2015 IEEE Symposium on Security and Privacy

GenoGuard: Protecting Genomic Data against Brute-Force Attacks

Zhicong Huang*, Erman Ayday†, Jacques Fellay‡, Jean-Pierre Hubaux*, Ari Juels§

* School of Computer and Communication Sciences, EPFL, Switzerland
† Bilkent University, Turkey
‡ School of Life Science, EPFL, Switzerland
§ Jacobs Institute, Cornell Tech, USA

***Abstract*—Secure storage of genomic data is of great and increasing importance. The scientific community's improving ability to interpret individuals' genetic materials and the growing size of genetic database populations have been aggravating the potential consequences of data breaches. The prevalent use of passwords to generate encryption keys thus poses an especially serious problem when applied to genetic data. Weak passwords can jeopardize genetic data in the short term, but given the multi-decade lifespan of genetic data, even the use of strong passwords with conventional encryption can lead to compromise.**

We present a tool, called *GenoGuard*, for providing strong protection for genomic data both today and in the long term. GenoGuard incorporates a new theoretical framework for encryption called honey encryption (HE): it can provide information-theoretic confidentiality guarantees for encrypted data. Previously proposed HE schemes, however, can be applied to messages from, unfortunately, a very restricted set of probability distributions. Therefore, GenoGuard addresses the open problem of applying HE techniques to the highly non-uniform probability distributions that characterize sequences of genetic data.

In GenoGuard, a potential adversary can attempt exhaustively to guess keys or passwords and decrypt via a brute-force attack. We prove that decryption under any key will yield a plausible genome sequence, and that GenoGuard offers an information-theoretic security guarantee against message-recovery attacks. We also explore attacks that use side information. Finally, we present an efficient and parallelized software implementation of GenoGuard.

I. Introduction

Due to major advances in genomic research and to the plummeting cost of high-throughput sequencing, the use of human genomic data is rapidly expanding in several domains, including healthcare (e.g., genomic-based personalized medicine), research (e.g., genome-wide association studies), direct-to-consumer (DTC) services (e.g., ancestry determination), legal cases (e.g., paternity tests), and forensics (e.g., criminal investigation). For example, it is now possible for physicians to adjust the prescription of certain drugs based on the genetic makeup of their patients, for individuals to learn about their genetic predisposition to serious diseases, and for couples to find out if their potential offspring has an increased likelihood of developing rare genetic diseases. Major stakeholders are entering the game; for example, Google is building a cloud platform for storing, processing and sharing genomic data [1].

However, such a vast exploitation of genomic data comes with critical privacy issues. Because genomic data includes valuable and sensitive information about individuals, leakage of such data can have serious consequences, including discrimination (e.g., by a potential employer), denial of services due to genetic predisposition (e.g., by an insurance company), or even blackmail (e.g., using sensitive paternity information). Thus it is crucial to store and manage genomic data in a privacy-preserving and secure way.

Existing mechanisms for protecting the privacy of genomic data include (i) anonymization, which has proven to be ineffective for genomic data [2], [3], (ii) adding noise to published genomic data or statistics for medical research (e.g., to guarantee differential privacy [4], [5], [6]), (iii) computation partitioning [7], and (iv) cryptography (e.g., homomorphic encryption [8], [9], private set intersection [10], etc.). In this work, we focus mainly on the personal use of genomic data, such as healthcare or DTC services.

Appropriately designed cryptographic schemes can preserve the utility of data, but they provide security based on assumptions about the computational limitations of adversaries. Hence they are vulnerable to brute-force attacks when these assumptions are incorrect or erode over time. Given the longevity of genomic data, serious consequences can result. Compared with other types of data, genomic data has especially long-term sensitivity. A genome is (almost) stable over time and thus needs protection over the lifetime of an individual and even beyond, as genomic data is correlated between the members of a single family. It has been shown that the genome of an individual can be probabilistically inferred from the genomes of his family members [11].

In many situations, though, particularly those involving direct use of data by consumers, keys are weak and vulnerable to brute-force cracking *even today*. This problem arises in systems that employ password-based encryption (PBE), a common approach to protection of user-owned data. Users' tendency to choose weak passwords is widespread and well documented [12].

Recently, Juels and Ristenpart introduced a new theoretical framework for encryption called *honey encryption* (HE) [13]. Honey encryption has the property that when a ciphertext is decrypted with an incorrect key (as guessed by an adversary),

This research was undertaken while Erman Ayday was at Ecole Polytechnique Fédérale de Lausanne.

DOI 10.1109/SP.2015.34

the result is a plausible-looking yet incorrect plaintext. Therefore, HE gives encrypted data an additional layer of protection by serving up fake data in response to every incorrect guess of a cryptographic key or password. Notably, HE provides a hedge against brute-force decryption in the long term, giving it a special value in the genomic setting.

However, HE relies on a highly accurate distribution-transforming encoder (DTE) (Section II-B) over the message space. Unfortunately, this requirement jeopardizes the practicality of HE. To use HE in any scenario, we have to understand the corresponding message space quantitatively, that is, the precise probability of every possible message. When messages are not uniformly distributed, characterizing and quantifying the distribution is a highly non-trivial task. Building an efficient and precise DTE is the main challenge when extending HE to a real use case, and it is what we do in this paper. Hopefully, the techniques proposed in this paper are not limited to genomic data; they are intended to inspire those who want to apply HE to other scenarios, typically when the data shares similar characteristics with genomic data.

In this paper, we propose to address the problem of protecting genomic data by combining the idea of honey encryption with the special characteristics of genomic data in order to develop a secure genomic data storage (and retrieval) technique that is (i) robust against potential data breaches, (ii) robust against a computationally unbounded adversary, and (iii) efficient.

In the original HE paper [13], Juels and Ristenpart propose specific HE constructions that rely on existing generation algorithms (e.g. for RSA private keys), or operate over very simple message distributions (e.g., credit card numbers). These constructions, however, are inapplicable to plaintexts with considerably more complicated structure, such as genomic data. Thus substantially new techniques are needed in order to apply HE to genomic data. Additional complications arise when the correlation between the genetic variants (on the genome) and phenotypic side information are taken into account. This paper is devoted mainly to addressing these challenges.

A. GenoGuard

We propose a scheme called GenoGuard. In GenoGuard, genomic data is encoded, encrypted under a patient's password[1], and stored at a centralized biobank. We propose a novel tree-based technique to efficiently encode (and decode) the genomic sequence to meet the special requirements of honey encryption. Legitimate users of the system can retrieve the stored genomic data by typing their passwords.

