $\kappa = 128$ in our protocols, while κ is in the range 1024-2048 in [Ver11, BEDM+12] protocols (due to their use of public-key primitives).

t. Hazay and Toft [HT10, HT14] proposed a malicious-secure solution with $O(nt)$ communication and $O(nm)$ computation costs.

Outsourcing pattern matching. In this setting, parties outsource their encrypted data and computation to an untrusted server, while maintaining data privacy. The main goal here is to minimize the communication and computational overhead of the parties by relying on the powerful resources of the untrusted server. The first work that considered secure pattern matching in the cloud setting can be traced back to Faust et al. [FHV13]. Other follow-up works are [WJW+15]. Recently, Wei et al. [WZX17] proposed an efficient solution by combining a secret sharing scheme and oblivious transfer which requires $O(\kappa)$ computation and $O(mn)$ communication costs. Outsourcing pattern matching can be viewed as substring searchable encryption which are studied in [CS15, ÇCL+17]

2 Overview of Our Results & Techniques

In this work we present SWiM (**S**ecure **Wi**ldcard Pattern **M**atching), a protocol for WPM based on two fast cryptographic tools: oblivious transfer and secure string equality test (given two strings of equal lengths, without wildcards, determine whether they are equal). Thanks to recent optimizations in oblivious transfer protocols [Bea96, IKNP03, KK13, ALSZ13], it is possible to realize a large number of OT instances with amortized cost of only a few μs. Kolesnikov et al. [KKRT16] give a protocol for secure string equality test based on techniques for efficient OT. With their protocol, one can perform many private equality tests with amortized cost of 5 μs.

Overview of techniques. Suppose the sender holds a string $\boldsymbol{x} \in \{0,1\}^*$ and the receiver holds a pattern $\boldsymbol{p} \in \{0,1,\star\}^*$.

As a very simple warm up, consider the case that $|\boldsymbol{x}| = |\boldsymbol{p}| = 1$. The receiver will first encode its pattern $\boldsymbol{p} \in \{0,1,\star\}$ as a pair of bits $(\overset{\star}{\boldsymbol{p}}, \bar{\boldsymbol{p}})$ ("p-star & p-bar"), using the following encoding:

$\boldsymbol{p}$	$\overset{\star}{\boldsymbol{p}}$	$\bar{\boldsymbol{p}}$
$\star$	1	0
1	0	1
0	0	0

(1)

The significance of this encoding is the following:

$$\boldsymbol{x} \text{ matches pattern } \boldsymbol{p} \iff \boldsymbol{x} = \overset{\star}{\boldsymbol{p}} \cdot \boldsymbol{x} \oplus \bar{\boldsymbol{p}} \tag{2}$$

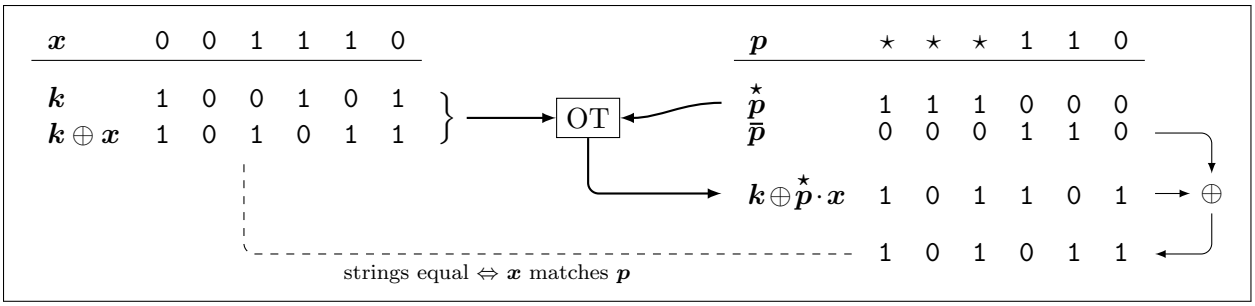

Figure 1: Illustration of the main idea behind our protocol: using oblivious transfer and private string equality test to perform private string equality with wildcards.

Indeed, if $\boldsymbol{p} = \star$, then $(\overset{\star}{\boldsymbol{p}}, \bar{\boldsymbol{p}}) = (1, 0)$, so the RHS of (2) simplifies to $\boldsymbol{x}$ and the two sides equal (regardless of $\boldsymbol{x}$). On the other hand, if $\boldsymbol{p} \neq \star$, then $(\overset{\star}{\boldsymbol{p}}, \bar{\boldsymbol{p}}) = (0, \boldsymbol{p})$, so the RHS simplifies to $\boldsymbol{p}$ and the two sides equal if and only if $\boldsymbol{p} = \boldsymbol{x}$.

Our next trick is to blindly evaluate equation (2) using a single OT evaluation. The parties invoke an instance of 1-out-of-2 bit-OT, where the sender gives inputs $(\boldsymbol{k}, \boldsymbol{k} \oplus \boldsymbol{x})$, and the receiver gives input $\overset{\star}{\boldsymbol{p}}$. Here $\boldsymbol{k}$ is a random bit chosen by the sender. Note that the receiver's output from this OT is $\boldsymbol{k} \oplus \overset{\star}{\boldsymbol{p}} \cdot \boldsymbol{x}$.

Now, adding $\boldsymbol{k}$ to both sides of the equation in (2), we have that $\boldsymbol{x}$ matches pattern $\boldsymbol{p}$, if and only if $\boldsymbol{k} \oplus \boldsymbol{x} = (\boldsymbol{k} \oplus \overset{\star}{\boldsymbol{p}} \cdot \boldsymbol{x}) \oplus \bar{\boldsymbol{p}}$. Importantly, the LHS of this equation is known to the sender, while the RHS is known to the receiver. At the same time, the random mask $\boldsymbol{k}$ hides all information about $\boldsymbol{x}$ from the receiver. We can summarize the above gadget as follows: using a single OT of bits, the sender and receiver each compute a bit which is *the same bit if and only if* $\boldsymbol{x}$ *matches pattern (possibly wildcard)* $\boldsymbol{p}$.

This technique can be easily extended to the case of WPM with $|\boldsymbol{x}| = |\boldsymbol{p}| = n$ by simply doing the above gadget n times, bit-by-bit. After doing so, each party will hold an n-bit string (without wildcards); these two strings will be equal if and only if $\boldsymbol{x}$ matches the pattern $\boldsymbol{p}$. An example is given in Figure 1 (we simply extend the notation $\oplus$ and $\cdot$ to bit-vectors). In short, we have reduced the problem of WPM with $|\boldsymbol{x}| = |\boldsymbol{p}|$ to the problem of secure (exact, no wildcards) equality test of strings. We complete the wildcard pattern matching by actually testing the equality of these strings, using the efficient protocol of Kolesnikov et al. [KKRT16].

