DNA Screening Technical Note

Cryptographic Aspects of DNA Screening

Carsten Baum
Hongrui Cui
Ivan Damgård
Kevin Esvelt
Mingyu Gao
Dana Gretton
Omer Paneth
Ron Rivest
Vinod Vaikuntanathan
Daniel Wichs
Andrew Yao
Yu Yu

Cryptographic Aspects of DNA Screening

Carsten Baum [1]
Hongrui Cui [2]
Ivan Damgård [1]
Kevin Esvelt [3]
Mingyu Gao [4]
Dana Gretton [3]
Omer Paneth[3,5]
Ron Rivest [3]
Vinod Vaikuntanathan [3]
Daniel Wichs[3,5]
Andrew Yao [4]
Yu Yu[2,4]

[1]Aarhus University
[2]Shanghai Jiao Tong University
[3]Massachusetts Institute of Technology
[4]Tsinghua University
[5]Northeastern University

Institute for Interdisciplinary Information Sciences
Tsinghua University

This publication is available free of charge from:
https://securedna.org

Jan 2020

DNA Screening Technical Note
Tech. Note , 27 pages (Jan 2020)

Foreword

To prevent the construction of hazardous biological agents from synthetic DNA without disclosing information on potential bioweapons, a screening method that compares query sequences against a database of potential bioweapon sequences must achieve some level of provable security guarantee. To encourage universal adoption, the privacy of clients' queries should be protected. Thus, we describe in this document the cryptographic aspects of the screening protocol we propose. While we will provide characterization of security guarantees and rigorous mathematical proofs, this document also includes explanations concerning the reasoning behind our choice of cryptographic components and security guarantees for an audience without an extensive background in cryptography. Nevertheless, familiarity with basic mathematical notations is helpful throughout this report.

Abstract

Securely screening synthetic DNA orders is crucial to minimizing the number of individuals and groups capable of accessing biological weapons of mass destruction, but it must be accomplished without disclosing information on potential bioweapons. Accomplishing this goal requires: 1) screening orders against a database of hazardous sequences, whose secrecy should be protected at the highest possible level while preserving usability; 2) protecting the privacy of the client synthesizer's queries. In this document, we propose a cryptographic screening protocol that accomplishes these objectives, providing accurate complexity-theoretical assumptions, precise security guarantees, and rigorous mathematical proofs. In addition to technical details, we also explain the reasoning behind our protocol design for the benefit of non-cryptographers.

Key words

DNA Screening, Cryptography, Multiparty Computation.

Table of Contents

List of Tables

List of Figures

1. Introduction

This technical note provides a detailed but non-specialist-friendly presentation of the DNA screening protocol, including a brief note on its motivations, a formal description of the protocol, and some experimental results on a preliminary implementation.

We will begin by presenting the cryptographic formulation of DNA screening, followed by a high-level idea of the current version of screening protocol. We also include some preliminary comparisons between different cryptographic techniques and explain why the project chooses the current version as its privacy-enforcing means.

1.1 Motivation

While cheap, accessible DNA synthesis has revolutionized molecular biology since its first practical availability in the early 2000s, it also presents a serious risk to the health and safety of people everywhere. Accidental or malicious use of DNA synthesis to create self-replicating pathogens could lead to global pandemics. It is imperative to introduce an effective screening system for all DNA synthesizers that checks sequence orders against a database of known hazardous sequences to prevent users from "printing out" destructive bioweapons. The global hazard database should be up-to-date, secure against attackers seeking to learn its contents, and capable of keeping screened sequences confidential.

Present DNA screening efforts are slow, expensive, and suffer incomplete adoption among DNA providers. Much of the time and money consumed by today's DNA screening protocols is spent on "fuzzy matching" between orders and hazard sequences. In contrast with exact-match screening, which concerns whether a sequence appears in its entirety in a database, fuzzy matching assesses the degree of similarity between a query sequence and a database. In screening protocols that use fuzzy matching, a threshold of similarity is established for DNA sequences that are deemed too close to hazards. Due to the high degree of variability among DNA sequences, even those that have the same biological function, it is widely assumed that this inexact matching is critical to catch subtle variants on known hazards.

However, fuzzy matching comes with costs that make it untenable as a candidate for global screening. Many innocuous sequences look similar to parts of hazards, leading them to be identified incorrectly as malicious, an event we refer to as a "false positive." All false positives produced by modern screening systems must be forwarded to a human expert who can examine the sequences to determine whether they are dangerous. Since nearly all synthesis orders are not bioweapons, this expert intervention is almost completely wasted effort. Furthermore, screening protocols with a human component cannot scale to meet the demands of an ever growing global DNA synthesis market.

Instead, we phrase DNA synthesis screening as an exact match problem between short (e.g. 42 base pair) windows selected from the orders and the hazard sequences. These windows are constant-length strings of the DNA bases A, T, C, and G. If a small sequence window from an order exactly matches some window from a hazard, the order is not synthesized. To contend with variability in genetic sequences, we include a selection of likely variations on these windows in the database. The effectiveness of this approach is supported by the observation that all pathogens have many critical regions in their genomes, and that extensive changes in all such regions simultaneously will almost certainly yield an inactive agent. As we will demonstrate, an exact-match screening system can be fast, fully automated, and secure. Exact-match screening can also keep the rate of false positives low enough that global adoption is feasible.

In this document, we examine the DNA screening problem in the exact-match setting. Details on the sequence, window, and variation selection strategies used to construct the database are not discussed, as the cryptographic treatment is independent of any such choices.

1.2 Problem-Specific Challenges

As a query of the form, "Does any sequence window in the order appear in the hazard database?", exact-match DNA screening amounts to set membership testing, where both the query and the database are private. Private set membership testing is an extensively studied topic in cryptographic literature that already has solutions under a variety of privacy constraints. Nevertheless, we found that no single existing technique will overcome the unique challenges posed by the DNA screening problem due to the precise details of its intended application. These challenges are:

- **Small maximum element size.** Short sequence windows, which are the elements of the database set, are limited in length. This constraint makes it extremely difficult to maintain database privacy, given arbitrary set membership queries, because it may be possible to guess short sequences.

  > The window size limitation comes from the physical characteristics of DNA. Long strands of DNA are routinely assembled from short overlapping pieces by ligation. Since exact-match screening cannot screen sequences up to, but short of, the window size in length, the window size must be short enough that unscreened pieces cannot be assembled. For DNA, this limit is around 50-60 base pairs due to thermodynamic properties of DNA binding. Sequence windows must not exceed this limit.

  The challenge could be restated as: *The input space has low entropy*.

- **High query rate.** The present global synthesis market is estimated at around a trillion bases per year. By 2029, queries are expected to exceed 10^{15} queries per year, or over 10 million per second. Many orders consist of quasi-random sequences for DNA libraries.

  > This challenge has two main repercussions:
  >
  > 1. Much of the input space of short windows is effectively searched each year,

raising the minimum false positive rate.

2. The performance demands on such a global networked screening system are immense, requiring many servers and robustness to server failure.

To ensure that the servers hosting the hazard database cannot be hacked to obtain its dangerous contents, we store the database as hashes, or results of a pseudorandom function (PRF) applied to the hazard sequences.

The only way to address the low entropy of the input space to the PRF is to limit evaluation of the PRF itself. We accomplish rate limiting by requiring the participation of multiple trusted third parties each time the PRF is evaluated.

To enable the required level of distribution and robustness to server downtime, we require some, but not all, of the trusted "key-holder" third parties to participate, a property known as "threshold evaluation." We settled on the *distributed pseudorandom function* as the core component.

Our method is private in that no party, whether the querier, database holder, or key-holder, ever learns the input sequence strings for the query *or* the database elements, a very strong property. It has *imperfect privacy* in a strict cryptographic sense, in that it is possible for a server to deduce that the same query was executed twice by a particular client, a relatively weak concession that we make for performance reasons.

In the rest of this section, we first fix the notations to be used in this document. Then, we briefly summarize the formal security requirements for DNA screening, which frames further specification of the ideal functionalities to be realized with provable security. Finally, we provide rationales for our choices of cryptographic building blocks wherever these diverge from established norms, mainly due to scalability issues.

1.3 Formulation

At the core of the DNA screening problem is the private membership query, in which a server P^s maintains a database D and a client P^c makes queries of the form "Is x in D?"[1], with the privacy of both server and client respected. In addition to P^s and P^c, our solution additionally includes some key-holders P^{kh} to facilitate an efficient solution.