A computationally unbounded adversary can break into the biobank protected by GenoGuard, or remotely try to retrieve the genome of a victim. The adversary could exhaustively try all the potential passwords in the password space for any genome in the biobank. However, for each password he tries, the adversary will obtain a plausible-looking genome without knowing whether it is the correct one. We also consider the case when the adversary has side information about a victim (or victims) in terms of his physical traits. In this case, the adversary could use genotype-phenotype associations to determine the real genome of the victim. GenoGuard is designed to prevent such attacks, hence it provides protections beyond the normal guarantees of HE.

GenoGuard is highly efficient and can be used by the service providers that offer DTC services (e.g., 23andMe) to securely store the genomes of their customers. It can also be used by medical units (e.g., hospitals) to securely store the genomes of patients and to retrieve them later for clinical use.

[1] A patient can choose a low-entropy password that is easier for him/her to remember, which is a common case in the real world [12].

B. Contributions

Our main contributions in GenoGuard are summarized as follows:

- We propose a novel technique to secure genomic data against data breaches that involve a computationally unbounded adversary (an essential requirement given the longevity of genomic data);
- We design and analyze several distribution models for genome sequences;
- We propose and analyze techniques for preventing an adversary from exploiting side information (physical traits of victims) in order to decrypt genomes;
- We present a formal security analysis of our proposed techniques;
- We implement and show the efficiency of GenoGuard.

Organization

The rest of the paper is organized as follows. In the next section we provide a brief background on genomics and honey encryption. In Section III, we introduce the system model for GenoGuard. In Section IV, we describe in detail the techniques underpinning GenoGuard and analyze their security in Section V. In Section VI, we study the robustness of GenoGuard against adversaries with side information (namely, physical traits of victims). In Section VII, we consider performance, use cases, and other details. In Section VIII, we review related work. Section IX concludes the paper.

II. Background

In this section, we briefly introduce some basic concepts of genomics, as well as the honey encryption scheme [13]. To facilitate future references, frequently used notation is listed in Table I.

A. Genomics

1) Genetic Locus, Allele, and Single Nucleotide Variant: In this paper, we consider a genetic **locus** (plural **loci**) as a position on a chromosome. One of a number of alternative forms at a given locus is called an **allele**. Most of the genome is conserved, in comparison to the reference human sequence, in any given individual. The most abundant type of genetic variants are **single nucleotide variants** (SNVs), in which different alleles are observed at the same chromosomal position. Only about 4 million SNVs are observed per individual; they represent the sensitive information that should be protected. In most cases, there are two alleles at a locus, a **major allele**,

$\mathcal{M}$	sequence (plaintext) space
M	a sequence (message), $\mathsf{M} \in \mathcal{M}$
n	number of SNVs in M
$\mathcal{S}$	seed space
$\mathcal{K}$	key space
$\mathcal{C}$	ciphertext space
p_k	key (password) distribution
p_m	original message distribution
p_d	DTE message distribution
h	storage overhead parameter
$\mathcal{A}$	the adversary against the DTE scheme
$\mathrm{Adv}^{\mathrm{dte}}_{\mathsf{DTE},p_m}(\mathcal{A})$	adversary $\mathcal{A}$'s advantage of distinguishing p_m from p_d
$\mathcal{B}$	the adversary against the HE scheme
$\mathrm{Adv}^{\mathrm{mr}}_{\mathsf{HE},p_m,p_k}(\mathcal{B})$	adversary $\mathcal{B}$'s advantage of recovering the correct sequence

TABLE I: Notations and definitions.

which is observed with a high frequency in the population, and a **minor allele**, which is observed with low frequency. The frequency of an allele in a given population is denoted as the **allele frequency** (AF). An allele takes a value from the set $\{A, T, C, G\}$. We represent a major allele as 0, and a minor allele as 1. Human chromosomes are inherited in pairs, one from the father and the other from the mother, hence each SNV position has a pair of alleles (nucleotides). For example, the i-th SNV (on the DNA sequence) can be represented as $\mathrm{SNV}_i = xy$, where x (and y) is an allele. As the ordering of x and y does not matter, we represent the value of an SNV_i from the set $\{0, 1, 2\}$, based on the number of minor alleles it has. For example, if locus i has major allele A and minor allele G, we represent AA as 0, AG (or GA) as 1, GG as 2.

2) Diploid Genotype and Haploid Genotype: To be consistent throughout the paper, given a sequence of loci, we interpret an individual's **diploid genotype** as a corresponding sequence of SNVs, each of which takes values in $\{0, 1, 2\}$, and a **haploid genotype** as a corresponding sequence of alleles, each of which takes values in $\{0, 1\}$.

3) Linkage Disequilibrium and Recombination: Because chromosomal segments are inherited as blocks, SNVs on a sequence are usually correlated, especially when they are physically close to each other. This correlation is measured by **linkage disequilibrium** (LD) [14]. The strength of LD between two SNVs is usually represented by r^2, where $r^2 = 1$ represents the strongest LD relationship. At meiosis, two DNA sequences exchange genetic information, leading to a novel combination of alleles that is passed on to the progeny. This process is called **recombination**. The recombination rates vary on the different regions of a chromosome.

B. Honey Encryption

Honey encryption [13] is a recently proposed encryption scheme that has the advantage of providing security beyond the brute-force bound over conventional ciphers. In our case, this is a highly desirable property, considering the longevity of genomic data. Suppose a message M is sampled from a distribution p_m over the message space $\mathcal{M}$ and honey encrypted under key $K \in \mathcal{K}$ to yield a ciphertext $C \in \mathcal{C}$. Decryption under an incorrect key $K' \neq K$ yields a fake message M' also from the distribution p_m. In a conventional cipher, when decrypting a ciphertext using a wrong key, the scheme usually produces an invalid[2] message (often denoted by special symbol $\perp$); thus the adversary can easily eliminate wrong keys via a brute-force attack. However, in honey encryption, the adversary does not have such an advantage because the output of the decryption under a wrong key is equivalent to random sampling from p_m. Honey encryption is proposed with a notion called *distribution-transforming encoder* (DTE), as we briefly describe below.