The security of this protocol (in the semi-honest model) is easy to understand: the only new information is that the receiver obtains output $\boldsymbol{k} \oplus \overset{\star}{\boldsymbol{p}} \cdot \boldsymbol{x}$, which leaks no information about the sender's input $\boldsymbol{x}$ since $\boldsymbol{k}$ is uniform.

Now consider extending this approach to the general case of WPM with $|\boldsymbol{x}| > |\boldsymbol{p}|$. The idea is the natural one: for each $i \in \{1, |\boldsymbol{x}| - |\boldsymbol{p}| + 1\}$ simply perform the above approach on the substring $x[i \ldots i+|\boldsymbol{p}|-1]$ and $\boldsymbol{p}$. Unpacking the abstractions reveals room for optimizations, as follows. While the previous constructions were presented in terms of OT of *bits,* the OT of strings is significantly more efficient in practice. We observe that in each subprotocol, the receiver's OT choice bits are always the same $\overset{\star}{\boldsymbol{p}}$, allowing corresponding OT instances to be combined easily. Hence instead of $|\boldsymbol{p}|(|\boldsymbol{x}| - |\boldsymbol{p}| + 1)$ instances of bit-OT, we can use $|\boldsymbol{p}|$ instances of string-OT, with strings of length $|\boldsymbol{x}| - |\boldsymbol{p}| + 1$. This optimization actually reduces costs by a multiplicative factor of the security parameter. The details are given in Section 4.

In Section 4.1 we present additional optimizations and extensions, such as moving almost all of

the cost to the offline, amortization and efficient handling of non-binary alphabets.

Efficiency. SWiM requires only $O(\kappa)$ public-key operations (all in the offline phase). In terms of communication, our protocol requires $O(mn)$ in the offline phase, but only $O(m+(\lambda+\kappa)n)$ in online phase. Here, κ, λ are the computational and statistical security parameters, respectively. As noted previously, all constants under the big-O are small, as we use fast optimized building blocks. We describe the performance of representative Secure Wildcard Pattern Matching protocols in Table 1.

We note that SWiM is efficient *concretely*. This is because we carefully optimize both computation and communication. Further, we use algorithmically- and implementation-optimized building blocks, namely the OT extension of [ALSZ13] and private equality test of [KKRT16]. In particular, the [KKRT16] equality test is *independent* of the length of the players' inputs.

This significantly improves over the state-of-the-art secure wildcard pattern matching protocol of [BEDM+12]. In Section 5, we report in detail on implementation and evaluation, and find that SWiM is a $2-499\times$ faster than 5PM, and continues to scale well on larger instances. 5PM considers WPM instances on DNA text of length up to 10^5 and pattern of length up to 10^3. These larger instances require only 1.96 seconds in our protocol, in comparison with 304.53 seconds with 1024-bit key and 978.94 seconds with 2048-bit key using [BEDM+12].

3 Preliminaries

3.1 Notation

Throughout the paper we use the following notation: The length of the text is n, while the length of the pattern is m. Wildcard is denoted by $\star$. The computational and statistical security parameters are denoted by κ, λ, respectively. $[m]$ to denote a set $\{1, \ldots, m\}$.

The notation OT_m denotes a 1-out-of-2 OT where the string is m bits long. We denote vectors in bold $\boldsymbol{a}$, and matrices in capitals A. For the vector, we let $\boldsymbol{a}_{[i,j]}$ denote the sub-vector of $\boldsymbol{a}$ from i-th bit to j-th bit, and a_i denote the i-th bit of vector $\boldsymbol{a}$. Given vectors $\boldsymbol{a} = a_1 \| \cdots \| a_n$ and $\boldsymbol{b} = b_1 \| \cdots \| b_n$, we define $\oplus$ and $\cdot$ operations as follows. We use the notation $\boldsymbol{a} \oplus \boldsymbol{b}$ to denote the vector $(a_1 \oplus b_1) \| \cdots \| (a_n \oplus b_n)$. Similarly, the notation $\boldsymbol{a} \cdot \boldsymbol{b}$ denotes the vector $(a_1 \cdot b_1) \| \cdots \| (a_n \cdot b_n)$. Let $c \in \{0,1\}$, then $c \cdot \boldsymbol{a}$ denotes the vector $(c \cdot a_1) \| \cdots \| (c \cdot a_n)$. For a matrix A, we let $\mathbf{a}_i$ denote the i-th row of A, $\mathbf{a}^j$ denote the j-th column of A; A_i^j denote the entry of A at the i-th row and the j-th column.

Consider an alphabet Σ. We define a **pattern matching relation** $\preceq$ via the following rules: (1) $a \preceq a$ for $a \in \Sigma$; (2) $\star \preceq a$ for $a \in \Sigma$. We extend the notation to vectors as $\boldsymbol{x} \preceq \boldsymbol{y} \Leftrightarrow (\forall i) x_i \preceq y_i$. If $\boldsymbol{p} \preceq \boldsymbol{x}$ we say that $\boldsymbol{x}$ **matches the pattern** $\boldsymbol{p}$**.**

3.2 Oblivious Transfer

Oblivious Transfer (OT) is a ubiquitous cryptographic primitive, and necessary for secure computation, which was introduced by Rabin [Rab05]. In OT, a sender with two input strings (x_0, x_1) interacts with a receiver who has a input choice bit b. In a privacy-preserving way, the receiver learns x_b without learning anything about x_{1-b}, while the sender learns nothing about b. Rabin's protocol requires expensive public key cryptography. Ishai et al. [IKNP03] proposed OT extension, an efficient protocol that evaluates a small number of expensive OTs, from which a large number of OTs can be performed using only cheap symmetric-key operations. OT extension, to which

PARAMETERS: A bit length m, and two parties: sender $\mathcal{S}$ and receiver $\mathcal{R}$

FUNCTIONALITY:

- Wait for pair-input $(\boldsymbol{x}_0, \boldsymbol{x}_1) \subseteq \{0,1\}^m$ from the sender $\mathcal{S}$
- Wait for bit-input $b \in \{0,1\}$ from the receiver $\mathcal{R}$
- Give output $\boldsymbol{x}_b$ to the receiver $\mathcal{R}$.

