The Mastermind Attack on Genomic Data

Michael T. Goodrich
Dept. of Computer Science
Secure Computing and Networking Center
University of California, Irvine
http://www.ics.uci.edu/~goodrich/

Abstract

In this paper, we study the degree to which a genomic string, Q, leaks details about itself any time it engages in comparison protocols with a genomic querier, Bob, even if those protocols are cryptographically guaranteed to produce no additional information other than the scores that assess the degree to which Q matches strings offered by Bob. We show that such scenarios allow Bob to play variants of the game of Mastermind with Q so as to learn the complete identity of Q. We show that there are a number of efficient implementations for Bob to employ in these Mastermind attacks, depending on knowledge he has about the structure of Q, which show how quickly he can determine Q. Indeed, we show that Bob can discover Q using a number of rounds of test comparisons that is much smaller than the length of Q, under various assumptions regarding the types of scores that are returned by the cryptographic protocols and whether he can use knowledge about the distribution that Q comes from, e.g., using public knowledge about the properties of human DNA. We also provide the results of an experimental study we performed on a database of mitochondrial DNA, showing the vulnerability of existing real-world DNA data to the Mastermind attack.

Keywords: *mitochondrial DNA, genomic databases, privacy, mastermind, attacks.*

1. Introduction

As high-throughput genome sequencing technologies continue to improve, genome sequence data continue to accumulate at an exponential pace. Not only do scientists already have the genome sequence of thousands of viruses and bacteria and dozens of multicellular organisms from plants to humans, but we are rapidly approaching the day when sequencing individual diploid human genomes will be economically affordable. The milestone of the first human genome sequence draft in 2001 [12], [39] has recently been followed by the first diploid human genome sequence [26]. Moreover, a project to fully sequence 1,000 human genomes in the next few years is already under way [24], and the race for the capability to sequence an individual human genome for less than $1,000 within a few years is on [33]. In addition, the Personal Genome Project [10] is currently directed at the publication of the complete genomes and medical records of several volunteers. In the foreseeable future, it is anticipated that millions of individuals could have their diploid genome fully sequenced in the United States alone, and many more than that are likely to have partial DNA information sequenced, with this data tied to deeply personal information about these individuals. These coming scenarios obviously raise a number of serious privacy issues that are going to arise when many individuals have their complete genomes stored in various genomic databases.

In fact, these privacy issues are already here with respect to one type of human DNA—mitochondrial DNA. We are already at the point where hundreds of thousands of people have had their mitochondrial DNA (mtDNA) sequenced [5], [29], which is typically about 16,500 base pairs (bp) long, whereas the entire diploid human genome is roughly 6 billion bp long. Interestingly, mtDNA is transferred only along the maternal line; hence, scientists have used differences from a reference mtDNA sequence as a way to plot human migration from the earliest days of the modern human species. (See Figure 1.)

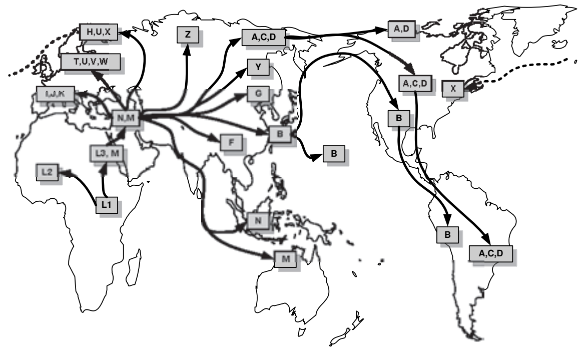

Figure 1. A confluent illustration [14] of the pattern of human migration implied by mtDNA mutations [5], [29]. Each letter stands for a major human mitochondrial haplogroup, that is, a canonical set of genetic mutations from a common ancestor.

Because of this knowledge of migration patterns and its correlation to known mtDNA mutations, given someone's mtDNA sequence, it is possible to trace their maternal ancestry back to individual villages [5], just by identifying differences in their mtDNA to a reference sequence, rCRS (see Figure 2). In other words, mtDNA alone is sufficient to determine a person's ethnic background with incredible accuracy.

```
GATCACAGGTCTATCACCCTATTAA
CCACTCACGGGAGCTCTCCATGCAT
TTGGTATTTTCGTCTGGGGGGTATG
CACGCGATAGCATTGCGAGACGCTG
GAGCCGGAGCACCCTATGTCGCAGT
ATCTGTCTTTGATTCCTGCCTCATC
...
ATCTGGTTCCTACTTCAGGGTCATA
AAGCCTAAATAGCCCACACGTTCCC
CTTAAATAAGACATCACGATG
```

Figure 2. A portion of the Revised Cambridge Reference Sequence, rCRS (GenBank accession number: AC 000021), which is 16,568 bp long.

In addition to ethnicity, there are, of course, other privacy concerns with respect to genomic data, including sensitive information related to disease susceptibility, and possible genetic influences on sexual orientation, personality, addiction, and intelligence. Concerns that employers or insurers will use genetic information to screen those at high risk for a disease already loom large and are widespread in the press. Indeed, the U.S. government and several states have already created laws dealing with DNA data access, and many more are considering such legislation. Thus, there is a need for technologies that can safeguard the privacy and security of genomic data.

Fortunately, several researchers have started exploring privacy-preserving data querying methods that can be applied to genomic sequences (e.g., see [2], [16], [17]). That is, cryptographic techniques can be used to allow for queries to be performed in a way that answers the specific question—such as a score rating the quality of a query for DNA matching or sequence alignment—but does not reveal any other information about the data, such as race or disease risk of the individual whose DNA is being queried.

The purpose of this paper is to show that, while being sufficient for single-shot comparisons of DNA sequences, such cryptographic techniques have a weakness when they are employed repeatedly. Specifically, we explore in this paper an attack on genomic data, which we call the *Mastermind attack*, because of its similarity with the game, Mastermind [11], [25]. This attack allows a genomic querier, Bob, to iteratively discover the full identity of a genomic query sequence, Q, with surprising efficiency, even if each comparison of Q with Bob's sequences are done using cryptographic privacy-preserving protocols. It is not surprising that iterated privacy-preserving string comparisons leak information about the strings being compared; what is surprising is how quickly the Mastermind attack can work, especially on genomic data.

1.1. Mastermind

Mastermind [11], [25] is a game played between two players—a *codemaker* and a *codebreaker*—using colored pegs. (See Figure 3.)

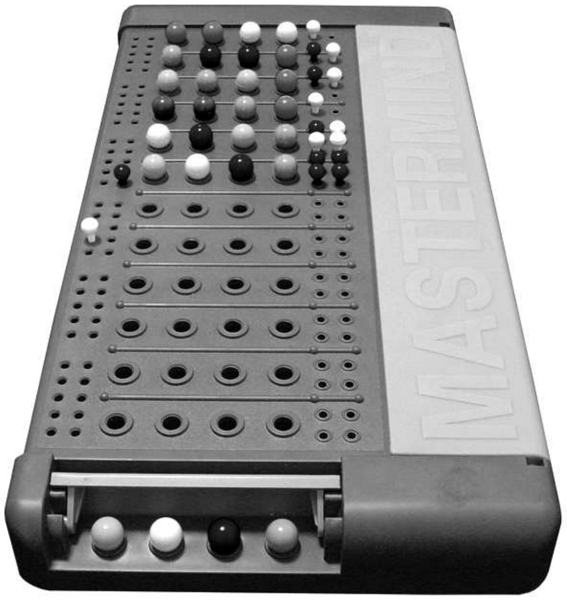

Figure 3. The Mastermind game. The four large pegs in the middle are used for guessing. The four smaller peg locations on the right are used to score each guess—with black-peg and white-peg scores. And the two pegs on the left are used to keep score across multiple games. (This image is adapted from http://en.wikipedia.org/wiki/File:Mastermind.jpg, by User:ZeroOne, under the Creative Commons Attribution ShareAlike 2.0 License.)

Viewed mathematically, Mastermind is abstracted as a game where the codemaker selects a plaintext vector, V, of length N, whose elements are selected from an alphabet of size K. For consistency with the board game, the members of this alphabet are often referred to as "colors." The codemaker and codebreaker both know the values of N and K, and play consists of the codebreaker repeatedly making guesses, $V_1, V_2, \ldots$, about the identity of V. For each guess, V_i the codemaker provides a score on how well V_i matches V. In *double-count* Mastermind, which is the standard version based on the board game, this score consists of a pair of two numbers:

- A *black* count, $b(V_i)$, which is the number of elements in V_i and V that match in both value and location. That is,
$$b(V_i) = |\{j \colon V_i[j] = V[j]\}|.$$
- A *white* count, $w(V_i)$, which is the number of elements in V_i that appear in V but in different locations than their locations in V_i. That is, letting π denote an arbitrary permutation,
$$w(V_i) = \max_{\pi} |\{j \colon V_i[\pi(j)] = V[j]\}| - b(V_i).$$

In *single-count* Mastermind, which has been less studied, the codebreaker is given only the black count, $b(V_i)$, for each guess, V_i. (Note that it is impossible to solve the problem given only white-count scores.) The goal is for the codebreaker to discover V using a small a number of guesses.