Distribution-Transforming Encoder: A DTE works by transforming the potentially non-uniform message distribution p_m into a uniform distribution over a *seed space* $\mathcal{S}$. Formally, it is a pair of algorithms represented as $\mathsf{DTE} = (\mathsf{encode}, \mathsf{decode})$: encode takes as input a message M and outputs a value in $\mathcal{S}$, whereas decode takes as input a value in $\mathcal{S}$ and outputs a message. encode is probabilistic: A message M can potentially be mapped to one of many possible values that make up a set $\mathcal{S}_{\mathsf{M}} \subseteq \mathcal{S}$, and $\mathcal{S}_{\mathsf{M}} \neq \emptyset$. For any pair of different messages M and M' (where $\mathsf{M} \neq \mathsf{M}'$), $\mathcal{S}_{\mathsf{M}} \cap \mathcal{S}_{\mathsf{M}} = \emptyset$. Moreover, $\bigcup_{\mathsf{M}\in\mathcal{M}} \mathcal{S}_{\mathsf{M}} = \mathcal{S}$. Therefore, encode needs to choose a value randomly in $\mathcal{S}_{\mathsf{M}}$ when transforming M, but decode is deterministic. A good DTE has the property that a randomly selected seed, mapped to the message space, yields roughly the underlying message distribution p_m ($\frac{|\mathcal{S}_{\mathsf{M}}|}{|\mathcal{S}|} \approx p_m(\mathsf{M})$), where $p_m(\mathsf{M})$ is the probability of message M. We further discuss the benefits of this property in Section V.

In the DTE-then-encrypt paradigm proposed in [13], encryption of a message M involves two steps: (i) application of encode to M to yield a seed s, and then (ii) encryption of s under a conventional symmetric cipher SE. HE does not provide IND-CCA (indistinguishability under chosen-ciphertext attack) security. It provides the weaker but still useful property of *message-recovery* (MR) security, described below and formally defined in Section V. Consider the scenario in which an adversary wants to guess the key (K) used for the encryption. Given an ideal cipher model for SE, a randomly selected key corresponds to a permutation selected uniformly at random. Hence, if the adversary tries to decrypt a ciphertext C with a randomly guessed key K', he will obtain a value uniformly sampled from $\mathcal{S}$. If he decodes this value, the output message is equivalent to one sampled from the distribution p_m. Given a good DTE, the adversary cannot distinguish a correct key K from an incorrect one K' with a significant advantage over guessing the key (without knowledge of the ciphertext).

We use the DTE-then-encrypt construction in honey encryption. The setup is described as follows:

- Let p_m denote the distribution over the message space $\mathcal{M}$, p_k denote the distribution over the key (password) space $\mathcal{K}$, $\mathcal{S} = \{0, 1\}^l$ denote the seed space with bit length l, and $\mathcal{C}$ denote the ciphertext space.
- Let $\mathsf{DTE} = (\mathsf{encode}, \mathsf{decode})$ be a DTE scheme. Specifically, $\mathsf{encode}(\mathsf{M}) = S$ and $\mathsf{decode}(S) = \mathsf{M}$, where M is a message and $S \in \mathcal{S}$.

[2]Here "invalid" means a message with an extremely low probability in p_m.

$\mathsf{HEnc}(K, \mathsf{M})$	$\mathsf{HDec}(K, (r, C))$
$S \leftarrow\!\!\$\ \mathsf{encode}(\mathsf{M})$	$S \leftarrow \mathsf{decrypt}(K, C, r)$
$r \leftarrow\!\!\$\ \{0,1\}^B$	$\mathsf{M} \leftarrow \mathsf{decode}(S)$
$C \leftarrow\!\!\$\ \mathsf{encrypt}(K, S, r)$	return M
return (r, C)	

Fig. 1: DTE-then-encrypt construction using a symmetric encryption. $\mathsf{M} \in \mathcal{M}, K \in \mathcal{K}, S \in \mathcal{S}$, and $C \in \mathcal{C}$. The symbol '$' implies randomness of the function. r is a random salt of length B.

- Use a conventional symmetric encryption scheme $\mathsf{SE} = (\mathsf{encrypt}, \mathsf{decrypt})$ with plaintext space $\mathcal{S}$ and ciphertext space $\mathcal{C}$. For block ciphers without padding, $\mathcal{C}$ is the same as $\mathcal{S}$. SE uses random bits uniformly sampled from $\{0,1\}^B$ during encryption, where B is the length of the random bits.

The honey encryption construction $\mathsf{HE}[\mathsf{DTE}, \mathsf{SE}] = (\mathsf{HEnc}, \mathsf{HDec})$ is also shown in Figure 1. However, as we will show, the application of HE to genomes, is far from straightforward. Constructing a good DTE for genetic sequences, one that yields an HE scheme with good MR security bounds, is the main challenge addressed in this paper. Addressing the problem of side information is also a significant challenge.

III. System Model

We consider a scenario where individuals' genomic data is stored in a database (e.g., a biobank) and used for various purposes, such as clinical diagnosis or therapy, or DTC services. In the data collection phase, patients provide their biological samples to a certified institution (CI) that is responsible for the sequencing. Furthermore, each patient also chooses a password (we assume patients can choose low-entropy passwords). The CI pre-processes the sequence data; the most important step is the application of protection mechanisms to the data, such as encryption using the passwords of the patients. The CI then sends the processed data to the biobank. To efficiently protect the data, we assume there are two layers of protection:

- The inner-layer protection is provided by using cryptographic techniques. This layer is necessary for defending against attacks from insiders or someone who hacks into the system and steals the database. This is the focus of this paper.
- The outer-layer protection is the access control; it decides various permissions on the data. Access control has been extensively investigated in the literature [15] and is out of the scope of this paper.

During data retrieval, a user (such as a doctor or the patient himself) first authenticates himself to the system using a passcode[3], or biometric information (e.g., face). After authentication, the user can send a data request to the biobank that processes the request according to access control rules and the biobank responds with the authorized data. Figure 2 gives an overview of the considered architecture.

[3]Chosen by the user or generated by a one-time passcode generator. Note that the passcode used for authentication cannot be the same as the password used for PBE (if PBE is used in GenoGuard that is introduced in Section IV), as the former would require storing a hash of the passcode on the system.

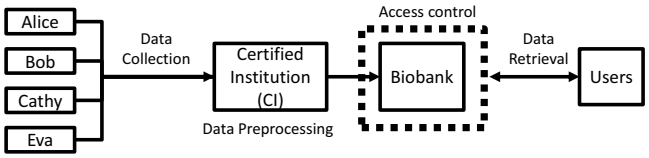

Fig. 2: System model of genomic data storage and retrieval. Patients provide their samples to CI for sequencing. Encrypted sequence data is sent to the biobank and retrieved for various purposes by the users.

A. Genomic Data Representation

We represent each patient's genomic data as a sequence of genetic variants (SNVs) that take values from the set $\{0,1,2\}$, as we discussed before. We assume a sequence M with n SNVs, and we represent such a sequence as $(m_1, m_2, \cdots, m_n)$, where m_i represents an SNV. We use $\mathsf{M}_{i,j}$ to represent the subsequence including all the SNVs between (and including) the i-th and the j-th.

