Title

Innocent until proven guilty: Privacy-preserving search over a central CODIS criminal database from the field

Authors

Jacob A. Blindenbach[1,*], Karthik A. Jagadeesh[2,*], Gill Bejerano[3,4,5,6,7], David J. Wu[1,7]

Affiliations

[*] These authors contributed equally to this work and will list themselves first on their CVs.

[1] Department of Computer Science, University of Virginia, Charlottesville, VA
[2] Klarman Cell Observatory, Broad Institute of MIT and Harvard, Cambridge, MA
[3] Department of Computer Science, Stanford University, Stanford, CA
[4] Department of Developmental Biology, Stanford University, Stanford, CA
[5] Department of Pediatrics (Medical Genetics), Stanford University, Stanford, CA
[6] Department of Biomedical Data Science, Stanford University, Stanford, CA
[7] Corresponding authors: bejerano@stanford.edu (G.B) and dwu4@virginia.edu (D.J.W)

Abstract

The presumption of innocence (i.e., the principle that one is considered innocent until proven guilty) is a cornerstone of the criminal justice system in many countries, including the United States. DNA analysis is an important tool for criminal investigations[1]. In the U.S. alone, it has already aided in over half a million investigations using the Combined DNA Index System (CODIS) and associated DNA databases[2]. CODIS includes DNA profiles of crime scene forensic samples, convicted offenders, missing persons and more. The CODIS framework is currently used by over 50 other countries[3] including much of Europe, Canada, China and more. During investigations, DNA samples can be collected from multiple individuals who may have had access to, or were found near a crime scene, in the hope of finding a single criminal match[4]. Controversially, CODIS samples are sometimes retained from adults and juveniles despite *not* yielding any database match[4–6]. Here we introduce a cryptographic algorithm that finds any and all matches of a person's DNA profile against a CODIS database *without* revealing anything about the person's profile to the database provider. With our protocol, matches are immediately identified as before; however, individuals who do *not* match anything in the database retain their full privacy. Our novel algorithm runs in 40 seconds on a CODIS database of 1,000,000 entries, enabling its use to privately screen potentially-innocent suspects even in the field.

Introduction

DNA-based forensic analysis is a powerful tool used by law enforcement agencies around the world for solving crimes[1–3]. With today's technology, local police stations[7] and even agents in the field[8] can generate a suspect's DNA profile to search against central criminal DNA databases in an impressive 90 minutes[9,10]. At the same time, the increased prominence of DNA-based forensics opens up new avenues for misuse including social control and racial profiling[11,12].

In particular, the storage of DNA samples from potentially innocent individuals *permanently* links their genetic identities to criminal databases (without due process). Within the United States, these controversial practices have included collecting DNA from individuals who have been arrested but not convicted or even charged with a crime[4], people who are not even arrested (so-called "stop-and-spit" and "swab-and-go" practices[13]), detained immigrants and asylum seekers[6], and even children brought in for questioning[5].

Here we develop a novel solution (Figure 1) whereby an agent in the field, using modest computational resources, could privately query a suspect's DNA profile against a large central database of DNA profiles. In less than 40 seconds, the agent learns whether the suspect's profile matches, based on CODIS rules, against any profile in the central database of 1,000,000 profiles, while learning nothing else about the contents of the central database. More importantly, the central database itself learns absolutely *nothing* about the DNA profile being searched. Any match discovered can be investigated further as before. However, should no match be made, the suspect's DNA profile, now exonerated, can be disposed of on the spot, with zero risk that the central database provider chose to retain it.

Results

CODIS system of DNA profiles and profile matches

The United States CODIS system originally established a set of 13 loci across the genome coinciding with short tandem repeats (STRs) as a method for comparing genetic data for identification purposes[3]. These were expanded to a set of 20 loci in 2017. Many countries have adopted a similar system with a combination of existing core STR loci and region-specific STR loci. For example, a European Union (EU) system of 16 loci, a UK system of 11 loci and a Chinese system of 20 loci have all been described[14,15] (see Online Methods and Supplementary Table 1). At each locus, individuals have 2 alleles, one inherited (with possible personal modification) from each parent. The allele at each locus is represented by a varying range and represents the number of repeats of a 2 to 6-character (base pair) generic sequence observed in the individual's genome. Based on frequency statistics collected by the FBI, the probability that two unrelated individuals share the same STR profile across all 13 core loci positions of the older US system is 1 in 575 trillion[16]. This probability further decreases with the expanded set of 20 loci introduced in 2017.

The CODIS system describes several ways to query a database of STR profiles. The standard and default method is a "high-stringency search with one mismatch," which requires that both of the alleles appearing in at least 19 out of the 20 STR loci between the query profile and the database profile match exactly[17,18]. Central CODIS databases holding thousands to millions of DNA profiles that can be used for such queries are maintained in the US at the national, state, and sometimes even municipal level[2] (as well as very similar databases and matching rules in dozens of other countries[3]).

Encoding an STR profile for secure computation

We encode an STR profile as a binary string. For each STR locus, we propose a public dictionary that maps every pair of STR alleles to a unique binary string. The number of bits used to encode each allele is determined based on the number of unique alleles expected at the locus[14]. We assume that all entries in the central database (unknown to the agent making the query) and the suspect's STR profile held by the agent (unknown to the central database) are encoded in this commonly-agreed upon manner.

Secure protocol overview

It suffices to construct a secure protocol for comparing a single entry in the central database with the suspect's profile. By running this protocol over and over against all entries in the central database, the agent will learn the indices of the complete set of matching records, and nothing else, while the central database will learn nothing (Figure 1C).

We start by fitting a computational model to the task, before later securing it: given two binary strings encoding two STR profiles as above, decide whether they correspond to a match according to the CODIS specification or not. Our key observation is that one can efficiently compute this using a compact deterministic finite automaton[19] (DFA). In a DFA, the computation begins at an initial state, and at each step of the computation, the DFA reads a bit of the input and advances the state. After reading all of the input bits, the DFA ends in either an "accepting" state or a "rejecting" state. For example, a DFA can be constructed to test for equality between two equal length bit-strings by defining two sets of states: a set of "matching" states and another for "mismatching" states. The computation begins in the "matching" set, and as each bit of the input is read, it is compared against the target bit. As long as the current state is in the matching set, if the two bits match, the program transitions to the next state in the matching set, and otherwise, it transitions to a state in the mismatching set. Once a single bit mismatches, one enters the mismatching set, from which all inputs lead only to the next state in the mismatching set. The input bit string is equal to the target string if the computation concludes in a state in the matching set, and is otherwise unequal (see Supplementary Figure 1). We use this DFA to check for matches at a single STR locus. A similar DFA can be used to test for equality with up to one mismatch (see Online Methods and Supplementary Figure 2). We use this one to compute a CODIS match of at least 19 of 20 STR loci.

In our setting, the central database owner constructs a DFA for each profile in its database. The input to the DFA is the agent-held suspect's STR profile. The DFA computation ends in an accepting state if the suspect's profile is a CODIS match to the central database profile; otherwise, the DFA computation ends in a rejecting state.

Our protocol now proceeds as follows. At the beginning, the agent knows the initial state of the DFA as well as the suspect's STR profile (hidden from the central database). The central database holds the DFA corresponding to a database entry (hidden from the agent). Using a cryptographic protocol called "oblivious transfer[20,21]," the agent and the central database now jointly perform the evaluation of the DFA on the agent's input STR profile.

Specifically, at each step of the DFA evaluation, the central database enumerates *all* possible states of the DFA that the agent might be in and all possible states the agent will end up in based on the next bit of the agent's input. These statements are of the form "if you are in state X and the next bit of your input is 0, then you will proceed to state Y." Using the oblivious transfer protocol, the agent can secretly choose to learn exactly one of these statements *without* revealing which one she chose to the database

server. In this case, the agent chooses the statement corresponding to her current state and the next bit of her input; this in turns reveals to the agent her next state in the DFA evaluation. The oblivious transfer protocol hides all of the other statements from the agent, so the agent cannot learn what would have happened had she been in a different state or had a different input.

At the very end of the protocol, the agent arrives at either an accepting state or a rejecting state, which indicates whether the STR profile she provided matched against the central database's profile or not. As described so far, the agent learns the full execution path in the DFA as well as whether her input profile matches the database profile or not. To ensure that the execution path does not leak additional information about the profiles on the database server, the central database additionally encrypts all of the intermediate steps of the computation (in a way that still enables the above evaluation procedure). Then, at each of the intermediate steps, the agent no longer knows where she is in the actual DFA execution. The central database only provides a single intelligible state: the very last state which reveals whether the two profiles match or not match.