In this framework, a solution can be specified by:

1. An *oblivious distributed pseudorandom function* F (DPRF) where the key-holders P^{kh} first generate a shared key $k = (k_1, k_2, \ldots, k_n)$, so that $F_k(x)$ can be computed by the client P^c when at least t key holders (t being a security parameter) supply certain information $a(k_i, x)$ to it.

2. A two-party protocol P between client P^c (also known as the receiver) and server P^s (the sender) for membership query.

[1]Readers interested in technical details can refer to Sec. 2 for a complete list of notations in this report.

Initially, P^{kh} generate a shared secret key k, and then set up a database D to contain all $F_k(y)$ for the known harmful strings y (in encrypted form). For P^c to ask a query x, it first interacts with t keyholders P^{kh} to obtain $F_k(x)$, and then uses the two-party protocol P to obtain the answer to whether $F_k(x)$ is in D.

1.4 Technical Details of the Implementation

We begin with a high level view of our initial implementation, which uses a highly efficient protocol with imperfect privacy of client queries. We then describe why the approach is superior for the purpose of DNA synthesis screening to alternatives based on different paradigms.

A basic model:

1. For the DPRF, we use the Naor-Pinkas-Reingold [10] scheme which is based on the hardness assumption of DDH (Decisional Diffie-Hellman). The DDH assumption is well-studied and used in many deployed cryptographic systems.

2. For security and privacy, we represent each string x by a one-way hashed image $H(x)$. The membership query simply does a binary search for $F_k(H(x))$ against the database containing $F_k(H(y))$ of all known harmful strings y.

Two remarks:

1. It is assumed that $H(x)$ is a random oracle. Even then, the client's privacy is not perfect, as the server can detect when two queries x, y are the same (as $H(x) = H(y)$).

2. This DPRF is not quantum resistant. There exist DPRFs designed for post-quantum security, but they are presently too inefficient for practical use. We discuss our approach to making the system quantum-resistant in Sec. 6.

We refer to Sec. 2 through Sec. 5 of this report for the details about the preliminary design of the screening protocol.

1.5 Other Cryptographic Solutions

We believe the privacy tradeoff of revealing identical queries is necessary for an application of our scale: we estimate that the size of a single client request and database size are around 10^3 and 10^9 entries respectively. Our approach to DNA screening can be viewed as *private membership testing* or, more generally, *private set intersection*(PSI), which has received considerable attention in the field of practical secure multiparty computation. As an alternative to PSI, one could also employ a technique which is called *oblivious RAM* (ORAM). We will now survey the state of both PSI and ORAM and point out why either of these methods, albeit providing stronger security guarantees, would lead to an unacceptable performance deterioration.

1.5.1 PSI-based Solutions

A *Private Set Intersection (PSI)* protocol runs between two parties which each hold sets and securely computes the intersection of these two sets without revealing the any values outside of the intersection to each other. It is immediate that such a PSI protocol implies a privacy-preserving screening protocol. By running the PSI protocol, the client learns if his query intersects (matches) with (any record in) the database or not. By its definition using such a PSI protocol as a building block would additionally hide repeated queries.

There is a rich literature about the PSI problem (in the semi-honest setting in particular), e.g., the state-of-the-art FHE-based PSI [3] in the asymmetric setting and OT-based PSI in the symmetric setting [12], where the receiver's input set is significantly smaller and roughly the same as the sender's input set respectively. Our scenario is clearly asymmetric since the server's database stores up to $N = 10^9$ sequences and the client makes a single query each time. In order to better amortize the cost, we assume that each client makes 10^3 queries per minute (the input size is 10^3). The length of the sequence is 57 base pairs (114 bits). We analyze the cost of the two PSI protocols in this setting.

OT-based PSI protocols. We follow the OT-based PSI protocol in [12] with the concrete parameter choice in Tab. 1. Altogether, the sender and the receiver perform $(b+s)$ invocations of l-bit 2^σ-choose-1 random oblivious transfer(OT) protocol instances. In addition, the server sends $(k \cdot N + s \cdot N) \cdot l$ bits to the receiver. Using the OT extension protocol of [8], this translates to the receiver sending $424 \cdot (b+s)$ bits to the server. Thus, in terms of communication cost, the amortized communication cost per query is about 400 MB. As for the computation cost, the server needs to perform $5,000,000$ PRF evaluations per query. Even on a high-end server CPU, one can easily expect that these PRF queries would take approximately 1s to complete using SHA-256 and not considering the cost of memory accesses[2]. We believe that a communication complexity of 400 MB for a single request is too high for any practical deployment.

Table 1. The choice of parameters for the OT-based PSI Protocol from [12]

Symbol	Value	Meaning
M	10^3	Set size of the receiver
N	10^9	Set size of the sender
b	2.4×10^5	Hash table slot number
k	2	Hash function number
s	3	Cuckoo hashing stash size
l	77	Mask length
σ	58	Item representation length
λ	30	Hashing failure parameter

[2]Compare e.g. to [6] for a single thread on a modern AMD EPYC 7702 CPU at 2000 MHz for 64 byte blocks.

FHE-based PSI Protocols. The PSI protocol from [3] uses fully homomorphic encryption and cuckoo hashing, which is relatively efficient when the receiver's set size is significantly smaller than the sender's. Based on the performance evaluation in [3], we use linear interpolation to estimate the performance for our setting. In particular, when the receiver's set size is $M = 10^3$ and the sender's set size is $N = 10^9$, the sender's processing time would be more than 5 hours, and the total communication cost would be around 74 MB. In particular the running time will be a bottleneck of this approach.

1.5.2 Oblivious RAM

Oblivious RAM protocols allow a party to securely read information from and store data in an database held by a server (or a number of servers). For this server it will be impossible to identify the actual read/write pattern that the client has performed. A recent work by Doerner and shelat [5] gives a novel design called FloRAM that performs membership query as a binary search on ORAM quite efficiently. Experimentally, it clocks per query 0.79s (for $N = 2^{15}$), 2.04s (for $N = 2^{20}$), and 14.37s (for $N = 2^{25}$) per membership query. Similar to the PSI approaches this technique would require a substantial amount of fundamental research before it would support the workload that we anticipate.

2. The Screening Functionality

In this section we introduce the mathematical formulation of the efficient (but with imperfect privacy) screening protocol, which is presented in the form of ideal functionalities. These functionalities capture the actions of a virtual trusted third party that a cryptographic protocol seeks to simulate. We first list all the notations to be used in this report in Sec. 2.1, and then present the functionalities in Sec. 2.2.

2.1 Notations

This section summarizes all the symbols we use in the white paper.

2.1.1 Mathematical Symbols

We use $[n]$ to denote the set of integers $\{1,\ldots,n\}$. $S = \langle x_1,\ldots,x_n\rangle$ denotes an ordered list, $S[i]$ refers to the i^{th} entry of S, and $|S|$ denotes the number of entries in S. In particular, we use `DB` to denote the set of all database entries.

We use $\mathbb{Z}_q$ to represent the set $\{0,1,\ldots,q-1\}$, and we denote by $x \overset{\$}{\leftarrow} S$ sampling an element from the finite set S uniformly at random. We use $poly(\cdot)$ to refer to some polynomial function.

We use κ to denote the security parameter. Let p and q be two prime numbers such that the length of q is at least κ bits and $q|p-1$. Let $\mathbb{G}$ denote a multiplicative subgroup of $\mathbb{Z}_p^*$ of order q, and we use g to denote a generator in that subgroup. The *Decisional Diffie-Hellman* assumption for group $\mathbb{G}$ holds if for all probabilistic polynomial-time algorithms

D, the distinguishing advantage

$$\left|\Pr[a,b \xleftarrow{\$} \mathbb{Z}_q : \mathsf{D}(g,g^a,g^b,g^{ab}) = 1] - \Pr[a,b,c \xleftarrow{\$} \mathbb{Z}_q : \mathsf{D}(g,g^a,g^b,g^c) = 1]\right|$$

is negligible in the security parameter κ.

We use "sequence" to denote a string of base pairs which make up a biological DNA sequence. We denote by Q the genetic alphabet $Q = \{\mathrm{A},\mathrm{T},\mathrm{C},\mathrm{G}\}$, and we use $|x|$ to denote the length of a sequence x. Our protocol mainly considers a sequence of fixed length w, and we use Q^w to denote the set that consists of all sequences of length w.

We use H to denote a random oracle that maps any sequence of length w to a group element in $\mathbb{G}$. A random oracle is a popular cryptographic heuristic that emulates a truly random function and is typically instantiated by a cryptographic hash function.