1.2. Previous Related Work

The original Mastermind game was invented in 1970 by Meirowitz, as a board game having holes for sequences of length $N = 4$ and $K = 6$ colored pegs. Knuth [25] subsequently showed that this instance of the Mastermind game can be solved in five guesses or less. Chvátal [11] studied the combinatorics of general Mastermind, showing that it can be solved in polynomial time, in the $K \geq N$ case, using $2N\lceil \log K\rceil + 4N$ guesses, and Chen *et al.* [9] showed how this bound can be improved, in this case, to $2N\lceil \log N\rceil + 2N + \lceil K/N\rceil + 2$ guesses. Stuckman and Zhang [34] showed that is NP-complete to determine if a sequence of guesses and responses in general double-count Mastermind is satisfiable. Goodrich [20] shows that single-count (black-peg) Mastermind satisfiability is NP-complete and that a specific vector V can be guessed using a single-count (black-peg) query sequence that is of length $N\lceil \log K\rceil + \lceil (2 - 1/K)N\rceil + K$.

As mentioned above, several researchers have started exploring privacy-preserving data querying methods that can be applied to genomic sequences (e.g., see [2], [16], [17]). In particular, Atallah *et al.* [2] and Atallah and Li [3] studied privacy-preserving protocols for edit-distance sequence comparisons, such as in the longest common subsequence (LCS) problem [21], [22], [37], where each party learns the score for the comparison, but neither learns the contents of the string of the other party. Such comparisons are common in DNA sequence alignment comparisons, for example. Troncoso-Pastoriza *et al.* [36] described a privacy-preserving protocol for searching for a certain regular-expression pattern in a DNA sequence. In last-year's Oakland conference, Jha *et al.* [23] give privacy-preserving protocols for computing the edit distance and Smith-Waterman similarity scores between two genomic sequences, improving the privacy-preserving Smith-Waterman comparison algorithm of Szajda *et al.* [35]. Single-count matching results between two strings can be done in a privacy-preserving manner, as well, using privacy-preserving set intersection, e.g., using the method of Freedman *et al.* [17], Vaidya and Clifton [38] or Sang and Shen [31], [32]. The string matching problem can also be done using privacy-preserving dot product computations [1] or even general multi-party computation protocols (e.g., see [13], [19], [40]) or systems [6].

In terms of the framework of this paper, the closest previous work is that of Du and Atallah [15], who studied a privacy-preserving protocol for querying a string Q in a database of strings, D, where comparisons are based on approximate matching (but not sequence-alignment). Their protocols assume that the parties are honest-but-curious, however, so that, for instance, the database owner cannot introduce fake strings in his database whose intent is to discover the identity of the query string, Q. The attack model we explore in this paper, on the other hand, allows for "cheating" in the comparison protocol, so that D can introduce strings whose sole purpose is to help him discover the identity of Q.

1.3. Attack Scenarios

In this paper we study the *Mastermind attack* on genomic data, which is a way that a genomic querier, Bob, can "play" a type of Mastermind game with a genomic string Q–for which Q's owner, Alice, thinks that she is comparing with Bob in a privacy-preserving manner—but instead Bob is discovering the full identity of Q.

The attack scenario is that Alice repeatedly participates in privacy-preserving comparisons of Q to iteratively compare Q with strings provided by Bob. All is learned from each comparison is the score measuring the similarity of the two strings (Q and a string V_i provided by Bob), with the score for each string comparison being revealed to Bob (and possibly also Alice) before the next comparison begins. Bob's goal is to learn the complete identity of Q with a reasonably small of comparisons.

We distinguish two versions of this attack scenario. In the first scenario, the comparison between Q and each string V_i provided by Bob is scored according to the single-count (black-peg) straight-match score,
$$b(V_i) = |\{j \colon V_i[j] = Q[j]\}|.$$
In our second scenario, which is more common in genomic databases, the comparison between Q and each V_i provided by Bob is scored according to a sequence-alignment score,
$$a(V_i) = |\{(j, k) \in \mathcal{I} \colon V_i[j] = Q[k]\}|,$$

where $\mathcal{I}$ is an ordered index set of pairs of integers so that if (j,k) appears before (l,m) in $\mathcal{I}$, then $j < l$ and $k < m$. This is also known as the *longest common subsequence* (LCS) [21], [22], [37] score between Q and V. (See Figure 4.) Incidentally, the edit distance and Smith-Waterman scores are strongly related to the LCS score, and our attack scenarios should be able to be translated to these other measures, depending on the weights given to various edit actions.

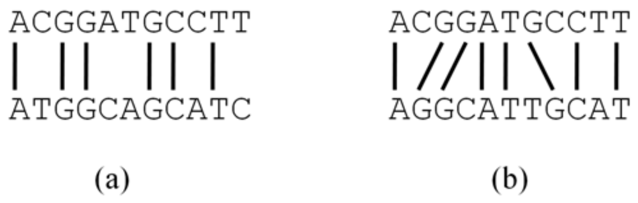

Figure 4. Illustrating two types of matches between two strings. (a) A single-count (black-peg) straight-match. Note that the second “A” in the bottom string is not matched, since it doesn’t line up exactly with the second “A” in the top string. (b) A sequence-alignment match. In going from the top string to the botttom string, the first “C” in the top string corresponds to a *deletion* event, the first “C” in the bottom string corresponds to an *insertion* event, and the penultimate characters in each string correspond to a *substitution* event.

There are a number of motivating usage environments that could be susceptible to Mastermind attacks. For example, Bob could be a genomic database owner, storing genomic sequences for a number of individuals, and Alice could be a database user who is searching Bob’s database to find the closest match to a string Q of interest. Bob could, for instance, be the owner of a database of DNA from every male attending a certain university and Alice could be an FBI agent searching through that database for a match with DNA evidence gathered after a sexual assault. Both parties in this example are likely to be under legal restrictions not to reveal the complete identity of their strings unless there is a match. In another example, Alice could be the owner of a database of genomic sequences and Bob could be an attacker trying to learn the identity of a string Q in Alice’s database, e.g., which Bob can identify only by an anonymized index, j. In this case, Bob repeatedly does queries with each of his strings, V_i, indexing into Alice’s database using the name “j” to locate Q and get Alice to do a privacy-comparison of Q with V_i. Bob could, for instance, be an employer trying to learn the genomic sequence of a prospective employee, Charlie, by querying a university DNA sequence database owned by Alice, which he could query simply knowing the index of Charlie’s DNA in Alice’s database (e.g., Bob might be able to infer this index from Charlie’s student number). In every case, Bob gets to ask Alice to compare her string, Q, to each of his query strings, V_i, in a privacy-preserving manner until these comparisons have leaked enough information that he can easily infer the identity of Q.

1.4. Our Results

We begin our technical discussion in this paper by showing that, in fact, the problem of determining whether a sequence of Mastermind responses has a valid solution is NP-complete, even if each response is a sequence-alignment response. At first, this might seem to provide some security for the privacy of the genomic string, Q, for it implies a degree of intractability to the problem of learning a query string Q just from Mastermind responses involving Q. Unfortunately, as was learned with Knapsack cryptosystems [28], having the security of a system be based on the difficulty of solving an NP-complete problem is no guarantee that it is safe in practice. Indeed, such is the case for the security of genomic strings being susceptible to the Mastermind attack. We show that genomic strings can be discovered by surprisingly short sequence of guesses. In particular, we also provide the following results:

- We adapt a method of Goodrich [20] for solving Mastermind games using black-peg results to genomic strings, showing how an arbitrary query string, Q, of length N from an alphabet of size K, can be discovered with $N\lceil\log K\rceil + \lceil(2-1/K)N\rceil + K$ guesses, each of which is a single-count response, and we observe that this method is still valid even if all such responses are privacy-preserving.
- We show that an arbitrary query string, Q, of length N from an alphabet of size K, can be discovered with $(N+1)K$ queries, each of which reports the result of a sequence-alignment test. Such queries are, of course, different than single-count tests, but they are more common in genomic applications. We also show that this bound can be further improved if the distribution of characters in the alphabet follows Zipf’s Law [27].
- We show how a Mastermind attacker can take advantage of known distributional information for genomic data. Armed with distributional knowledge about a query string, Q, with respect to a reference string, R, such as the Revised Cambridge Reference Sequence, rCRS (GenBank accession number: AC 000021), the Mastermind attacker can discover Q much quicker than in the general cases, using either single-count or sequence-alignment responses.
- We provide experimental analysis of the distribution-based Mastermind attack for genomic data, showing that, for either single-count responses or sequence-alignment responses, the attack works surprisingly well. Given the relative abundance of mtDNA data, and its ethnic sensitivity, we focus our experiments on 1000

human mtDNA sequences, showing that most can be discovered with a Mastermind attack of just a few hundred guesses, even though mtDNA strings are typically over 16,500 bp long. Given that current mtDNA databases already have thousands of members (e.g., see [5]), this experimental analysis shows that it would be relatively easy for an attacker, Bob, to interleave an undetected Mastermind attack with privacy-preserving responses to actual strings.