By repeating the above protocol with each profile in the central database, while changing the encryption of intermediate states every run, the agent has learned just one thing: the database indices of any and all profile matches. She learns nothing else about any of the centrally-stored profiles. On the flip side, the central database learns nothing at all about the suspect's STR profile; at every step, they only provide an exhaustive list of "if you are here, and have this bit next, then go there." We have thus achieved the desired goal of Figure 1C. We encourage our readers to refer to the Online Methods for the full technical details and security analysis.

Performance measurements

With our protocol implementation, an agent in the field in Northern California can privately query a CODIS database containing 1 million STR records located 3,000 miles away in Northern Virginia in just 38 seconds, using 180 MB of online communication. In our experiments, we represent each STR profile as a vector of 20 biallelic components (212 bits in total) based on the current US CODIS specification[2]. Our protocol additionally requires preparing 116 million oblivious transfer correlations[20,21] and 122 MB of client-side storage (see Online Methods and Supplementary Table 2). These can be generated in a separate preprocessing phase on commodity hardware in about a minute using existing state-of-the-art oblivious transfer extension protocols[22]. Since these correlations are independent of both the query and the database contents, they can be prepared concurrently with the 90 minutes needed for STR profile derivation[9,10], and thus, contributes no extra latency.

We also measured the performance of our protocol on CODIS specifications reported for the UK[14], the EU[14,18], and China[15] (see Online Methods for specifications details). In all cases, the cost of the protocol is smaller or comparable to that of the US CODIS system (up to 40 seconds and under 200 MB of communication; see Table 1). To illustrate the scalability of our approach, we also measured the performance for a CODIS system with 40 loci and an encoding length of 492 bits, a setting based on a system previously tested by NIST[23]. Performing a CODIS search in this setting over a database of a million records completes in just 72 seconds and requires only 340 MB of communication. Thus, our protocol also scales favorably to future scenarios with an expanded set of STR loci.

We also tested the performance of our protocol on central databases of different sizes (Figure 2). Both the online bandwidth and execution time of our protocol scale linearly with the size of the STR profile (a function of both the number of loci and the number of bits needed to represent the allele values at each locus) and with the size of the central database. For example, our protocol can search over a 20-STR U.S. database with 10 million records in just under 6 minutes.

Discussion

DNA profiles are an extremely powerful tool in forensics and crime solving[24]. Our law enforcement agencies have a duty to both serve and protect their communities. How can they use DNA databases to find criminals while simultaneously protecting the privacy of innocent individuals who make up the majority of each society[12]? It is natural to think that searching a suspect's DNA profile against a master database either requires the database to see the suspect's profile, or for the agent holding the suspect profile to have a local copy of the master database. Both scenarios compromise privacy (Figure 1). Here we show a third way, whereby a field agent searches the remote master database without learning anything about it except any possible match. Moreover, the central database aids the search while learning *nothing* about the suspect's profile. Should the search come up empty, if the field agent disposes of the sample, they have both served their community and protected its privacy.

DNA profiling machines can now routinely produce a searchable profile in a matter of 90 minutes, and can even be carried to the field[8]. Integrating our privacy-preserving protocol with such a system adds minimal overhead and can easily fit into existing workflows. In fact, the computational requirements to run the protocol are so modest, it is likely they can be performed on a modern smartphone. The performance of our protocol also compares favorably against generic approaches for privacy-preserving computation. For instance, a direct implementation of a query protocol using Yao's garbled circuits[25] would require communicating, storing, and evaluating a circuit that is around 8 GB (≈ 260 million AND gates) in size for a database with 1 million profiles (and increase to 80 GB of communication and storage for a 10 million entry database). In comparison, our protocol requires less than 125 MB of offline storage for the OT correlations (which can be generated in about a minute[22]) for a database of 1 million profiles.

Our implementation follows the CODIS guidelines for high-stringency matching (the default mode of searching)[17,18].The deterministic finite automata at the root of our approach can be easily extended to also support moderate and low-stringency matches as well as partial match queries, with modest increases in computation time and communication. In fact, similar operations like paternity testing and ethnicity identification can also be formulated as a similar string-matching problem and implemented using a similar approach. In all cases, the correct answer is obtained while the input DNA profiles to the computation remain private.

Our protocol is relevant not only in the US, but also in any of dozens of countries that use a CODIS-like system[3]. It scales well with the size of the central database (Figure 2), on current hardware that will only get faster, and most importantly, it gives the agent in the field or local office, the ability to destroy an exonerated profile that has yielded no incriminating match. Whether they are instructed to do so or not is a civil rights matter that each country must resolve for itself[11,12,26,27]. The importance of our work is in showing that accurate practical implementations to enable these fundamental rights are already doable.

Online Methods

STR profiles and representation

The US National Institute of Standards and Technology (NIST) has enumerated all possible allele values for each STR core locus in the current US system[14]. While over 50 other countries reportedly use a local variant of the same CODIS system[3], details of each country's system, or even a list of these 50+ countries are not easily found. Possibly current versions of the EU and UK systems are described by NIST[14], while a partial description of the Chinese system is provided in a paper co-authored by officials from the Chinese Ministry of Justice[15]. Similarly, the high-stringency matching rule with at most one mismatching locus is published for the US[17] and EU[18]. As the exact CODIS system parameters are immaterial for the essence of our proposed solution, when necessary, we infer the number of possible values at a locus based on the existing NIST standards, and also assume a default search configuration of high-stringency matching of all but one locus for all systems.

In 2017, the US 13 loci system was replaced by the 20 loci system[3]. It is unclear if or when this set will be expanded on. For purposes of illustrating the scalability of our solution we use a 40 loci system described in a NIST paper partly funded by the FBI[23].

Security model

We work in a two-party setting where the server holds a profile $\boldsymbol{v} = (v_1, \dots, v_n)$ and the client holds a query vector $\boldsymbol{w} = (w_1, \dots, w_n)$, where each of the components $v_i, w_i \in \{0,1\}^\ell$ can be represented by ℓ-bit strings. We define the "threshold matching" function $\mathrm{TM}_k(\boldsymbol{v}, \boldsymbol{w})$ to be the following Boolean-valued function:

$$\mathrm{TM}_k(\boldsymbol{v}, \boldsymbol{w}) = \begin{cases} 1, & |\{v_i \neq w_i \mid 1 \leq i \leq n\}| \leq k \\ 0, & |\{v_i \neq w_i \mid 1 \leq i \leq n\}| > k \end{cases}$$

In words, $\mathrm{TM}_k(\boldsymbol{v}, \boldsymbol{w})$ outputs 1 if the vectors $\boldsymbol{v}$ and $\boldsymbol{w}$ disagree on at most k components. In this work, we design a secure protocol such that at the end of the protocol, the client learns the value of $\mathrm{TM}_k(\boldsymbol{v}, \boldsymbol{w})$ and nothing more about the server's profile $\boldsymbol{v}$, while the server learns nothing about the client's query $\boldsymbol{w}$**.** Importantly, the client only learns whether the number of differing components between $\boldsymbol{v}$ and $\boldsymbol{w}$ is greater than k or not, but nothing about the exact number of differing components or the indices of the differing components.

We can naturally extend this notion to the setting where the server holds a database with many vectors $\{\boldsymbol{v}_1, \dots, \boldsymbol{v}_t\}$ and the client's goal is to learn all of the indices $i \in \{1, \dots, t\}$ where $\mathrm{TM}_k(\boldsymbol{v}_i, \boldsymbol{w}) = 1$, but nothing more about the vectors $\boldsymbol{v}_1, \dots, \boldsymbol{v}_t$. Since a protocol for private evaluation of $\mathrm{TM}_k(\boldsymbol{v}, \boldsymbol{w})$ for a single entry $\boldsymbol{v}$ suffices for searching over a database of values (by repeating the protocol for each database entry $\boldsymbol{v}_1, \dots, \boldsymbol{v}_t$), we focus on the single-instance setting for the remainder of this section.

Throughout this work, we assume that the client and the server are semi-honest or "honest-but-curious;" namely, both the client and the server will follow the protocol as described, but may try to infer additional information about the other party's private inputs based on the messages they receive in the protocol. We note that our protocol actually ensures privacy of the client's query even against a *malicious* database server that may arbitrarily deviate from the protocol execution, provided that the underlying "oblivious transfer" protocol we use (see below) is secure against a malicious sender.

