Carrier Status Comparison Using Secure Two-Party Computation

Jacob Gasparich jgaspari@mit.edu
Grant Prater pratergc@mit.edu
Dennis Grishin dennis@nebula.org

May 15, 2019

1 Introduction

We are in the age of genetic testing. In the past six years, over 25 million people have purchased direct-to-consumer genetic tests. Additionally, the number of tests sold per year has been increasing exponentially. Because of this growth, it is imperative that secure applications of the data produced from these tests are developed. Genetic data is personal health information that should not be shared with unwanted parties.

The secure application that we are proposing specifically deals with an individual's carrier status for genetic diseases. Most genetic diseases are recessive, meaning that an individual can be any of the following: an unaffected non-carrier who has zero "broken" gene copy and does not exhibit symptoms, an unaffected carrier who has one "broken" gene copy and does not exhibit symptoms, or an affected individual who has two "broken" gene copies and exhibits symptoms. So, two carrier partners who are looking to have a child run a risk of their child receiving two "broken" gene copies and therefore having symptoms of these often terrible genetic diseases.

Having a child with a genetic disease can have a very large impact, so we are discussing a secure application that would allow an individual to check that a potential partner is not a carrier for any of the same genetic diseases as the individual. However, it is very important that the potential partner does not learn which genetic diseases the individual is a carrier for, as this is personal health information that the individual may not want shared.

2 Yao's Protocol

The application we are suggesting makes use of Yao's Protocol, which is an algorithm for doing secure two-party computation. Later on, we will discuss how to apply Yao's protocol to carrier status specifically. But, for now, we will simply be giving an overview of how Yao's protocol works.

2.1 Garbled Circuits

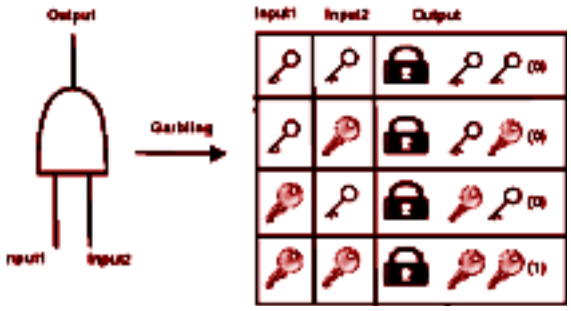

Image Source: https://www.semanticscholar.org/paper/GeSLV-%3A-Garbled-circuit-based-Software-License-Singh-Rajan/53d60c8cff7bf83733daae36e563f72e604c47b3

The first step of using Yao's protocol is converting the problem into a Boolean circuit. A Boolean circuit is a connected collection of logic gates such as AND, OR, XOR, and NOT. Note that a logic gate can be represented by a small truth table that enumerates the output for any possible combination of inputs. Most problems that are looking for a binary output can be converted into a Boolean circuit. However, in order to evaluate this Boolean circuit in a secure manner through Yao's protocol, we must convert this Boolean circuit into a garbled circuit.

A Boolean circuit can be converted into a garbled circuit in the following way. First, each wire (input or output of a logic gate) in the circuit must be given two randomly generated strings, typically referred to as labels. One of these strings will represent a 0 on that wire, and the other will represent a 1. Next, in the truth table for each gate (the logic that guides what the output is for the gate given its inputs), all of the 0's should be replaced by the 0 label for the wire that is associated with that column in the truth table. Essentially, the gates are being changed such that instead of taking in 0's and 1's and outputting a 0 or 1, they now are taking in two randomly generated labels that represent 0's and 1's and outputting a label that represents a 0 or 1. This makes it such that an adversary looking into the circuit that does not know the random labels cannot determine the inputs or outputs at any step.