Figure 2: Oblivious Transfer functionality OT_m.

PARAMETERS: Two parties: sender $\mathcal{S}$ and receiver $\mathcal{R}$

FUNCTIONALITY:

- Wait for input $\boldsymbol{x}_0 \in \{0,1\}^*$ from the sender $\mathcal{S}$.
- Wait for input $\boldsymbol{x}_1 \in \{0,1\}^*$ from the receiver $\mathcal{R}$.
- Give the receiver $\mathcal{R}$ output 1 if $\boldsymbol{x}_0 = \boldsymbol{x}_1$ and 0 otherwise.

Figure 3: The Private Equality ideal functionality $\mathcal{F}_{\mathsf{peqt}}$

PARAMETERS: A text length n, a pattern length m, and two parties: sender $\mathcal{S}$ and receiver $\mathcal{R}$

FUNCTIONALITY:

- Wait for text $\boldsymbol{x} \in \{0,1\}^n$ from the sender $\mathcal{S}$
- Wait for pattern $\boldsymbol{p} \in \{0,1,\star\}^m$ from the receiver $\mathcal{R}$
- Give the receiver $\mathcal{R}$ output $\{i \in [n-m+1] \mid \boldsymbol{p} \preceq \boldsymbol{x}_{[i,i+m-1]}\}$ (see Section 3.1 for notation)

Figure 4: Wildcard Pattern Matching functionality $\mathcal{F}_{\mathsf{wpm}}^{n,m}$.

we sometimes refer as IKNP, has become a core building block in many aspects of secure computation such as Garble Circuit, Private Set Intersection [PSZ14, KKRT16, KMP+17], Hamming Distance [BCP13]. We describe the ideal functionality for OT in Figure 2.

Despite the wide use of OT, there are very few improvement of IKNP OT protocol in semi-honest setting. In 2013, Kolesnikov and Kumaresan [KK13] proposed an generalization of IKNP OT extension for short secrets, which brought $O(\log(\kappa))$ factor performance improvement in communication and computation, where κ is security parameter. They also proposed an IKNP optimization, saving on the auxiliary matrix transfer. Later same year, Asharov et al. [ALSZ13] proposed several IKNP optimizations (one of which was the optimization independently discovered by [KK13]). Importantly, [ALSZ13] also provided optimized implementation of (improved) IKNP OT protocol.

[ALSZ13] also presented optimizations for a useful variant of OT. In Correlated OT (COT), the sender's OT inputs x_0, x_1 are chosen randomly subject to $x_0 \oplus x_1 = \Delta$, where Δ is chosen by the sender (possibly a different Δ for each OT instance). In this case, it is possible to let the protocol itself "choose" the value x_0 randomly. Doing so reduces the bandwidth requirement by approximately half. It is easy to see that we require only this weaker variant of OT for pattern matching, hence our implementation takes advantage of this optimization.

3.3 Private Equality Test

Definition. A **Private Equality Test (PEQT)** is a 2-party protocol in which the sender with input string $\boldsymbol{x}_0$ interacts with a receiver with input string $\boldsymbol{x}_1$ in the following way. The receiver learns a bit indicating whether $\boldsymbol{x}_0 = \boldsymbol{x}_1$ and nothing else, while the sender learns nothing about $\boldsymbol{x}_1$. We describe the ideal functionality for an PEQT in Figure 3.

To our knowledge, PEQT was first introduced in 1996 by Fagin, Naor, and Winkler [FNW96]. Follow-up works[NP99, BST01, Lip03] improved the efficiency of PEQT, while still relying on expensive public-key operations. PEQT is heavily used in two-party private set intersection (PSI) protocols [FNP04]. Recently, Kolesnikov et al. [KKRT16], in the context of PSI proposed an efficient PEQT, which was constructed by applying novel encodings inside the OT extension matrix. Their protocol, cast as a variant of Oblivious PRF, executes many PEQT instances by using only cheap symmetric cryptographic operations, apart from base OTs. Concretely, the amortized cost of each PEQT instance with unbounded input domain $\{0,1\}^*$ is only a few symmetric-key operations and 488 bits in communication. We heavily rely on the high-performing PEQT protocol of [KKRT16] in this work.

4 SWiM: the Main Construction

We present SWiM, our main construction for the WPM functionality in Figure 4. It closely follows and formalizes the high-level overview presented in Section 2. For readability, we present SWiM for binary alphabet $\Sigma = \{0,1\}$. In Section 4.1 we show how to easily extend it to an arbitrary Σ. We first run OT extension with the chosen inputs defined in Figure 2, which will allow the receiver to compute $\boldsymbol{\alpha} = \boldsymbol{k} \oplus \overset{\star}{\boldsymbol{p}} \cdot \boldsymbol{x} \oplus \bar{\boldsymbol{p}}$. Recall, as discussed in Section 2, $\boldsymbol{x}$ matches $\boldsymbol{p}$, iff $\boldsymbol{\alpha}$ equals to $\boldsymbol{k} \oplus \boldsymbol{x}$ held by the sender. This equality is efficiently checked in bulk by calling instances of Private Equality Test defined in Figure 3, with the result delivered to the receiver and output. The SWiM protocol is presented in Figure 5 and is proven secure against semi-honest adversaries.

Correctness. The main observation of OT-extension is that the receiver obtains output $\boldsymbol{q}_i$ such that:

$$\boldsymbol{q}_i = \boldsymbol{k}_i \oplus \overset{\star}{p}_i \cdot \boldsymbol{x}_{[i,i+n'-1]} = \begin{cases} \boldsymbol{k}_i, & \text{if } \overset{\star}{p}_i = 0 \\ \boldsymbol{k}_i \oplus \boldsymbol{x}_{[i,i+n'-1]}, & \text{if } \overset{\star}{p}_i = 1 \end{cases}$$

Therefore, the i-th row of U is equal to $\boldsymbol{u}_i = \boldsymbol{k}_i \oplus \overset{\star}{p}_i \cdot \boldsymbol{x}_{[i,i+n'-1]} \oplus C(\bar{p}_i)$. Let K denote the $m \times n'$ matrix such that the i-th row of K is the vector $\boldsymbol{k}_i$. When viewing the matrices U and T column-wise, we see that the receiver holds $\boldsymbol{u}^i = \boldsymbol{k}^i \oplus \overset{\star}{\boldsymbol{p}} \cdot \boldsymbol{x}_{[i,i+m-1]} \oplus \bar{\boldsymbol{p}}$ while the sender holds $\boldsymbol{t}^i = \boldsymbol{k}^i \oplus \boldsymbol{x}_{[i,i+m-1]}$. Following the high-level idea described Section 2, and specifically the pattern match test of Equation 2, it is clear that the pattern matches the text $\boldsymbol{x}$ at the i-th position if and only if $\boldsymbol{u}^i = \boldsymbol{t}^i$.