B. Threat Model

We assume the CI to be trusted in order to perform sequencing on patients' samples. An adversary can be anyone (except the CI) who has access to the protected data, such as the biobank, a user who has been granted access permission on part of the data, or an attacker who breaks into the biobank and downloads a snapshot of the database. As a consequence, the adversary can be assumed to have a copy of encrypted sequences. We further assume that the adversary has access to public knowledge about genomics, i.e., AF, LD, recombination and mutation rates. A stronger adversary could even have some side information about a given patient, such as his phenotype, and even some of his SNVs. We represent the adversary's *background knowledge* as $\mathsf{BK} = \{\mathrm{AF}, \mathrm{LD}, \text{recombination and mutation rates}, [\text{side info}]\}$, where "[side info]" means the type and amount of side information depend on the power of an adversary. We also study the effect of phenotype as side information (in Section VI) and propose a general solution in this regard. We emphasize that more side information could result in stronger attacks. Throughout this paper, we assume a *computationally unbounded adversary* who has the capability to efficiently enumerate all keys in $\mathcal{K}$ and to use them to decrypt the data, also called a *brute-force attack*. We also assume that the adversary is honest-but-curious (i.e., follows the protocols honestly, but tries to learn more information than he is authorized for). The adversary's main goal is to break the inner-layer protection and gain access to the plaintext sequences of the patients.

IV. GenoGuard

We describe *GenoGuard*, our solution based on honey encryption, for the secure storage of genomic data. We show the main steps of the protocol in Figure 3. We represent the patient and the user as two separate entities, but they can be

the same individual, depending on the application. We discuss more about the application scenarios in Section VII. Step by step, we discuss the protocol in this section, emphasizing the encoding (Step 3) and decoding (Step 9) steps that are the major features of GenoGuard.

Initially, a patient provides his biological sample (e.g., blood or saliva) to the CI and chooses a password that is used for the encryption (Step 1). The CI does the sequencing on the sample and produces genomic data represented as discussed in Section III-A (Step 2).

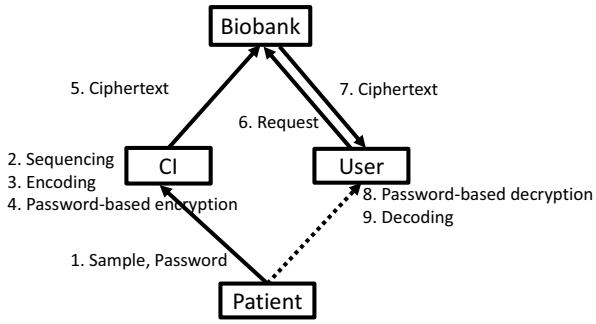

Fig. 3: GenoGuard protocol. A patient provides his biological sample to the CI, and chooses a password for honey encryption. The CI does the sequencing, encoding and password-based encryption, and then sends the ciphertext to the biobank. During a retrieval, a user (e.g., the patient or his doctor) requests for the ciphertext, decrypts it and finally decodes it to get the original sequence.

A. Encoding

We introduce a novel DTE scheme that can be applied efficiently on genome sequences. The general idea is to estimate the conditional probability of an SNV given all preceding ones. In other words, the proposed scheme estimates $P(m_i|\mathsf{M}_{1,i-1})$, the conditional probability of the i-th SNV given preceding SNVs. The probability of a complete sequence M can be decomposed as follows:

$$p_m(\mathsf{M}) = P(m_n|\mathsf{M}_{1,n-1})P(m_{n-1}|\mathsf{M}_{1,n-2})\cdots P(m_2|m_1)P(m_1). \quad (1)$$

The main challenge is to find an efficient way to encode a sequence M into a uniformly distributed seed, which defines the deterministic mapping from M to $\mathcal{S}_\mathsf{M}$ (then we can uniformly pick a value from $\mathcal{S}_\mathsf{M}$). A naive and impractical method would be to enumerate all possible sequences, compute their corresponding probabilities, calculate the cumulative distribution function (CDF) of each sequence in a pre-defined order, and finally assign the corresponding portion of seeds to a sequence. However, given that there are three possible states for each SNV on a sequence of length n, this method incurs both time and space complexity of $O(3^n)$.

Therefore, we propose a novel approach for efficiently encoding such a sequence. The approach works by assigning subspaces of $\mathcal{S}$ to the prefixes of a sequence M. The prefixes of a sequence M are all the subsequences in the set $\{\mathsf{M}_{1,i}|1 \le i \le n\}$. For example, the prefixes of the sequence ATTCG are $\{\mathsf{A}, \mathsf{AT}, \mathsf{ATT}, \mathsf{ATTC}, \mathsf{ATTCG}\}$. We first describe the basic setup as follows:

- Seed space $\mathcal{S}$ corresponds to the interval $[0, 1)$. Each seed is a real number in this interval. In practice, we need to use only sufficient precision (l bits as indicated by the definition $\mathcal{S} = \{0, 1\}^l$) to distinguish between the seeds of different sequences. But, for simplicity of presentation in the rest of this subsection, we assume there is infinite precision.
- To calculate the CDFs, we define a total order $\mathcal{O}$ of all sequences in $\mathcal{M}$, i.e., $\mathcal{O} : \mathcal{M} \to \mathbb{N}$. For any two different sequences M and M', scanning from the first SNV, suppose they begin to differ at the i-th SNV, m_i and m'_i correspondingly (i.e., $\mathsf{M}_{1,i-1} = \mathsf{M}'_{1,i-1}$ and $m_i \neq m'_i$). If the value (0, 1, or 2) of m_i is smaller than that of m'_i, then $\mathcal{O}(\mathsf{M}) < \mathcal{O}(\mathsf{M}')$, otherwise $\mathcal{O}(\mathsf{M}) > \mathcal{O}(\mathsf{M}')$. The CDF of a sequence M is $\mathsf{CDF}(\mathsf{M}) = \sum_{\substack{\mathsf{M}\in\mathcal{M} \\ \mathcal{O}(\mathsf{M})\le\mathcal{O}(\mathsf{M})}} p_m(\mathsf{M}')$ where $p_m(\mathsf{M}')$ is the probability of sequence M'.

In a nutshell, we can encode a sequence with the help of a perfect ternary tree (an example in Figure 4). For a sequence M, starting from the root, (i) if an SNV m_i is 0, we move down to the left branch; (ii) if it is 1, we move down to the middle branch; (iii) if it is 2, we move down to the right branch. As a consequence, each internal node represents a prefix of a sequence, whereas each leaf node represents a complete sequence. We also attach an interval $[L_i^j, U_i^j)$ to each node, where i represents the depth of the node in the tree, and j represents the order of the node at a given depth i, both starting from 0. This interval is the sub seed space that can be assigned to the sequences that start with the prefix represented by the corresponding node.