Next, for each gate the logic is changed further so that instead of simply outputting the label representing a 0 or 1, the gate outputs the encryption of the label representing a 0 or 1. The encryption is done using a double-key symmetric encryption, using both of the input labels for that row as keys. (Note that now the truth table can be represented simply as the output column since rather than giving two inputs to a table and looking up the output that share the inputs' row, we can try the two inputs as keys for all four encrypted labels and select the one for which decryption was successful). Finally, the truth table that stores the logic of the gate has its rows permuted randomly such that an adversary would not know an output value based solely on row position.

These steps result in a garbled circuit which, through Yao's Protocol, can be evaluated in a manner in which neither individual giving inputs learns of the other individual's inputs or the outputs at any intermediate step.

2.2 Oblivious Transfer

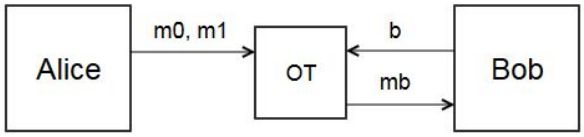

Image Source: https://www.alexirpan.com/2016/02/11/secure-computation.html

Note that one individual ("Individual A") must set up the garbled circuit and create the random labels for the inputs and outputs. This means that Individual A knows both input labels that the other individual ("Individual B") will be using. In order for this circuit to be executed securely, oblivious transfer must be used so that Individual B can select one of the input labels without Individual A knowing which label is selected.

The oblivious transfer protocol as used in Yao's protocol can be described in the following way. First, Individual A generates an a public key and a private key using RSA encryption and sends the public key to Individual B. Individual A also makes two more random strings and indexes them as s_0 and s_1. These strings are sent to Individual B. Next, individual B chooses the string whose index represents the input label they wish to receive (call this s_b for now). Individual B creates a random message m, encrypts it using the RSA algorithm, adds s_b to it, and send the result to Individual A.

Individual A does not know whether s_b is s_0 or s_1, so he or she computes two separate decryptions. One of these is the decryption of the result sent over by Individual B minus s_0, and the other is the decryption of the result sent over by Individual B minus s_1. The result of one of these decryptions will be the random message m that was created by Individual B, but Individual A will not know which one. Then, individual A adds the values of these decryptions to the labels with their respective indices, and sends both altered labels over to individual B. Finally, individual B selects the altered label with the index that he desires and subtracts m. This gives Individual B the label that he or she desires.

Note that in this oblivious transfer protocol, individual A never learns which input label individual B plans to use. Additionally, individual B never gains access to the input label that he or she will not be using.

2.3 Evaluation

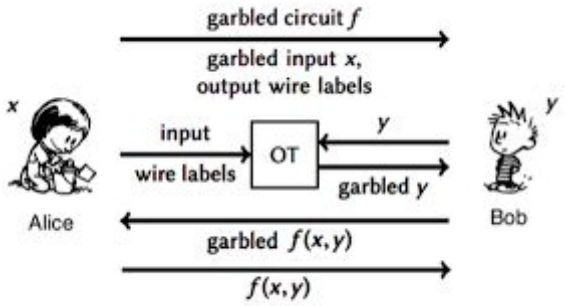

Image Source: https://www.researchgate.net/figure/Communication-flow-in-the-semi-honest-Yaos-protocol_fig5_314797270

Finally, we will discuss how to actually evaluate the garbled circuit with the desired inputs from each of the parties using Yao's protocol.

Individual A sets up the garbled circuit as explained in section 2.1 and sends all of the final truth tables of the garbled circuit to Individual B, along with all of the input labels that represent the data he or she is trying to send.

Individual B uses oblivious transfer as explained in section 2.2 to receive all of the labels for the input data that he or she is trying to send. He or she now has all of input labels and all of the truth tables representing the circuit, so he or she must simply try to decrypt each of the entries of the truth tables with both input labels at that gate, and stop when the decryption is successful and yields an output label. This step is applied on the truth tables for all gates until only a single output label is given.

Individual B then shares this output label and Individual A shares which value (0 or 1) that this label maps to. Both people know the output of the computation, but neither know the other's inputs, and, while evaluating, individual B only had the labels of the intermediary computations. So, neither individual knows the values of the intermediary computations either. Thus, Yao's protocol is successful as a secure two-party computation.