Theorem 1. *The SWiM protocol in Figure 5 securely computes the WPM functionality (Figure 4) in semi-honest setting, given the ideal OT and $\mathcal{F}_{peqt}$ primitives defined Figure 2 and Figure 3, respectively.*

Proof. The proof of security of our construction is based on the fact that the OT and $\mathcal{F}_{\text{peqt}}$ are secure.

PARAMETERS:

1. Two parties: sender $\mathcal{S}$ and receiver $\mathcal{R}$
2. A length n of text, a length m of pattern. Define $n' = n - m + 1$
3. A repetition encoding $C : \{0,1\} \to \{0,1\}^{n'}$ defined by $C(a) = a^{n'}$ for $a \in \{0,1\}$.
4. Ideal OT and $\mathcal{F}_{\mathsf{peqt}}$ primitives defined in Figure 2 and Figure 3, respectively.

INPUT OF $\mathcal{S}$: a text $\boldsymbol{x} \in \{0,1\}^n$

INPUT OF $\mathcal{R}$: a pattern $\boldsymbol{p} \in \{0,1,\star\}^m$ encoded into $\bar{\boldsymbol{p}}, \overset{\star}{\boldsymbol{p}} \in \{0,1\}^m$, as described in Section 2.

PROTOCOL:

1. **[Random Keys]** $\mathcal{S}$ chooses $\{\boldsymbol{k}_i\}_{i\in[m]} \leftarrow \{0,1\}^{n'}$ at random
2. **[OT]** For each $i \in [m]$, $\mathcal{S}$ and $\mathcal{R}$ invoke $\mathsf{OT}_{n'}$-functionality
   (a) $\mathcal{R}$ acts as *receiver* with a input-bit $\overset{\star}{p}_i$.
   (b) $\mathcal{S}$ acts as *sender* with a ordered pair input $(\boldsymbol{k}_i, \boldsymbol{k}_i \oplus \boldsymbol{x}_{[i,i+n'-1]})$
   (c) $\mathcal{R}$ receives output $\boldsymbol{q}_i$
3. **[Matrix Form]**
   (a) $\mathcal{S}$ forms $m \times n'$ matrix T such that the i-th row of T is the vector $\boldsymbol{t}_i = \boldsymbol{k}_i \oplus \boldsymbol{x}_{[i,i+n'-1]}$
   (b) $\mathcal{R}$ forms $m \times n'$ matrix U such that the i-th row of U is the vector $\boldsymbol{u}_i = \boldsymbol{q}_i \oplus C(\bar{p}_i)$.
4. **[PEQ]**
   (a) For each $i \in [n']$, $\mathcal{S}$ and $\mathcal{R}$ invoke the $\mathcal{F}_{\mathsf{peqt}}$-functionality:
      - $\mathcal{S}$ acts as *sender* with input $\boldsymbol{t}^i$ as the i-th column of T
      - $\mathcal{R}$ acts as *receiver* with input $\boldsymbol{u}^i$ as the i-th column of U
   (b) $\mathcal{R}$ outputs $\{i \in [n'] \mid i\text{th instance of } \mathcal{F}_{\mathsf{peqt}} \text{ outputs } 1\}$

Figure 5: SWiM: **S**ecure **Wi**ldcard Pattern **M**atching Protocol for $\Sigma = \{0,1\}$.

Simulating $\mathcal{S}$. It is easy to argue that the view of the sender $\mathcal{S}$ can be perfectly simulated since the semi-honest $\mathcal{S}$ receives nothing from the protocol.

Simulating $\mathcal{R}$. The view of the receiver $\mathcal{R}$ consists of two kinds of messages: (1) output of the form $\boldsymbol{q}_i$ from the OT primitive in Step 2c, which is equal to $\boldsymbol{k}_i \oplus \overset{\star}{p}_i \cdot \boldsymbol{x}_{[i,i+n'-1]}$ and hence information-theoretically hides $\boldsymbol{x}$; (2) outputs of $\mathcal{F}_{\mathsf{peqt}}$ in step 4b, which correspond exactly to the WPM protocol output itself. Hence both can be perfectly simulated. □

Cost. Using OT extension, some initial "base OT" instances are required. These base OTs consist of $O(\kappa^2)$ communication and $O(\kappa)$ exponentiations. Thereafter, any number of OTs can be obtained with communication and computation proportional only to total size of parties' inputs. The computation consists of only *symmetric-key* operations. In our case, there are m OT instances,

each on strings of length n', so $O(n'm)$ total communication and symmetric-key operations.

The $\mathcal{F}_{\mathsf{peqt}}$ protocol of [KKRT16] has a statistical security parameter which we denote λ. Specifically, the protocol allows for a false positive (output 1 for input strings which are different) with probability $2^{-\lambda}$. The protocol also uses OT extension, but the base OTs can be shared/reused from the base OTs mentioned above. The amortized cost of an equality test is $448 + \lambda$ bits of communication (using typical parameters) and a constant number of symmetric-key operations.

4.1 Additions, optimizations

Online/offline phase. We briefly describe how the protocol can be modified so that most of the cost can be incurred in an offline phase, before the parties' inputs are known.

First, we can run all OTs in Step 2 of the protocol before the receiver's input $\boldsymbol{p}$ is known, by taking advantage of a well-known technique of Beaver [Bea95]. The following modifications are required: First, the receiver uses a random $\boldsymbol{\pi} \in \{0,1\}^m$ (rather than $\overset{\star}{\boldsymbol{p}}$) as its OT choice bits in Step 2 (note that $\overline{\boldsymbol{p}}$ is not used until Step 3). Later, upon learning $\boldsymbol{p}$, the receiver sends $\boldsymbol{\delta} = \boldsymbol{p} \oplus \boldsymbol{\pi}$ to the sender. The sender sets $\boldsymbol{k}'_i = \boldsymbol{k}_i \oplus \delta_i \cdot \boldsymbol{x}_{[i,i+n'-1]}$. It is easy to see that the receiver holds

$$\boldsymbol{q}_i = \boldsymbol{k}_i \oplus \pi_i \cdot \boldsymbol{x}_{[i,i+n'-1]} = \boldsymbol{k}_i \oplus (\delta_i \oplus \overset{\star}{p}_i) \cdot \boldsymbol{x}_{[i,i+n'-1]} = \boldsymbol{k}'_i \oplus \overset{\star}{p}_i \cdot \boldsymbol{x}_{[i,i+n'-1]}.$$

In other words, $\boldsymbol{k}'_i$ and $\boldsymbol{q}_i$ satisfy the appropriate condition, now with respect to the receiver's true input $\boldsymbol{p}$. The rest of the protocol continues as usual, with $\boldsymbol{k}'_i$ instead of $\boldsymbol{k}_i$.