Oblivious transfer and the OT-hybrid model

An oblivious transfer (OT) protocol[20,21,28] is a two-party protocol between a sender and a receiver. In a 1-out-of-k OT protocol on t-bit messages, the sender holds k messages $m_1, \ldots, m_k \in \{0,1\}^t$ while the receiver holds an index $i \in \{1, \ldots, k\}$. At the end of the protocol, the receiver learns the i^{th} message m_i and nothing else about any of the remaining messages, while the sender learns nothing. To facilitate the analysis of our protocol, we will work in the "OT-hybrid" model where we assume that the parties have access to a *trusted party* that implements the above 1-out-of-k OT functionality[20]. We can then replace the trusted party with a cryptographic implementation of a 1-out-of-k OT protocol[28]. If our protocol provides semi-honest security in the OT-hybrid model, and we instantiate the OT protocol with a cryptographic protocol that is secure against semi-honest adversaries, then the overall protocol is also secure against semi-honest adversaries (*without* relying on any trusted party)[29].

Oblivious transfer correlations

We can significantly reduce the *online* cost of oblivious transfer by first precomputing *input-independent* "oblivious transfer correlations" in an offline (or preprocessing phase). Because the correlations are input-independent, they can be precomputed without knowledge of the client's query or the server's database. These OT correlations can be generated efficiently using a technique called OT extension in a separate input-independent preprocessing step[30] (this can even be done with low communication using a recent approach called silent OT extension[31]). Alternatively, they can be generated ahead of time by a trusted dealer or a secure hardware platform (observe that in both of these settings, the party generating the correlations does *not* need to know anything about the query or the database entries).

Very briefly, a 1-out-of-k OT correlation for t-bit messages consists of the following: (1) a tuple of k random values $r_1, \ldots, r_k \in \{0,1\}^t$ for the server; and (2) a random index $\beta \in \{1, \ldots, k\}$ together with the value r_β for the receiver. Once the client and server have this OT correlation, it is straightforward to implement a 2-message 1-out-of-k OT (on an arbitrary collection of sender messages $m_1, \ldots, m_k \in \{0,1\}^t$ and receiver index $i \in \{1, \ldots, k\}$). We recall the construction below:

- **Receiver message:** The receiver sends the index $j = i + \beta \pmod k$ to the sender. Observe that since β is uniformly random over $\{1, \ldots, k\}$ and unknown to the sender, this message perfectly hides the receiver's index i. For notational convenience, we will consider the output of arithmetic modulo k to be a value between 1 and k (as opposed to 0 and $k-1$).
- **Sender response:** On input an index $j \in \{1, \ldots, k\}$ from the receiver, the sender computes the blinded message $c_i \leftarrow m_i \oplus r_{j-i \pmod k}$, where $\oplus$ denotes the bitwise exclusive-or operator (i.e., bitwise xor). The sender sends the blinded messages $c_1, \ldots, c_k$ to the receiver. Since $r_1, \ldots, r_k$ are uniform over $\{0,1\}^t$ and the receiver knows exactly one of these values (i.e., r_β), $k-1$ out of the k values are perfectly hidden from the receiver.
- **Receiver reconstruction:** The receiver computes its message as $c_i \oplus r_\beta$.

Correctness of the protocol follows from the following simple relation:

$$c_i \oplus r_\beta = m_i \oplus r_{j-i \pmod k} \oplus r_\beta = m_i \oplus r_{i+\beta-i \pmod k} \oplus r_\beta = m_i \oplus r_\beta \oplus r_\beta = m_i.$$

To summarize, a 1-out-of-k OT correlation on t-bit values yields a 1-out-of-k OT on t-bit messages in two rounds of interaction where the total communication is $\lceil \log k \rceil + kt$ bits. The only necessary computation is integer arithmetic and bitwise operations, so this is a very lightweight protocol.

Representing threshold matching as a deterministic finite automaton

Given two vectors $\boldsymbol{v} = (v_1, \dots, v_n)$ and $\boldsymbol{w} = (w_1, \dots, w_n)$ where $v_i, w_i \in \{0,1\}^\ell$, our objective is to securely compute the threshold matching function $\mathrm{TM}_k(\boldsymbol{v}, \boldsymbol{w})$ defined above. In our setting, we assume the database server holds $\boldsymbol{v}$ while the client holds $\boldsymbol{w}$. The starting point of our design is to express the computation of TM_k as a composition of two deterministic finite automatons (DFAs):

- The first DFA checks equality of ℓ-bit strings. Namely, for a string $v \in \{0,1\}^\ell$, the machine computes the function $g_v(w)$ that outputs 1 if $v = w$ and 0 otherwise.
- The second DFA computes a threshold function. Namely, for a target sequence of bits $a_1, \dots, a_n \in \{0,1\}$ and a threshold k, the machine computes the function $h_{(a_1,\dots,a_n),k}(b_1, \dots, b_n)$ that outputs 1 if $a_i \neq b_i$ on at most k indices $1 \leq i \leq n$ and 0 otherwise.

By definition, the threshold matching function TM_k can now be expressed as

$$\mathrm{TM}_k(\boldsymbol{v}, \boldsymbol{w}) = h_{(1,1\dots,1),k}\left(g_{v_1}(w_1), \dots, g_{v_n}(w_n)\right).$$

We note that while we can construct a *single* DFA that combines both functionalities, decomposing the computation into two separate steps enables a more efficient privacy-preserving protocol.

We now show how to construct simple DFAs for computing the functions g_v and $h_{(a_1,\dots,a_n),k}$. For both functions, we design a "layered" DFA, which can be more efficiently computed in a privacy-preserving manner. First, a DFA consist of a tuple $M = (Q, \Sigma, \delta, q_0, S)$, where Q denote the set of states, Σ is the alphabet, $\delta: Q \times \Sigma \to Q$ is the state-transition function, $q_0 \in Q$ is the start state, and $S \subseteq Q$ is the set of accepting states. On input $x = x_1 x_2 \cdots x_n \in \Sigma^n$, the output $M(x)$ is 1 if $\delta(q_{n-1}, x_n) \in S$ where $q_i = \delta(q_{i-1}, x_i)$ for all $1 \leq i < n$, and 0 otherwise. Finally, we say that $M = (Q, \Sigma, \delta, q_0, S)$ is a layered DFA if the following properties hold:

- The set of states Q can be partitioned into $\bigcup_{i=0}^{n} Q_i$ where $Q_0, \dots, Q_n$ are pairwise disjoint, $Q_0 = \{q_0\}$, and $S \subseteq Q_n$.
- The state-transition function δ can be decomposed into a collection of functions $(\delta_1, \dots, \delta_n)$ where $\delta_i: Q_{i-1} \times \Sigma \to Q_i$ and $\delta(q, \sigma) = \delta_i(q, \sigma)$ for all $1 \leq i \leq n, q \in Q_{i-1}, \sigma \in \Sigma$.

In words, a layered DFA is one whose states can be partitioned into a collection of $n + 1$ pairwise disjoint sets (i.e., "layers") $Q_0, \dots, Q_n$. On input $x \in \Sigma^n$, the state of the DFA after reading the first i bits of x is contained in layer i (i.e., in the set Q_i). We now describe how to represent g_v and $h_{(a_1,\dots,a_n),k}$ as layered DFAs:

- Let $g_v(w)$ denote the function that outputs 1 if $v = w$ and 0 otherwise, where $v, w \in \{0,1\}^\ell$. It is easy to construct a layered DFA for the equality function. The DFA consists of two branches, each with ℓ states: an "accept" branch that corresponds to a matching input and a "reject" branch that corresponds to a non-matching input. Evaluation begins on the accept branch and successively compares the bits of w to the bits of v (encoded in the DFA transitions). If a mismatch is encountered, the DFA transitions to the reject branch. The output is 1 if the final state is on the accept branch and 0 otherwise. We illustrate this in Figure 1. Thus, for $v \in \{0,1\}^\ell$, the function g_v can be computed by a layered DFA with $\ell + 1$ layers, each containing up to 2 states. Note that the vector v is entirely encoded in the transition function δ of the DFA. Our

protocol for privacy-preserving layered DFA evaluation assumes that the topology of M is public, but will hide the transition function δ (and thus, the value of v) from the client.