3 Application to Carrier Status

Now that we have discussed a method to do secure two-party computation, we can discuss applying this method to make carrier status comparison possible.

We desire the answer to the following problem. Are two individuals carriers for any of the same genetic diseases? (Note that we are not asking the question: Are two individuals carriers for a

specific genetic diseases? This is because asking that question could immediately reveal the other individual's carrier status for that disease, and therefore would no longer be secure.)

As stated in section 2.1, problems such as this one can be expressed as a Boolean circuit with specific inputs. The following discussions will show how.

3.1 Creating an Input Bit-vector

23andMe and other genetic testing companies typically give users their own genetic sequence and a reference genetic sequence that is an unaffected non-carrier for all genetic diseases. Given both of these sequences, one could use comparison to compute a bit-vector that has 1's for the locations in the sequence where the individual is a carrier and 0's for the locations in the sequence where the individual is not a carrier. This will produce the input data for each of the individuals.

There are an estimated 20,000 genes[1] in the human genome, so we would like the application to be able to support input bit-vector lengths of up to 20,000. However, currently genetic testing sites are only sequencing a small fraction of these genes. Additionally, only a small fraction of the genes that are being sequenced are currently linked to known genetic diseases. For the remainder of this discussion, we will use 20,000 as the length of the input bit-vectors; however, in practice this length would be much smaller.

3.2 Creating the Boolean circuit

The Boolean circuit for this problem would consist of the following. There would be 20,000 two input AND gates that take as input the pair of bits at each index in the bit-vectors. Then, there would be 19,999 two input OR gates that can be treated as a 20,000 input OR gate applied to the results of the AND gates. Using this Boolean circuit as the starting point for Yao's protocol as discussed in section 2 would lead to a secure computation of the problem.

3.3 Use Case

A good application of this process would lead to a mobile app that could have a person's carrier status bit-vector uploaded and, when connected to another individual's device, use Yao's

[1] "What is a gene? - Genetics Home Reference - NIH." 14 May. 2019, https://ghr.nlm.nih.gov/primer/basics/gene. Accessed 18 May. 2019.

protocol to see if the individuals are carriers for any of the same diseases. This could lead to a simple, convenient use case where a couple could quickly get this information, which would help them make decisions about whether or not to move their relationship forward.

4 Security Properties

The two main portions of Yao's protocol which are subject to attack are the oblivious transfer and each gate evaluation. For the 1-out-of-2 oblivious transfer implemented with RSA, the difficulty of the discrete log problem (or invertibility of the cryptographic function used to blind more generally) is the only thing preventing Alice from discovering which input bit Bob chose. As this holds, there is no known security vulnerability for properly implemented oblivious transfer.

For the evaluation, security of the two-key symmetric hash functions used to generate each row of each garbled circuit table is required. If there is potential for an incorrect but valid label to be produced as the output of a gate by the evaluating party, then virtually any output value can be produced. Proving the security properties of double-key encryption is somewhat unintuitive and outside the scope of this paper, but a full, correct proof of Yao's protocol for two parties security is given here.[2] This is all assuming a semi-honest adversary model. Of course, under the basic protocol, if the garbler decides to construct rows in the garbled circuit which do not actually correspond to the intended function to be evaluated, potentially information about the evaluator's input can be gleaned. There are advanced techniques in use to secure Yao's protocol against a malicious adversary, but they are beyond the scope of this paper.

5 Performance Analysis

A number of optimizations to Yao's basic protocol are commonly done and required for short runtime on systems where memory and clock cycles need to be conserved. The most significant speedup can be done by not sending the entire garbled circuit at once, instead doing a topological sort on the gates beforehand then streaming them to be evaluated as they arrive.[3] This not only parallelizes the circuit generation and evaluation, but conserves a significant amount of memory — recall each wire requires two random labels — and can even lead to much of the computing being possible on the cache.