There is also a standard Beaver technique for preprocessing OTs before the sender's OT input is known. Applying here naively would require the sender to send online correction strings of total length $O(|\boldsymbol{p}||\boldsymbol{x}|)$ since that is the combined length of all the sender's OT inputs.

Instead, we propose the following technique that is similar in spirit but takes advantage of the fact that the sender's OT inputs are derived from a single $\boldsymbol{x}$ value. The parties run step 1, but with the sender using a random $\boldsymbol{\chi} \in \{0,1\}^n$ instead of the true input $\boldsymbol{x}$ (which is not yet known). After the online phase described above, the sender will have $\boldsymbol{k}'_i$ strings and the receiver will have $\boldsymbol{q}_i = \boldsymbol{k}'_i \oplus \overset{\star}{p}_i \cdot \boldsymbol{\chi}_{[i,i+n'-1]}$. As the sender learns its input $\boldsymbol{x}$, it sends $\boldsymbol{\gamma} = \boldsymbol{x} \oplus \boldsymbol{\chi}$ to the receiver. The receiver can compute

$$\begin{aligned}
\boldsymbol{q}'_i \overset{\text{def}}{=} \boldsymbol{q}_i \oplus \overset{\star}{p}_i \cdot \boldsymbol{\gamma}_{[i,i+n'-1]} &= (\boldsymbol{k}'_i \oplus \overset{\star}{p}_i \cdot \boldsymbol{\chi}_{[i,i+n'-1]}) \oplus \overset{\star}{p}_i \cdot \boldsymbol{\gamma}_{[i,i+n'-1]} \\
&= \boldsymbol{k}'_i \oplus \overset{\star}{p}_i \cdot (\boldsymbol{\chi}_{[i,i+n'-1]} \oplus \boldsymbol{\gamma}_{[i,i+n'-1]}) \\
&= \boldsymbol{k}'_i \oplus \overset{\star}{p}_i \cdot \boldsymbol{x}_{[i,i+n'-1]}
\end{aligned}$$

In other words, $\boldsymbol{k}'_i$ and $\boldsymbol{q}'_i$ satisfy the appropriate condition, now with respect to the sender's true input $\boldsymbol{x}$. The protocol can proceed, using $\boldsymbol{q}'_i$ instead of $\boldsymbol{q}_i$.

By having precomputation, we are able to shift the bulk of the $O(nm)$ communication to the offline phase. In the online phase, each party only sends a "correction string" whose length is proportional to its input size, followed by the equality tests. Similarly to the standard Beaver's technique, it is easy to see that the resulting protocol is secure, namely that the separation of the offline and online phases can be simulated.

Amortization. In certain multiple-execution scenarios, the cost of our protocol can be further significantly reduced by reusing the OT/PEQT outputs.

First, notice that in SWiM (Figure 5), the OT step is independent of the non-wildcard characters of the pattern string (i.e., independent of $\overline{\boldsymbol{p}}$). Therefore, if the *positions of wildcards* in the

receiver's pattern (i.e., $\overset{\star}{\boldsymbol{p}}$) are the same across several executions, OT in subsequent executions can be implemented as length extension of the OT in the first execution. Further, if additionally the sender's text is the same across the executions (and the only variation is the non-$\star$ pattern), then only the equality tests need to be run in the subsequent executions.

Further, in the PEQT protocol of [KKRT16], the receiver can check his input for equality against a polynomial number sender's inputs at the cost λ per check (vs $4\kappa + \lambda$ for full KKRT PEQT). Indeed, on the KKRT BaRK-OPRF output $(R, S) \leftarrow (F_k(x), k)$, KKRT sender $\mathcal{S}$ can send to receiver $\mathcal{R}$ a set of $\{F_k(y_i)\}$, and R will determine $x = y_i \iff F_k(x) = F_k(y_i)$.

To use this in the amortization, we let the WPM sender play the role of PEQT's receiver. We note that this amortization will reveal whether the WPM receiver has used the same pattern in different instances. Additionally, PEQT receiver learns the comparison output, and so will the WPM sender. Both restrictions may be acceptable in certain scenarios.

Non-binary alphabets. The protocol extends naturally to alphabets Σ beyond $\Sigma = \{0, 1\}$. Without loss of generality let $\Sigma = \mathbb{Z}_b$ for some b. The receiver holds a pattern $\boldsymbol{p} \in (\Sigma \cup \{\star\})^m$ and will encode the pattern into $\overset{\star}{\boldsymbol{p}} \in \{0, 1\}^m$ and $\overline{\boldsymbol{p}} \in \Sigma^m$, as follows:

$\boldsymbol{p}_i$	$\overset{\star}{\boldsymbol{p}}_i$	$\overline{\boldsymbol{p}}_i$
$\star$	1	0
$a \neq \star$	0	a

Consider the corresponding amendment to SWiM (Figure 5), where the parties hold strings of length m and n, both over the alphabet Σ. The parties still perform m 1-out-of-2 OT, using $\overset{\star}{\boldsymbol{p}}$ as the receiver's choice bits. All other vectors ($\boldsymbol{k}$, $\boldsymbol{q}$, etc) become vectors over Σ, and the $\oplus$ operation is replaced by component-wise addition mod $|\Sigma|$. Note that the "$\cdot$" operation in the protocol is only used between a *binary* vector $\overset{\star}{\boldsymbol{p}}$ and a Σ-vector, so its meaning can still be taken as component-wise multiplication. Finally, the KKRT PEQT can be naturally amended to support equality tests of non-binary strings, e.g. by translating the strings into binary.

5 SWiM Implementation and Performance

Our SWiM implementation uses code from [KKRT16, Rin, WMK16]. All running times are reported as the average over 10 trials. Our complete implementation is available on https://github.com/osu-crypto/PatternMatching.