Here, we describe the details of encoding process (step 3 in Figure 3). Assume we encode a sequence M. It is obvious that the root has an interval $[0, 1)$, namely, $[L_0^0, U_0^0) = [0, 1)$. Depending on the value of SNV m_{i+1}, encoding proceeds from the node that represents $\mathsf{M}_{1,i}$ with order j at depth i to depth $i + 1$ as follows:

- If $m_{i+1} = 0$, go to the left branch and attach an interval $[L_{i+1}^{3j}, U_{i+1}^{3j}) = [L_i^j, L_i^j + (U_i^j - L_i^j) \times P(m_{i+1} = 0|\mathsf{M}_{1,i}))$.
- If $m_{i+1} = 1$, go to the middle branch and attach an interval $[L_{i+1}^{3j+1}, U_{i+1}^{3j+1}) = [L_i^j + (U_i^j - L_i^j) \times P(m_{i+1} = 0|\mathsf{M}_{1,i}), L_i^j + (U_i^j - L_i^j) \times (P(m_{i+1} = 0|\mathsf{M}_{1,i}) + P(m_{i+1} = 1|\mathsf{M}_{1,i})))$.
- If $m_{i+1} = 2$, go to the right branch and attach an interval $[L_{i+1}^{3j+2}, U_{i+1}^{3j+2}) = [L_i^j + (U_i^j - L_i^j) \times (P(m_{i+1} = 0|\mathsf{M}_{1,i}) + P(m_{i+1} = 1|\mathsf{M}_{1,i})), U_i^j)$.

So far, we have not devoted much content to the discussion of computing the conditional probability $P(m_{i+1}|\mathsf{M}_{1,i})$, which will be elaborated later. For now, we focus on how the encoding scheme works on the high level. Finally, when we reach the leaf node with the interval $[L_n^j, U_n^j)$, we pick a seed S uniformly from this range to encode the corresponding

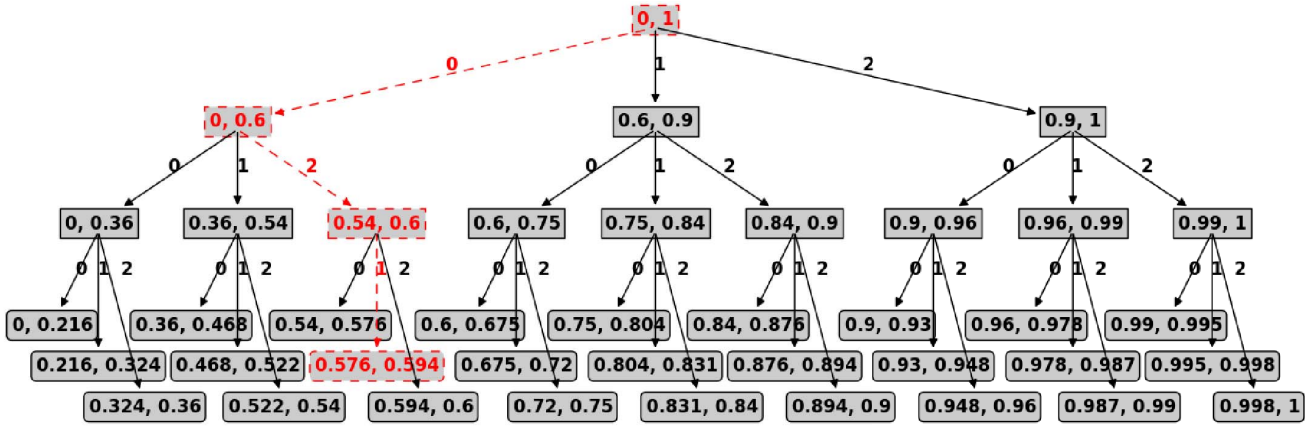

Fig. 4: A toy example of the encoding process. The sequence is of length 3. The sequence that needs to be encoded is $(0, 2, 1)$, shown in red dashed line. Take the second step as an example. We have $P(m_2 = 0|m_1 = 0) = 0.6, P(m_2 = 1|m_1 = 0) = 0.3, P(m_2 = 2|m_1 = 0) = 0.1$, and $[L_1^0, U_1^0) = [0, 0.6)$. Hence the next three intervals are: **(i)** $[L_2^0, U_2^0) = [L_1^0, L_1^0 + (U_1^0 - L_1^0) \times P(m_2 = 0|m_1 = 0)) = [0, 0.36)$; **(ii)** $[L_2^1, U_2^1) = [L_1^0 + (U_1^0 - L_1^0) \times P(m_2 = 0|m_1 = 0), L_1^0 + (U_1^0 - L_1^0) \times (P(m_2 = 0|m_1 = 0) + P(m_2 = 1|m_1 = 0))) = [0.36, 0.54)$; **(iii)** $[L_2^2, U_2^2) = [L_1^0 + (U_1^0 - L_1^0) \times (P(m_2 = 0|m_1 = 0) + P(m_2 = 1|m_1 = 0)), U_1^0) = [0.54, 0.6)$. Note that the intervals in black solid line do not need to be computed when encoding $(0, 2, 1)$. When we reach the leaf $[0.576, 0.594]$, we pick a seed randomly from this range, e.g., 0.583.

sequence. In the following, we give a toy example of this encoding process.

Example (Encoding): Suppose all sequences are of length 3. The sequence M that needs to be encoded is $(0, 2, 1)$. Assume $P(m_1 = 0) = 0.6, P(m_2 = 2|m_1 = 0) = 0.1$, and $P(m_3 = 1|\mathsf{M}_{1,2}) = 0.3$. The encoding process is illustrated in Figure 4.

In Step 4 (in Figure 3), after the encoding is finished, the seed, as a plaintext, is fed into a conventional password-based encryption (PBE) [16] by using the password chosen by the patient (at Step 1). This step is a direct application of PBE, so we skip the details here. The encrypted seed is then sent to the biobank (step 5) that, as a centralized database, receives requests (step 6) from users and responds with the corresponding encrypted data (step 7).

B. Decoding

When an encrypted seed is sent to the user, the user first performs a password-based decryption by using the patient's password (step 8). As discussed, the user could be the patient himself, or the patient can provide his password on behalf of the user. We discuss more on these scenarios in Section VII. Once the user has the plaintext seed, the decoding process (step 9) is the same as the encoding process. Given a seed $S \in [0, 1)$, at each step, the algorithm computes three intervals for the three branches, chooses the interval in which the seed S falls, and goes down along the ternary tree. Once it reaches a leaf node, it outputs the path from the root to this leaf with all chosen SNVs.