We conclude by discussing some of the issues that would have to be addressed in order to defeat Mastermind attacks on genomic data, as well as some possible directions for future research.

2. NP-Completeness of Sequence-Alignment Mastermind Satisfiability

As mentioned above, Stuckman and Zhang [34] show that double-count Mastermind satisfiability is NP-complete and Goodrich [20] shows that single-count (black-peg) Mastermind satisfiability is also NP-complete.

In the Sequence-Alignment Mastermind Satisfiability problem, we are given a collection of Mastermind queries, $V_1, V_2, \ldots, V_N$, and the responses, $a(V_1), a(V_2), \ldots, a(V_N)$, each of which is said to report the sequence-alignment (LCS) score between each V_i and an unknown vector, V. We are asked to determine if there indeed exists a vector V that satisfies all of these responses.

Theorem 1: *Sequence-Alignment Mastermind Satisfiability is NP-complete.*

Proof: See appendix. ■

Thus, it is extremely unlikely that we will be able to find a polynomial-time algorithm that can always satisfy arbitrary Mastermind sequence-alignment query sequences, or even single-count queries [20]. Unfortunately, this is not the same as a guarantee of security for the kinds of query sequences that would result from an interaction between a Mastermind attacker, Bob, and a genomic string owner, Alice, where Bob is trying to learn Alice's string, Q, through a sequence of privacy-preserving string comparisons. For we show, in the sections that follow, that such query strings, Q, can be discovered fairly efficiently using the Mastermind attack.

3. The Mastermind Attack for Single-Count Straight-Match Queries

In this section, we explore the version of the Mastermind attack where the attacker, Bob, engages in a series of privacy-preserving protocols with Alice, each of which reveals only the single-count straight-match score between Alice's string, Q, and strings provided by Bob, in an iterative online fashion (recall Figure 4a). In the attack model we consider, Bob is allowed to use self-constructed strings in comparisons with Q. That is, we do not restrict Bob to use only the strings present in a certain genomic database, D, or that belong to a certain probability distribution. In the framework of the Mastermind attack, this scenario is equivalent to the situation where Bob constructs a deterministic plan for discovering Q from the single-count responses returned by the privacy-preserving protocol used for each comparison. Here, we show that Bob can learn Q with a sequence of $N\lceil \log K\rceil + \lceil (2-1/K)N\rceil + K$ guesses, where N is the length of Q and K is the size of the alphabet (whose members we call "colors"). Our algorithm is an adaptation of an algorithm of Goodrich [20] for solving the boardgame version of Mastermind to the specific case of a Mastermind attack on a genomic string Q.

We begin the attack for Bob by having him perform a query against Q with a reference string, R. For example, R could be a genomic string derived from a sequencing of the DNA of a specific reference human or it could be a canonical genomic reference string derived from analyzing commonalities among a number of human sequences. Even though few humans have presently had their complete genomes sequenced [12], [26], [39], any of these could serve as a reference, R, for a Mastermind attack on a complete genome sequence. For the more wide-spread instances of mitochondrial DNA, the Revised Cambridge Reference Sequence (rCRS) (GenBank accession number: AC 000021) is commonly used as a mtDNA reference sequence [7], [8], [30], and it could serve as the string R in a Mastermind attack on a mitochondrial DNA string.

Imagine that we cyclically order the K characters in our alphabet, so, for instance, if our alphabet is {A,C,G,T}, then we could use the cyclic ordering (A,C,G,T,A,C,G,T,...). Note that this ordering allows us to choose any character as a base color, i.e., a "color 0," and then specify all other characters as offsets from that base. For example, in the DNA case, we could pick "C" as the base, color 0, in which case "G" becomes color 1, "T" becomes color 2, and "A" becomes color 3. Or we could pick "T" as the base, color 0, in which case "A" becomes color 1, "C" becomes color 2, and "G" becomes color 3.

In the context of a Mastermind attack, we consider each character, R_i, in the reference string, R, to be color "0" for that position, i. Viewed Mathematically, we can then number the $K-1$ remaining characters, according to our cyclic ordering, as offsets from these respective color 0's. Assuming that Bob's first guess, of R, is not a perfect match for the query string, Q, then we can view Bob's remaining task as that of determining the cardinality and location of all the non-zero offset values for positions in R. In fact, if we think of the characters in the respective positions of R as the respective color 0's for those positions, then we can view the remaining task as that of determining the locations of the colors 0 through $K-1$.

After Bob makes his initial guess using R, we then have him perform $K-1$ additional queries, each of which is a vector of elements that are all the same offset from R, i.e., a vector of all the same "colors" with respect to R. This allows us to initially know the cardinality, $c_0, c_1, \ldots, c_{K-1}$, of every (offset) color in the unknown vector, Q. If any $c_i = 0$, then we remove the color i from our alphabet of colors, and update the value of K accordingly. The remainder of Bob's computation proceeds as a recursive divide-and-conquer algorithm, which is similar in structure to the approach of [11], [20].

The generic problem is to determine the offset values of all the elements in a range $Q[l..r]$, which initially is the entire vector $Q = Q[0..N-1]$, assuming we know the values of $c_0, c_1, \ldots, c_{K-1}$, of every color in $Q[l..r]$, and each $c_i > 0$. If $K \leq 1$, we are done; so let us assume without loss of generality that $K \geq 2$. In addition, we assume inductively that we know, d, the number of instances of color 0 outside of the range $Q[l..r]$. Initially, of course, $d = 0$.

Given this initial setup, we split $Q[l..r]$ into $Q[l..m]$ and $Q[m+1..r]$, where m is in the middle of the interval $[l, r]$. The main challenge, then, is to provide for $Q[l..m]$ and $Q[m+1..r]$ the same setup we had for $Q[l..r]$. This setup can be accomplished by determining the cardinalities, $x_0, x_1, \ldots, x_{K-1}$ and $y_0, y_1, \ldots, y_{K-1}$, of every color that respectively appears in $Q[l..m]$ and $Q[m+1..r]$. We do this with a series of $K-1$ additional queries, where we guess that the elements in $Q[l..m]$ are of color i, for $i = 1, 2, \ldots, K-1$, and that the rest of Q is of color 0. Let the values of these queries be denoted as $b_1, b_2, \ldots, b_{K-1}$, and note that, at this point, we know the following:

$$x_i + y_i = c_i, \quad \text{for } i = 0, 1, \ldots, K-1 \qquad (1)$$

$$x_i + y_1 = b_i - d, \quad \text{for } i = 1, 2, \ldots, K-1 \qquad (2)$$

$$x_0 + x_1 + \cdots + x_{K-1} = m - l + 1. \qquad (3)$$

Thus, we can determine y_0, as

$$y_0 = \frac{c_0 + \sum_{i=1}^{K-1}(b_i - d) - (m - l + 1)}{K},$$

for y_0 is counted K times in the sum of c_0 and all the $(b_i - d)$'s, and the sum of the x_i's is $m - l + 1$, by Equation (3). Given the value of y_0, we can then determine all the x_i values, by using Equation (1) for x_0 and Equation (2) for $x_1, x_2, \ldots, x_{K-1}$. Moreover, once we have all these x_i values, we can determine the values, $y_1, y_2, \ldots, y_{K-1}$, using Equation (1). Finally, we can determine the values $d' = d + y_0$ and $d'' = d_{x_0}$ and use these respectively for the role of d in $Q[l..m]$ and $Q[m+1..r]$. This gives us all the values necessary to then recursively determine $Q[l..m]$ and $Q[m+1..r]$. Of course, if the c_i values for either of these subproblems are all 0, except for one (which would be equal to the size of this problem), then there is no need to recursively solve this problem; so we would not perform a recursive call in this case.

Let us, therefore, analyze the number, $G(N, K)$, of vector guesses performed by this algorithm. Ignoring for the time being the initial set of K guesses, we can bound this parameter using the following recurrence:

$$G(N, K) = 2G(N/2,\ K) + \min\{N, K-1\}.$$

Thus, adding the initial K queries back in, we get that the total number of guesses is at most

$$N\lceil \log K \rceil + \lceil (2 - 1/K)N \rceil + K.$$

Thus, we have the following.

Theorem 2: *Given an unknown length-N genomic string Q, defined on an alphabet of size K, then, starting from a reference string, R, a malicious Mastermind attacker can discover Q in polynomial time using at most $N\lceil \log K \rceil + \lceil (2 - 1/K)N \rceil + K$ tests against Q, in the worst case, each of which reveals only the number of positions where Q and the test string match.*

One thing to notice in this theorem is that it provides a worst-case upper bound on the information leakage risks of the Mastermind attack. As we explore later, in Section 5, this worst-case can significantly over-estimate these risks, particularly when the reference string, R, is already a reasonably good match for Q, which is often the case.