5.1 Experimental Performance: Comparison with Prior Work

We compare our prototype to the state-of-art WPM protocols [BEDM+12, YSK+14]. While the implementations [BEDM+12, YSK+14] are not publicly available, [BEDM+12] reports experimental numbers in the semi-honest model. Further, as we discussed in Section 1.1, [YSK+14] numbers can be indirectly estimated to be around $2 - 2.5\times$ faster than 5PM. We give detailed comparisons to 5PM protocol [BEDM+12]; comparison to other works can be appropriately derived.

Runtime Comparison. For the most direct comparison, we matched the test system's computational performance to that of [BEDM+12], as reported in their Table 13. Since 5PM [BEDM+12] experiments were performed on Intel dual quad-core 2.93GHz Linux machine with 8GB RAM, we evaluate our protocol on a virtual Linux machine with 8GB RAM and 2 cores (the host machine

is Intel Core i7 2.60GHz with 12GB RAM). Table 2 presents the running time of our protocol compared with 5PM [BEDM+12]. For our protocol, we report both the total running time and the online time. We use $\lceil \log(\Sigma) \rceil$ bits to encode the text and pattern alphabet into binary alphabet.

When comparing the two protocols, we find that the total running time of SWiM is significantly less than that of the prior works, requiring 1.96 seconds to perform a wildcard pattern matching with 4-symbol alphabet for text size $n = 10^5$ and pattern size $m = 10^3$. This is a 155× improvement in running time compared to 5PM [BEDM+12] which used 1024-bit key length. When considering 5PM [BEDM+12] with 2048-bit key length (which better corresponds to our security level), our improvement is 499×.

SWiM is optimized for the typical use case, where the length of the text is greater than that of the pattern. If this doesn't hold (indeed, an unusual setting for the motivating examples we consider), our performance improvement is moderate. For instance when $m = n = 10^3$, our protocol requires 0.61 seconds. Using the same parameters, the protocol of [BEDM+12] results in an execution time of 1.39 seconds. The moderate 2× improvement is due to the constant-cost overheads of OT extension and PEQT, which do not pay off without amortization in a larger execution. Even in these cases, our protocol achieves great improvement in the online phase (e.g., running in just 3ms for $m = n = 10^3$).

Bandwidth Comparison. We calculate the bandwidth requirements of our protocol on the range of the length text $n \in \{2^{16}, 2^{18}, 2^{20}, 2^{22}\}$ and the length pattern $m \in \{2^8, 2^{10}, 2^{12}, 2^{14}\}$, for the binary alphabet. For comparison, we calculate the communication cost of 5PM [BEDM+12], for the same parameters. 5PM bandwidth requirements is independent on the length of pattern, and is roughly $(n+2)\kappa$ bits. 5PM protocol relies on public-key operations, and needs 1024-2048-bit key lengths.

Table 4 reports the communication overhead of the protocols. Our protocol requires less communication for smaller pattern sizes. Concretely, for $n = 2^{22}$ and $m = 2^8$, our protocol requires 392.1 MB of communication, a 1.37 to 2.7× improvement compared to 5PM [BEDM+12]. Increasing the pattern length to $m = 2^{12}$ the communication cost of 5PM protocol (at a great performance penalty!) becomes preferable to ours, since their bandwidth is independent of the length of pattern. Note, the bulk of the communication cost in our protocol is OT extension in the offline phase.

We note that Table 4 does not show off SWiM algorithmic improvement for non-binary alphabet, which reduces the number of OT calls. For larger Σ, we (but not other approaches, to our knowledge) get factor $\approx \log |\Sigma|$ bandwidth reduction in the *offline* phase over the simple mapping of Σ to a binary alphabet.

5.2 SWiM performance at scale: experiments and discussion

To understand the scalability of SWiM, we evaluate it on the range of the text/pattern lengths $n \in \{2^{16}, 2^{18}, 2^{20}, 2^{22}, 2^{24}\}$, $m \in \{2^8, 2^{10}, 2^{12}, 2^{14}\}$, for the binary alphabet. We report SWiM detailed performance results in Table 3, showing total running time and online time in both LAN and WAN settings.

This set of experiments was ran on a larger machine (a single server with 2x 36-core Intel Xeon 2.30GHz CPU and 256GB of RAM), whose resources were carefully limited by us to provide a good understanding of the performance. Specifically, we ran each party *single threaded*, both on the same machine, communicating via `localhost` network. We simulated a network connection using the Linux `tc` command. We configured LAN setting with 0.02ms round-trip latency, 10 Gbps network bandwidth, and WAN setting with a simulated 40ms round-trip latency, 400 Mbps network bandwidth.

Protocol	Bit key length	Pattern length m	Text length n 10^3	10^4	10^5
5PM	1024	10	0.42	4.08	40.43
		10^2	0.67	6.81	64.76
		10^3	0.39	29.15	304.53
	2048	10	1.50	14.18	140.52
		10^2	2.27	22.37	216.27
		10^3	1.39	92.29	978.94
SWiM	128	10	**0.29 (0.006)**	**0.36 (0.03)**	**0.76 (0.32)**
		10^2	**0.37 (0.005)**	**0.62 (0.09)**	**1.82 (0.49)**
		10^3	**0.61 (0.003)**	**0.73 (0.04)**	**1.96 (0.39)**

Table 2: 5PM vs SWiM. Comparison of 5PM and SWiM of the total runtime (in seconds) for wildcard pattern matching of length n, the pattern of length m, and the alphabets of sizes 4 (DNA). In SWiM, the online time is presented in parenthesis. Best results marked in bold. SWiM experiment ran on Intel Core i7 2.60GHz with 8GB RAM. 5PM timings reported on comparable hardware.

Setting	Pattern length m	Text length n 2^{16}	2^{18}	2^{20}	2^{22}	2^{24}
LAN	2^8	**0.21 (0.04)**	0.40 (0.15)	0.94 (0.48)	4.07 (2.78)	16.11 (11.38)
	2^{10}	0.24 (0.03)	0.48 (0.12)	1.41 (0.57)	5.21 (2.38)	20.61 (10.00)
	2^{12}	0.37 (0.03)	0.97 (0.17)	3.40 (0.78)	12.92 (3.34)	**51.88 (14.44)**
	2^{14}	1.02 (0.07)	3.91 (0.37)	15.14 (1.66)	**60.10 (6.46)**	246.24(43.51)
WAN	2^8	**1.04 (0.40)**	1.90 (1.02)	5.10 (3.10)	17.84 (12.04)	70.45 (48.43)
	2^{10}	1.28 (0.40)	2.81 (0.95)	8.62 (3.04)	31.29 (12.00)	127.92 (48.08)
	2^{12}	2.28 (0.36)	6.46 (0.96)	21.61 (3.17)	84.52 (12.48)	**363.06 (50.15)**
	2^{14}	6.16 (0.34)	22.24 (1.07)	85.98 (3.87)	318.23 (15.45)	1,382.03 (65.86)

Table 3: SWiM scaling. Total running time and online time (in parenthesis) in second of SWiM for the text of length n, the pattern of length m, binary alphabet. The results mentioned in the discussion is marked in bold. Experiment ran sender and receiver *single-threaded* on 2x 36-core Intel Xeon 2.30GHz CPU and 256GB of RAM.

