# **Privacy-preserving, Error-resilient DNA Searching through Oblivious Automata[1]**

## The problem

DNA sequencing is a process by which we can record the sequence of nucleotide base pairs in a person’s genome. This sequence is a string over {A, C, G, T}, each character representing a nucleotide. Every three nucleotides forms a “base pair”, which codes for a particular amino acid.

The Human Genome Project, after almost 13 years and $3 billion USD, has prototyped a typical human genome. This has led to significant progress in anatomy and the medical field at large. Examples of the applications made possible by the human genome include testing whether a patient is predisposed to a genetic condition.

However, unsecured access to a person’s genome information poses significant risks. For example, a person shown to be genetically predisposed to a particular disease may be considered a financial liability by insurance companies, leading to denial of coverage. Along with the more generic risks associated with exposing private health information, genome data is important to keep as secure as possible.

Current DNA sequencing techniques do not guarantee total accuracy, and are faulted through symbol substitutions, insertions and deletions. Nucleotide triplets are also not unique, meaning there are many possible different combinations of triplets that make the same amino acid. As humans evolve, there is never a complete genomic database for all DNA sequences, which leave many sequences to be undetermined.

With all of this in mind, a model for DNA sequencing that both protects the users’ privacy and is not prone to errors is needed.

## Relevance to the course

The solution proposed by the paper is a direct application of a Deterministic Finite Automaton, one of the three models learned in this course. We have learned of more powerful models with even more capabilities like Pushdown Automaton and Turing Machines, but the researchers had chosen DFA to solve this seemingly complex problem. This decision is possibly based on the two goals the Oblivious Automaton set out for. First is security. Any amount of memory (like the stack in a PDA) is prone to compromise, so using a model without one limits that risk. The second is simplicity. More complex solutions may cause more errors than it is worth, not to mention there is little opportunity for cross-validation in the case of the protocol used. Therefore, a simple model like the DFA is well-suited for the task. This shows how powerful the DFA is in real-life applications, even with lesser power and memory than PDAs or TMs.

## General Significance

As mentioned in a previous part, the need for a secure and error-resilient automata protocol for checking private genome information is very important for the healthcare industry. Both parties, the provider and the patient, has the right to privacy in regards to their information. There are already existing laws that protects against discrimination in healthcare coverage [2]. Some of the issues include, but are not limited to, discrimination on the basis of race, origin, pre-existing condtions, which are predictable from human genome information. This automata solution provides extra safeguard on top of regulations.

## Solution

The core of the solution is a protocol consisting of three parts. The first subprotocol initializes the first state, starting with the first input. From then on, the second subprotocol continues progress by distributing states of the following states to both parties. Lastly, when input is complete, the third subprotocol announces the result.

Let A be the party with the DFA’s information, and B the party with the genetic information, both wants to know the acceptance of the DFA on the DNA, using all conventional symbols for the DFA (Q, Σ, Δ, q, F)

1. Subprotocol 1: performs the first state transition of the automaton. Party A creates a random matrix ra and selects a column q of the transition functions as a vector, and blinds every element. After this step, both parties share the next state q of the automaton
2. Subprotocol 2: shares k-th state transition as input. The protocol continues essentially the same as the last subprotocol. To share progress, the protocol lets both parties share the states as input, using an encryption technique on a matrix vector. Each party will only receive the encoded matrix, containing just the length of input, or the number of states, based on the receiver.
3. Subprotocol 3: announcing the result. Once all symbols from B’s string are consumed, A’s DFA determines if it is in a final state or not. The sharing process is still the same as previous steps.

As we discuss the problems of privacy preserving approximate DNA searching and matching, we also need to go further in depth on how this can be solved using the protocol for oblivious automata execution and judge whether it is practical or not. One of the reasons we choose an automaton instead of a dynamic programming algorithm resides in the fact that an automaton has predefined transitions, and it does not need any comparisons while traversing the input sequence. Comparisons are one of the most expensive operations so by using a finite automaton, all the comparisons can be avoided, because they are all hard-wired in the automaton itself. Furthermore, using an automaton allows the implementation of any matching problem represented in the form of a regular expression, endowing our privacy preserving solution with a strong generality.

In order to solve matching, we create a levenshtein automata LEVd(xA) that accepts all strings that have at most Levenshtein distance d from xA. With this, we are able to calculate the Levenshtein distance between two sequences xA and xB and compare it to a given threshold d on input xB which solves the matching problem. Once the Levenshtein automaton for a given sequence is generated, we extend it to accept the language Σ\*Ld(xA)Σ\*. Thus, the resulting automaton accepts any string that contains as substring any of the sequences accepted by the Levenshtein automaton, thus solving the problem of approximate string searching, when the automaton is run on xB. Extending the Levenshtein automaton comprises two concatenations with Σ∗ , the right one being trivial, as it only involves adding self-loops in all the final states. For a fixed d, the algorithm for generating LEVd(xA) given xA is linear in time and space complexity in the length of the string xA, the dependency on d can be at worst exponential but is usually a small parameter compared to the length of xA for practical applications like DNA searching. We know that this protocol allows the efficient privacy-preserving execution of an automaton, but it does not have to be used just for, in fact it can be applied to any problem with a need of privacy preservation that can be stated in terms of a regular expression like password format validation or data parsing. All these applications can also be handled by the protocol presented in this work when there is the need of protecting the recognized sequence.

## Group members

Hugh Hoang - 1833106

Omar Watad - 2036321

Manh Nguyen - 2005493

Jacob Higdon - 1908262

## References

[1] <https://dl-acm-org.ezproxy.lib.uh.edu/doi/pdf/10.1145/1315245.1315309>

[2] <https://www.hhs.gov/civil-rights/for-providers/laws-regulations-guidance/laws/index.html>