C. Moving to Finite Precision

As we mentioned, the current seed space $\mathcal{S}$ is a real number domain with infinite precision. However, considering the size of a DNA sequence, with infinite precision, we could end up having a very long floating-point representation for a sequence, which could cause a high storage overhead. Also, we cannot afford to enumerate all possible sequences to find the smallest precision to represent all the corresponding real numbers. Moreover, if we work with finite precision and decide on the precision *a priori* (without enumerating the sequences), this could result in an inaccurate representation of the sequence distribution, thus causing a security loss. In this subsection, we describe how our proposed DTE scheme can be implemented with finite precision and with negligible effect on security.

For a sequence of length n, with each SNV taking three possible values, we require at least $(n \cdot \log_2 3)$ bits to store the sequence.[4] To optimally implement the scheme, we first select a storage overhead parameter h ($h > \log_2 3$). We use hn bits to encode one sequence. As before, the algorithm works by segmenting intervals based on conditional probabilities. In this case, however, an interval is represented by integers, and not by real numbers of infinite precision. The root interval is $[0, 2^{hn} - 1]$. To better describe the scheme, suppose (during the encoding) we reach the j-th node at depth i on the tree (the root has depth 0 and the leaves have depth n). The interval of this node is denoted by $[L_i^j, U_i^j]$ (U_i^j inclusive, which is different from the infinite-precision case). The segmentation rules are described in the following.

We compute the conditional probabilities for the three branches, P_L (left branch), P_C (middle branch) and P_R (right branch) respectively. Without loss of generality, we assume the three probabilities are ordered as $P_L \geq P_C \geq P_R$ (the following algorithm is similar for different orderings). We initialize a variable $\mathbf{avail} = U_i^j - L_i^j + 1$ to denote the size of the seed space available for allocation. The sizes of seed space that will be allocated to the three branches are denoted by $\mathbf{alloc}_L$ (left branch), $\mathbf{alloc}_C$ (middle branch), and $\mathbf{alloc}_R$ (right branch). Note that $\mathbf{alloc}_L + \mathbf{alloc}_C + \mathbf{alloc}_R = U_i^j - L_i^j + 1$. The algorithm advances as follows:

[4] We do not consider compression techniques here.

(i) If $P_R < \frac{3^{n-i-1}}{\mathbf{avail}}$, then $\mathbf{alloc}_R = 3^{n-i-1}$, otherwise $\mathbf{alloc}_R = \lceil P_R \cdot \mathbf{avail} \rceil$. Then, we update $\mathbf{avail}$ as $\mathbf{avail} = \mathbf{avail} - \mathbf{alloc}_R$.

(ii) If $\frac{P_C}{P_C+P_L} < \frac{3^{n-i-1}}{\mathbf{avail}}$, then $\mathbf{alloc}_C = 3^{n-i-1}$, otherwise $\mathbf{alloc}_C = \lceil P_C \cdot \mathbf{avail} \rceil$. And, we set $\mathbf{alloc}_L = \mathbf{avail} - \mathbf{alloc}_C$.

(iii) Finally, we set the three sub-intervals as:

- $[L_{i+1}^{3j}, U_{i+1}^{3j}] = [L_i^j, L_i^j + \mathbf{alloc}_L - 1]$;
- $[L_{i+1}^{3j+1}, U_{i+1}^{3j+1}] = [L_i^j + \mathbf{alloc}_L, L_i^j + \mathbf{alloc}_L + \mathbf{alloc}_C - 1]$;
- $[L_{i+1}^{3j+2}, U_{i+1}^{3j+2}] = [L_i^j + \mathbf{alloc}_L + \mathbf{alloc}_C, U_i^j]$.

The intuition behind the above conditions is that we need to allocate at least one integer (seed) for one sequence. To ensure this, when we want to move down to a branch, we need to guarantee that the size of the seed space allocated for this branch is not smaller than the total number of sequences belonging to this branch. The requirement is satisfied from the beginning by setting the root interval as $[0, 2^{hn} - 1]$ and never violated in the algorithm. This method causes a deviation from the original sequence distribution. In Section V, we quantify the security loss due to such deviation and prove that it is negligible.

D. Modeling Genome Sequences

To compute the conditional probabilities in Equation (1) efficiently, we introduce several models and compare their goodness of fit in real genome datasets.

1) Modeling with linkage disequilibrium and allele frequency: With LD and AF, we can compute the joint probability of two SNVs, $P(m_i, m_j)$. However, to compute the conditional probability $P(m_{i+1}|\mathsf{M}_{1,i})$, we have to simplify the model (Equation (1)) because public LD values are always given pairwise in the literature. Although there could be multiple pairwise LD relations for SNV_{i+1}, we adopt the following heuristic method: We consider only the previous SNV that has the strongest LD with SNV_{i+1}. Such an LD usually occurs between neighboring SNVs on the DNA sequence, hence we have $P(m_{i+1}|\mathsf{M}_{1,i}) \approx P(m_{i+1}|m_i) = \frac{P(m_{i+1}, m_i)}{P(m_i)}$. This is the first-order Markov chain that was considered also in genomics [17].

This model fails to capture the correlation between distant SNVs. However, we argue that it approximates the genome sequence model better than the uniform distribution model used in conventional encryption, as we will see later in model comparison with real datasets.

2) Modeling by building k-th-order Markov chains on a dataset: With this method, we assume the correlation in a genome sequence can be captured by a k-th-order Markov chain, where the conditional probability of SNV_{i+1} depends on the k preceding SNVs. In other words, we estimate the conditional probability as

$$P(m_{i+1}|\mathsf{M}_{1,i}) \approx P(m_{i+1}|\mathsf{M}_{i-k+1,i}). \tag{2}$$

Researchers have tried to build such a genetic Markov model in a different context [18]. However, to the best of our knowledge, there is no public data (like LD) available for these models. In a similar manner, we build the k-th-order Markov model on a real dataset, for different k values. Assume the dataset has N sequences. We use $F(\mathsf{M}_{i,j})$ to represent the frequency of subsequence $\mathsf{M}_{i,j}$ between SNVs i and j in the dataset. The k-th-order Markov model is built by computing

$$P(m_{i+1}|\mathsf{M}_{i-k+1,i}) = \begin{cases} 0 & \text{if } F(\mathsf{M}_{i-k+1,i}) = 0, \\ \frac{F(\mathsf{M}_{i-k+1,i+1})}{F(\mathsf{M}_{i-k+1,i})} & \text{if } F(\mathsf{M}_{i-k+1,i}) > 0. \end{cases} \tag{3}$$

Due to the constraint of the dataset size, k normally can only take small values to avoid overfitting of the model. For example, in HapMap diploid genotype datasets, N is smaller than 200 for each population. For $k = 3$, there are 81 possible configurations for $\mathsf{M}_{i-k+1,i+1}$, which makes the average frequency for each configuration quite small, hence the model has modest statistical significance due to this sparsity problem. We introduce this model as a possible direction and use it to emphasize the importance of higher-order correlation, which will be shown in the evaluation. The k-th-order Markov chain serves as a bridge to the next more promising model.