- Let $h_{(a_1,\ldots,a_n),k}(b_1,\ldots,b_n)$ be the threshold function that outputs 1 if there are at most k indices $1 \le i \le n$ where $a_i \neq b_i$ and 0 otherwise. It is also straightforward to construct a layered DFA for $h_{(a_1,\ldots,a_n),k}$. Namely, we consider a DFA with $k+2$ branches. We label each branch with an integer $j = 0,1\ldots,k,k+1$, corresponding to the number of mismatches encountered thus far. Evaluation begins on branch 0 and whenever the DFA reads in an input bit $b_i \neq a_i$, then the state transitions from branch j to branch $j+1$; otherwise, evaluation continues on branch j. All state transitions on the final branch $j = k+1$ (corresponding to an input that differs on more than k indices), remain on the branch irrespective of the input bit. We illustrate this in Supplementary Figure 2 (for the case where $k = 1$). Thus, the function $h_{(a_1,\ldots,a_n),k}$ can be implemented by a layered DFA with $n+1$ layers, each containing up to $k+2$ states. Note that the sequence of bits $(a_1,\ldots,a_n)$ is entirely encoded in the transition function δ of the DFA. The topology of the DFA is a function of the dimension n and the threshold k, both of which will be assumed to be publicly known in our protocol; our final protocol will require that the target bits $(a_1,\ldots,a_n)$ be hidden, which holds as long as the protocol for layered DFA evaluation hides δ.

It is easy to see that the two layered DFAs described above can be combined into a single layered DFA for computing TM_k. However, the number of layers in the resulting DFA will be the *product* of the number of layers in the two underlying DFAs. When developing our cryptographic protocol for privacy-preserving layered DFA evaluation, both the round complexity and the communication complexity scales with the number of layers. Thus, privately-evaluating two smaller DFAs yields a significantly more efficient protocol compared with evaluating a single larger layered DFA.

Privacy-preserving evaluation of a layered DFA

We now describe how to use oblivious transfer to construct a privacy-preserving protocol for computing TM_k. We leverage our implementation of the threshold matching function as a *layered* DFA to enable a more efficient protocol. While there already exist protocols for private evaluation of general (i.e., not necessarily layered) DFAs [32,33], in this work, we show that the layered structure of our DFAs are amenable to a simpler and direct OT-based evaluation procedure (without needing to additionally rely on heavier cryptographic primitives such as homomorphic encryption).

Let $M = (Q, \Sigma, \delta, q_0, S)$ be a layered DFA: namely, $Q = \bigcup_{i=0}^{n} Q_i$ and $\delta = (\delta_1, \ldots, \delta_n)$. We will label the states Q_i in layer i by indices $1,2,\ldots,|Q_i|$. In our model, the server holds the layered DFA M while the client holds the input $x \in \Sigma^n$. At the end of the protocol, the server should learn nothing while the client should learn the value $M(x)$. In addition, we assume that the *topology* of the DFA M is publicly-known (i.e., the client knows the number of layers in M as well as the number of states in each layer of M). What is hidden is the transition function δ. This assumption is true for the setting we consider in this work: here, the CODIS algorithm itself is public (and this determines the topology of the DFA), but the records within the server's database determine the exact transition function, which is precisely what our protocol hides.

Our protocol relies on the following simple observation: the transition function δ_i can be described by a truth table of size $|Q_i| \cdot |\Sigma|$. This yields the following general approach for evaluating M privately:

- The client initializes her state to $s_0 \leftarrow q_0 \in Q_0$.

- Let $x \in \Sigma^n$ be the client's input. We maintain the following invariant. At the beginning of round $i = 1, \dots, n-1$, the client knows the state $s_{i-1} \in Q_{i-1}$ of the DFA after reading the prefix $x_1 \cdots x_{i-1}$. The client wants to learn $s_i := \delta_i(r_{i-1}, x_i)$. To do so, the client and server run a 1-out-of-$(|Q_i||\Sigma|)$ OT where the receiver's input is (s_{i-1}, x_i) and the server associates the message $y_i = \delta_i(q, \sigma)$ with index $(q, \sigma) \in Q_{i-1} \times \Sigma$ is. At the end of this protocol, the client learns the state $s_i = \delta_i(s_{i-1}, x_i) \in Q_i$ of the DFA after reading $x_1 \cdots x_i$.
- After $n-1$ rounds, the client learns the state $s_{n-1} \in Q_{n-1}$. For the final round, the client and server run a 1-out-of-$(|Q_{n-1}||\Sigma|)$ OT protocol where the client's input is (s_{n-1}, x_n) and the server associates the value 1 with the index $(q, \sigma) \in Q_{n-1} \times \Sigma$ if $\delta_n(q, \sigma) \in S$ (i.e., $\delta_{n-1}(q, \sigma)$ is an accepting state) and value 0 for the remaining indices (q, σ) where $\delta_n(q, \sigma) \notin S$.

Correctness of the above protocol follows by correctness of the underlying OT protocol. In particular, OT is used to iteratively evaluate the layered transitioned functions $\delta_1, \dots, \delta_n$. Moreover, privacy for the client's input is ensured by security of the OT protocol since the server's view in the entire protocol execution only consists of its view in the OT queries (which hide the client's input).

However, this protocol does *not* provide privacy for the server. Namely, the client learns the sequence of states corresponding in the DFA evaluation on her input, which could reveal information about the state-transition functions. Consider for instance a layered DFA with k branches where the node in branch j of layer i is always labeled with the index j. Then, the sequence of states the client obtains by executing the above protocol completely reveals which branch of the computation her input takes at each step in the DFA evaluation. This in turns leaks information about the transition function δ_i (e.g., the client learns whether $\delta_i(s_{i-1}, \sigma)$ outputs a state in the *same* branch or in a *different* branch). In the case of our threshold matching DFA, this in turn reveals to the client partial matches between her query and the server's database entry.

The above attack shows that additional randomization is *necessary* to ensure security for the server. The fix is simple: the server simply *blinds* the index of each state in each round of the protocol. The full protocol is described below:

1. The client initializes her current state to $s_0 \leftarrow 1$. The server initializes $\alpha_0 \leftarrow 0$.
2. On round $i = 1, \dots, n-1$ of the protocol, the client makes an OT query on the index (s_{i-1}, x_i). On round i of the protocol, the server samples a random blinding factor $\alpha_i \leftarrow \{1, \dots, |Q_i|\}$. It prepares a table of size $|Q_i| \cdot |\Sigma|$ where the message y_i associated with index $(q, \sigma) \in Q_{i-1} \times \Sigma$ is $y_i = \delta_i(q - \alpha_{i-1} \bmod |Q_{i-1}|, \sigma) + \alpha_i \bmod |Q_i|$. The server uses this collection of $|Q_i| \cdot |\Sigma|$ as its set of messages in the OT protocol. Let $s_i \in \{1, \dots, |Q_i|\}$ be the client's output in the OT protocol.
3. On the final round of the protocol, the client and server run a 1-out-of-$(|Q_{n-1}||\Sigma|)$ protocol where the client's input is (s_{n-1}, x_n) and the server associates the message 1 with index $(q, \sigma) \in Q_{n-1} \times \Sigma$ if $\delta_n(q - \alpha_{n-1} \bmod |Q_{n-1}|, \sigma) \in S$ and 0 otherwise.

Correctness. Let $z_0 = 1, z_1, \dots, z_n$ be the sequence of states in the evaluation of $M(x)$. Namely, $z_i = \delta_i(z_{i-1}, x_i)$ for each $1 \le i \le n$. First, we show that for each $0 \le i \le n$, $s_i - \alpha_i = z_i \bmod |Q_i|$. We proceed inductively. The claim trivially holds for $i = 0$. By correctness of the OT protocol,

$$s_{i+1} = \delta_{i+1}(s_i - \alpha_i \bmod |Q_i|, x_{i+1}) + \alpha_{i+1} \bmod |Q_{i+1}| = \delta_{i+1}(z_i \bmod |Q_i|, x_{i+1}) + \alpha_{i+1} \bmod |Q_{i+1}|.$$

Thus,

$$s_{i+1} - \alpha_{i+1} = \delta_{i+1}(z_i, x_{i+1}) \bmod |Q_{i+1}| = z_{i+1} \bmod |Q_{i+1}|,$$

and the claim holds. In the final OT, the output is 1 if

$$\delta_n(s_{n-1} - \alpha_{n-1} \bmod |Q_{n-1}|, \sigma) = \delta_n(z_{n-1}, x_n) = z_n \in S,$$

and 0 otherwise. Similarly, the output of $M(x)$ is 1 if and only if $z_n \in S$, and correctness follows.

