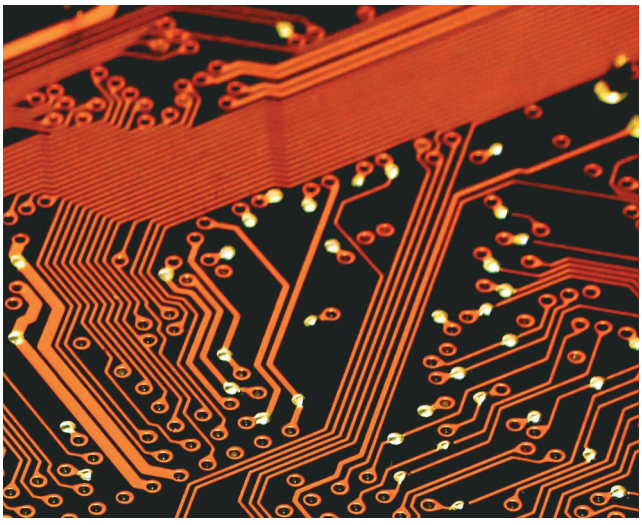

Blindfolded Data Search via Secure Pattern Matching

Karim El Defrawy, *HRL Laboratories*

Sky Faber, *University of California, Irvine*

Balancing security and privacy concerns with information sharing is a top priority for corporations, law enforcement agencies, governments, and other organizations. Secure pattern matching (SPM) addresses some of the challenges faced in sharing and searching private data.

As more organizations and users adopt software and computation as a service instead of building their own computation infrastructures, the need arises for secure solutions for sharing and searching private data. Secure computation protocols, which have attracted considerable attention, could provide a solution. Secure computation protocols generally allow two or more distrusting parties to collectively perform computation that requires inputs from those parties and delivers outputs to some, or all, of them.

Two properties must typically be guaranteed during these protocols' operation: correctness of the performed computation, and privacy of parties' inputs and outputs. This article presents a case for the use of secure pattern matching (SPM) in blindfolded data search and describes several new techniques based on SPM for securely searching data while guaranteeing both computational correctness and data privacy protection.

THE NEED FOR BLINDFOLDED DATA SEARCH

SPM allows an information requester to privately query an information provider's database, which retrieves the records matching the query pattern without revealing the query or its results to the provider. At the same time, the provider avoids revealing any information about its database other than the matched records. We describe two application domains in which SPM can be used to support privacy-preserving data search.

Intelligence and data sharing in law enforcement communities

In the past decade, the desire to share data with law enforcement; state, local, and international governments; and first responders has met resistance due to potential privacy issues. For example, in May 2006, the European Court of Justice ordered European airlines to stop sharing passenger records with the US Department of Homeland Security. Although the data requested was relatively innocuous, it took two years to implement a new agreement for sharing personal name records (PNRs), and some privacy advocates in Europe still oppose sharing data without stronger data privacy safeguards.[1] More difficult is sharing sensitive intelligence or law enforcement data collected from federal programs with the tens of thousands of state, local, and tribal governments within the US.

 Published by the IEEE Computer Society

Balancing security concerns with information sharing remains a top priority for intelligence and law enforcement agencies. An effective solution must simultaneously provide strong privacy protection and security while allowing access to the latest intelligence and law enforcement databases. SPM enables secure evaluation of an expressive set of queries to search such data. SPM solutions include exact matching, substring and approximate matching, and range queries for numerical data. Figure 1a illustrates how SPM can support blindfolded search of a passenger list for a variant of the name "Rob."