2.1.2 Parties

Since screening is essentially a multi-party computation task, we need to model different roles in screening as separate parties. There are four types of parties in our protocol.

- **Clients.** Synthesizers who query the database with their input to find out whether it matches any of the entries in the database or not.
- **Server.** Cloud servers that hold a database of encrypted DNA sequences.
- **Key holders.** Parties that hold the key shares of the main key.
- **Administrator.** The trustworthy party who owns the entries in cleartext. To add an entry to the database, he first interacts with key holders to get the entry encrypted and then stores the encrypted value in the database.

We can further divide clients into two groups: 1) large commercial DNA synthesis providers (e.g. BGI) who typically share dedicated and fast network connection with other parties (i.e. the server and key holders); 2) desktop clients who make queries less frequently with restricted bandwidth connections and are more likely to be compromised. We model client, server, and key holders as semi-honest for now, and extend our protocol to handle malicious clients (drawing from existing work on maliciously secure oblivious DPRFs) in the next version.

We use superscript to distinguish the roles of the parties, and subscript to index different parties of the same class. We use P^c to denote client, P^s to denote the server, P_i^{kh} to denote key holders and P^{adm} to denote the administrator. Throughout this note, we restrict our discussion to the setting of one client, one administrator, one server and n key holders, although our construction can be easily extended to the multiple server setting.

The server holds a database where all entries are encrypted under the main secret key. As a slight abuse of terminology, encryption refers to the evaluation of a pseudorandom function (under the main secret key) on the cleartext to produce the corresponding output.

Strictly speaking, it is not an encryption as its inverse operation, decryption, may not exist and neither is it needed in our protocol. In order to protect the main key, we run Shamir's (t,n)-secret sharing among the n key holders, where any t (or more) parties can efficiently reconstruct the secret key from their shares, and any less than t parties cannot. Further, the administrator (or client) runs a protocol with any t key holders in an oblivious manner such that he gets his input encrypted under the main key without revealing any information about his query to the key holders. The administrator (resp., client) then adds his encrypted entry to the database (resp., runs an exact matching with the database).

2.1.3 Provable Security

Following the conventions and notations of multiparty computation, we use $\mathscr{F}$ to denote an ideal functionality to be realized and use Π to denote a multiparty protocol. We use subscript to further specify the functionality or protocol.

We consider security against a static, semi-honest adversary. More specifically, the adversary chooses the parties to corrupt prior to the protocol execution and the corruption status does not change throughout the course of protocol execution. The corrupted parties follow the protocol specification exactly. However, the adversary tries to learn more information than allowed by looking at the transcript of messages that it received and the internal state of corrupted parties.

A protocol that is secure in the presence of semi-honest adversaries guarantees that there is no inadvertent leakage of information; when the parties involved essentially trust each other but want to make sure that no record of their input is found elsewhere, then this can suffice. Beyond this, protocols that are secure for semi-honest adversaries are often designed as the first step towards achieving stronger notions of security.

2.2 Security Requirement and Ideal Functionality

To support efficient and secure screening, we need four ideal functionalities.

- **Initialization.** Register and assign key shares to all the key holders and initialize the database;
- **Adding a new sequence.** Add a new sequence to the database;
- **Querying a sequence.** Decide whether a sequence being queried exists in the database or not;
- **Refreshing key shares.** Update the key shares of the key holders.

Notice that these ideal functionalities do not correspond to the actual operations executed by the protocol, but instead they specify the functionalities to be realized. As new sequences will be added to the database and possibly queried subsequently, the functionality is by nature reactive, which means that a trusted third party must maintain state in order to realize this functionality. Therefore, we consider the security of a session where the initialization functionality is first invoked followed by addition, querying, and key share refreshing operations. Indeed, the *sid* field in all the messages presented below is used to distinguish different sessions.

In the following, the server keeps two lists S_1 and S_2 that are of the same size at all times. S_1 keeps a record of the sequences that have appeared (either queried by client or input by the data owner) in the view of the server, while S_2 indicates whether such sequence has been actually added or not. More specifically, let i be an index in S_1 and S_2, then ($S_1[i] = x$, $S_2[i] = 0$) indicates that x has already been queried, but it does not exist in the database, whereas ($S_1[i] = x$, $S_2[i] = 1$) indicates that x has been added. While seemingly unnecessary, the former case is useful in the simulation-based proof (to simulate queries in a consistent manner). We use “append” to denote the operation of adding an element (a sequence for S_1 or a bit for S_2) to the end of a list.

2.2.1 Initialization

The key holders and the server participate in the initialization functionality $\mathscr{F}_{\text{init}}$, which proceeds as follows:

Functionality $\mathscr{F}_{\text{init}}$

1. Upon receiving $(\mathsf{init}, \mathsf{P}^{kh}_i, sid)$ from P^{kh}_i for all $i \in [n]$, and $(\mathsf{init}, \mathsf{P}^{s}, sid)$ from the server, the functionality chooses a new key id *kid* and record $(sid, kid, S_1 = \langle\rangle, S_2 = \langle\rangle)$.

2. The functionality sends to all key holders their respective $(\mathsf{init.receipt}, \mathsf{P}^{kh}_i, sid, kid)$, and to the server $(\mathsf{init.receipt}, \mathsf{P}^{s}, sid)$.

2.2.2 Adding a New Sequence

The administrator, server and key holders participate in the new sequence adding functionality $\mathscr{F}_{\text{add}}$, which proceeds as follows:

Functionality $\mathscr{F}_{\text{add}}$

1. Upon receiving $(\mathsf{add}, \mathsf{P}^{adm}, sid, x)$ from the administrator, $(\mathsf{add}, \mathsf{P}^{kh}_i, sid, kid)$ from t different key holders, and $(\mathsf{add}, \mathsf{P}^{s}, sid)$ from the server, the functionality checks if *sid* and *kid* agrees with the stored (sid, kid, S_1, S_2) pair. Otherwise, it terminates and outputs ‘$\perp$’.

2. If $x \notin S_1$, the functionality appends x to S_1, and 1 to S_2. If $x \in S_1$ and $S_2[i] = 0$, where i is the index of x in S_1, it sets $S_2[i]$ to 1. It then sends $(\mathsf{add.receipt}, \mathsf{P}^{kh}_i, sid)$ back to the key holders, $(\mathsf{add.receipt}, \mathsf{P}^{s}, sid, i)$ to the server, where i is the index of the sequence x in S_1.

2.2.3 Making a Query

We consider the querying functionality, which involves a client, a server and any t key holders. The querying functionality $\mathscr{F}_{\text{query}}$ is described below, which preserves the input privacy by definition.

Functionality $\mathscr{F}_{\text{query}}$

1. Upon receiving $(\mathsf{query}, \mathsf{P}^c, sid, x)$ from the client P^c, $(\mathsf{query}, \mathsf{P}^s, sid)$ from the server, and $(\mathsf{query}, \mathsf{P}^{kh}_i, sid, kid)$ from t key holders, the functionality checks whether all the (sid, kid) pairs match with the stored (sid, kid, S_1, S_2) pair. Otherwise, it terminates and outputs '$\perp$'.

2. It then checks if x appears in the set S_1 or not. If not, it appends x to S_1 and 0 to S_2.

3. Finally, the functionality sends $(\mathsf{query.receipt}, \mathsf{P}^c, sid, S_2[i])$ to the client, $(\mathsf{query.receipt}, \mathsf{P}^s, sid, i)$ to the server and $(\mathsf{query.receipt}, \mathsf{P}^{kh}_i, sid)$ to the key holders, where i is the index such that $S_1[i] = x$.

2.2.4 Refreshing Key Shares

In order to achieve proactive security, we refresh the key shares periodically to mitigate the risk of leaking t or more key shares in a *single epoch*, compared to the case with no time limit at all (where no key share refreshing is in place). This operation is formally captured by the key refreshing functionality $\mathscr{F}_{\text{refresh}}$, which proceeds as follows:

Functionality $\mathscr{F}_{\text{refresh}}$

1. Upon receiving $(\mathsf{refresh}, \mathsf{P}^{kh}_i, sid, kid)$ from all n key holders, and $(\mathsf{refresh}, \mathsf{P}^s, sid)$ from the server, the functionality checks whether all the (sid, kid) values match with the stored (sid, kid) value. If not, it outputs '$\perp$' and terminiates.