Another observation to make about the above Mastermind attack is that even though Bob is incrementally making progress, with each guess, in his goal of discovering Q, the scores between his guesses and Q need not ever be all that close to N. He can always be focusing on a specific recursive call in the above divide-and-conquer algorithm, dealing with a certain region of the vector, keeping the other regions unchanged, which will keep the scores relatively low. In other words, he can run the above divide-and-conquer algorithm, implementing each iteration using a privacy-preserving protocol with Alice, where both he and Alice learn the result, and these scores should not tip off Alice that he is running a Mastermind attack.

4. The Mastermind Attack for Sequence-Alignment Queries

In addition to queries that report the number of matching locations between two strings, which we explored in the previous section, another type of query that is quite common in genomic databases is the *sequence-alignment* query. In this query, we wish to compare two sequences Q and V, where the score for a match is the length of the longest common subsequence (LCS) [21], [22], [37] between Q and V. Several researchers have studied this problem and have come up with privacy-preserving protocols to determine such scores (e.g., see [2]). In this section, we show that performing such a series of sequence-alignment queries with Bob is susceptible to a type of Mastermind attack of its own.

Suppose we are given an unknown string Q of length N over an alphabet of size K, the members of which we call "colors." Suppose further that we are going to engage in a protocol with Bob to test Q against strings provided by Bob, where each test returns the length of a longest common substring between Q and one of Bob's strings. That is, we score matches using the sequence-alignment scoring function, $a(V)$, for a guess vector V, which is the length of a longest common substring between V and Q. We are interested in this section on studying an efficient scheme for Bob to discover Q using this query scheme.

A Mastermind-attack algorithm for Bob is as follows. Bob begins by guessing K vectors, $V_1, V_2, \ldots, V_K$, with each vector V_i consisting of elements of all the same color, i. (Here we are not starting with a reference string, R, as we did in the previous section; we explore the improvements that can come from using a reference string and genetic distribution knowledge with sequence-alignment queries in Section 5.2.)

The substring alignment score for each of the initial guesses will tell Bob the cardinality of each color in Q. Let us now imagine that we reorder the colors so that they are listed 1 to K in nondecreasing order of how often they each appear in Q. Thus, color 1 is now the least frequent color in Q and K is the most frequent color. Let c_i denote the cardinality of color i in Q. Our algorithm for Bob's Mastermind attack continues by repeatedly using the following observation:

Observation 3: *If we create a subsequence S_1 of c_1 elements of color* 1*, then all the other elements of Q appear between matches with the consecutive elements in S.*

Using a sequence S_1 of c_1 elements of color 1, we next focus on how the elements of color 2 interleave with the elements of S_1. Let us therefore define c_1 variables, $c_{2,1}, c_{2,2}, \ldots, c_{2,n_2}$, where $n_2 = c_1 + 1$ is the number of "slots" determined by the consecutive elements in S_1. Our goal is for each variable, $c_{2,i}$, to store the number of elements of color 2 that appear in the ith slot determined by S_1. We determine these variables by a sequence of guesses, where the first guess is a vector:

$$W_1 = (2, 2, \ldots, 2, 1, 1, \ldots, 1),$$

where we have $N - c_1$ instances of color 2 and c_1 instances of color 1. Note, then, that the number of elements of color 2 in the first slot is $c_{2,1} = a(W_1) - c_1$. So we next form

$$W_2 = (2, 2, \ldots, 2, 1, 2, 2, \ldots, 2, 1, \ldots, 1),$$

where we have $c_{2,1}$ elements of color 2, followed by one 1, followed by $N - c_1 - c_{2,1}$ elements of color 2, followed by $c_1 - 1$ elements of color 1. From this, we can determine $c_{2,2}$ as $c_{2,2} = a(W_2) - c_1 - c_{2,1}$. We then continue in this way, forming W_i as an alternating sequence of elements of color 2 and single instances of color 1, with the jth sequence of 2's having $c_{2,j}$ instances of color 2. At the end of W_i we place $N - c_1 - \sum_{j=1}^{i-1} c_{2,j}$ elements of color 2 followed by $c_1 - i + 1$ elements of color 1. Thus, after performing n_1 guesses, we will know the values of all the $c_{2,i}$ variables. From this, we form a sequence S_2 of elements of colors 2 and 1 that forms the largest such sequence that is a subsequence of Q.

Inductively, we can assume we have a sequence, S_{s-1}, of elements of colors, $1, 2, \ldots, s-1$, that is largest such sequence that is a subsequence of Q. We then form variables $c_{s,1}, c_{s,2}, \ldots, c_{s,n_s}$, where $n_s = 1 + \sum_{j=1}^{i-1} c_j$, which is the number of "slots" formed between elements in S_{s-1}. Each variable $c_{s,i}$ denotes the number of elements of color s that appear in the ith slot determined by S_{s-1}. We then form n_s guesses, similar to as above, with the ith such guess allowing us to determine the value of $c_{s,i}$. When we have completed this set of guesses for color K, we will know the vector Q in its entirety.

In terms of the analysis of this Mastermind-attack algorithm, note that the total number of queries is

$$\begin{aligned} K + \sum_{i=2}^{K} n_i &= K + \sum_{i=1}^{K-1} \left(1 + \sum_{j=1}^{i} c_j\right) \\ &= K + N + \sum_{i=1}^{K-1} (K-i) c_i. \end{aligned}$$

Let us perform a substitution of variables, where we let $d_1, d_2, \ldots, d_K$ denote the cardinalities of the colors in Q in nonincreasing order, so d_1 is the most frequent color and d_K is the least frequent. Then the total number of queries performed becomes

$$K + N + \sum_{i=1}^{K-1} i d_{i+1}.$$

Note that, by definition, $d_i \leq N/i$, for otherwise, d_i could not be the ith largest-cardinality color. Thus, since $d_{i+1} \leq d_i$, the total number of queries is at most

$$\begin{aligned} K + N + \sum_{i=1}^{K-1} i(N/i) &= K + N + N(K-1) \\ &= (N+1)K. \end{aligned}$$

This is the number of tests done by Bob, the Mastermind attacker, making no additional assumptions about the distribution of colors in the query string, Q.

This analysis can be refined, however, if the colors are distributed in Q according to Zipf's Law [27], which in this context would imply that

$$d_i \leq \frac{N}{i^s H_{N,s}},$$

where $H_{N,s}$ is the N-th Harmonic number of order s, $H_{N,s} = \sum_{i=1}^{N} 1/i^s$, and s is typically between 1 and 2. In this case, the total number of guesses done by Bob would be at most

$$\begin{aligned} K + N + \sum_{i=1}^{K-1} \frac{iN}{i^s H_{N,s}} &\leq K + N + \frac{N(K-1)}{H_{N,s}} \\ &= \frac{(N+1)K}{H_{N,s}}, \end{aligned}$$

for $s \geq 1$. Thus, we have the following:

Theorem 4: *Given an unknown length-N string Q, defined on an alphabet of size K, a malicious Mastermind attacker can discover Q in polynomial time using $(N+1)K$ sequence-alignment tests tests against Q, each of which reveals only the length of a longest common subsequence between Q and the test string match. If the cardinalities of elements of Q follow Zipf's Law, with parameter s, then a malicious Mastermind attacker can discover Q using at most $(N+1)K/H_{N,s}$ sequence-alignment tests.*

5. Exploiting Genomic Data Distributions

Up to this point, we have focused on how the Mastermind attacker, Bob, could learn a general string Q using the types of queries typically asked of genomic databases, even if those queries are privacy preserving. In this section, we explore how Bob can significantly improve the effectiveness of the Mastermind attack if he exploits information, which is publicly available, about the distributions of characters in genomic sequences. Moreover, to drive the point home, we provide experimental evidence of the effectiveness of such Mastermind attacks on a real-world genomic database, in the section that follows.

Genomic strings typically have a great deal of similarity. Indeed, recent compression schemes have shown that it is effective to view a genomic string with respect to a compression scheme that represents a string in terms of its differences with a reference string, R (e.g., see [4]). That is, we can start from a reference string, R, which contains the most common components of a typical genomic string. Then we define each other string, Q, in terms of its differences with R. Each difference is defined by an index location, i, in R and an operation to perform at that location, such as a substitution, insertion, or deletion.

This difference pattern is present, for example, in human mitochondrial DNA, which is the type of genomic data we use in our experimental studies. This type of of DNA, which, as we have already mentioned, is inherited only through the maternal line and is already available in sequenced form in sizeable enough quantities to support obfuscated Mastermind attacks.

As shown in recent work of Baldi *et al.* [4], mitochondrial DNA sequences can be encoded in significantly-compressed form by using a standard reference sequence [7], [30]. This reference sequence, R = rCRS, is 16,568 bp long. So, in terms of the notation used above, we have $N = 16568$ and $K = 4$, since there are 4 types of base pairs possible. But these parameters suggest that there is more variation in the data than actually occurs. For example, using only this information and the Mastermind attack algorithm summarized in Theorem 2, we would need roughly $4 + (16568)2 + (1.75)16568 = 62170$ guesses to determine the identity of a query string.