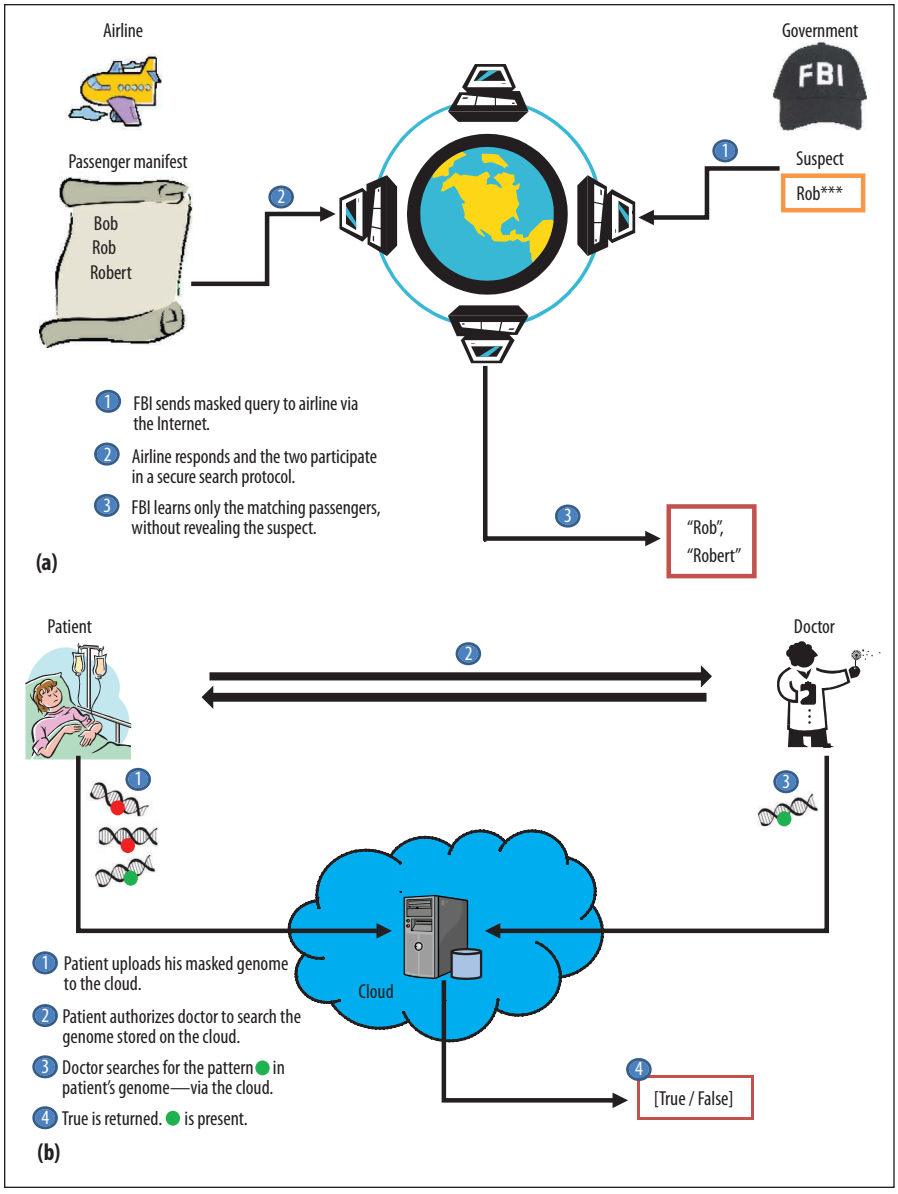

Figure 1. Examples of blindfolded data search. (a) The US Federal Bureau of Investigation privately examines a passenger list for a suspect with a variant of the name "Rob." (b) A patient visits a doctor who privately examines his genome for treatment.

Medical and health data sharing

To implement the data sharing requirements in the Health Insurance Portability and Accountability Act of 1996 (HIPAA), the US Department of Health & Human Services established the "Standards for Privacy of Individually Identifiable Health Information." These standards address the use and disclosure of individuals' health information by heathcare providers, as well as standards for individuals' rights to understand and control how their health information is used. Such standards aim to ensure that individuals' health information is properly protected while allowing the flow of health information needed to provide and promote high-quality healthcare.

SPM can be used to protect such personal health information while allowing doctors, hospitals, and other health organizations to use it in medical care. Consider, as an example, secure and private search of personal DNA sequences.[2] Preventing leakage of an individual's DNA sequencing data is obviously important; among other things, such information might indicate pre-exposure to certain health risks that could cause insurance providers to deny coverage or increase premiums. Figure 1b shows how a doctor might use blindfolded data search to test a patient's genome for a specific disease-causing mutation.

BACKGROUND

Pattern matching is fundamental to computer science. Application areas include database search, networking, security, and, recently, bioinformatics and DNA analysis.[2] Pattern-matching research has resulted in several efficient algorithms and approaches to solving the insecure variants thereof, none of which guarantee the secrecy of the input.[3,4] Nonetheless, no single algorithm performs well in all cases, is sufficiently fast, uses memory efficiently, and can handle exact matches, approximate matches, and wildcards.

The pattern-matching problem

The most common interpretation of the pattern-matching problem is, given an alphabet Σ and a text $T \in \Sigma^n$, the exact *pattern-matching decision problem* requires the user to decide if a pattern appears at all in the string.

For the exact pattern-matching search problem, the requirement is to find all indices i of T (if any) where P occurs as a substring. If $\overline{T}_i$ denotes the m-character substring of T starting at position i, the output should be the set $\{i | \overline{T}_i = P\}$. Throughout the article, we use DNA as our alphabet—that is, $\Sigma = \{A, C, G, T\}$—to simplify illustrations. Several generalizations of the exact matching problem exist.

In *single-character wildcard pattern matching*, a special character, $* \notin \Sigma$, represents a wildcard that is not part of the alphabet and could match any single character of the alphabet. This special character might repeat several times in a pattern P. The output in this case should also be the set of indices $\{i | T_i = P\}$. Using such a wildcard character lets users specify one pattern that could match several sequences of characters. For example, the pattern *TA** would match any of the following words in a text: *TAA*, *TAC*, *TAG*, and *TAT*.