2. It chooses an unused key id kid', and updates (sid, kid, S_1, S_2) to (sid, kid', S_1, S_2), and then sends $(\mathsf{refresh.receipt}, \mathsf{P}^{kh}_i, sid, kid')$ to the key holders, and $(\mathsf{refresh.receipt}, \mathsf{P}^s, sid)$ to the server.

3. The Screening Protocol

In this section we present the screening protocol. We first introduce a helper functionality $\mathscr{F}_{\text{prf}}$ for oblivious threshold evaluation of a pseudorandom function and a protocol that implements this functionality in the semi-honest model. The screening protocol is then introduced in the $\mathscr{F}_{\text{prf}}$-hybrid model, which facilitates the modular presentation of the protocol and as well as security analysis.

3.1 The Oblivious Threshold PRF Protocol

First we present an ideal functionality that defines the oblivious threshold evaluation of a pseudorandom function. The pseudorandom function we consider here is a variant of (with minor adaption to) the Naor-Pinkas-Reingold PRF introduced in [10]. The PRF $F : \mathbb{Z}_q \times Q^w \to \mathbb{G}$ is defined as

$$F(\alpha, x) = f_\alpha(x) = H(x)^\alpha,$$

where $\mathbb{G}$ is a multiplicative group of order q used in the DDH assumption, and H is a random oracle that maps a sequence to a group element.

Two types of parties participate in this protocol, namely the key holders P_i^{kh} and a receiver P^r. For convenience, we assume that the key holders in the protocol are the same as those in the screening protocol and thus inherit the notations for the key holders. Each key holder holds his respective (t,n)-Shamir secret share of the PRF master key, and the receiver evaluates the PRF on his input under the master secret key by interacting with t key holders.

The functionality consists of three functions, the PRF secret key generation, oblivious evaluation of the PRF under the secret key and refreshing the key shares of key holders.

Functionality $\mathscr{F}_{\textbf{prf}}$

The functionality keeps a mapping F that maps sequences to elements in $\mathbb{G}$. Initially, it sets F to an empty mapping.

- Upon receiving $(\mathsf{gen}, sid, \mathsf{P}_i^{kh})$ from all key holders, the functionality chooses a new key id kid and sends $(\mathsf{gen.receipt}, sid, \mathsf{P}_i^{kh}, kid)$ to P_i^{kh}. It ignores any subsequent calls under the same sid.

- Upon receiving $(\mathsf{eval}, sid, \mathsf{P}_{i_j}^{kh}, kid)$ from t distinct key holders $\mathsf{P}_{i_1}^{kh}, \dots, \mathsf{P}_{i_t}^{kh}$ and $(\mathsf{eval}, sid, P^r, x)$ from the receiver, the functionality checks if x appears in F. If x has already been added to F, let y be the corresponding value in F (i.e., $y = F(x)$). Otherwise, it samples $y \xleftarrow{\$} \mathbb{G}$ and adds $\langle x, y \rangle$ to F. The functionality sends $(\mathsf{eval.receipt}, sid, P^c, y)$ to the receiver, and $(\mathsf{eval.receipt}, sid, \mathsf{P}_{i_j}^{kh})$ to the t key holders.

- Upon receiving $(\mathsf{refresh}, sid, \mathsf{P}_i^{kh}, kid)$ from all key holders, the functionality samples an unused key id kid' and sends $(\mathsf{refresh.receipt}, sid, \mathsf{P}_i^{kh}, kid')$ to the respective key holders.

We present a protocol Π_{prf} that securely computes the functionality $\mathscr{F}_{\text{prf}}$ in the semi-honest model. Since the functionality consists of key generation, evaluation and refreshing, we present three sub protocols accordingly.

3.1.1 PRF Key Generation

The key generation protocol $\Pi_{\text{prf.gen}}$ uses the additive homomorphic property of Shamir's (t,n)-threshold secret sharing.

PRF Key Generation Protocol $\Pi_{\text{prf.gen}}$

- **Parties:** All key holders $\mathsf{P}_1^{kh}, \cdots, \mathsf{P}_n^{kh}$.
- **Common Parameters:** Group description $(\mathbb{G}, q, g)$.

1. *Round 1:*

   Each key holder P_i^{kh} generates a uniformly random $(t-1)$-degree polynomial in $\mathbb{Z}_q[x]/(x^n+1)$. Specifically, for $i \in [n]$, P_i^{kh} chooses $a_{i,j} \xleftarrow{\$} \mathbb{Z}_q$ for each $j \in [0, t-1]$, and computes
   $$f_i(x) = a_{i,0} + a_{i,1}x + \ldots + a_{i,t-1}x^{t-1} \mod q.$$
   Then, each P_i^{kh} computes and sends $f_i(j)$ to every other P_j^{kh} for $j \in [n]$.

2. *Output:*

   Each key holder P_i^{kh} now has $f_j(i)$ for all $j \in [n]$ and thus he computes and stores $\alpha_i = \sum_{j=1}^n f_j(i)$, which corresponds to the evaluation of the following function on input i:
   $$f(x) = \sum_{i=1}^n f_i(x) = \underbrace{\sum_{i=1}^n a_{i,0}}_{=\alpha} + \sum_{i=1}^n a_{i,1}x + \ldots + \sum_{i=1}^n a_{i,t-1}x^{t-1} \mod q \ .$$
   This jointly forms the (t,n)-secret sharing of the uniformly random key $\alpha = \sum_{i=1}^n a_{i,0} \mod q$, namely the constant term of the above polynomial.

3.1.2 Oblivious Evaluation of the PRF

In subprotocol $\Pi_{\text{prf.eval}}$, the receiver first computes $H(x)$ and masks it by raising it to a random power β. It then sends this masked value $H(x)^\beta$ to t key holders $\mathsf{P}_{i_1}^{kh} \ldots, \mathsf{P}_{i_t}^{kh}$. Each such key holder computes its respective $H(x)^{\beta\alpha_{i_j}}$ where α_{i_j} is $\mathsf{P}_{i_j}^{kh}$'s key share. On receiving the computed values from the t key holders, the receiver outputs

$$f_\alpha(x) = \left(\prod_{j=1}^t H(x)^{\beta\alpha_{i_j}\lambda_{i_j}}\right)^{\beta^{-1} \mod q},$$

for the appropriate Lagrange coefficient λ_{i_j} such that $\sum_{j=1}^t \lambda_{i_j}\alpha_{i_j} = \alpha \mod q$.

Oblivious PRF Evaluation Protocol $\Pi_{\textbf{prf.eval}}$

- **Parties:** t key holders $\mathsf{P}^{kh}_{i_1}\ldots,\mathsf{P}^{kh}_{i_t}$ $(1 \le i_1 < \ldots < i_t \le n)$ and receiver P^r.
- **Common Parameters and Oracle:** Group description $(\mathbb{G}, q, g)$ and a random oracle $H : Q^w \to \mathbb{G}$.
- **Internal State:** For each key holder $\mathsf{P}^{kh}_{i_j}$ $(1 \le j \le t)$, the state information is his share α_{i_j} (of the PRF secret key α).
- **Input:** The receiver P^c's private input x.
- **Output:** The receiver outputs $y = f_\alpha(x)$.

1. *Round 1:*

   The receiver chooses a random element from the group $\mathbb{Z}_q$, i.e. $\beta \overset{\$}{\leftarrow} \mathbb{Z}_q$, raises $H(x)$ to the power of β, and sends this value to the key holders $\mathsf{P}^{kh}_{i_1},\ldots,\mathsf{P}^{kh}_{i_t}$.

2. *Round 2:*

   Upon receiving the value $H(x)^\beta$, the key holder $\mathsf{P}^{kh}_{i_j}$ raises it to the power of α_{i_j} and sends this value back to the receiver.

3. *Output:*

   Upon receiving the value from t key holders, the receiver computes

   $$f_\alpha(x) = \left(\prod_{j=1}^{t} H(x)^{\beta\alpha_{i_j}\lambda_{i_j}}\right)^{\beta^{-1} \mod q},$$

   where λ_{i_j} is the appropriate Lagrange coefficient such that $\sum_{j=1}^{t} \lambda_{i_j}\alpha_{i_j} = \alpha \mod q$.[a] and outputs $f_\alpha(x)$.

[a]Note that the values of coefficients $\{\lambda_{i_j}\}_{j\in[t]}$ actually depend on the subset of key holders participating in the PRF evaluation protocol, but they should be known to the receiver who chooses the subset of key holders.