In fact, the vulnerability of DNA strings to the Mastermind is much worse than this in practice. For example, there are a limited number of locations along the reference sequence where any changes appear statistically in the mitochondrial DNA data. So let us use M to denote the number of different possible locations where any query string might differ from the reference string, R. Worse yet, from a privacy-preservation standpoint, the average number of difference between any human DNA string and the reference is orders of magnitude smaller than M in practice. (We explore these statistics in detail below.) Here we show how a Mastermind attack can exploit these statistical properties of genomic data.

5.1. The Substitution-Only Case

Given the above additional knowledge of the distributional properties of DNA data, we can refine the Mastermind attack to take this knowledge into consideration. We begin, in this subsection, with our algorithm that performs single-count Mastermind tests, adapting the algorithm of Section 3. In this case, we make the assumption that the query string, Q, differs from the reference string R only through substitutions, which is true for example, for 45% of the mitochondrial DNA data. We will explore the more general case later in this section.

For any query string, Q, let $s(Q)$ denote the number of substitutional differences Q has with the reference string, R, and let $\hat{D}$ denote the average of the $s(Q)$ values, taken over the population of genomic strings under consideration.

We perform our Mastermind attack algorithm, in this case, using a slight modification of our algorithm in Section 3. Note that already after the very first query, using the reference string R, we know $s(Q)$, using the formula $s(Q) = N - b(R)$. As before, we now view each color in each position of R as the base color for that position, and each other color as an offset from that base (using modular arithmetic). Thus, we can view colors as being numbered as $0, 1, 2, \ldots, K-1$, and our remaining job is to determine the $s(Q)$ locations in Q that have non-zero colors. So far, this sounds like the same algorithm as we described in Section 3.

Here is where things become different. We follow our initial guess of the reference string, R, with $K-1$ additional guesses for each of the offset values $1, 2, \ldots, K-1$, but we only do so for the M statistically modifiable locations in R. This tells us the cardinality of each offset value in Q, which we denote with $c_0, c_1, \ldots, c_{K-1}$.

Next we begin a divide-and-conquer algorithm, but instead of splitting at the midpoint of a range that is initially from locations 0 to $N-1$ in Q, we perform the splitting in the midpoint of a range of the M values that we know could possibly be locations of substitutions. To set up the recursive calls, we perform $K-1$ additional queries, like in Section 3, with the first half being an offset that ranges from 1 to $K-1$. Using the results of these guesses, we then recursively solve any subproblems that we have just determined have a difference with the reference string, R, reducing the set of offset colors used if we discover any of them have cardinality of zero in the subproblem. When we reach the base case of one remaining location, we stop, having identified its color (offset). Likewise, if all the colors of a subproblem are determined from the cardinalities of the colors for that subproblem, then we stop the recursion at that point, since that subproblem is now solved.

In spite of the similarity of this algorithm to the Mastermind attack method of Section 3, we analyze this variation differently. In particular, we can view the recursive calls for this divide-and-conquer algorithm in terms of a recursion trace, where we create a node in a binary tree for each recursive call that we make. Thus, each leaf of this tree corresponds to a location that we have discovered that differs from the reference string. Each internal node in this tree corresponds to a recursive call. Moreover, the number of guesses that we make at such an internal node v is equal to the minimum of $K-1$ and the number of leaf descendents of v in this tree. Therefore, we can more than account for every guess that is made (other than the initial $K-1$) by charging each leaf w with one guess for each of w's ancestors in this tree. Since we always perform splits according to the M known locations of possible deviation from the reference string, R, this implies that each leaf is charged at most $\lceil \log M \rceil$ times. That is, the total number of guesses made by this modified algorithm is at most $s(Q)\lceil \log M \rceil + K - 1$. We can summarize this method and its performance, then, as follows.

Theorem 5: *Given an unknown length-N string Q, defined on an alphabet of size K, with Q having M possible locations of deviation from a reference string, R, a malicious Mastermind attacker can discover Q in polynomial time using $s(Q)\lceil \log M \rceil + K - 1$ guesses, each of which reveals only the number of positions where Q and the test string match and where $s(Q)$ denotes the number of substitutions that would transform R into Q.*

Thus, the average number of tests made by this algorithm is $\hat{D}\lceil \log M \rceil + K - 1$. As we note in Section 6, this performance is more than adequate to show that nearly half of all mitochondrial DNA data is vulnerable to this version of the Mastermind attack. Before we provide those statistics, however, let us study how the Mastermind attack with sequence-alignment queries can be streamlined to exploit DNA data distributions.

5.2. The Sequence-Alignment Case

As mentioned above, roughly half of the sequences in the mitochondrial DNA data set include insertions and/or deletions in addition to substitutions in the reference string, R. Thus, we discuss in this subsection how we can modify the Mastermind attack algorithm of Section 4 to take advantage of the distributional properties common in genomic data sets, so as to discover a query string that can have arbitrary kinds of differences with the reference string, R. In this case, we view differences with R procedurally as *events*, each of which is either a singleton deletion, a singleton deletion, or an arbitrary-length insertion, which would transform R into the query string, Q.

In this case, we run the attack algorithm in two phases. In Phase 1, we aim to discover all the deletion and substitution events, and in Phase 2, we aim to discover all the insertion events. In both phases, we make the simplifying assumption that insertion and deletion events are disjoint. That is, they don't overlap or interfere with one another. This assumption is based on the fact that these events come from a statistical characterization of genomic strings, which is designed to keep events disjoint (for overlapping events are better subdivided further and considered as separate sub-events). So, for example, we assume that there is no insertion event that is then followed by a deletion event that then removes part of the string that was just inserted.

Since we can view each substitution as a deletion followed by a singleton insertion, we first discover all the deletion and substitution events as if they all were deletion events and we then determine which of them are really substitutions. In addition, we assume that all deletion events are singleton deletions, since multiple-length deletions can be broken down into multiple singleton deletions.

We begin by performing a guess for the reference string, R. Armed with the sequence-alignment score, $a(R)$, for R, we then perform a divide-and-conquer computation to find all the deletion events that occur in going from R to Q. Note that if we next perform a guess V for a collection of deletion events at some subset of the M statistically possible (deletion) locations in R, then we can detect how many deletions actually occurred at these locations. Namely, for each deletion event that is present in one of these locations, then our score will not change with respect to the score for R, and, for each location that should not be deleted, we will record a score for V that is one worse than that for R. Thus, we can determine the number of deletion events for any test we do by the difference between the score we observe and the score we would expect if all of the deletions are removing actual matches. That is, if we test for r singleton deletion events in V, then the number that actually occur is $a(V) - (a(R) - r)$, where $a(\cdot)$ is the sequence-alignment count.

Let $Z_{1,M} = \{z_1, z_2, \ldots, z_M\}$ be a set of Boolean variables, such that z_i is 1 if and only if the ith statistically possible deletion event in R actually occurs in going from R to Q. We can perform a divide-and-conquer search in $Z_{1,M}$ to determine which of the z_i's are 1. We begin by testing for all the deletion events in $Z_{1,M}$. This gives us the number of 1's in $Z_{1,M}$. We then perform a test for every deletion event in $Z_{1,M/2} = \{z_1, \ldots, z_{M/2}\}$, which by deduction gives us the number in $Z_{M/2+1,M} = \{z_{M/2+1}, \ldots, z_M\}$. We then recursively determine the number in either or both of these two sets so long as there is at least one deletion event in that set. Thus, we perform a divide-and-conquer parallel "binary" search for each of the exact locations of singleton deletions. Once we have completed this computation for R, with queries against Q, we will have determined the locations of all the deletion events from R to Q, including those deletions that are really substitution events. Thus, this set of guesses uses at most $1 + (d(Q) + s(Q))\lceil \log M \rceil$ tests, where $d(Q)$ is the set of (singleton) deletion events in going from R to Q and $s(Q)$ is the set of substitution events in going from R to Q.

Once we know the locations of all the deletions in going from R to Q, we perform a second set of binary searches, just among these locations, to find the locations among this group that are actually the sites of substitution events. In particular, we perform a binary search for each of the K colors, searching, for each color i, the locations among the found deletion locations where we improve our score by adding a character of color i. Thus, the set of additional guesses we do in this wrap-up set is at most $K + s(Q)\lceil \log(s(Q) + d(Q)) \rceil$. Therefore, the total number of guesses that we do in Phase 1 is at most $1 + K + (d(Q) + s(Q))\lceil \log M \rceil + s(Q)\lceil \log(s(Q) + d(Q)) \rceil$.

Let us now define R' to be the reference string resulting from performing the events we discovered in Phase 1. We begin Phase 2 by performing a guess of the reference string, R'. Now we know that if $R' \neq Q$, then the only remaining events are insertion events. Moreover, we know from our statistical analysis, that there are only M locations where these insertions can occur. Next, we consider each character as a separate "color" and we number them $1, 2, \ldots, K$. As in the deletion case, we let M denote the (relative-to-R') locations where insertion and substitution events are statistically possible. We then perform K guesses, one for each color, with each guess i consisting of an insertion of character i at each of the M statistically possible locations in R'. Note that some insertion events actually involve the addition of strings longer than a single character, but each will match at least one of these guesses.