3) Modeling with recombination rates: Although higher-order Markov models might better model genome sequences, these models seem unlikely to be practical because of the difficulty of accurately estimating all the necessary parameters in available datasets. Inspired by the modeling method used by Li and Stephens [19], we can address the problem from a different viewpoint. Given a set of k existing haploid genotypes $\{h_1, h_2, ..., h_k\}$, another haploid genotype h_{k+1} to be observed is an imperfect mosaic of $h_1, h_2, ..., h_k$, due to genetic recombination and mutation (Figure 5). This reproduction process is actually a hidden Markov model with a sequence of n states (the number of loci in a haploid genotype):

- **Markov chain states:** State j, X_j, can take a value from 1 to k, representing the original haploid genotype for locus j;
- **Symbol emission probabilities:** $h_{i,j}$ denotes the allele (0 or 1) at locus j in haploid genotype i. To produce h_{k+1}, at state j, an allele $h_{k+1,j}$ is output with a certain probability, depending on the allele of the original haploid genotype (X_j) and the mutation rate;
- **Transition probabilities:** Transition probabilities from state j to state $j+1$ depend on the recombination rate between locus j and $j+1$.

With this model, we can compute the probability of a haploid genotype h_{k+1}, that is, $P(h_{k+1}|h_1, ..., h_k)$. The computation is done with the well-known forward-backward algorithm for hidden Markov models [20]. The probability of a genome sequence M, which is the coupling of two haploid genotypes, can be computed similarly by extending this hidden Markov model so that state j will take a value pair (X_j^1, X_j^2), where X_j^1 denotes the first original haploid genotype and X_j^2 denotes the second. Such an extension technique has been detailed in a genotype imputation scenario [21]. The conditional probability $P(m_{i+1}|\mathsf{M}_{1,i})$ can then be computed in the intermediate steps of the forward algorithm. Model and algorithm details are given in Appendix A.

The correlation between two SNVs, which is considered in the previous two models, is essentially the result of recombination in genome sequences. With this recombination model,

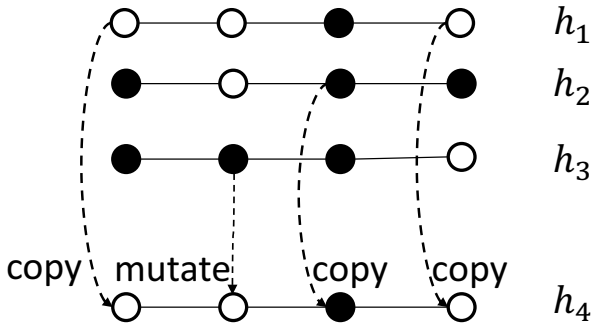

Fig. 5: An example showing how the haploid genotype h_4 is interpreted as an imperfect mosaic of a given set of haploid genotypes $\{h_1, h_2, h_3\}$, based on recombination and mutation. Each haploid genotype can be as long as the whole genome, but we show only four loci here to explain the idea. White circle means allele 0 for that locus, whereas black circle means allele 1. The first allele of haploid genotype h_4 is copied from h_1. Though the second allele comes from h_3, it mutates to a different allele. The third allele is copied from h_2, and the fourth is copied from h_1. Note that this shows just one possible process to get h_4 from $\{h_1, h_2, h_3\}$, and as there are many other possibilities, the task of this model is to compute the probability of observing h_4 by taking all the possible underlying processes into account, which constitutes a hidden Markov model.

we are able to capture the high-order correlation efficiently, without having to estimate a large number of parameters.

4) Goodness of fit of the models: To evaluate the models, we used different types of real genomic datasets from HapMap, for the population CEU (Utah residents with Northern and Western European ancestry from the CEPH collection) [22], including:

- A diploid genotype dataset that contains 165 individuals, each having 22 pairs of autosomes (different from sex chromosomes that are discussed in Section VI-A). The shortest chromosome contains 17304 SNVs, whereas the longest one contains 102157 SNVs;
- A haploid genotype dataset that contains 234 haploid genotypes, each of which has the same sequence of loci as that in the diploid genotype dataset on the 22 chromosomes;
- Allele frequency and linkage disequilibrium datasets for each chromosome;
- Recombination rates for each chromosome.

We performed a chi-square goodness-of-fit test to show how well each model fits the diploid genotype dataset. We divided the sequence space $\mathcal{M}$ into B bins with equal probability. The chi-square statistic is defined as

$$\chi^2 = \sum_{i=1}^{B} \frac{(O_i - E_i)^2}{E_i}, \tag{4}$$

where O_i is the observed frequency for bin i, and E_i is the expected frequency for bin i. The null hypothesis H_0 is that the

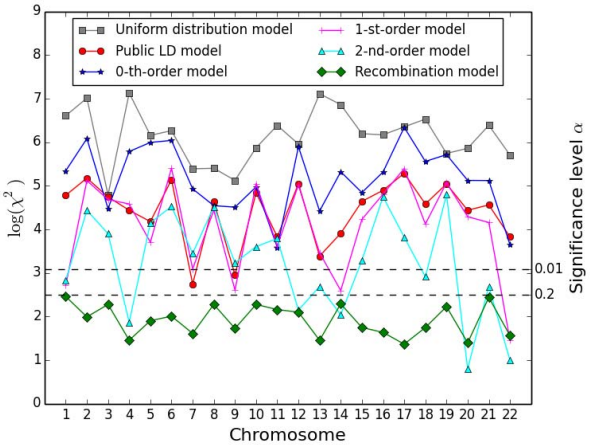

Fig. 6: Chi-square goodness-of-fit tests for different genome sequence models on 22 chromosomes. The x-axis is the chromosome number, from 1 to 22. To graphically show the results at a fine scale, the left y-axis is transformed to the logarithm of chi-squared statistic. The right y-axis shows one frequently used significance level, $\alpha = 0.01$, and another significance level, $\alpha = 0.2$. The uniform distribution model is the one used in conventional encryption. The "public LD model" is built with public LD and AF data. The "0-th", "1-st", "2-nd"-order models are the Markov models built on the dataset. Finally, the "recombination model" is built based on genetic recombination and mutation. Most models are rejected at $\alpha = 0.01$, whereas the recombination model cannot be rejected even at $\alpha = 0.2$, which shows a good fit of this model on real datasets.