3.1.3 Refreshing PRF Key Shares

In order to achieve proactive security, the key shares need to be "refreshed" periodically, which is achieved by the key share refreshing protocol $\Pi_{\text{prf.refresh}}$. The protocol re-randomizes the shares of the key holders by generating a new random polynomial with zero constant term (but uniformly random values for other terms), and masking the original polynomial with the new one.

Key Share Refreshing Protocol $\Pi_{\textbf{prf.refresh}}$

- **Parties:** All n key holders $\mathsf{P}_1^{kh}, \ldots, \mathsf{P}_n^{kh}$.
- **Internal State:** For the key holders, the state information is their respective shares α_i of the PRF key α for $i \in [n]$.
- **Common Parameters:** Group description $(\mathbb{G}, q, g)$.
- **Outputs:** Each key holder's new share α_i' of the master PRF secret key α.

1. *Round 1:*

   Each key holder P_i^{kh} generates a uniformly random $(t-1)$-degree polynomial with constant term 0 in $\mathbb{Z}_q[x]/(x^n+1)$. Specifically, for $i \in [n]$, P_i^{kh} chooses $a_{i,j} \overset{\$}{\leftarrow} \mathbb{Z}_q$ for $j \in [t-1]$, and computes

   $$f_i(x) = a_{i,1}x + \ldots + a_{i,t-1}x^{t-1} \mod q.$$

   Then, each key holder P_i^{kh} computes and sends $f_i(j)$ to P_j^{kh} for $j \in [n]$.

2. *Output:*

   Upon receiving $f_j(i)$ for every $j \in [n]$, P_i^{kh} computes $\alpha_i' = \alpha_i + \sum_{j=1}^{n} f_j(i) \mod q$, and produces it as output.

3.2 Screening Protocol in the $\mathscr{F}_{\text{prf}}$-hybrid Model

Now we present the screening protocol in the $\mathscr{F}_{\text{prf}}$-hybrid model, which consists of the following sub-protocols, namely database initialization, adding a new entry to the database, making database queries and refreshing key shares.

3.2.1 Initialization Protocol

The initialization protocol Π_{init} implements the functionality $\mathscr{F}_{\text{init}}$, and is executed among all key holders and the server. The protocol is explained in detail in Fig. 1.

Initialization Protocol $\Pi_{\textbf{init}}$

- **Parties:** All n key holders $\mathsf{P}_1^{kh}, \ldots, \mathsf{P}_n^{kh}$ and the server P^s
- **Oracle:** The ideal functionality $\mathscr{F}_{\mathrm{prf}}$.

1. *Round 1:*

   Each key holder P_i^{kh} sends $(\mathsf{gen}, sid, \mathsf{P}_i^{kh})$ to the functionality $\mathscr{F}_{\mathrm{prf}}$.

2. *Output:*

   Upon receiving the response $(\mathsf{gen.receipt}, sid, \mathsf{P}_i^{kh}, kid)$, key holder P_i^{kh} outputs kid. The server sets up two empty lists $S_1, S_2 = \langle\rangle$.

Fig. 1. The Initialization Protocol Π_{init}

3.2.2 Adding New Sequence Protocol

Adding new sequence involves evaluating the pseudorandom function (under the main secret key) on the sequence to be added, and storing the encrypted value in the database. The detailed process is explained in Fig. 2. Note that this sub-protocol involves only t key holders, the administrator and the server. The protocol follows a two-step process: the first step is an oblivious evaluation of the pseudorandom function (by invoking the ideal functionality $\mathscr{F}_{\mathrm{prf}}$) between the administrator P^{adm} and t key holders where the administrator plays the role receiver in $\mathscr{F}_{\mathrm{prf}}$. Then the administrator sends y (the result returned from $\mathscr{F}_{\mathrm{prf}}$) to the server P^s, who stores it in the database.

Add New Sequence Protocol Π_{add}

- **Parties:** t key holders $\mathsf{P}^{kh}_{i_1}, \ldots, \mathsf{P}^{kh}_{i_t}$ $(1 \le i_i < \ldots < i_t \le n)$, the administrator P^{adm} and the server P^{s}.
- **Internal States:** For the server, the state information is the lists S_1, S_2. For the key holders, the state information is the key id kid.
- **Input:** For the administrator, the private input is the sequence x to be added.
- **Oracle:** The ideal functionality $\mathscr{F}_{\text{prf}}$.

1. *Round 1:*

   Key holder $\mathsf{P}^{kh}_{i_j}$ sends $(\mathsf{eval}, sid, \mathsf{P}^{kh}_{i_j}, kid)$ to ideal functionality $\mathscr{F}_{\text{prf}}$ and the administrator P^{adm} sends $(\mathsf{eval}, sid, \mathsf{P}^{adm}, x)$ to the ideal functionality $\mathscr{F}_{\text{prf}}$.

2. *Round 2:*

   Upon receiving the response $(\mathsf{eval.receipt}, sid, \mathsf{P}^{adm}, y)$ from the ideal functionality, the administrator P^{adm} sends y to the server P^{s}.

3. *Output:*

   Upon receiving y from the administrator, the server searches in S_1 for y.

   - If $y \in S_1$, let i be the index such that $S_1[i] = x$, then if $S_2[i] = 0$, the server sets $S_2[i] = 1$.
   - If $y \notin S_1$, the server appends y to S_1 and 1 to S_2.

Fig. 2. The New Sequence Adding Protocol Π_{add}

3.2.3 Querying Protocol

The querying protocol involves a client, t key holders and the server. The protocol is similar to the two-step new sequence adding protocol except that this time the client acts as the receiver as in the ideal functionality $\mathscr{F}_{\text{prf}}$ in the first step and acquires y (the result returned from $\mathscr{F}_{\text{prf}}$). Then, it sends y to the server, who checks whether the received value is in the database or not. Finally, the server sends to the client a Boolean value indicating whether the matching query exists or not.

Query Protocol $\Pi_{\textbf{query}}$

- **Parties:** t key holders $\mathsf{P}^{kh}_{i_1}, \ldots, \mathsf{P}^{kh}_{i_t}$ $(1 \leq i_i < \ldots < i_t \leq n)$, the server P^s and the client P^c.
- **Internal States:** For the server, the state information is the lists S_1, S_2. For the key holders, the state information is the key id kid.
- **Private Inputs:** For the client, the private input information is the sequence x to be queried.
- **Oracle:** The ideal functionality $\mathscr{F}_{\text{prf}}$.
- **Output:** The client outputs a bit b indicating whether x is in the database or not.

1. *Round 1:*

   Key holder $\mathsf{P}^{kh}_{i_j}$ sends $(\mathsf{eval}, sid, \mathsf{P}^{kh}_{i_j}, kid)$ to ideal functionality $\mathscr{F}_{\text{prf}}$ and the client P^c sends $(\mathsf{eval}, sid, \mathsf{P}^c, x)$ to the ideal functionality $\mathscr{F}_{\text{prf}}$.

2. *Round 2:*

   Upon receiving the response $(\mathsf{eval.receipt}, sid, \mathsf{P}^c, y)$ from the ideal functionality, the client P^c sends y to the server P^s.

3. *Round 3:*

   Upon receiving y from the client, the server searches in S_1 for y.

   - If $y \in S_1$, let i be the index such that $S_1[i] = y$, the server sets $b = S_2[i]$.
   - If $y \notin S_1$, the server appends y to S_1 and 0 to S_2, and sets $b = 0$.

   The server sends b to the client.

4. *Output:*

   Upon receiving b from the server, the client outputs b.

Fig. 3. The Querying Protocol Π_{query}

3.2.4 Key Share Refreshing Protocol

In order to achieve proactive security, the key shares need to be "refreshed" periodically, which is achieved by the key share refreshing protocol. Now that the ideal functionality $\mathscr{F}_{\text{prf}}$ already supports key share refreshing operation, this protocol simply invokes it. The details of this protocol is specified in Fig. 4.

Key Share Refreshing Protocol $\Pi_{\textbf{refresh}}$

- **Parties:** All key holders $\mathsf{P}_1^{kh}, \cdots, \mathsf{P}_n^{kh}$.
- **Internal States:** For the key holders, the state information is the key id kid.
- **Oracle:** The ideal functionality $\mathscr{F}_{\text{prf}}$.

1. *Round 1:*

   Each key holder P_i^{kh} sends $(\mathsf{refresh}, sid, \mathsf{P}_i^{kh}, kid)$ to the ideal functionality $\mathscr{F}_{\text{prf}}$.