We then perform a divide-and-conquer search algorithm, like those we have described above, using the M potential event locations as the ranges we split when doing recursive calls. We perform such a search for each color i to determine all the locations where there is an insertion that includes a character of color i. That is, we split the set of location in half and perform a guess that counts how many of the known number of insertions of color i occur in the first half (and by implication how many occur in the second half). This set of queries gives us the cardinalities of each "color" on the two halves. So we then recur on either of the two subproblems that have at least one event that changes it from the reference string. Moreover, with each such subproblem, we only consider the events that have non-zero cardinality in that subproblem region (defined with relative indices from the reference string, R'). Thus, the total number of guesses done by this part of our Phase 2 algorithm is $1 + K + e(Q)\lceil \log M \rceil$, where $e(Q)$ denotes the number of insertion events in Q relative to the reference string, R'.

To complete the computation, then, we perform a miniature version of our algorithm from Section 4 at each location determined to be to site of an insertion event. Each such computation requires $(m + 1)K$ guesses, where m is the length of the insertion. Thus, the total number of guesses made in Phase 2 is $1 + K + e(Q)\lceil \log M \rceil + (\varepsilon(Q) + 1)K$, where $\varepsilon(Q)$ is a weighted sum of all the insertion events, where each is given a weight equal to its length. Therefore, we have the following.

Theorem 6: *Given an unknown length-N string Q, defined on an alphabet of size K, with Q having M possible locations of deviation from a reference string, R, a malicious Mastermind attacker can discover Q in polynomial time using $2 + (d(Q) + s(Q) + e(Q))\lceil \log M \rceil + (\varepsilon(Q) + 3)K + s(Q)\lceil \log(s(Q) + d(Q)) \rceil$ guesses, each of which reveals only the number of positions where Q and the test string match, using sequence-alignment LCS tests.*

6. Experimental Analysis

To demonstrate the vulnerability of real-world DNA data to the Mastermind attack, we have performed an experimental analysis of our distribution-based Mastermind attack algorithms. We used 1000 human mitochondrial sequences downloaded from a recent version of GenBank (http://www.ncbi.nlm.nih.gov/Genbank/index.html). We focused on the sequences alone, ignoring any header and other information, and have simulated Mastermind attacks on each one. The Revised Cambridge Reference Sequence (rCRS) (GenBank accession number: AC 000021) was also downloaded and used as the reference sequence [7], [8], [30]. The reference sequence is 16,568 bp long. All the sequences were aligned to the reference sequence and, for each sequence, the indices of the location of each variation were recorded together with the type (substitution, insertion, deletion) and content of each variation. This step is also essential if one is interested in compressing the data [4], for example. Statistics for the number of substitutions, deletions, and insertions for this data set of 1000 mtDNA sequences is given in Table 1.

	mean	**standard dev.**
Substitutions	28.00	18.38
Deletions	0.90	2.46
Insertions	0.95	1.10

Table 1. Frequency statistics for 1000 mtDNA sequences. Mean and standard deviation statistics are given for the frequency of substitutions, deletions, and insertions in going from the reference string, $R =$ rCRS, to each sampled sequence.

Of the 1000 sequences, 453 have only substitution events with respect to the reference string, $R =$ rCRS. So we used this subset of 453 sequences to test the simulated performance of the method of Theorem 5. The distribution of the number of substitutions in each of these sequences is shown in Figure 5.

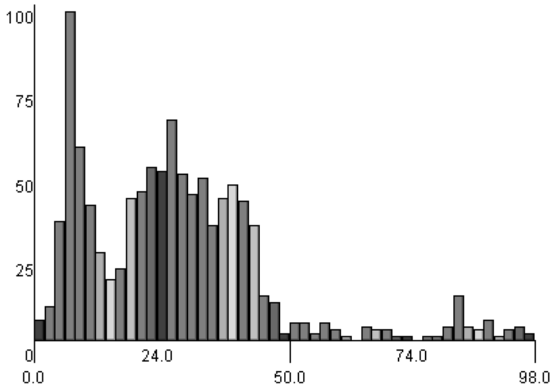

Figure 5. Histogram of number of substitutions in 1000 mtDNA with respect to the reference string, $R =$ rCRS.

Note that these frequencies do not follow a normal distribution, which shows the importance of our using real-world data, such as this, rather than randomly-generated or simulated data. The statistical diversity of the mtDNA data is actually a reflection of the racial diversity of the people whose mtDNA data is included in our data set. That is, edit distance from the reference string, $R =$ rCRS, across the human species, is not uniformly or normally distributed. Instead, edit distance from rCRS is a reflection of human migration patterns, as illustrated in Figure 1.

The 45.3% of the sampled mtDNA sequences with substitution-only modifications from rCRS are exactly the set of sequences that can be effectively discovered by the single-count Mastermind attack of Theorem 5. Thus, we simulated the performance of this attack on each one of these sequences and tabulated the number of guesses that would be needed in each case in order to discover the complete identity of each sequence. Interestingly, 90% of the simulated substitution-only Mastermind attacks completed with 375 guesses or less. The complete distribution of single-count Mastermind attack lengths for this data set are shown in Figure 6.

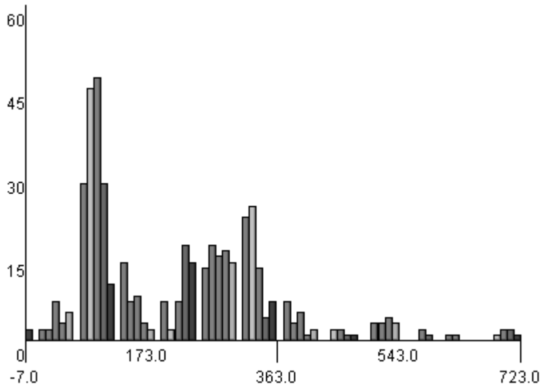

Figure 6. Histogram of Mastermind attack lengths for 453 substitution-only mtDNA strings with standard single-count Mastermind scores. The mean attack length for this data set was 219.6 and the standard deviation was 139.1.

All 1000 sampled mtDNA sequences were then used to test the performance of the method of Theorem 6. Sequence-alignment Mastermind attacks were simulated for each such mtDNA sequence while the number of sequence-alignment tests were counted for each. Interestingly, 90% of these simulated subsequence-alignment Mastermind attacks completed with 875 guesses or less. And some completed with much fewer than this. The complete distribution of sequence-alignment Mastermind attack lengths for this data set is shown in Figure 7.

7. Discussion and Future Directions

We have shown that, even though the single-count and sequence-alignment Mastermind satisfiability problems are NP-complete, one can effectively mount Mastermind attacks on arbitrary character strings just by knowing basic information about the length of the strings and the number of characters in the alphabet used to construct those strings. Moreover, if these strings are genomic sequences and one has some basic statistical information about these strings, relative to a reference string, then one can mount the Mastermind attack with surprising effectiveness. In fact, we have shown experimentally that such attacks are already possible and surprisingly efficient for mtDNA sequences.

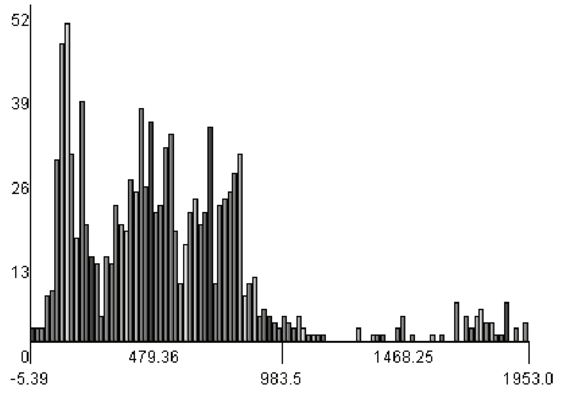

Figure 7. Histogram of simulated Mastermind attack lengths for 1000 mtDNA strings with sequence-alignment scores. The mean sequence-alignment simulated Mastermind attack length was 536.3 with a standard deviation of 373.9.

One conclusion to draw from this work is that privacy-preserving protocols for performing a query with a sequence, Q, against a genomic database, D, should take into account the entire set of comparisons [15], with Q and the strings in D, rather than relying on the privacy-preservation of each individual comparison in turn. For example, in the usage model where Bob is a user querying a genomic database, the Mastermind attack is weakened if it is difficult for Bob to know the index of the strings he is comparing against—for example, if the database owner, Alice, presents her strings in a different random order each time. Such an obfuscation does not defeat the Mastermind attack, however, if Bob is able to use other reasoning inferences to match scores of his query strings across multiple queries in Alice's database of strings.