Protocol	Bit key length	Pattern length m	Text length n 2^{16}	2^{18}	2^{20}	2^{22}
5PM	1024	$\{2^8, 2^{10}, 2^{12}, 2^{14}\}$	8.4	33.5	134.2	536.9
	2048	$\{2^8, 2^{10}, 2^{12}, 2^{14}\}$	16.8	67.1	268.4	1073.7
SWiM	128	2^8	**7.6(3.9)**	**25.9(16.1)**	**99.2(64.1)**	**392.1(256.4)**
		2^{10}	**13.7(3.9)**	**50.9(15.9)**	**199.7(64.1)**	**794.6(256.3)**
		2^{12}	36.7(3.7)	149.5(15.8)	600.2(63.8)	2403.1(256.1)
		2^{14}	105.2(2.9)	519.9(15.1)	2178.6(63.1)	8813.3(255.4)

Table 4: Bandwidth. calculation of communication (in MB) for wildcard pattern matching of text length n, pattern length m, binary alphabet. In SWiM, the online communication cost is presented in parenthesis. Compared to 5PM, best results marked in bold.

The step of forming the matrices in SWiM is relatively costly. We push it into the preprocessing phase, which will include creating OT matrices and performing the matrix transposition. Our experiments show that the offline phase takes $60 - 90\%$ of the total running time. For instance, with text size $n = 2^{22}$ and pattern size $m = 2^{14}$ our overall running time is 60.10 seconds with an offline phase of 53.64 seconds, a 89% of the overall cost.

We find that SWiM scales well in the experiments. For text size $n = 2^{16}$ and pattern size $m = 2^8$, our protocol takes only 0.21 seconds in which 0.04 seconds is for online time. When increasing the lengths to $n = 2^{24}$ and $m = 2^{12}$, we see that our protocol requires roughly 52 seconds in total.

When evaluating our implementation in the WAN setting, we still have a fast online phase due to the fact that OTs can be precomputed in the offline phase. For $n = 2^{24}$ and $m = 2^{12}$, we obtain an overall running time of 363.06 seconds and an online time of 50.15 seconds which contains only 13% of the total cost. For the small text and pattern, the protocol requires only a few seconds. With $n = 2^{16}$ and $m = 2^8$, our protocol takes an overall running time of 1.04 seconds with the online phase requiring 0.4 seconds.

Acknowledgments

The first author was supported by Office of Naval Research (ONR) contract number N00014-14-C-0113. The second and third authors were supported by NSF awards #1149647 and #1617197.

References

[ALSZ13] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More efficient oblivious transfer and extensions for faster secure computation. In *ACM CCS 13*, 2013.

[AWY15] Amir Abboud, Richard Ryan Williams, and Huacheng Yu. More applications of the polynomial method to algorithm design. In Piotr Indyk, editor, *26th SODA*, pages 218–230. ACM-SIAM, January 2015.

[BCP13] Julien Bringer, Hervé Chabanne, and Alain Patey. *SHADE: Secure HAmming DistancE Computation from Oblivious Transfer*. 2013.

[Bea95] Donald Beaver. Precomputing oblivious transfer. In *CRYPTO'95*, 1995.

[Bea96] Donald Beaver. Correlated pseudorandomness and the complexity of private computations. In *28th ACM STOC*, 1996.

[BEDM+12] Joshua Baron, Karim El Defrawy, Kirill Minkovich, Rafail Ostrovsky, and Eric Tressler. 5pm: Secure pattern matching. SCN'12, 2012.

[BGVV14] Philip Bille, Inge Li Gørtz, Hjalte Wedel Vildhøj, and Søren Vind. String indexing for patterns with wildcards. *Theory of Computing Systems*, 2014.

[BI14] Carl Barton and Costas S. Iliopoulos. On the average-case complexity of pattern matching with wildcards. *CoRR*, abs/1407.0950, 2014.

[BST01] Fabrice Boudot, Berry Schoenmakers, and Jacques Traoré. A fair and efficient solution to the socialist millionaires' problem. *Discrete Applied Mathematics*, 2001.

[ÇCL+17] Gizem S. Çetin, Hao Chen, Kim Laine, Kristin Lauter, Peter Rindal, and Yuhou Xia. Private queries on encrypted genomic data. *BMC Medical Genomics*, 2017.

[CEPR09] Raphaël Clifford, Klim Efremenko, Ely Porat, and Amir Rothschild. From coding theory to efficient pattern matching. In *20th SODA*, 2009.

[CH02] Richard Cole and Ramesh Hariharan. Verifying candidate matches in sparse and wildcard matching. In *34th ACM STOC*, 2002.

[CLI07] Simple deterministic wildcard matching. *Information Processing Letters*, 2007.

[CS15] Melissa Chase and Emily Shen. Substring-searchable symmetric encryption. *PoPETs*, 2015.

[CWZ+06] Gong Chen, Xindong Wu, Xingquan Zhu, Abdullah N. Arslan, and Yu He. Efficient string matching with wildcards and length constraints. *Knowledge and Information Systems*, 2006.

[DCFT13] Emiliano De Cristofaro, Sky Faber, and Gene Tsudik. Secure genomic testing with size- and position-hiding private substring matching. WPES '13, 2013.

[DF13] K. El Defrawy and S. Faber. Blindfolded data search via secure pattern matching. *Computer*, 2013.

[FHV13] Sebastian Faust, Carmit Hazay, and Daniele Venturi. Outsourced pattern matching. In *ICALP 2013, Part II*, 2013.

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

[FNW96] Ronald Fagin, Moni Naor, and Peter Winkler. Comparing information without leaking it. *Commun. ACM*, 1996.

[Fri09] Keith B. Frikken. *Practical Private DNA String Searching and Matching through Efficient Oblivious Automata Evaluation.* 2009.