In *substring pattern matching* (sometimes called approximate or threshold pattern matching), a parameter $k \leq m$ is fixed. A match for P is found whenever there exists in T an m-length string that differs in $m - k$ characters from P (that is, has Hamming distance $m - k$ from P). For example, for the pattern *TAC*, $m = 3$. If $k = 2$, any of the following words would match (* indicates that any character from the alphabet could be substituted): **AC*, *T*C*, or *TA**. Substring matching will also capture results from single-character wildcard matching—for example, a substring matching with the pattern *TAC* and $k = 2$ will also capture all occurrences of *TA**.

Security and adversary models

A protocol that performs SPM on a text T held by a server and using a pattern P held by a client can be defined as an *interactive two-party protocol*. Such protocols typically assume that both parties are probabilistic polynomial time algorithms. The view of a party in this interactive protocol is defined as its private input and its private randomness as well as the protocol transcript.

SPM's informal security requirements dictate that the party holding the text learns only the upper bound on the pattern length, while the party holding the pattern receives a binary (yes/no) answer for the decision problem or the matching positions (if any), but nothing else.

Cryptographers typically demonstrate the security of secure computation protocols, and thus SPM, using the *ideal/real-world simulation paradigm*. To prove security using this paradigm, one imagines what properties a protocol should have in an ideal world. In this ideal world, all parties send their input to a trusted third party (also called ideal functionality) that computes the functionality in question and returns the output to each party. The trusted third party is assumed to be incorruptible, so by definition the ideal world is secure. An adversary cannot influence the inputs of any parties other than the ones it corrupts, nor can it learn the outputs of any other parties. A protocol is considered secure if its views are similar to those in the corresponding ideal world.

Protocol designers must consider several adversary models when constructing secure protocols using the ideal/real-world simulation paradigm.

Honest-but-curious (HBC) adversaries do not deviate from the steps prescribed by the protocol specifications. Such adversaries might try to violate the other party's privacy by storing exchanged messages, snooping on messages, or trying to analyze whatever data they receive to infer more information than they are supposed to. Such adversaries are often called passive or semi-honest adversaries, because they passively listen to messages and try to violate privacy and confidentiality but do not actively modify, fabricate, or delete messages to launch active attacks.

Malicious adversaries launch different types of passive and active attacks but do not fear being caught cheating. They might deviate arbitrarily from the prescribed protocol and do not necessarily perform their computations correctly. They might send garbage or inconsistent data, and even replay messages. Malicious adversaries typically are not stealthy—that is, they are not trying to hide the fact that they cheat. Protecting against malicious adversaries provides a strong level of security, but it usually requires using expensive cryptographic primitives such as zero-knowledge proofs and commitment schemes.

Other models capture adversaries that are more powerful than HBC but less powerful than malicious adversaries. Covert adversaries, for example, might deviate arbitrarily from the protocol specification in an attempt to cheat, but they do not wish to be caught.[5]

SPM PROTOCOLS

Table 1 compares the primary features of several recently developed SPM protocols. Recall that the secure exact pattern-matching setting consists of a server holding text $T \in \Sigma^n$ and a client holding $P \in \Sigma^m$. We let $\overline{T}_i$ represent the m-length substring rooted at position i of the text. Further, the wildcard pattern-matching problem allows an additional wildcard character, denoted as $*$.

Integer comparison-based pattern-matching protocols

Carmit Hazay and Tomas Toft present protocols that are secure against malicious adversaries and can compute variants of the pattern-matching problem.[6] Their protocols' security is based on the security of (additive) ElGamal

Table 1. Comparison of secure pattern-matching (SPM) protocols.

Protocol	Bandwidth		Total computation complexity		Rounds	Wildcard and substring matching	Security models
	Client	Server	Client	Server			
Integer comparison-based pattern matching	$O(mnk^2)$	$O(mnk^2)$	$O(mnk^2)\log^2 k)$	$O(mnk \log k)$	$O(1)$	Yes	Malicious models/honest-but-curious model
Fast Fourier transform (FFT)-based pattern matching	$O((n+m)k^2)$	$O((n+m)k^2)$	$O(mnk^2)\log^2 k)$	$O(mnk \log k)$	$O(1)$	Yes	Malicious models/honest-but-curious model
Matrix multiplication-based pattern matching	$O((n+m)k^2)$	$O((n+m)k^2)$	$O(mnk^2)\log^2 k)$	$O(mnk \log k)$	$O(1)$	Yes	Malicious models/honest-but-curious model
Garbled circuit-based text processing	$O(mk^2)$	$O((n+m)k^2)$	$O(n+mk\log^2 k)$	$O((n+m)k\log^2 k)$	$O(1)$	No	One-sided simulation/ honest-but-curious model

Note: the comparison assumes that the protocols requiring public-key operations use ElGamal encryption with a k-bit security parameter.

encryption, a well-known and vetted cryptosystem. Hazay and Toft offer three protocols, each solving a specific variant of the pattern-matching problem.