Security. We show that in the OT-hybrid model, the above protocol is secure in the presence of semi-honest adversaries. Recall that in the OT-hybrid model, the client and server interact with an "ideal" OT functionality (i.e., the client sends an index $i \in \{1, \dots, k\}$ to the OT functionality, the server sends a collection of messages $m_1, \dots, m_k$, and the OT functionality replies to the client with m_i). If we then instantiate the OT with any concrete protocol that is secure against semi-honest adversaries, we obtain a secure protocol for evaluating layered DFAs. We consider client and server security separately:

- **Client Security (Query Privacy).** In the OT-hybrid model, the server only provides inputs to the ideal OT functionality. It does not receive any message from either the client or the ideal OT functionality. Thus, the view of the server is *independent* of the client's query so security for the client against a semi-honest server is immediate.
- **Server Security (Function Privacy).** In the OT-hybrid model, the client receives a sequence of values $s_1, \dots, s_n$ from the ideal OT functionality. From the above correctness analysis, we have that $s_n = M(x) \in \{0,1\}$. By construction (and correctness of the OT protocol), each of the intermediate indices s_i for $1 \leq i < n$ satisfies $s_i = \delta_i(s_{i-1}, x_i) + \alpha_i \bmod |Q_i|$. Here, the server samples each α_i uniformly from the set $\{1, \dots, |Q_i|\}$, and in particular, independently of δ_i. This means that the value of each s_i is uniform and independent over $\{1, \dots, |Q_i|\}$. (Formally, we can construct a simulator that on input $M(x)$ outputs random indices $s_i \leftarrow \{1, \dots, |Q_i|\}$ for $1 \leq i < n$ and $s_n = M(x)$. The output of this simulator is identically distributed as the receiver's view in the protocol. Thus, the protocol provides perfect security in the OT-hybrid model.)

Efficiency. For a layered DFA with $n + 1$ layers, the protocol as described requires n rounds of communication, where on round $1 \leq i \leq n$, the client and server perform a single 1-out-of-$(|Q_{i-1}||\Sigma|)$ OT on $\lceil \log |Q_i| \rceil$-bit messages. Using precomputed OT correlation, the total communication in bits is then

$$\sum_{i=1}^{n} \lceil \log(|Q_{i-1}||\Sigma|) \rceil + \sum_{i=1}^{n} |Q_{i-1}||\Sigma| \lceil \log |Q_i| \rceil.$$

Secure evaluation of the threshold matching function

We now describe how to use our layered DFAs evaluation protocol to obtain a secure protocol for the threshold matching function $\mathrm{TM}_k(\boldsymbol{v}, \boldsymbol{w})$. As discussed above, we can write $\mathrm{TM}_k(\boldsymbol{v}, \boldsymbol{w}) = h_{(1,1,\dots,1),k}\left(g_{v_1}(w_1), \dots, g_{v_n}(w_n)\right)$ and also represent the functions g_{v_i} and $h_{(1,1,\dots,1),k}$ as layered DFAs. We can leverage our protocol for secure layered DFA evaluation to evaluate g_{v_i} and $h_{(1,1,\dots,1),k}$. However, directly applying our privacy-preserving protocol for layered DFAs does not yield a secure protocol for evaluating TM_k since such a protocol would leak the intermediate values $g_{v_1}(w_1), \dots, g_{v_n}(w_n)$ to the client. This means the client would learn the *exact* set of components that match between its vector $\boldsymbol{w}$ and the server's vector $\boldsymbol{v}$ (irrespective of whether the threshold is satisfied or not). We address this by *blinding* the output of the equality-checking circuits.

As currently defined $g_{v_i}(w_i)$ outputs 1 if $v_i = w_i$ and 0 otherwise. We can blind the equality bit by having the server sample a uniform bit $a_i \leftarrow \{0,1\}$ and define $g'_{v_i,a_i}(w_i) = a_i$ if $v_i = w_i$ and $1 - a_i$ otherwise. We make a few simple observations:

- If we have a layered DFA for computing g_{v_i}, then the same DFA (from Figure 1) can be used to compute g'_{v_i,a_i} by swapping the accept and reject states in the final layer of the DFA.
- Let $\beta_i \leftarrow g'_{v_i,a_i}(w_i)$. By construction, this means that $\beta_i \oplus a_i \oplus 1 = g_{v_i}(w_i)$. If the client and server apply the privacy-preserving protocol for evaluating a layered DFA to g'_{v_i,a_i}, then at the end of the protocol, the client learns β_i (and nothing more) while the server learns nothing. In this case, β_i is uniformly random (it is perfectly hidden by a_i), and thus, the client does not learn anything about the value of $g_{v_i,a_i}(w_i)$. (More precisely, we say that the client and the server have a "secret sharing" of the negated equality bit $g_{v_i,a_i}(w_i) \oplus 1$.)
- The client and server use the privacy-preserving protocol for computing layered DFAs to compute the quantity

$$h_{(a_1,\dots,a_n),k}\left(g'_{v_1,a_1}(w_1),\dots,g'_{v_n,a_n}(w_n)\right) = h_{(1,1,\dots,1),k}\left(g_{v_1}(w_1),\dots,g_{v_n}(w_n)\right).$$

The overall protocol is now given as follows:

1. The server begins by sampling $a_1,\dots,a_n \leftarrow \{0,1\}$. For each $i = 1,\dots,n$, the client and server execute the protocol for layered DFA evaluation for the function g'_{v_i,a_i}. On the i^{th} iteration of the protocol, the client provides w_i as its input. Let $b_1,\dots,b_n \in \{0,1\}$ be the client's output in the n protocol executions.
2. The client and server execute the protocol for layered DFA evaluation for the function $h_{(a_1,\dots,a_n),k}$ where the client uses $b_1,\dots,b_n$ as its input. The output z is the result of the threshold matching function $\mathrm{TM}_k(\boldsymbol{v},\boldsymbol{w})$.

Correctness. Correctness of the above protocol follows via correctness of the underlying layered DFA evaluation protocol. Take any two inputs $\boldsymbol{v}$ and $\boldsymbol{w}$. Then, we have that $b_i = g'_{v_i,a_i}(w_i)$ for all $1 \le i \le n$. The client's output at the end of the computation is

$$h_{(a_1,\dots,a_n),k}(b_1,\dots,b_n) = h_{(a_1,\dots,a_n),k}\left(g'_{v_1,a_1}(w_1),\dots,g'_{v_n,a_n}(w_n)\right) = h_{(1,1,\dots,1),k}(g_{v_1}(w_1),\dots,g_{(v_n)}(w_n),$$

which is precisely the value of $\mathrm{TM}_k(\boldsymbol{v},\boldsymbol{w})$.

Security. Security of the protocol also follows by security of the underlying protocol for evaluating layered DFAs. As before, we consider client and server security separately:

- **Client Security (Query Privacy).** The threshold matching protocol consists of $n + 1$ invocations of the protocol for layered DFA evaluation. Security of our protocol for layered DFA evaluation ensures that the server does not learn anything about the client's input in any of these protocol executions, and security follows. (More formally, security of the layered DFA evaluation protocol implies that the server's view can be simulated *without* knowledge of the client's input or output, and correspondingly, the server's view in the threshold matching protocol can be simulated by simply running the simulator for the underlying layered DFA evaluation protocol).
- **Server Security (Database Privacy).** Security for the server follows similarly from security of the underlying layered DFA evaluation protocol. In the above protocol, the client learns the values

of $b_1, \ldots, b_n$ and z. From the correctness analysis, we have that $z = \mathrm{TM}_k(\boldsymbol{v}, \boldsymbol{w})$. Next, by security of the layered DFA evaluation protocol (as applied to the computation of $h_{(a_1,\ldots,a_n),k}$), the client learns no additional information other than the value z. In particular, the protocol does not leak any information about the values of $a_1, \ldots, a_n$ to the client. By correctness of the layered DFA evaluation protocol, each bit b_i the client obtains satisfies $b_i = g_{v_i}(w_i) \oplus a_i \oplus 1$. Security of the layered DFA evaluation protocol (as applied to the computation of g'_{v_i,a_i}) implies that the client learns nothing more about v_i, a_i other than the value $b_i = g_{v_i}(w_i) \oplus a_i \oplus 1$. In particular, this means that the client does not learn anything about the value of a_i from the layered DFA protocol executions. Since the server samples $a_i \leftarrow \{0,1\}$ to be a uniformly random bit, the distribution of b_i is also uniform (even conditioned on the client's view of the protocol execution, which is independent of a_i). As such, the client's view in the protocol execution can be described by a sequence of uniform random bits $b_1, \ldots, b_n$ together with the output $z = \mathrm{TM}_k(\boldsymbol{v}, \boldsymbol{w})$. Thus, the client does not learn anything more about v other than the value $\mathrm{TM}_k(\boldsymbol{v}, \boldsymbol{w})$. (More formally, we can simulate the client's view of the protocol based on the value of $z = \mathrm{TM}_k(\boldsymbol{v}, \boldsymbol{w})$ and then sampling the bits $b_1, \ldots, b_n \leftarrow \{0,1\}$).