[2] "A Proof of Security of Yao's Protocol for Two-Party Computation." 26 Jun. 2006, https://eprint.iacr.org/2004/175.pdf.
[3] "Faster Secure Two-Party Computation Using Garbled Circuits - Usenix." https://www.usenix.org/events/sec11/tech/full_papers/Huang.pdf.

5.1 Garbled Table Optimizations

Several optimizations to garbled table representations can be done without affecting the security properties of the protocol as a whole. These include point-and-permute, free-XOR, half-AND, and row reduction. Point-and-permute involves a select bit being randomly selected to be appended at the beginning of each wire label for each party. Instead of the rows being permuted randomly, they are ordered by the select bit of the first party, and secondarily by that of the second party. As long as the select bit is chosen randomly and sent obliviously, this does not impact security, and speeds up evaluation by a factor of 2-4, as the evaluator no longer needs to decrypt each row, but can jump immediately to the correct row.

5.2 Free XOR

For free XOR, the labels for each wire are not both randomly generated. Instead, one is randomly generated and an additional string of the same length is randomly generated, with the other label being the XOR of these two. This allows XOR to be computed by XOR-ing the two input labels. A free-XOR full proof of security can be found here.[4]

5.3 Half-AND

Half-AND works on a similar principle. The original paper proposing it[5] gave a thorough analysis of likely speedup, which can be around 30-40% overall. Essentially, by representing AND gates as two XOR's which can both be evaluated for free, only two ciphertexts per AND gate are required.

5.4 Garbled Row Reduction

Garbled row reduction involves setting the ciphertext of the first row of each circuit representation as 0, so it does not need to be sent at all. This is compatible with all of the other optimizations mentioned.

[4] "Improved Garbled Circuit: Free XOR Gates and Applications." http://www.cs.toronto.edu/~vlad/papers/XOR_ICALP08.pdf.
[5] "Reducing Data Transfer in Garbled Circuits using Half Gates." 28 Sep. 2014, https://eprint.iacr.org/2014/756.

6 Implementation Details

Numerous libraries implementing Yao's protocol efficiently exist. The one most appropriate to use for our project was determined to be OblivC[6], due to a number of factors, including the expressiveness of the language, performance benchmarks, and abundance of code examples. A modification of CIL converting OblivC to plain C, OblivC introduces primarily the obliv modifier for C primitives. Conversion of a computation into a garbled circuit, and execution of Yao's protocol are both largely automated.

Conversion of the C code produced by OblivC to be run natively in an Android application is done using the Android Native Development Kit (NDK)[7] for Android Studio. The application is developed by means of a standard SDK, of which Android Studio is the most prominent and straightforward. Deployment via the Google Play store and later Apple Store is the surest method to encourage mass adoption, and a simple intuitive interface which makes the purpose and security properties of the application clear could educate as well as increase ease of use. Ultimately the performance increase and simplicity of a 2-party application, as well as our specific use case make 2-party support the natural route to take, but multi-party extensions, as well as multiple comparison options with variable protection quotients could be included as well, and may be useful.

7 Conclusion

As genetic data becomes increasingly available and analysis increasingly enticing, safeguards against data leaks should be an option. Using the most flexible and powerful Multi-party secure transfer protocol, Yao's protocol, and optimizations to the basic format derived after years of research, our proposed application provides this very safeguard in a convenient way. Unfortunately due to time constraints we were unfortunately unable to complete a working implementation of our proposed application, but are confident that it is possible to do so and the simplicity of the evaluation would lead to a performant application.

[6] "Obliv-C: A Language for Extensible Data ... - Cryptology ePrint Archive." https://eprint.iacr.org/2015/1153.pdf.
[7] "Using C and C++ Code in an Android App with the NDK — SitePoint." 5 Feb. 2016, https://www.sitepoint.com/using-c-and-c-code-in-an-android-app-with-the-ndk/.