The first two protocols use a function $h(x)$ to convert the pattern, and each of the m-length substrings of text, into an encryption of a single integer. They then compare these integers obliviously. Each protocol can support alphabets of any size, and without loss of generality we assume that the alphabet is a subset of the integers. Thus, each substring is encoded as a base $|\Sigma|$ integer. That is,

$$h(x) = \sum_{i=0}^{m} x_i * |\Sigma|^i .$$

For exact matching, the client encodes the pattern as

$$P' = E\left(\sum_{i=1}^{m} P_i * |\Sigma|^{i-1}\right)$$

and the server computes for each $\bar{T}_i$:

$$\bar{T}_i' = E\left(\sum_{j=1}^{m-1} T_{i+j} * |\Sigma|^j\right) .$$

Both protocols then compare P to each $\bar{T}_i$ to identify a match. They can make this comparison using the homomorphic properties of (additive) ElGamal encryption. For all i, the two parties compute $(P')^{-1} * \bar{T}_i'$, which is an encryption of 0 if and only if the two are equal. They then jointly check, in zero knowledge, if this is indeed an encryption of 0, and thus a matching position in T. Figure 2a illustrates this process. To ensure correctness under a malicious adversary, both parties perform all computations. Further, the parties use zero-knowledge proofs to verify the correctness of each other's inputs and computation. The exact matching protocol requires $O(n + m)$ cryptographic computation and communication.

Hazay and Toft also present a slightly modified protocol to handle wildcard matching.[6] Wildcards in the pattern will be selectively removed before they are encoded as in the previous two protocols. The client replaces wildcard characters (*) in the pattern with 0, and sends an m-length vector of encrypted bits—the selection vector—to "select" the *nonwildcard characters* from each $\bar{T}_i$. Thus, the client sends

$$sv_i = \begin{cases} E(0) \text{ if } P_i = * \\ E(1) \text{ otherwise} \end{cases} .$$

The server verifies this vector and returns the encrypted text T. The client then computes, verifies, and randomizes for the ith position of the jth $\bar{T}_j$:

$$\hat{T}_{i,j} = T_{j+i-1}^{sv_i} .$$

Again, due to the homomorphism, any character paired against a wildcard in the pattern is an encryption of 0, as is the wildcard. With wildcards removed, the protocol continues as before, computing

$$P' = E\left(\sum_{i=1}^{m} P_i * |\Sigma|^{i-1}\right)$$

and

$$\bar{T}_i = E\left(\sum_{j=0}^{m-1} \hat{T}_{i+j} * |\Sigma|^j\right) .$$

Figure 2b illustrates this process on a small DNA string.