In terms of further exploration of the vulnerability of genomic data to the Mastermind attack, one interesting direction for future work would be to test the vulnerability of entire human genomes to the Mastermind attack, once we have enough completed genomes to do such an experimental study. In addition, other directions for future research therefore could include new, efficient privacy-preserving schemes for querying entire genomic databases with respect to sequence-alignment queries. Such results would negate the privacy-exposing vulnerabilities of the Mastermind attack.

Acknowledgments

We would like to thank Pierre Baldi for suggesting the security of genomic data as an important research question and for providing the mitochondrial DNA data used in our experiments, including the characterizations in terms of the reference string, rCRS. We would also like to thank David Eppstein, Daniel Hirschberg, Stas Jarecki, and Michael Nelson for helpful discussions regarding the topics of this paper. This research was supported in part by the National Science Foundation under grants 0724806, 0713046, and 0847968.

References

[1] A. Amirbekyan and V. Estivill-Castro. A new efficient privacy-preserving scalar product protocol. In *AusDM '07: Proceedings of the sixth Australasian conference on Data mining and analytics*, pages 209–214, Darlinghurst, Australia, Australia, 2007. Australian Computer Society, Inc.

[2] M. J. Atallah, F. Kerschbaum, and W. Du. Secure and private sequence comparisons. In *WPES '03: Proceedings of the 2003 ACM workshop on Privacy in the electronic society*, pages 39–44, New York, NY, USA, 2003. ACM.

[3] M. J. Atallah and J. Li. Secure outsourcing of sequence comparisons. *Int. J. Inf. Secur.*, 4(4):277–287, 2005.

[4] P. Baldi, R. W. Benz, D. Hirschberg, and S. Swamidass. Lossless compression of chemical fingerprints using integer entropy codes improves storage and retrieval. *Journal of Chemical Information and Modeling*, 47(6):2098–2109, 2007.

[5] D. M. Behar1, S. Rosset, J. Blue-Smith, O. Balanovsky, S. Tzur1, D. Comas, R. J. Mitchell, L. Quintana-Murci, C. Tyler-Smith, and R. S. Wells. The genographic project public participation mitochondrial DNA database. *PLoS Genetics*, 3(6), 2005. doi:10.1371/journal.pgen.0030104.

[6] A. Ben-David, N. Nisan, and B. Pinkas. FairplayMP - a system for secure multi-party computation. In *Proceedings of the ACM Computer and Communications Security Conference (ACM CCS)*, pages 257–266, New York, NY, USA, 2008. ACM.

[7] M. Brandon, M. Lott, K. Nguyen, S. Spolim, S. Navathe, P. Baldi, and D. Wallace. MITOMAP: a human mitochondrial genome database - 2004 update. *Nucleic Acids Research*, 33:D611–D613, 2005. Database Issue.

[8] M. C. Brandon, E. Ruiz-Pesini, D. Mishmar, V. Procaccio, M. T. Lott, K. C. Nguyen, S. Spolim, U. Patil, P. Baldi, and D. C. Wallace. MITOMASTER: A bioinformatics tool for the analysis of mitochondrial DNA sequences. *Human Mutation*, 0:1–6, 2008.

[9] Z. Chen, C. Cunha, and S. Homer. Finding a hidden code by asking questions. In *COCOON '96: Proceedings of the Second Annual International Conference on Computing and Combinatorics*, volume 1090 of *LNCS*, pages 50–55. Springer, 1996.

[10] G. M. Church. The personal genome project. *Molecular Systems Biology*, 1, 2005.

[11] V. Chvátal. Mastermind. *Combinatorica*, 3(3/4):325–329, 1983.

[12] I. H. G. S. Consortium. Initial sequencing and analysis of the human genome. *Nature*, 409:860–921, 2001.

[13] I. Damgård, M. Fitzi, E. Kiltz, J. B. Nielsen, and T. Toft. Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In S. Halevi and T. Rabin, editors, *Theory of Cryptography*, volume 3876 of *Lecture Notes in Computer Science*, pages 285–304. Springer, 2006.

[14] M. Dickerson, D. Eppstein, M. T. Goodrich, and J. Meng. Confluent drawings: Visualizing non-planar diagrams in a planar way. In *Proc. 11th Int. Symp. on Graph Drawing*, volume 2912 of *Lecture Notes in Computer Science*, pages 1–12. Springer-Verlag, 2003.

[15] W. Du and M. J. Atallah. Protocols for secure remote database access with approximate matching. In A. K. Ghosh, editor, *E-Commerce Security and Privacy: Advances in Information Security, Volume 2*, pages 87–112. Kluwer Academic Publishers, 2001.

[16] W. Du and M. J. Atallah. Secure multi-party computation problems and their applications: a review and open problems. In *NSPW '01: Proceedings of the 2001 workshop on New security paradigms*, pages 13–22, New York, NY, USA, 2001. ACM.

[17] M. Freedman, K. Nissim, and B. Pinkas. Efficient private matching and set intersection. In *Advances in Cryptology — EUROCRYPT 2004.*, 2004.

[18] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, New York, NY, 1979.

[19] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *STOC '87: Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 218–229, New York, NY, USA, 1987. ACM.

[20] M. T. Goodrich. On the algorithmic complexity of the mastermind game with black-peg results. *Information Processing Letters*, ?:to appear, 2009.

[21] D. S. Hirschberg. A linear space algorithm for computing maximal common subsequences. *Commun. ACM*, 18(6):341–343, 1975.

[22] C. S. Iliopoulos and M. S. Rahman. Algorithms for computing variants of the longest common subsequence problem. *Theor. Comput. Sci.*, 395(2-3):255–267, 2008.

[23] S. Jha, L. Kruger, and V. Shmatikov. Towards practical privacy for genomic computation. In *Proceedings of the 2008 IEEE Symposium on Security and Privacy*, pages 216–230, Washington, DC, USA, 2008. IEEE Computer Society.

[24] J. Kaiser. A plan to capture human ddveristy in 1000 genomes. *Science*, 319:395, 2008.

[25] D. Knuth. The computer as a master mind. *Journal of Recreational Mathematics*, 9:1–5, 1977.

[26] S. Levy, G. Sutton, P. C. Ng, and et al. The diploid genome sequence of an individual human. *PLOS Biology*, 5(10):2113–2144, 2007.

[27] M. Newman. Power laws, Pareto distributions, and Zipf's law. *Contemporary Physics*, 46(5):323–351, 2005.

[28] A. M. Odlyzko. The rise and fall of knapsack cryptosystems. In C. Pomerance, editor, *Cryptology and Computational Number Theory*, pages 75–88. Am. Math. Soc., 1990.

[29] B. Pakendorf and M. Stoneking. Mitochondrial DNA and human evolution. *Annual Rev. Genomics Hum. Genet.*, 6:165–183, 2005.

[30] E. Ruiz-Pesini, M. T. Lott, V. Procaccio, J. Poole, M. C. Brandon, D. Mishmar, C. Yi, J. Kreuziger, P. Baldi, and D. C. Wallace. An enhanced MITOMAP with a global mtDNA mutational philogeny. *Nucleic Acids Research*, 35:D823–D828, 2007. Database Issue.

[31] Y. Sang and H. Shen. Privacy preserving set intersection protocol secure against malicious behaviors. In *PDCAT '07: Proceedings of the Eighth International Conference on Parallel and Distributed Computing, Applications and Technologies*, pages 461–468, Washington, DC, USA, 2007. IEEE Computer Society.

[32] Y. Sang and H. Shen. Privacy preserving set intersection based on bilinear groups. In *ACSC '08: Proceedings of the thirty-first Australasian conference on Computer science*, pages 47–54, Darlinghurst, Australia, Australia, 2008. Australian Computer Society, Inc.

[33] R. F. Service. The race for the $1000 genome. *Science*, 311:1544–1546, 2006.

[34] J. Stuckman and G.-Q. Zhang. Mastermind is np-complete, 2005. http://arxiv.org/abs/cs/0512049.

[35] D. Szajda, M. Pohl, J. Owen, and B. G. Lawson. Toward a practical data privacy scheme for a distributed implementation of the Smith-Waterman genome sequence comparison algorithm. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2006.

[36] J. R. Troncoso-Pastoriza, S. Katzenbeisser, and M. Celik. Privacy preserving error resilient dna searching through oblivious automata. In *CCS '07: Proceedings of the 14th ACM conference on Computer and communications security*, pages 519–528, New York, NY, USA, 2007. ACM.

[37] J. D. Ullman, A. V. Aho, and D. S. Hirschberg. Bounds on the complexity of the longest common subsequence problem. *J. ACM*, 23(1):1–12, 1976.

[38] J. Vaidya and C. Clifton. Secure set intersection cardinality with application to association rule mining. *J. Comput. Secur.*, 13(4):593–622, 2005.

[39] J. C. Venter, M. D. Adams, E. W. Myers, and P. W. Li et al. The sequence of the human genome. *Science*, 291:1304–1351, 2001.

[40] A. C. Yao. Protocols for secure computations. In *Proc. of 23rd Symp. on Foundations of Computer Science*, pages 160–164, Washington, DC, USA, 1982. IEEE Computer Society.