2. *Output:*

   Upon receiving $(\mathsf{refresh.receipt}, sid, \mathsf{P}_i^{kh}, kid')$ from $\mathscr{F}_{\text{prf}}$, key holder P_i^{kh} updates its key id from kid to kid'.

Fig. 4. The Key Share Refreshing Protocol Π_{refresh}

4. Security Analysis

We analyze the security of our screening protocols presented in Sec. 3. In particular, we prove that in a session where the initialization protocol is called first followed by subsequent invocations to all other sub protocols, the four sub-protocols securely compute the corresponding ideal functionalities (defined in Sec. 2.2) in the semi-honest model. In cryptography, the simulation paradigm (see a tutorial in [9]) comes in handy to argue security statement such as "the adversary learns no substantial information beyond knowledge K", by showing that there exist efficient simulators who take as input K and produce a fake view, which looks indistinguishable from the real view of the adversary in the protocol. More specifically, we prove that in such a session, the view of any static, semi-honest adversary $\mathscr{A}$ corrupting the client, the server and $t-1$ key holders (or any subset of these parties) can be efficiently simulated in the ideal world (where the simulator has access to the ideal functionality).

Firstly, we state that under the Decisional Diffie-Hellman assumption the function defined in protocol Π_{prf} is a pseudorandom function.

Lemma 1. *Assume that DDH holds for group $\mathbb{G}$ of order q and that $H : Q^w \to \mathbb{G}$ is a random oracle, then the function $F : \mathbb{Z}_q \times Q^w \to \mathbb{G}$ defined as*

$$F(\alpha, x) = f_\alpha(x) = H(x)^\alpha$$

is a pseudorandom function.

Proof sketch. The authors of [10] proved that the function $f_\alpha := x^\alpha$ is a weak pseudorandom function (i.e., on uniformly random input x). The random input condition can be enforced by applying a random oracle prior to exponentiation, which gives rise up to a standard pseudorandom function (with key homomorphism). $\square$

Next, we prove that the protocol Π_{prf} in Sec. 3.1 securely computes the functionality $\mathscr{F}_{\text{prf}}$ in the semi-honest model, where the adversary $\mathscr{A}$ corrupts the receiver P^r and $t-1$ key holders (or any subset of these parties). This is formally stated as the following lemma:

Lemma 2. *Assuming that H is a random oracle, the protocol Π_{prf} securely computes the functionality $\mathscr{F}_{prf}$ in the semi-honest model, where a static adversary $\mathscr{A}$ can corrupt either the receiver P^r and any $t-1$ key holders $\mathsf{P}^{kh}_{i_1}, \ldots, \mathsf{P}^{kh}_{i_{t-1}}$ (or less).*

Proof Sketch. The case when $\mathscr{A}$ only corrupts key holders is trivial, since the only view of the adversary consists of uniformly random elements in $\mathbb{G}$ and they are independent of the input x and the PRF key α. Therefore, the simulator can simply sample $\tilde{y} \xleftarrow{\$} \mathbb{G}$ and output it as $\mathscr{A}$'s view. The simulation is perfect.

As for the case where $\mathscr{A}$ corrupts the receiver and key holders, it suffices to consider that $\mathscr{A}$ corrupts the receiver P^r and $t-1$ key holders $\mathsf{P}^{kh}_{i_1}, \ldots, \mathsf{P}^{kh}_{i_{t-1}}$.

We first describe the simulator, and then prove the indistinguishablity of the simulation. Firstly, the simulator fixes the random tape of the corrupted parties $\mathsf{P}^{kh}_{i_1}, \ldots, \mathsf{P}^{kh}_{i_{t-1}}$. For the simulation of the key generation protocol, the simulator samples $\tilde{\alpha}_{i_1}, \ldots, \tilde{\alpha}_{i_{t-1}} \xleftarrow{\$} \mathbb{Z}_q$, and then chooses the incoming messages of $\mathsf{P}^{kh}_{i_1}, \ldots, \mathsf{P}^{kh}_{i_{t-1}}$ randomly on condition that the outputs of corrupted key holders in the key generation protocol would be $\tilde{\alpha}_{i_1}, \ldots, \tilde{\alpha}_{i_{t-1}}$ (this is trivial in the semi-honest model since all parties faithfully follow the protocol instructions).

Simulation for the evaluation sub-protocol proceeds as follows. The simulator gets the evaluation result y on x from the ideal functionality $\mathscr{F}_{\text{prf}}$. The simulator derives the random mask β from the receiver's random tape, and outputs $H(x)^{\beta}$ as the view of corrupted key holders. As for the view of the receiver, the simulator computes $H(x)^{\beta\alpha_l}$ (recall the receiver's view consists of t group elements from respective key holders of the form $H(x)^{\beta\cdot\alpha_l}$ where β is the random mask and α_l is the key share of P^{kh}_l) by

$$H(x)^{\beta\cdot\tilde{\alpha}_l} = y^{\beta\cdot\lambda_0} \cdot \prod_{j=1}^{t-1} H(x)^{\beta\cdot\lambda_{i_j}\cdot\tilde{\alpha}_{i_j}},$$

where $l \in [n]$ and $\lambda_0, \lambda_{i_1}, \ldots, \lambda_{i_{t-1}}$ are the appropriate Lagrange coefficients such that

$$\alpha_l = \lambda_0 \cdot \alpha_0 + \sum_{j=1}^{t-1} \lambda_{i_j} \cdot \alpha_{i_j}$$

where $\alpha_{i_1}, \ldots, \alpha_{i_{t-1}}$ are (t,n)-Shamir shares of the secret α_0.

Simulation for the share refreshing protocol proceeds similar to the key generation protocol. The simulator chooses the corrupted key holders' incoming messages randomly on condition that they sum to zero.

The indistinguishability for the key generation and share refreshing phase is straightforward, since no private input is involved in these two sub-protocols. For the evaluation phase, the view generated by the aforementioned simulator would be identical to the real view of $\mathscr{A}$ except that we replace the truly random value y with $H(x)^{\alpha}$, where α is the random PRF key used in the real execution. By the property of (t,n)-secret sharing, condition on the view of the adversary who holds at most $t-1$ shares, the PRF key α is uniformly random. Recall that a PRF $f_{\alpha}(\cdot)$ (see Lem. 1) keyed by uniformly random α behaves like a truly random function in the computational sense. Therefore, the real view and the simulated view are computationally indistinguishable. □

Now, we state the main theorem whose proof is significantly simplified thanks to the modularity of the security analysis (by working in the $\mathscr{F}_{\text{prf}}$-hybrid model).

Theorem 1 (main theorem)**.** *Assume that DDH holds for group $\mathbb{G}$ and that $H : Q^w \to \mathbb{G}$ is a random oracle (as defined in pseudorandom function f_{α}), we have that the four subprotocols in Sec. 3.2 securely compute the corresponding functionalities in Sec. 2.2 in the $\mathscr{F}_{prf}$-hybrid model where a static semi-honest adversary $\mathscr{A}$ can corrupt the server P^s, the client P^c and any $t-1$ key holders $\mathsf{P}^{kh}_{i_1}, \ldots, \mathsf{P}^{kh}_{i_{t-1}}$ (or less).*

Proof sketch. We first consider the cases for corrupting the server P^s or the client P^c or $t-1$ key holders separately, and show that the corresponding view can be efficiently simulated. We then prove that when corrupting any combination of the above, the joint view still can be simulated efficiently.

Corrupted Server. The simulator for the corrupted server needs to generate the view of P^s in new sequence adding protocol Π_{add} and querying protocol Π_{query}. In the ideal functionalities (that correspond to the two protocols), the server receives an index which allows the simulator to distinguish whether the sequence being added or queried has already been added or queried. This is crucial for the consistency of the simulation, for instance, if at some point in the simulation, the administrator adds a sequence that has been queried before, these two values, as part of the server's view, should be identical.

In real protocol executions, the messages P^s receives is uniformly random values (since the protocol is defined in the $\mathscr{F}_{\text{prf}}$-hybrid model where the PRF is replaced by an idealized functionality), these messages can be efficiently simulated. Specifically, the simulator for P^s keeps a list $\tilde{S_1}$. Upon receiving the response message ($\mathsf{add.receipt}$, P^s, sid, i) from $\mathscr{F}_{\text{add}}$ or ($\mathsf{add.receipt}$, P^s, sid, i) from $\mathscr{F}_{\text{query}}$, if $i > |\tilde{S_1}|$, the simulator samples a new random element $\tilde{y} \xleftarrow{\$} \mathbb{G}$, appends it to $\tilde{S_1}$ and outputs it as P^s's view. Otherwise, the simulator outputs $\tilde{y} = \tilde{S_1}[i]$ as P^s's view.