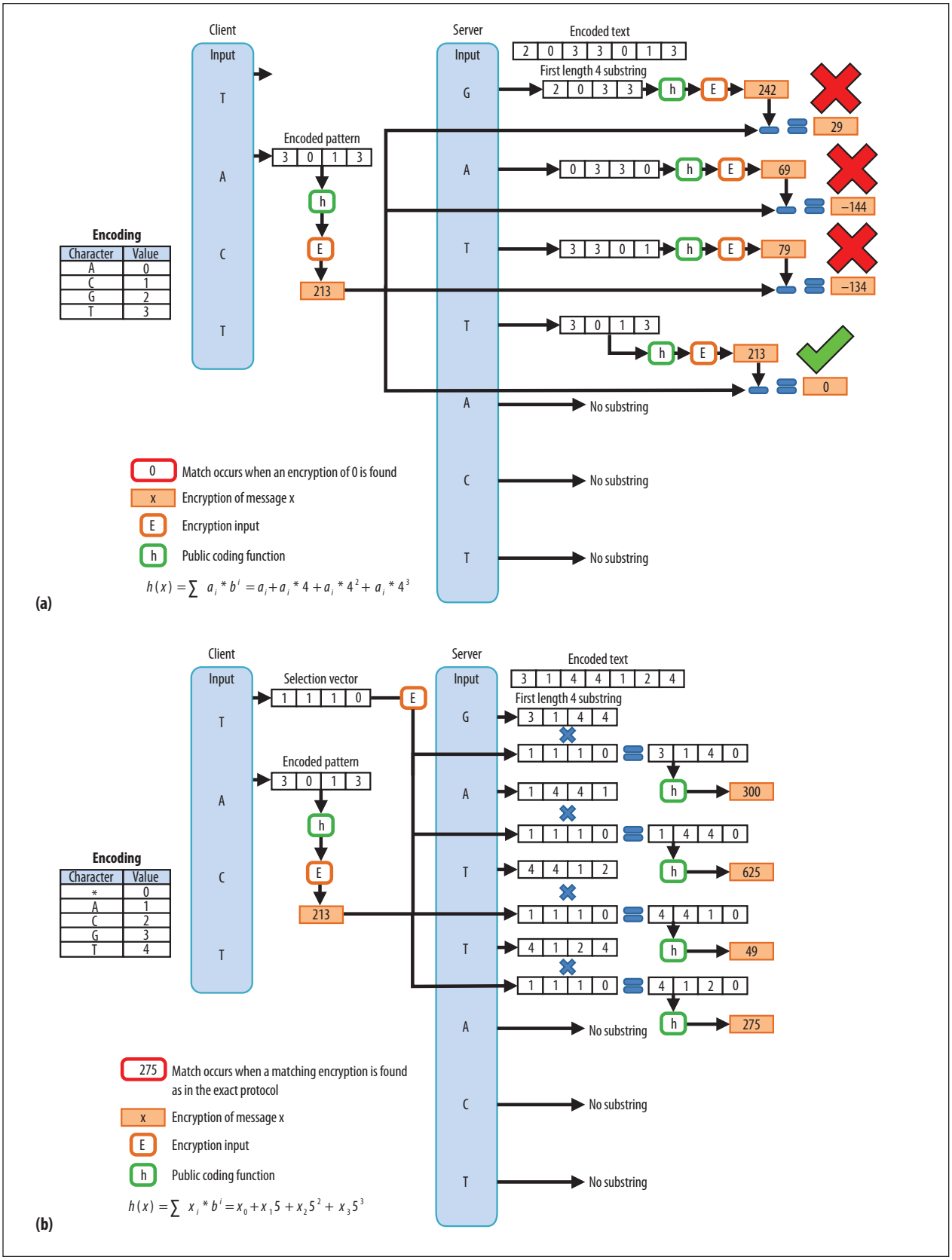

Figure 2. Secure pattern matching (SPM) of client queries using (a) Carmit Hazay and Tomas Toft's exact matching protocol (TACT) and (b) wildcard matching (TAC*). The server's text remains "GATTACT" with a single match in both cases.

For every possible matching position in the text, the protocol performs $O(m)$ cryptographic operations to remove the wildcards. This results in an overall complexity of $O(nm)$ communication and computation.

The final protocol securely computes the secure approximate matching problem, for bit strings only, in $O(nm)$ computation and communication. This limitation can severely affect its practical applications (a pattern in an encoded binary string does not necessarily correspond to a pattern in the original text). The protocol relies on securely computing the Hamming distance of P and each $\overline{T_i}$. A match is found if this distance is above a specified threshold τ. As before, the server sends the encrypted text as $T_i = E(t_i)$. The client computes and sends for the ith position of the jth substring:

$$\prod_{i,j} = T_{j+i=1}^{P_i}.$$

Effectively, this is an encryption of $P_i \wedge T_{j+i-1}$. They both then compute $X_{i,j} = T_{j+i-1} * P_i * 2\Pi_{i,j}$, which is the encryption of $P_i \oplus T_{j+i-1}$, or the Hamming distance in binary.

Fast Fourier transform-based protocols

Damien Vergnaud solved the pattern-matching problem using the fast Fourier transform (FFT).[7] He showed that you can use an FFT algorithm to quickly compute sums of products of entries in a vector. Vergnaud derived protocols for exact, approximate, and wildcard matching.[7] His wildcard-matching protocol runs with $O(n\log(m))$ exponentiations, $O(nm)$ multiplications, and $O(n)$ communication. It first maps wildcards in the text and pattern to 0 and every other character to a positive integer, as in the integer comparison-based protocols. Next, the protocol solves for the sum of squared error between the pattern and each possible matching substring of the text.

Both parties securely compute $\forall_i \in [n - m + 1]$:

$$\sum_{j=1}^{m} P_j * T_{i+j-1}\left(P_j - T_{i+j-1}\right)^2 =$$

$$\sum_{j=1}^{m} P_j^3 * T_{i+j} - 2P_j^2 * T_{i+j-1}^2 + P_j * T_{i+j-1}^3.$$

Note that if P_j or T_{j+i-1} is a wildcard, the jth term in the sum will always be 0. Similarly, if $P_j = T_{j+i-1}$, the jth term is also 0. Thus the entire sum evaluates to 0 if and only if the pattern matches the ith substring—that is, the error between strings is 0.

We can compute all of these sums securely in $O(n \log m)$ using a protocol that securely computes polynomial multiplication via convolution (FFT). To demonstrate this technique, we look at the first term in the sum $P_j^3 * t_{i+j-1}$. We treat each P_i^3 as a coefficient of an m-degree polynomial $A(x)$. Similarly, we let t_i be the coefficient of an n-degree polynomial $B(x)$. If we can quickly determine the coefficients of $A(x)B(x)$, we can calculate $P_j^3 * T_j \forall i,\ j$. Using these values, we easily compute the appropriate terms in the sum. We can compute the other two terms similarly. Vergnaud gives an in-depth explanation of polynomial multiplication using FFT and describes a protocol that is secure in the presence of malicious adversaries.[7] This protocol also relies on the security of (additive) ElGamal encryption.