Efficiency. We now analyze the cost of privately computing $\mathrm{TM}_k(\boldsymbol{v}, \boldsymbol{w})$ for two n-dimensional vectors $\boldsymbol{v}, \boldsymbol{w}$ with n-bit components where each component $v_i, w_i \in \{0,1\}^\ell$ is an ℓ-bit string.

- **Equality checking.** Computing the (blinded) equality-check functions $g'_{v_i,a_i}(w_i)$ requires evaluating a DFA with $\ell + 1$ layers, where each layer contains 2 nodes. This requires ℓ rounds of communication and 6ℓ bits of communication.
- **Thresholding.** Computing the threshold function $h_{(a_1,\ldots,a_n),k}$ requires evaluating a DFA with $n + 1$ layers, where each layer contains $k + 2$ nodes. This requires n rounds of communication and $n(1 + (2k + 5)\lceil \log(k + 2) \rceil)$ bits of communication.

Finally, we note that the equality checks can be conducted in *parallel*. Thus, the round complexity of the complete protocol is $n + \ell$ and the total communication (in bits) is

$$n\big((2k + 5)\lceil \log(k + 2) \rceil + 6\ell + 1\big) = O(nk \log k + n\ell).$$

We can achieve a tradeoff between communication complexity and round complexity by having each DFA read multiple bits of the input at a time. This is equivalent to considering a DFA with a larger alphabet (i.e., $\Sigma = \{0,1\}^m$ if the DFA is reading m bits of the input for each state transition). Reading multiple bits of the input per state transition decreases the number of layers in the DFA (thus decreasing the round complexity of the private layered DFA evaluation protocol), but increases the size of the transition table between each layer (thus increasing the communication complexity of the protocol). In particular, if we consider DFAs that read m-bits of input per transition, then the overall round complexity of the protocol becomes $\left\lceil \frac{\ell}{m} \right\rceil + \left\lceil \frac{n}{m} \right\rceil$ while the communication complexity (in bits) is

$$\left\lceil \frac{\ell}{m} \right\rceil n(m + 1 + 2^{m+1}) + \left\lceil \frac{n}{m} \right\rceil (m + (2^m(k + 2) + 1)\lceil \log(k + 2) \rceil).$$

When ℓ and m are even, then setting $m = 2$ (i.e., reading 2 bits at a time) yields a protocol with smaller round complexity compared to the $m = 1$ setting (by roughly a factor of 2) *and* slightly smaller communication complexity (compared to both the case where $m = 1$ and all $m > 2$). This is the setting we use in our implementation.

Implementation and evaluation

We implemented the protocol in C++. For our benchmarks, we conducted experiments using two Amazon EC2 instances (M4.2xlarge). Each instance has an 8-core 2.4 GHz Intel Xeon E5-2676 v3 (Haswell) processor and 32 GB of memory. The client instance is located on the West Coast (Northern California region) while the server is located on the East Coast (Northern Virginia region) to simulate a wide-area network (WAN). The network latency between the two instances is roughly 60ms and the bandwidth is roughly 25 MB/s. We use a single-threaded execution environment for all of our experiments.

Code availability

All code will be made publicly available on Github for non-commercial use upon publication.

Author Contributions

JAB, KAJ, GB, and DJW designed the study, analyzed results, and wrote the manuscript. JAB wrote software for the analysis with input from KAJ, GB, and DJW.

Competing Interests

The authors declare no competing interests.

Acknowledgements

We thank Benton Case and Dan Boneh for helpful discussions in an early phase of this project and Aviv Regev for support (K.A.J.). This work was also supported by the Joint University Microelectronics Program (JUMP) Undergraduate Research Initiative (J.A.B), the Stanford A.I. Lab (G.B), NSF CNS-1917414 (D.J.W) and a University of Virginia SEAS Research Innovation Award (D.J.W).

References

1. Arnaud, C. Thirty years of DNA forensics: How DNA has revolutionized criminal investigations. *Chemical and Engineering News* vol. 95 16–20 (2017).
2. CODIS - NDIS Statistics. *Federal Bureau of Investigation* https://www.fbi.gov/services/laboratory/biometric-analysis/codis/ndis-statistics.
3. Combined DNA Index System (CODIS). *Federal Bureau of Investigation* https://www.fbi.gov/services/laboratory/biometric-analysis/codis.
4. Ransom, J. & Southall, A. 'Race-Biased Dragnet': DNA From 360 Black Men Was Collected to Solve Vetrano Murder, Defense Lawyers Say. *The New York Times* (2019).
5. Ransom, J. & Southall, A. N.Y.P.D. Detectives Gave a Boy, 12, a Soda. He Landed in a DNA Database. *The New York Times* (2019).
6. Dickerson, C. U.S. Government Plans to Collect DNA From Detained Immigrants. *The New York Times* (2019).
7. Murphy, H. Coming Soon to a Police Station Near You: The DNA 'Magic Box'. *The New York Times* (2019).
8. Crowley, M. How Commandos Could Quickly Confirm They Got Their Target. *The New York Times* (2019).
9. What is Rapid DNA? *ANDE Rapid DNA* https://www.ande.com/what-is-rapid-dna/.
10. RapidHIT ID System for Human Identification - US. https://www.thermofisher.com/us/en/home/industrial/forensics/human-identification/forensic-dna-analysis/dna-analysis/rapidhit-id-system-human-identification.html.
11. Wee, S.-L. China Is Collecting DNA From Tens of Millions of Men and Boys, Using U.S. Equipment. *The New York Times* (2020).
12. Joly, Y. *et al.* Establishing the International Genetic Discrimination Observatory. *Nat. Genet.* **52**, 466–468 (2020).
13. NYPD's 'Knock-and-Spit' DNA Database Makes You a Permanent Suspect. *Newsweek* https://www.newsweek.com/police-dna-database-nypd-swab-testing-collection-new-york-1326722 (2019).
14. NIST. Core STR Loci Used in Human Identity Testing. https://strbase.nist.gov/coreSTRs.htm.
15. Wang, Z. *et al.* Developmental Validation of the Huaxia Platinum System and application in 3 main ethnic groups of China. *Sci. Rep.* **6**, 31075 (2016).
16. Norrgard, K. Forensics, DNA Fingerprinting, and CODIS. *Nat. Educ.* **1**, 35 (2008).

17. Frequently Asked Questions on CODIS and NDIS. *Federal Bureau of Investigation* https://www.fbi.gov/services/laboratory/biometric-analysis/codis/codis-and-ndis-fact-sheet.

18. ENFSI DNA Working Group. Matching rules. in *DNA DATABASE MANAGEMENT REVIEW AND RECOMMENDATIONS* 22–25 (ENFSI, 2017).

19. Hopcroft, J. E., Motwani, R. & Ullman, J. D. *Introduction to Automata Theory, Languages, and Computation*. (Pearson, 2006).

20. Kilian, J. Founding Cryptography on Oblivious Transfer. in *STOC* 20–31 (1988).

21. Rabin, M. O. How To Exchange Secrets with Oblivious Transfer. *IACR Cryptol. EPrint Arch.* **2005**, 187 (2005).

22. Kolesnikov, V., Kumaresan, R., Rosulek, M. & Trieu, N. Efficient Batched Oblivious PRF with Applications to Private Set Intersection. in *ACM CCS* 818–829 (2016).

23. O'Connor, K. L., Butts, E., Hill, C. R., Butler, J. & Vallone, P. Evaluating the effect of additional forensic loci on likelihood ratio values for complex kinship analysis. in *Proceedings of the 21st International Symposium on Human Identification* 10–14 (2010).

24. Doleac, J. L. The Effects of DNA Databases on Crime. *Am. Econ. J. Appl. Econ.* **9**, 165–201 (2017).

25. Yao, A. C.-C. Protocols for Secure Computations. in *FOCS* 160–164 (1982).

26. Hazel, J. W., Clayton, E. W., Malin, B. A. & Slobogin, C. Is it time for a universal genetic forensic database? *Science* **362**, 898–900 (2018).