Corrupted Key Holders. The simulation for $t-1$ corrupted server is straightforward, since in $\mathscr{F}_{\text{prf}}$-hybrid model, the key holders only receive key id's, which can be trivially simulated.

Corrupted Client. The client's view is restricted to the querying subprotocol, and therefore the simulator only needs to generate the corresponding view. The simulator simulates

$\mathscr{F}_{\text{prf}}$ by sampling random elements for the client's queries and outputs the querying result obtained from $\mathscr{F}_{\text{query}}$. Since the actual protocol is based on $\mathscr{F}_{\text{prf}}$-hybrid model, the simulation here is perfect.

Corrupted Server and Key Holders. In this case, the simulator combines the simulation of the two respective cases. In short, the simulator keeps a list of random values to simulate the view of the corrupted server. The server's messages are identically distributed to those in the real execution, and so are the key holders' messages. The simulated view is therefore indistinguishable from the real view.

Corrupted Client and Key Holders. The simulation for this case is also a combined simulation of the two respective cases. Since the two simulation have no correlations, the joint simulation is therefore perfect as well.

Corrupted Client and Server. The simulator needs to simulate the adding new sequence protocol and querying protocol. The simulation for this case can be built upon the simulation for a corrupted server alone. Specifically, the ideal functionality $\mathscr{F}_{\text{query}}$ returns an index to the server. If this index indicates the queried value has not appeared in the server's view, its corresponding random value can be sampled uniformly at random. Or else, it has already been sampled, and therefore should be extracted from previous simulations for adding new sequence protocol or querying protocol.

Corrupted Client, Server and Key Holders. In this case, we can base the simulation on that of the previous case. Since the key holders' view only consists of key id's that are not correlated to any other messages, the jointly simulated view is indistinguishable from the real view.

$\square$

5. Performance Evaluation

5.1 Testing Setup

There are many elliptic curves recommended by NIST [7] and more implemented by OpenSSL [13]. Our performance evaluation and comparison are based on the OpenSSL 1.1.1 implementation on a laptop equipped with Intel® Core™ i5-7400 CPU @ 3.00GHz x4 and the Ubuntu 18.04.2 LTS 64bit OS.

Table 2. The performance of various curves in terms of the number of base and random multiplications (i.e., the operations of $s \cdot P$ and $t \cdot Q$ respectively) per second and the number of $s \cdot P + t \cdot Q$ operations per second, where $s,t \in \mathbb{Z}_q$, q is the order of the curve, P is the generator, and Q is an arbitrary element.

curve name	bit sec.	#.base mul/s	#.random mul/s	#.s*P+t*Q/s
secp112r1	112	3305	6889	8982
wap-wsg-idm-ecid-wtls6	112	7692	7485	9175
sect113r1	113	12343	12290	6079
wap-wsg-idm-ecid-wtls1	113	13731	13142	6660
secp128r1	128	6572	6676	8953
sect131r1	131	7097	6998	3473
secp160k1	160	4063	4066	5340
wap-wsg-idm-ecid-wtls7	160	4257	4124	5683
brainpoolP160r1	160	4000	3854	5212
sect163k1	163	5339	5547	2687
wap-wsg-idm-ecid-wtls3	163	5569	5513	2783
c2pnb176v1	176	5429	5277	2621
c2tnb191v3	191	6019	5895	2917
prime192v1	192	3400	3347	4710
sect193r1	193	5154	5151	2565
secp224r1	224	26932	10703	7830
wap-wsg-idm-ecid-wtls12	224	2346	2332	3377
brainpoolP224r1	224	2147	2188	3064
secp256k1	256	1859	1859	2681
prime256v1	256	101937	20462	17406
brainpoolP256r1	256	1912	1920	2753
c2pnb272w1	272	2368	2324	1167
brainpoolP512r1	512	548	550	855
secp521r1	521	5322	2765	2032

5.2 Performance Evaluation

Let us mention that the PRF key generation $\Pi_{\text{prf.gen}}$ and key refreshing operations $\Pi_{\text{prf.refresh}}$ (and the corresponding Initialization and Key Share Refreshing Protocols that invoke them) involve only a few additions and multiplications, and they do not constitute the bottleneck of performance.

Obliviously, the main computationally intensive operation is the Oblivious PRF Evaluation $\Pi_{\text{prf.eval}}$, which is invoked by querying and adding new sequence protocols. It takes quite some group multiplications and exponentiation operations. An exponentiation operation can be computed by $\log q$ multiplications where q is the order of group. Thus, a client needs $2\log q + t$ multiplications and each key holder requires $\log q$ multiplications. As shown in Table 1, we compare the performance on various elliptic curves and highlight the (relatively) good ones in red. The evaluation of the PRF is interactive and thus it depends on the bandwidth too. Concretely, to handle 10^5 queries in the 256-bit security setting, the communication cost between client and server is approximately 14MB, which should not affect the performance too much.

In summary, the performance bottleneck is the basic operations over the elliptic curves despite many other less dominant factors. As illustrated in Table 3, our testing runs on an off-the-shelf (actually out-of-date) personal laptop, which takes about 27 minutes to answer 10^5 queries. We estimate that a high performance server should easily accomplish the task in less than a minute.

Table 3. The estimated numbers of basic protocol operations per second on our laptop implementing those curves highlighted in red in Table 1. Note that "bit Sec." uniquely identifies the corresponding red curve from Tab. 2.

bit Sec.	operation	#.operation/s
113	init	200
113	add	60
113	query	60
113	refresh	200
256	init	200
256	add	60
256	query	60
256	refresh	200
521	init	200
521	add	15
521	query	15
521	refresh	200

6. Quantum Resistance

In this section, we detail why the possibility of efficient quantum computation is relevant to the long-term security of our proposal, and our approach to securing our established privacy properties, especially of the hazard database, against future quantum attacks.

6.1 Rationale for Exploring Quantum-Resistant Approaches

There is considerable debate as to whether quantum computation will become a serious threat to certain techniques in non-quantum, or "classical," cryptography in the foreseeable future. Regardless of the strength of the arguments on each side, we must accept that there is some chance that any cryptographic primitive in our proposal whose hardness assumptions rely on a classical computation setting could be broken by a quantum-capable adversary. One example of a hardness assumption that is considered secure under classical computation, but not quantum computation, is DDH (Decisional Diffie-Hellman). Our oblivious DPRF construction relies on DDH for its security. Under quantum computation, an adversary could efficiently evaluate the oblivious DPRF on arbitrary inputs without involving the threshold number of participating servers. Free DPRF evaluation would permit oracle access to set membership in the database; due to the low entropy of the input sequence space, the database contents could be exfiltrated quickly, constituting a catastrophic failure of the screening system.

We have considered several possible extensions of our basic protocol to address such weaknesses with respect to future quantum attacks. For our purposes, it is sufficient to secure the database contents against a quantum adversary, for now setting aside query privacy, which is much less crucial. Our extensions fall into two categories: information-theoretic DPRF and MPC-embedded encryption.

6.2 PRF with Information-Theoretic Security.

Having obtained the result of the oblivious DPRF $F_k(\cdot)$, the client may participate in another round of PRF evaluation with multiple (possibly the same) parties to compute $G_{k^q}(F_k(\cdot))$ before participating in the two-party membership query protocol P. If this round is made quantum secure, e.g. by using fast block cipher based PRFs, query privacy is still preserved under classical computation and the database is secure against a quantum adversary.

The following approaches, with increasing sophistication, each achieve information-theoretic security (and thus quantum security) by requiring key information from extra sources:

1. In its simplest form, a single trusted server P^Q holds k^q, the key for the quantum-secure PRF. The client sends its computed result $F_k(x)$ to P^Q and receives $G_{k^q}(F_k(x))$.

2. Slightly more securely, several servers P_i^Q each hold pieces of $k^q = (k_0^q, k_1^q, \ldots, k_n^q)$. The quantum-secure "distributed" PRF is computed as the XOR of the independent keyed PRFs, $G_{k^q}(\cdot) = G'_{k_0^q}(\cdot) \oplus G'_{k_1^q}(\cdot) \oplus \cdots \oplus G'_{k_n^q}(\cdot)$. Evaluation of this $G_{k^q}(\cdot)$ requires all servers P_i^Q to be online, a robustness concern.