Matrix multiplication-based pattern-matching protocols

Joshua Baron and his colleagues construct a two-party protocol, 5PM, that can securely calculate the exact, wildcard, and approximate pattern-matching problems for any alphabet.[8] The protocol is secure against a malicious adversary provided that the underlying homomorphic encryption scheme is semantically secure. The crux of the protocol is a reduction from pattern matching to a non-FFT-based matrix multiplication. The algorithm requires $O(nm)$ cryptographic operations and $O(n + m)$ communication. This is the only protocol described here that has an accompanying implementation written in C++.

The 5PM protocol first preprocesses the pattern to obtain a character delay vector (CDV) for each letter in the alphabet. It uses these CDVs to compute, for each character in the text, where a matching pattern might end. A counter at this computed location is incremented. If a match occurs in the text, all m characters in the matching $\overline{T_i}$ would have caused the counter to increment. Thus, if at the end of the protocol any counter equals m, a match exists.

Without cryptographic machinery to guarantee security, the algorithm is as follows. For each character in the alphabet, the algorithm creates and initializes an m-length CDV to all 0s. Next, for each character P_i in the pattern, a 1 is set in the corresponding CDV, indicating a possible ending $m - i$ characters away. That is, i $CDV(P_i)[m - i] = 1$. Observe that for all CDVs and all i, the ith position is 1 in exactly one CDV. Next, the activation vector (AV), a length n array of counters, is created and initialized to 0 element wise. The activation vector is then computed as

$$\forall i \in [1..n], j \in [1..m]\ \ AV[i + j] = CDV(T_i)[j] + AV[i + j].$$

Finally, we output i if $AV[i] = m$. Figure 3 shows CDVs for the pattern *TACT*. We can easily extend the algorithm to handle wildcard pattern matching. Upon CDV creation, wildcard characters are simply skipped, and no corresponding CDV is created. Further, in the final matching phase, we check $AV[i]$ against $m - k$, where k is the number of wildcards in the pattern.

We can compute the same algorithm using a combination of three linear operators, equivalent to matrix multiplication. Briefly, $ColSum(M)$ computes the sum of each column and stores the result in a length n vector.

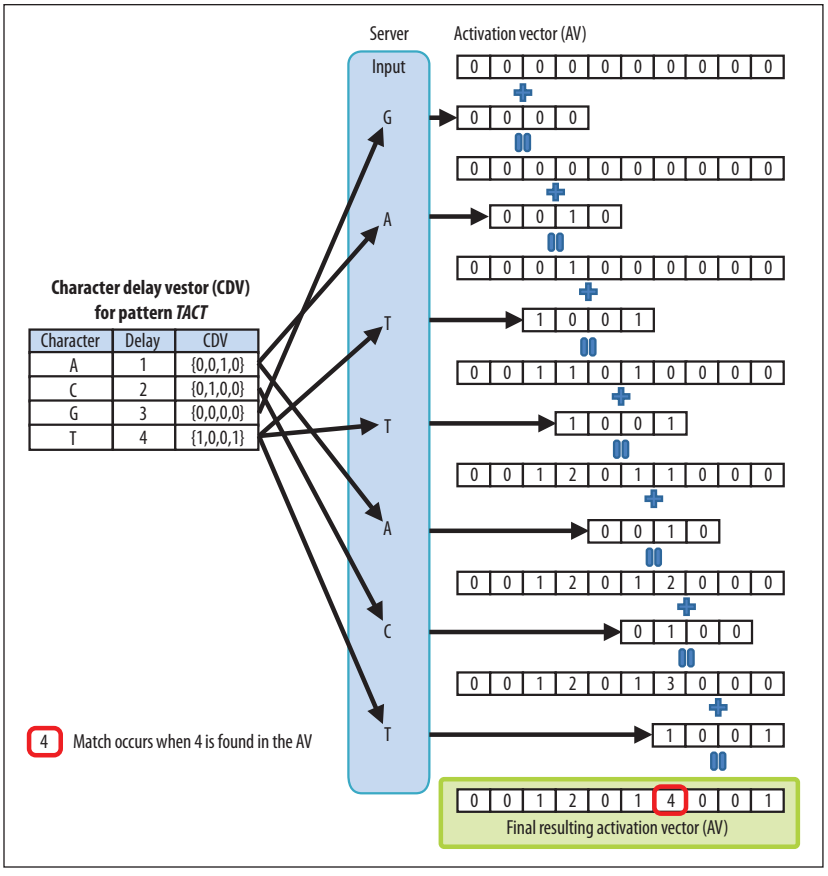

Figure 3. Character delay vectors (CDVs) for the pattern *TACT* and their application to the activation vector (AV) during processing.