27. Joly, Y., Marrocco, G. & Dupras, C. Risks of compulsory genetic databases. *Science* **363**, 938–940 (2019).

28. Naor, M. & Pinkas, B. Oblivious Transfer and Polynomial Evaluation. in *STOC* 245–254 (1999).

29. Canetti, R. Security and Composition of Multiparty Cryptographic Protocols. *J Cryptol.* **13**, 143–202 (2000).

30. Ishai, Y., Kilian, J., Nissim, K. & Petrank, E. Extending Oblivious Transfers Efficiently. in *CRYPTO* 145–161 (2003).

31. Boyle, E. *et al.* Efficient Two-Round OT Extension and Silent Non-Interactive Secure Computation. in *ACM CCS* 291–308 (2019).

32. Troncoso-Pastoriza, J. R., Katzenbeisser, S. & Celik, M. U. Privacy preserving error resilient dna searching through oblivious automata. in *ACM CCS* 519–528 (2007).

33. Sasakawa, H. *et al.* Oblivious Evaluation of Non-deterministic Finite Automata with Application to Privacy-Preserving Virus Genome Detection. in *Workshop on Privacy in the Electronic Society (WPES)* 21–30 (2014).

Tables and Figures

Table 1

CODIS System	End-to-End Time (seconds)	Bandwidth (MB)
US System, current (20 STRs)	38.4	172.4
US System, pre-2017 (13 STRs)	27.7	114.9
UK-like System (11 STRs)	23.4	97.3
EU-like System (16 STRs)	34.1	154.5
Chinese-like System (20 STRs)	40.0	189.0

Table 1. Runtime and network communication needed to privately compare a suspect CODIS profile in the field to a central office database far away. End-to-end protocol execution time and communication required to privately query a central database of 1,000,000 entries for CODIS systems deployed at different countries (see Online Methods). Here, the client and server are Amazon EC2 instances, with the client located on the West Coast of the U.S. and the server located on the East Coast of the U.S.

Figure 1

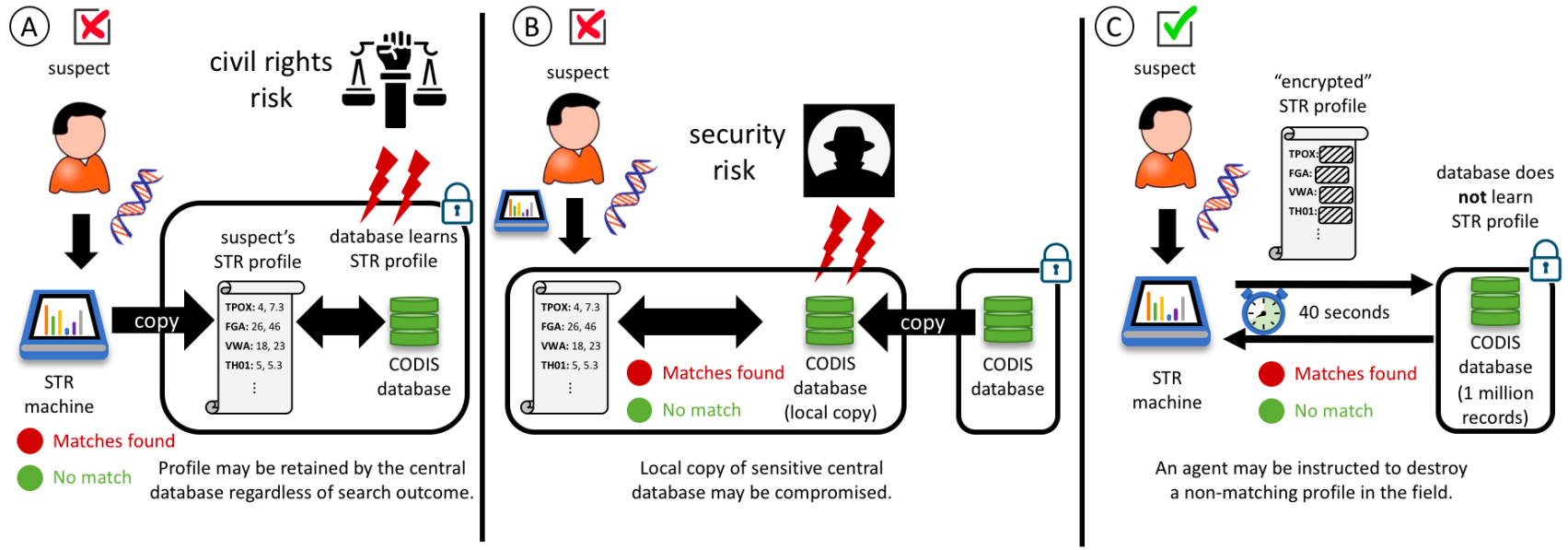


Figure 1. A novel privacy-preserving CODIS DNA profile matching protocol. Rapid STR profiling technologies enables genetic testing and matching from the field. They provide a valuable tool for crime solving but raise significant civil rights concerns regarding data retention and racial profiling. (A) Today, STR profiles collected from potentially innocent individuals are sent to a central CODIS database to check for matches. Here, the central database learns the full query profile and has the option to retain it, irrespective of whether the search yields any match or not. (B) To try and provide anonymity to exonerated (unmatched) profiles, one may load a private copy of the central database onto every field device. This way, an unmatched suspect profile may be destroyed in the field to retain suspect privacy. However, this approach risks exposure of all or parts of the sensitive central database to malicious parties who get ahold of a field device. (C) Our privacy-preserving search protocol enables a new approach where agents can still query a central CODIS database as in (A), but in a way that completely hides the query from the central database. The agent still learns the outcome of the query as before. However, an innocent profile, not matching anything in the database may safely be destroyed in the field. The central database can no longer store it, as it has learned nothing about it through the query.