3. In keeping with threshold, rather than obligatory, participation by trusted third parties being a desirable property, the keys for the second round can be distributed by replication. Using $\binom{n}{t}$ keys replicated across the servers P_i^Q such that every subset of size $n-t$ servers is assigned one unique key, any $t+1$ servers are guaranteed to have at least one copy of all n key pieces between them. $G_{k^q}(\cdot)$ is then computed by XOR of the individually communicated $G'_{k_i^q}(\cdot)$ as above. This approach is only feasible for small t, and does not scale well for t as a constant fraction of n. Still, even with many keys, an implementation utilizing block ciphers could achieve sufficient throughput.

One concern is that this approach also requires high communication overhead due to transmission of many redundant intermediate results.

4. Excessive communication overhead may be avoided by specifying some local computation on each server P_i^Q as in [4] to combine the PRF evaluations over all of its $\binom{n-1}{t-1}$ keys into one value before sending, such that the information transmitted to the querier collectively forms a Shamir secret sharing of the result $G_{k^q}(F_k(x))$.

6.3 PRF Evaluation using MPC.

It may be possible to replace the DPRF from Naor-Pinkas-Reingold (NPR [10]), utilized throughout this document, with a secure multiparty computation (MPC) of a function that serves the same purpose, but that has better security properties in a quantum setting. Like NPR, the query sequences remain private to the client. Unlike NPR, in which each key-holding server only communicates with the client directly and not with other key-holders, MPC requires multiple rounds of communication between the computing parties.

MPC protocols may be quantum-insecure or quantum-secure. Even if a quantum-insecure MPC protocol is used, the MPC could be used to evaluate a quantum-secure function, such a block cipher (e.g. AES-256), for the client.

In this setting, the client holds sequence x, and key holders P^{kh} have their respective secret shares of a secret key k. They jointly run a (classical or quantum) secure MPC that reconstructs the secret key k and then outputs e.g. $F_k(x) :=$ AES-256(k,x), i.e. it encrypts the oblivious DPRF input x under the key k using the symmetric encryption scheme AES-256.

This approach will be slower than the proposed design as it requires interaction among the key-holding servers, but in particular for small numbers of servers it is highly efficient. For example, in the 3-server setting, [2] achieved 1 million secure AES evaluations per second. Whether the extra MPC communication represents too great a slowdown for our application remains to be tested.

When using an information-theoretically secure MPC protocol it is safe to conjecture that the encrypted database is quantum-secure against exposure due to the use of AES-256. Furthermore, the entries are now encrypted under a standard algorithm (AES) and can be decrypted if this ever was required by stakeholders. Finally, recently proposed MPC-friendly ciphers such as LowMC [1] can be employed to achieve higher throughput than by relying on e.g. AES.

The optimal choice of a quantum-secure solution ultimately depends on the choices of n and t in the final implementation. If $\binom{n}{t}$ is small, the PRF with replicated keys and information-theoretic security is likely indicated. Otherwise, the extra communication overhead of a PRF evaluation using MPC may represent the right trade-off. We are confident that at least one of the above schemes will be sufficient to achieve database privacy against a quantum-capable adversary.

7. Splitting the Database

Our current setup consists of one server P^s which holds the database of encrypted DNA. We now discuss the potential problems which this setting poses as well as mitigation methods.

7.1 Rationale

Our current high-level design distributes the database between the server P^s and the key holders $\mathsf{P}^{kh}_1, \dots, \mathsf{P}^{kh}_n$ such that, unless t keyholders and the server collude, they cannot learn anything about the contents. This is because P^s holds oblivious DPRF outputs while the key holders P^{kh}_i have shares of the key. However, the current implementation only achieves this under computational assumptions that are broken in the quantum setting (as was already outlined in Section Sec. 6). Furthermore, if the secret key k that is shared among the key holders P^{kh} is ever stolen, then only a single machine must be corrupted to obtain the whole database. It is therefore potentially necessary to get a solution where the database server is distributed further among parties, perhaps using information-theoretic secret sharing such as Shamir's scheme that we already use in our DPRF construction.

7.2 Secret-Sharing the Database

With the current implementation, P^s holds the hashed values of all the $\approx 10^9$ harmful fragments. That is, the (hashed) values $H(x_i)$ are sorted, and a membership query $H(y)$ will be answered by P^s by binary search (and thus in logarithmic time). We do not attempt to hide the search path from P^s. Under this level of privacy, it is fine to replace P^s by a distributed set of m servers $\mathsf{P}^s_1, \dots, \mathsf{P}^s_m$, maintaining the order of the (hashed) item values, while each value $H(x_i)$ is broken into m secret shares. The membership query $H(y)$ can naturally be done in logarithmic number of steps, where each step involves running a secure computation among the m database servers $\mathsf{P}^s_1, \dots, \mathsf{P}^s_m$ that distributively makes a comparison: $H(y) > H(x_i)$? and branches accordingly.

The efficiency of this approach depends on the following task: Given two x, y (each with secret shares distributed among the servers), securely compute the Boolean value $z = [y > x?]$ with all m servers getting the output value z. This can either be done by converting this function into a circuit and using an off-the-shelf MPC scheme among $\mathsf{P}^s_1, \dots, \mathsf{P}^s_m$. Alternatively, we can use methods that were designed specifically for this particular task such as [11]. The concrete slowdown due to this approach remains to be investigated.

References

[1] M. R. Albrecht, C. Rechberger, T. Schneider, T. Tiessen, and M. Zohner. Ciphers for MPC and FHE. In E. Oswald and M. Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 430–454, Sofia, Bulgaria, Apr. 26–30, 2015. Springer, Heidelberg, Germany.

[2] T. Araki, J. Furukawa, Y. Lindell, A. Nof, and K. Ohara. High-throughput semi-honest secure three-party computation with an honest majority. In Weippl et al. [14], pages 805–817.

[3] H. Chen, Z. Huang, K. Laine, and P. Rindal. Labeled PSI from fully homomorphic encryption with malicious security. In D. Lie, M. Mannan, M. Backes, and X. Wang, editors, *ACM CCS 2018: 25th Conference on Computer and Communications Security*, pages 1223–1237, Toronto, ON, Canada, Oct. 15–19, 2018. ACM Press.

[4] R. Cramer, I. Damgård, and Y. Ishai. Share conversion, pseudorandom secret-sharing and applications to secure computation. In *Theory of Cryptography Conference*, pages 342–362. Springer, 2005.

[5] J. Doerner and a. shelat. Scaling ORAM for secure computation. In B. M. Thuraisingham, D. Evans, T. Malkin, and D. Xu, editors, *ACM CCS 2017: 24th Conference on Computer and Communications Security*, pages 523–535, Dallas, TX, USA, Oct. 31 – Nov. 2, 2017. ACM Press.

[6] eBASH team. ebacs: Ecrypt benchmarking of cryptographic systems. http://bench.cr.yp.to/results-hash.html. accessed 1/22/2020.

[7] C. F. Kerry, A. Secretary, and C. R. Director. Fips pub 186-4 federal information processing standards publication digital signature standard (dss), 2013.

[8] V. Kolesnikov, R. Kumaresan, M. Rosulek, and N. Trieu. Efficient batched oblivious PRF with applications to private set intersection. In Weippl et al. [14], pages 818–829.

[9] Y. Lindell. How to simulate it - a tutorial on the simulation proof technique. Cryptology ePrint Archive, Report 2016/046, 2016. https://eprint.iacr.org/2016/046.

[10] M. Naor, B. Pinkas, and O. Reingold. Distributed pseudo-random functions and KDCs. In J. Stern, editor, *Advances in Cryptology – EUROCRYPT'99*, volume 1592 of *Lecture Notes in Computer Science*, pages 327–346, Prague, Czech Republic, May 2–6, 1999. Springer, Heidelberg, Germany.

[11] T. Nishide and K. Ohta. Multiparty computation for interval, equality, and comparison without bit-decomposition protocol. In *International Workshop on Public Key Cryptography*, pages 343–360. Springer, 2007.

[12] B. Pinkas, T. Schneider, and M. Zohner. Scalable private set intersection based on ot extension. *ACM Transactions on Privacy and Security (TOPS)*, 21(2):7, 2018.

[13] The OpenSSL Project. OpenSSL: The open source toolkit for SSL/TLS. www.openssl.org, April 2003.

[14] E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi, editors. *ACM CCS 2016: 23rd Conference on Computer and Communications Security*, Vienna, Austria, Oct. 24–28, 2016. ACM Press.