Appendix

Proving that Sequence-Alignment Mastermind Satisfiability is NP-Complete

Recall that in the Sequence-Alignment Mastermind satisfiability problem, we are given a sequence of Mastermind queries, $V_1, V_2, \ldots, V_N$, and the responses, $a(V_1), a(V_2), \ldots, a(V_N)$, each of which is presumed to be the sequence-alignment score of V_i and an unknown string V. We are asked to determine if there exists a vector V that satisfies all of these responses. We give the complete proof here of the following theorem.

Theorem 1: Sequence-Alignment Mastermind Satisfiability is NP-complete.

Proof: Our proof is an adaptation of the NP-completeness proof of Goodrich [20] showing that single-count (black-peg) Mastermind Satisfiability is NP-complete. It is easy to see that Sequence-Alignment Mastermind Satisfiability is in NP. For example, we could nondeterministically guess a vector V and then test in polynomial time whether it satisfies all the responses, $a(V_1), a(V_2), \ldots, a(V_N)$.

To prove that Sequence-Alignment Mastermind Satisfiability is NP-hard, we provide a reduction from 3-Dimensional Matching (3DM), which is a well-known NP-complete problem (e.g., see [18]). In the 3DM problem, we are given three sets, $X = \{x_1, \ldots, x_n\}$, $Y = \{y_1, \ldots, y_n\}$, and $Z = \{z_1, \ldots, z_n\}$, of n elements each. In addition, we are given a set T of m triples, $\{(x_{i_1}, y_{j_1}, z_{k_1}), \ldots, (x_{i_m}, y_{j_m}, z_{k_m})\}$, whose elements are respectively taken from the three sets, X, Y, and Z. The problem is to determine if there is a subset of triples such that each element in X, Y, and Z appears in exactly one triple in this subset.

Suppose, then, that we are given an instance of the 3DM problem, as described above. We consider the unknown vector, V, to consist of the following sequence of variables:

$$(X_1, \ldots, X_{2n};\ Y_1, \ldots, Y_{2n};\ Z_1, \ldots, Z_{2n};\ T_1, \ldots, T_{2m-1}),$$

where the semi-colons are used for the sake of notation to separate the four sections in the unknown vector, V. We perform our reduction by constructing a sequence of guess vectors, $V_0, V_1, \ldots, V_N$, together with their sequence-alignment responses, $a(V_0), a(V_1), \ldots, a(V_N)$, so that there is a satisfying vector V for these responses if and only if there is a solution to the given instance of the 3DM problem.

Our construction begins by setting the number of colors, K, to be $m+2$. Intuitively, there is a color associated with each triple in T, plus a "null" color, ϕ, which is guaranteed to appear nowhere in our unknown vector, V, and a separator color, μ, which occurs in every other (even-indexed) position of V. We begin our sequence of queries with four special "enforcer" queries. The first two of these are

$$V_0 = (\phi, \ldots, \phi;\ \phi, \ldots, \phi;\ \phi, \ldots, \phi;\ \phi, \ldots, \phi),$$

which has response $a(V_0) = 0$, and

$$V_1 = (\mu, \ldots, \mu;\ \mu, \ldots, \mu;\ \mu, \ldots, \mu;\ \mu, \ldots, \mu),$$

which has response $a(V_1) = 3n + m - 1$. Intuitively, V_0 enforces the fact that the null color, ϕ, appears nowhere in the unknown vector, and V_1 enforces the fact that the separator color, μ, appears exactly often enough to separate every other (non-μ) character in the unknown vector. So as to better understand the characteristics of the other queries, let us set $h = 3n + m - 1$, the number of μ colors in our unknown vector V. We then define two additional enforcer queries,

$$\begin{aligned} V_2 \quad = \quad & (\phi, \mu, \ldots, \mu, \phi, \mu;\ \phi, \mu, \ldots, \phi, \mu; \\ & \phi, \mu, \ldots, \phi, \mu;\ 1, \mu, 1, \mu, \ldots, \mu, 1), \end{aligned}$$

which has response $a(V_2) = h + n$, and

$$\begin{aligned} V_3 \quad = \quad & (\phi, \mu, \ldots, \mu, \phi\mu;\ \phi, \mu, \ldots, \mu, \phi\mu; \\ & \phi, \mu, \ldots, \mu, \phi\mu;\ 0, \mu, 0, \mu, \ldots, \mu, 0), \end{aligned}$$

which has response $a(V_3) = h + m - n$. Intuitively, V_2 enforces a counting rule that exactly n of the T_i's will be set to 1, and V_3 enforces a counting rule that the remaining $m - n$ of the T_i's will be set to 0. For each triple, $T_s = (x_{i_s}, y_{j_s}, z_{k_s})$, we construct three query vectors, as follows.

$V_{s,1} =$
$(\phi, \mu, \ldots, \mu, \phi, \mu, s, \mu, \phi, \mu, \ldots, \mu, \phi, \mu;\ \phi, \mu, \ldots, \mu, \phi, \mu;$
$\phi, \mu, \ldots, \mu, \phi, \mu;\ \phi, \mu, \ldots, \mu, \phi, \mu, 0, \mu, \phi, \mu, \ldots, \mu, \phi),$

where the s is in position $2i_s - 1$ in the first group and the 0 is in position $2s - 1$ in the fourth group. This vector has response, $a(V_{s,1}) = h + 1$.

$V_{s,2} =$
$(\phi, \mu, \ldots, \mu, \phi, \mu;\ \phi, \mu, \ldots, \mu, \phi, \mu, s, \mu, \phi, \mu, \ldots, \mu, \phi, \mu;$
$\phi, \mu, \ldots, \mu, \phi, \mu;\ \phi, \mu, \ldots, \mu, \phi, \mu, 0, \mu, \phi, \mu, \ldots, \mu, \phi),$

where the s is in position $2j_s - 1$ in the second group and the 0 is in position $2s - 1$ in the fourth group. This vector has response, $a(V_{s,2}) = h + 1$.

$V_{s,3} =$
$(\phi, \mu, \ldots, \mu, \phi, \mu;\ \phi, \mu, \ldots, \mu, \phi, \mu;$
$\phi, \mu, \ldots, \mu, \phi, \mu, s, \mu, \phi, \mu, \ldots, \mu, \phi, \mu;$
$\phi, \mu, \ldots, \mu, \phi, \mu, 0, \mu, \phi, \mu, \ldots, \mu, \phi),$

where the s is in position $2k_s - 1$ in the third group and the 0 is in position $2s - 1$ in the fourth group. This vector has response, $a(V_{s,3}) = h + 1$. Intuitively, these three responses collectively form a "chooser" gadget, where we will either have $T_{2s-1} = 0$ or the three variables X_{2i_s-1}, Y_{2j_s-1}, and Z_{2k_s-1}, will each be set to have color s (and $T_{2s-1} = 1$). Moreover, note that there are m odd-index positions in the T, and each of them has to match either a 0 or 1 color.

This reduction can clearly be done in polynomial time. So all that remains is for us to show that it works. Suppose, then, that there is a possible solution to the given instance of 3DM. Then for each chosen triple, $T_s = (x_{i_s}, y_{j_s}, z_{k_s})$, we can assign colors $T_{2s-1} = 1$, $X_{2i_s-1} = s$, $Y_{2j_s-1} = s$, and $Z_{2k_s-1} = s$, which will satisfy each of the $V_{s,1}$, $V_{s,2}$, and $V_{s,3}$ vector responses for this value of s. Likewise, setting $T_{2s-1} = 0$ will satisfy each of the $V_{s,1}$, $V_{s,2}$, and $V_{s,3}$ vector responses for a triple T_{2s-1} that is not chosen. Finally, given that there are n chosen vectors, we will satisfy the four preliminary vector responses as well.

Suppose, alternatively, that we have a vector V that satisfies all our vector responses. We know that each X_i, Y_j, and Z_k must be assigned a color other than ϕ. Moreover, every even-indexed position in V must be assigned the color μ and every odd-indexed position must be a color other than μ, because there are exactly $h = 3n + m - 1$ instances of μ in V and we have introduced a query that enforces the fact that there is exactly one non-μ color between every consecutive pair of μ-colored positions. Since there are only $m+2$ colors, this implies each odd-indexed position X_{2i-1}, Y_{2j-1}, and Z_{2k-1} must be assigned a color corresponding to a triple number, s, that is, it is not assigned ϕ or μ. If the corresponding $T_{2s-1} = 1$, then in order to have satisfied the vectors $V_{s,1}$, $V_{s,2}$, and $V_{s,3}$, we must have set $X_{2i_s-1} = s$, $Y_{2j_s-1} = s$, and $Z_{2k_s-1} = s$, which implies we can include the triple $(X_{i_s}, Y_{j_s} Z_{k_s})$ in our matching. If $T_{2s-1} = 0$, then we do not include this triple in our matching. By the vector responses V_2 and V_3, we know that the number of triples chosen in this way is exactly n. Thus, we have found a valid 3-dimensional matching. ■