Cut(*M*) removes both the left *m* columns and the right *m* columns. *Stretch*(*M*) shifts the *i*th row of *M* right *i* columns, and places 0 in empty locations. The resulting matrix has an extra *n* columns, as Figure 4 illustrates. A more detailed explanation of these operations is available elsewhere.[8]

To use these operators, we first transform the text into a binary $n \times |\Sigma|$ matrix $\mathbf{M}_t$ such that $\mathbf{M}_t(i, T_i) = 1$. Essentially, each row in $\mathbf{M}_t$ encodes a 1 in the location corresponding to the correct letter in Σ. We also encode the CDVs as a matrix by concatenating each CDV together. We then calculate the matrix $\mathbf{M}_{t(CDV)}$ corresponding to a copy of the appropriate CDV in row *i* for each text position T_i. This is a simple multiplication problem, $\mathbf{M}_{t(CDV)} = \mathbf{M}_t * \mathbf{M}_{CDV}$. We then stretch $\mathbf{M}_{t(CDV)}$ to center the *i*th *CDV* at position (*i*, *i*). Finally, we compute the activation vector by cutting the extraneous rows where a match cannot occur and summing the columns to calculate the counter at each position *i*. In other words, $AV = ColSum(Cut(Stretch(\mathbf{M}_{t(CDV)})))$. As before, *i* is returned if $AV[i] = m$.

Using this construction, creating a secure protocol is straightforward. The server computes and sends an encryption of its text matrix $\mathbf{M}_t$, and the client computes and sends an encryption of the CDV matrix $\boldsymbol{M}_{CDV}$. Using an additive homomorphic encryption scheme, both the server and the client compute an encryption of the activation vector as $E(AV) = \text{ColSum}(\text{Cut}(\text{Stretch}(\mathbf{M}_{t(CDV)})))$, which, due to linearity of matrix multiplication, is computable if either $\mathbf{M}_t$ or $\mathbf{M}_{CDV}$ is encrypted. The two parties then

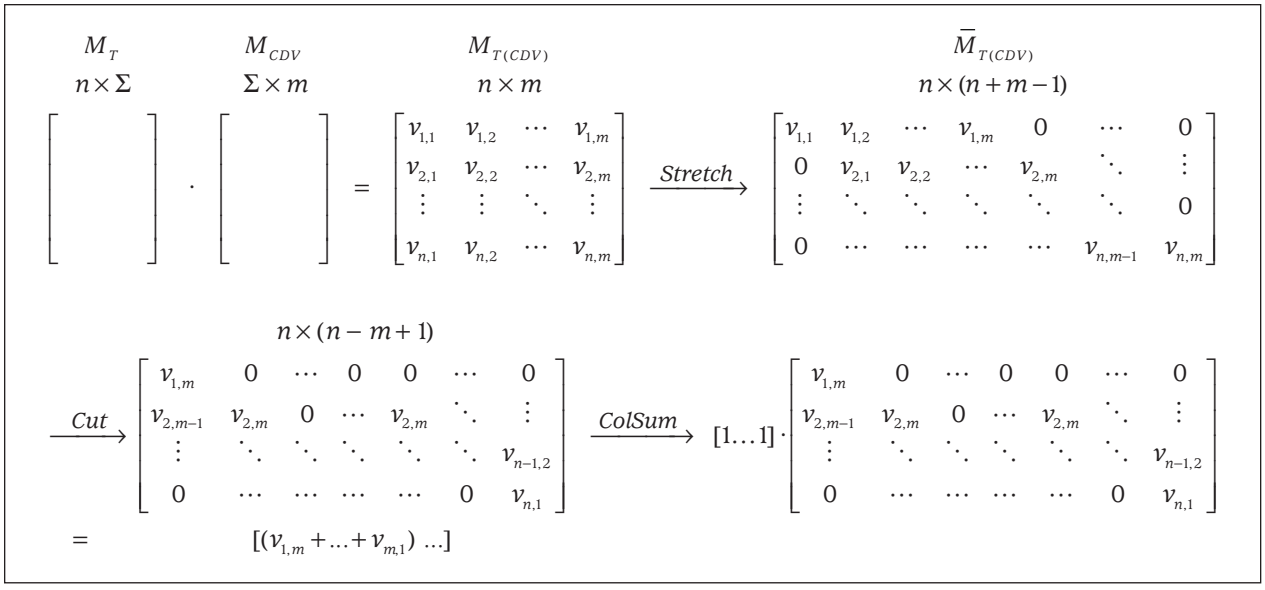

Figure 4. Insecure pattern matching expressed as linear matrix and vector operations.[8]

subtract m from each element in $E(AV)$ and jointly test for an encryption of 0, indicating a match. Baron and his colleagues include variants of this idea secure against HBC and malicious adversaries.[8]

Garbled circuit-based text-processing protocols

Some applications might require additional text processing after identifying a match. In this setting, the server and client will perform a secure protocol in which the client learns a function $f(P, T_i, i)$ for indices i, where the pattern matches $P = \overline{T}_i$. Jonathan Katz and Lior Malka[2] developed a protocol to securely compute this functionality by relying on two cryptographic primitives: keyword search and garbled circuits.[9] Keyword search lets a client query for matching keywords from a server's database, returning an arbitrary payload when there is a match. Garbled circuits allow for arbitrary secure function evaluation via Boolean circuits but require computation linear in the number of gates in the circuit.