Figure 2

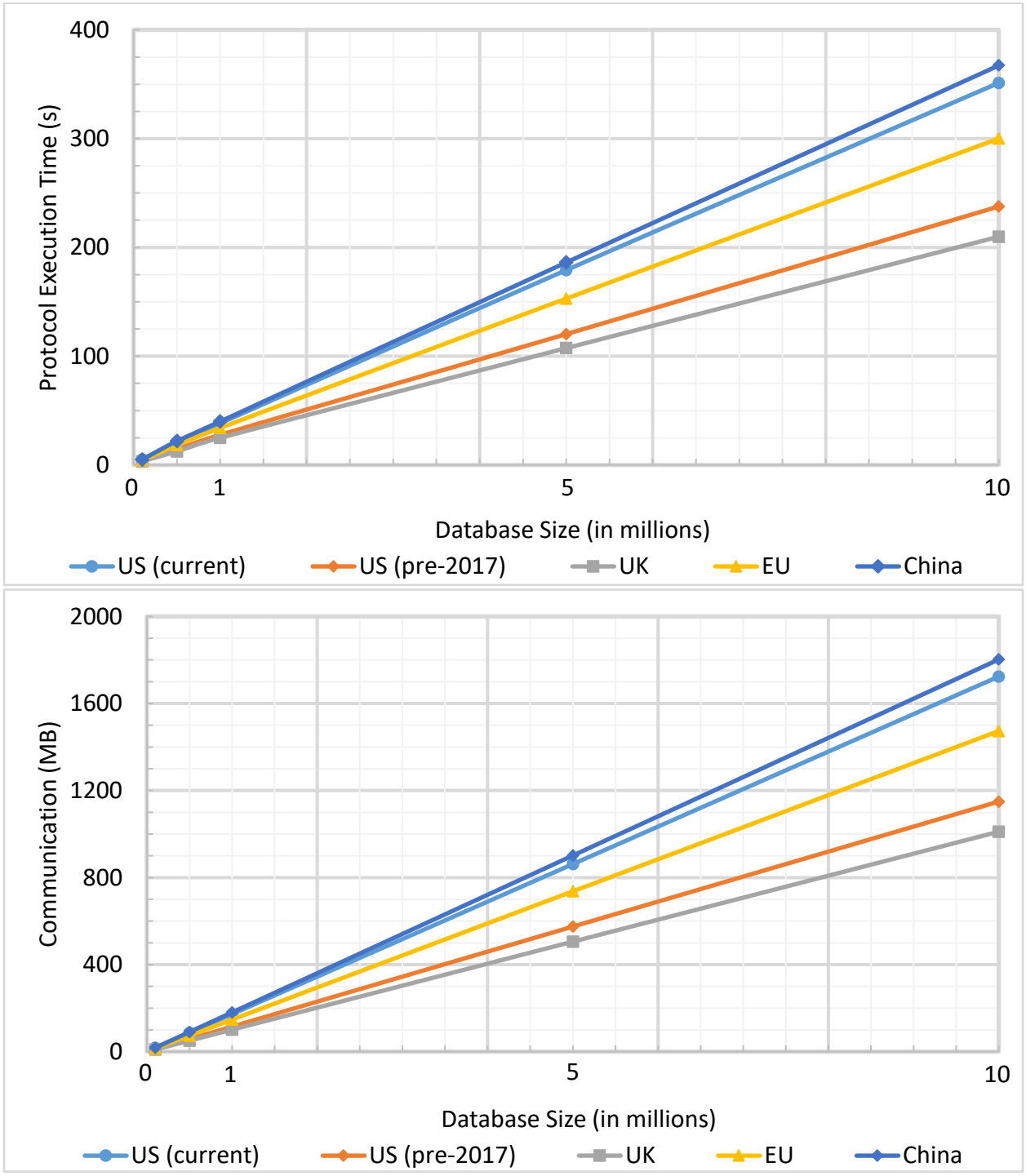


Figure 2. Single query search performance against an entire CODIS database as a function of database size. The number of STR loci and number of precision bits assumed for each system are described in Supplementary Table 1.

Supplementary Tables and Figures

Supplementary Table 1

CODIS System	# of Loci	# of Bits	Bits per Locus Breakdown
US System, current	20	212	10 bits (17 loci); 14 bits (3 loci)
US System, pre-2017	13	142	10 bits (10 loci); 14 bits (3 loci)
UK-like System	11	126	10 bits (7 loci); 14 bits (4 loci)
EU-like System	16	178	10 bits (12 loci); 14 bits (3 loci); 16 bits (1 locus)
Chinese-like System	20	224	10 bits (14 loci); 14 bits (6 loci)
NIST 40 Loci System	40	492	10 bits (17 loci); 14 bits (23 loci)

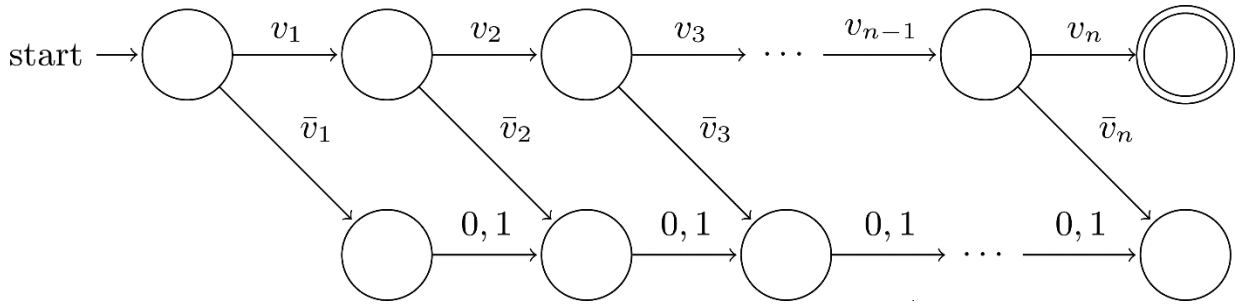
Supplementary Table 1. Approximate CODIS system specifications for select countries. Number of STR loci and number of bits used to encode alleles at each locus for a handful of the 50+ countries using the CODIS system (See Online Methods).

Supplementary Table 2

CODIS System	Precomputation Size (MB) Client	Precomputation Size (MB) Server	# of OT Correlations (in millions)
US System, current (20 STRs)	122	923	116
US System, pre-2017 (13 STRs)	87	616	78
UK-like System (11 STRs)	71	574	73
EU-like System (16 STRs)	104	763	96
Chinese-like System (20 STRs)	122	969	122
NIST 40 Loci System (40 STRs)	260	2105	266

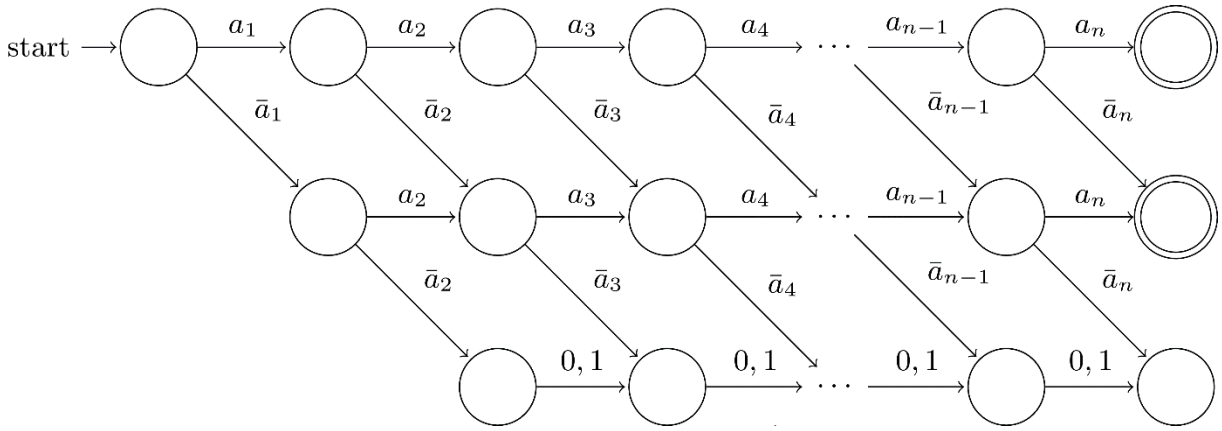
Supplementary Table 2. Offline precomputation cost for each setting. Number of oblivious transfer (OT) correlations, and the memory footprint of the OT correlations for the client and server needed to implement a single CODIS search query against a database with 1,000,000 records. Using state-of-the-art OT extension protocols[22], it is possible to setup $2^{24} > 16$ million OT correlations in 7.5 seconds over a wide-area network. For system specifications, see Online Methods.

Supplementary Figure 1



Supplementary Figure 1. Layered DFA for equality test. This DFA computes the equality-check function $g_{\boldsymbol{v}}(\boldsymbol{w})$ that outputs 1 if $\boldsymbol{v} = \boldsymbol{w}$ and 0 otherwise. In particular, for a vector $\boldsymbol{v} = (v_1, \ldots, v_n) \in \{0,1\}^n$, this DFA only accepts the input $\boldsymbol{w} = (w_1, \ldots, w_n) \in \{0,1\}^n$ where $v_i = w_i$ for all $1 \leq i \leq n$. We use this DFA to decide whether there is a match at a single STR locus. If we denote the single start state as "layer 0", the two states one can arrive at from layer 0 after reading the first bit as "layer 1", etc. we see that this DFA has $n + 1$ layers, such that after reading i bits, it can only be in one of the two states in layer i.

Supplementary Figure 2



Supplementary Figure 2. Layered DFA for thresholding. This DFA computes the threshold function $h_{(a_1,\ldots,a_n),k}$ for the $k = 1$ case. Namely, $h_{(a_1,\ldots,a_n),k}(b_1, \ldots, b_n)$ outputs 1 if $a_i = b_i$ for all but at most k indices $1 \leq i \leq n$. In other words, for any sequence of bits $(a_1, \ldots, a_n) \in \{0,1\}^n$, this DFA accepts if the input $b_1, \ldots, b_n$ satisfies $b_i = a_i$ for all but at most one index i. For instance, in this work, we use this DFA to decide whether a DNA profile matches against a database record on at least 19 out of 20 loci (i.e., the setting where $k = 1$ and $n = 20$) as well as the other configurations. Here, the i^{th} input bit $b_i \in \{0,1\}$ is the (blinded) equality bit denoting whether there is a match in the i^{th} STR locus (between the agent's query and the central database's record). In our protocol, this (blinded) equality bit is computed using the equality-test DFA from Supplementary Figure 1. The bits $a_1, \ldots, a_n$ in the function description $h_{(a_1,\ldots,a_n),k}$ are the blinding values chosen by the server. Recall that the blinding is introduced to hide from the client all information on whether there was a match at STR locus i between the database server's profile and the client's query. The client only learns whether her query matches the record or not, and nothing more. Much like Supplementary Figure 1, this DFA has $n + 1$ layers, such that after reading i bits, the computation can only be in one of the (at most) 3 states of layer i.