[GHS10] Rosario Gennaro, Carmit Hazay, and Jeffrey S. Sorensen. Text search protocols with simulation based security. In *PKC 2010*, 2010.

[GLMY16] Adam Groce, Alex Ledger, Alex J. Malozemoff, and Arkady Yerukhimovich. CompGC: Efficient offline/online semi-honest two-party computation. Cryptology ePrint Archive, Report 2016/458, 2016.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In *19th ACM STOC*, 1987.

[HHD11] H. Hoffmann, M. D. Howard, and M. J. Daily. Fast pattern matching with time-delay neural networks. In *The 2011 International Joint Conference on Neural Networks*, pages 2424–2429, July 2011.

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

[HT10] Carmit Hazay and Tomas Toft. Computationally secure pattern matching in the presence of malicious adversaries. In *ASIACRYPT 2010*, 2010.

[HT14] Carmit Hazay and Tomas Toft. Computationally secure pattern matching in the presence of malicious adversaries. *Journal of Cryptology*, 2014.

[IKNP03] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In *CRYPTO 2003*, 2003.

[JKS08] Somesh Jha, Louis Kruger, and Vitaly Shmatikov. Towards practical privacy for genomic computation. In *2008 IEEE Symposium on Security and Privacy*, 2008.

[KK13] Vladimir Kolesnikov and Ranjit Kumaresan. Improved OT extension for transferring short secrets. In *CRYPTO 2013, Part II*, 2013.

[KKRT16] Vladimir Kolesnikov, Ranjit Kumaresan, Mike Rosulek, and Ni Trieu. Efficient batched oblivious PRF with applications to private set intersection. In *ACM CCS 16*, 2016.

[KM10] Jonathan Katz and Lior Malka. Secure text processing with applications to private DNA matching. In *ACM CCS 10*, 2010.

[KMP+17] Vladimir Kolesnikov, Naor Matania, Benny Pinkas, Mike Rosulek, and Ni Trieu. Practical multi-party private set intersection from symmetric-key techniques. In *ACM CCS 17*, 2017.

[KNR+17] Vladimir Kolesnikov, Jesper Buus Nielsen, Mike Rosulek, Ni Trieu, and Roberto Trifiletti. DUPLO: Unifying cut-and-choose for garbled circuits. In *ACM CCS 17*, 2017.

[KS08] Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free XOR gates and applications. In *ICALP 2008, Part II*, 2008.

[Lip03] Helger Lipmaa. Verifiable homomorphic oblivious transfer and private equality test. In *ASIACRYPT 2003*, 2003.

[MNPS04] Dahlia Malkhi, Noam Nisan, Benny Pinkas, and Yaron Sella. Fairplay—a secure two-party computation system. In *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13*, SSYM'04, 2004.

[MNSS12] Payman Mohassel, Salman Niksefat, Seyed Saeed Sadeghian, and Babak Sadeghiyan. An efficient protocol for oblivious DFA evaluation and applications. In *CT-RSA 2012*, 2012.

[NN10] Kedar Namjoshi and Girija Narlikar. Robust and fast pattern matching for intrusion detection. In *Proceedings of the 29th Conference on Information Communications*, INFOCOM'10, 2010.

[NP99] Moni Naor and Benny Pinkas. Oblivious transfer and polynomial evaluation. In *Proceedings of the Thirty-first Annual ACM Symposium on Theory of Computing*, STOC '99, 1999.

[OPJM10] Margarita Osadchy, Benny Pinkas, Ayman Jarrous, and Boaz Moskovich. SCiFI - a system for secure face identification. In *2010 IEEE Symposium on Security and Privacy*, 2010.

[PSZ14] Benny Pinkas, Thomas Schneider, and Michael Zohner. Faster private set intersection based on OT extension. In *23rd USENIX Security Symposium (USENIX Security 14)*, 2014.

[Rab05] Michael O. Rabin. How to exchange secrets with oblivious transfer. Cryptology ePrint Archive, Report 2005/187, 2005.

[Rin] Peter Rindal. libOTe: an efficient, portable, and easy to use Oblivious Transfer Library. https://github.com/osu-crypto/libOTe.

[SK17] Tushar Kanti Saha and Takeshi Koshiba. *An Enhancement of Privacy-Preserving Wildcards Pattern Matching*, pages 145–160. Springer International Publishing, Cham, 2017.

[SOF10] Pattern matching with don't cares and few errors. *Journal of Computer and System Sciences*, 2010.

[SSSS15] Riku Saikkonen, Seppo Sippu, and Eljas Soisalon-Soininen. *Experimental Analysis of an Online Dictionary Matching Algorithm for Regular Expressions with Gaps*. 2015.

[Tha11] Chris Thachuk. *Succincter Text Indexing with Wildcards*. 2011.

[TPKC07] Juan Ramón Troncoso-Pastoriza, Stefan Katzenbeisser, and Mehmet Celik. Privacy preserving error resilient dna searching through oblivious automata. In *ACM CCS 07*, 2007.

[Ver11] Damien Vergnaud. Efficient and secure generalized pattern matching via fast fourier transform. In *AFRICACRYPT 11*, 2011.

[WJW+15] D. Wang, X. Jia, C. Wang, K. Yang, S. Fu, and M. Xu. Generalized pattern matching string search on encrypted data in cloud systems. In *INFOCOM*, 2015.

[WMK16] Xiao Wang, Alex J. Malozemoff, and Jonathan Katz. EMP-toolkit: Efficient Multi-Party computation toolkit. https://github.com/emp-toolkit, 2016.

[WRK17] Xiao Wang, Samuel Ranellucci, and Jonathan Katz. Authenticated garbling and efficient maliciously secure two-party computation. In *ACM CCS 17*, 2017.

[WZX17] Xiaochao Wei, Minghao Zhao, and Qiuliang Xu. Efficient and secure outsourced approximate pattern matching protocol. *Soft Computing*, 2017.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th FOCS*, 1986.

[YSK+13] Masaya Yasuda, Takeshi Shimoyama, Jun Kogure, Kazuhiro Yokoyama, and Takeshi Koshiba. Secure pattern matching using somewhat homomorphic encryption. CCSW '13, 2013.

[YSK+14] Masaya Yasuda, Takeshi Shimoyama, Jun Kogure, Kazuhiro Yokoyama, and Takeshi Koshiba. *Privacy-Preserving Wildcards Pattern Matching Using Symmetric Somewhat Homomorphic Encryption*. 2014.

[ZRE15] Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole - reducing data transfer in garbled circuits using half gates. In *EUROCRYPT 2015, Part II*, 2015.