Katz and Malka use keyword search to solve the exact pattern-matching problem, and garbled circuits to compute the generic functionality f on the matched results. When solving the exact pattern-matching problem, they require the pattern P to be a query and each $\overline{T}_i$ to be a keyword. By customizing the payload of the servers' keywords, they can transmit information allowing the client to securely compute $f(\overline{T}_i, i, P)$. Various information could be sent for specific functions; however, Katz and Malka observe that, in most cases, f is small enough to be easily computed using Yao's garbled circuits.[9] First, the server transmits u garbled circuits to the client, u being an upper bound on the number of matches of P in T. Next, the server sets the payload for each keyword $\overline{T}_i$ to the keys corresponding to its input of $f(\overline{T}_i; i)$. In this way, the client can add its input, P, and securely compute f for each match.

The protocol's novelty is its ability to quickly compute a large class of text-processing functions while remaining efficient. The protocol runs in $O(n + m)$ exponentiations for the keyword search phase, and with $O(u)$ gates. To secure circuits against HBC adversaries, each gate requires several symmetric key operations.

Still, while powerful, the protocol has limitations. Primarily, it does not provide full security against malicious adversaries. In addition, it cannot handle substring or wildcard pattern matching (although inefficient extensions exist). In addition, for efficient preprocessing of most circuits, the server must know the value of m before meeting with a client.

Despite the recent focus on SPM protocols, several problems remain. First, of all the SPM protocols, only 5PM has been implemented, so we know little about their concrete timing and memory requirements. Such analysis is critical in assessing these protocols' applicability to real-life scenarios. Second, most SPM protocols are designed for either the HBC adversary model or the malicious adversary model. To the best of our knowledge, no SPM protocols target other adversary models, such as covert adversaries. Third, given that multimedia and images constitute a significant fraction of today's (and future) data, it is natural to ask whether we can devise more efficient SPM protocols for multidimensional data (for example, biological data such as proteins that might require matching in multiple dimensions) instead of adopting protocols designed for 1D data. Finally, with the recent emphasis on streaming and big data, secure generic pattern matching must operate efficiently on large amounts of streaming data with constant memory and reasonable processing and timing constraints. This area remains largely unexplored.

References

1. W. J. Krouse and B. Elias, *Terrorist Watchlist Checks and Air Passenger Prescreening*, Congressional Research Service, 30 Dec. 2009; www.fas.org/sgp/crs/homesec/RL33645.pdf.
2. J. Katz and L. Malka, "Secure Text Processing with Applications to Private DNA Matching," *Proc. 17th ACM Conf. Computer and Comm. Security* (CCS 10), ACM, 2010, pp. 485-492.
3. T.-H. Tsai, "Average Case Analysis of the Boyer-Moore Algorithm," *Random Structures and Algorithms*, July 2006, pp. 481-498.
4. R.M. Karp and M.O. Rabin, "Efficient Randomized Pattern-Matching Algorithms," *IBM J. Research and Development*, vol. 31, Mar. 1987, pp. 249-260.
5. Y. Aumann and Y. Lindell, "Security against Covert Adversaries: Efficient Protocols for Realistic Adversaries," *Proc. Conf. Theory of Cryptography* (TCC 07), Springer, 2007, pp. 137-156.
6. C. Hazay and T. Toft, "Computationally Secure Pattern Matching in the Presence of Malicious Adversaries," *ASIACRYPT*, LNCS 6744, Springer, 2010, pp. 195-212.
7. D. Vergnaud, "Efficient and Secure Generalized Pattern Matching via Fast Fourier Transform," *Progress in Cryptology AFRICACRYPT 2011*, A. Nitaj and D. Pointcheval, eds., LNCS 6737, Springer, 2011, pp. 41-58.
8. J. Baron et al., "5PM: Secure Pattern Matching," *Security and Cryptography for Networks*, I. Visconti and R. Prisco, eds., LNCS 7485, Springer, 2012, pp. 222-240.
9. A.C. Yao, "Protocols for Secure Computations," *Proc. Ann. Symp. Foundations of Computer Science* (SFCS 82), IEEE CS, 1982, pp. 160-164.

Karim El Defrawy *is a research staff member in the Information Sciences and Systems Lab (ISSL) at HRL Laboratories, where he leads a project on secure computation. His research interests include secure computation, applied cryptography, and security and privacy in wired and wireless networks. El Defrawy received a PhD in networked systems from the University of California, Irvine. Contact him at kmeldefrawy@hrl.com.*

Sky Faber *is a graduate student at the University of California, Irvine. His research focuses on efficient privacy-preserving protocols, as well as network security. Faber received a BS in computer science from the University of Washington. Contact him at fabers@uci.edu.*