data follows the specified distribution model. B is chosen with an empirical formula in statistical theory [23] ($B = \lfloor 1.88N^{\frac{2}{5}} \rfloor$ where N is the sample size). We performed several rounds of the test for different B values around the empirical one and they all gave similar results. Hence we set B to be 10, and show the results in Figure 6. From the chi-square statistics, we can see that uniform distribution indeed gives a poor model of genome sequences. The 0-th-order model built on the dataset is also not appropriate because it does not take the correlation among SNVs into account. The model built with public LD and AF performs similarly with the first-order model built on the dataset, which is reasonable because they both consider only the first-order correlation. The second-order model is better than the previous four models, but it is not stable across different chromosomes: in many chromosomes, we can reject the null hypothesis H_0 at the significance level (α) of 0.01. The recombination model performs best among these models because it captures high-order correlations that are naturally caused by the underlying recombination mechanism. Moreover, the model is stable across all tests and cannot be rejected, even at the significance level of 0.2 in every chromosome, which shows a good fit of this model on real datasets. Therefore, we keep this model for our scheme.

V. Security Analysis

In this section, we prove the security of our proposed DTE scheme, with regard to the scheme in finite precision.

$\underline{\text{SAMP1}^{\mathcal{A}}_{\mathsf{DTE}}}$

$\mathsf{M}^* \leftarrow_{p_m} \mathcal{M}$
$S^* \leftarrow_{\$} \mathsf{encode}(\mathsf{M}^*)$
$b \leftarrow_{\$} \mathcal{A}(\mathsf{M}^*, S^*)$
return b

$\underline{\text{SAMP0}^{\mathcal{A}}_{\mathsf{DTE}}}$

$S^* \leftarrow_{\$} \mathcal{S}$
$\mathsf{M}^* \leftarrow \mathsf{decode}(S^*)$
$b \leftarrow_{\$} \mathcal{A}(\mathsf{M}^*, S^*)$
return b

Fig. 7: Game defining the DTE advantage. In $\text{SAMP1}^{\mathcal{A}}_{\mathsf{DTE}}$, sequence M^* is sampled according to p_m, whereas in $\text{SAMP0}^{\mathcal{A}}_{\mathsf{DTE}}$, M^* is equivalently sampled according to p_d. The adversary's output b is 0 or 1, indicating his guess on whether he is in $\text{SAMP0}^{\mathcal{A}}_{\mathsf{DTE}}$ or $\text{SAMP1}^{\mathcal{A}}_{\mathsf{DTE}}$.

Once the algorithm allocates seed space of size 3^{n-i-1} to a branch at step i (as in Section IV-C), each following step simply segments an input interval into three parts of equal size. Hence there is only one seed for each sequence in the sub-tree under the branch of step i. As discussed in Section IV-C, in such a case, the subinterval of the j^{th} node at depth i of the tree will contain 3^{n-i-1} integers that are exactly the number of sequences under that branch.

The goal in constructing a DTE is that decode applied to uniform points (in the seed space) provides sampling close to that of the target distribution p_m; this is the sequence distribution produced by the k^{th}-order Markov chain. The seed space $\mathcal{S}$ is the integer interval $[0, 2^{hn} - 1]$ (i.e., $l = hn$). We define p_d to be the DTE message distribution over $\mathcal{M}$ by

$$p_d(\mathsf{M}) = P[\mathsf{M}' = \mathsf{M} : S \leftarrow_{\$} \mathcal{S}; \mathsf{M}' \leftarrow \mathsf{decode}(S)].$$

The additional security provided by honey encryption depends on the difference between p_m and p_d. Intuitively, p_m and p_d are "close" in a secure DTE. Next, we quantify this difference for the proposed DTE scheme. Let P^i_m be the original probability of the prefix sequence $\mathsf{M}_{1,i}$, namely, $P^i_m = \sum_{\substack{\mathsf{M} \in \mathcal{M} \\ \mathsf{M}_{1,i} = \mathsf{M}_{1,i}}} p_m(\mathsf{M}')$.

We define P^i_d similarly in the distribution p_d. The complete proofs of the following analysis are available in the full version of this paper [24].

Lemma 1. $\forall \mathsf{M} \in \mathcal{M}, |p_m(\mathsf{M}) - p_d(\mathsf{M})| < \frac{1}{2^{(h - \log_2 3)n}}$.

Lemma 1 bounds the largest difference between $p_m(\mathsf{M})$ and $p_d(\mathsf{M})$. It gives rise to the following important theorem that bounds the DTE advantage of an adversary, introduced by honey encryption. The DTE advantage is formally defined by the following definition.

Definition 1. *Let $\mathcal{A}$ be an adversary attempting to distinguish between the two games shown in Figure 7. The advantage of $\mathcal{A}$ for the sequence distribution p_m and encoding scheme* $\mathsf{DTE} = (\mathsf{encode}, \mathsf{decode})$ *is*

$$\text{Adv}^{\text{dte}}_{\mathsf{DTE}, p_m}(\mathcal{A}) = |P[\text{SAMP1}^{\mathcal{A}}_{\mathsf{DTE}} \Rightarrow 1] - P[\text{SAMP0}^{\mathcal{A}}_{\mathsf{DTE}} \Rightarrow 1]|.$$

Theorem 1. *Let p_m be the sequence distribution and* $\mathsf{DTE} = (\mathsf{encode}, \mathsf{decode})$ *be the transformation scheme using hn bits. Let $\mathcal{A}$ be any sampling adversary, then*

$$\text{Adv}^{\text{dte}}_{\mathsf{DTE}, p_m}(\mathcal{A}) \leq \frac{1}{2^{(h - 2\log_2 3)n}}.$$

$\underline{\text{MR}^{\mathcal{B}}_{\mathsf{HE}, p_m, p_k}}$

$K^* \leftarrow_{p_k} \mathcal{K}$
$\mathsf{M}^* \leftarrow_{p_m} \mathcal{M}$
$C^* \leftarrow_{\$} \mathsf{HEnc}(K^*, \mathsf{M}^*)$
$\mathsf{M} \leftarrow_{\$} \mathcal{B}(C^*)$
return $\mathsf{M} = \mathsf{M}^*$

Fig. 8: Game defining MR security. Given ciphertext C^* (encrypted from M^*), adversary $\mathcal{B}$ is allowed to guess the message by brute-force attack. $\mathcal{B}$ wins the game if his output message M is the same as the original message M^*.

Proof Sketch: The proof follows Theorem 6 in [13]. ■

The last step of the security analysis is the quantification of *message recovery (MR) security* for any adversary $\mathcal{B}$ against the encryption scheme HE.

Definition 2. *Let $\mathcal{B}$ be the adversary attempting to recover the correct sequence given the honey encryption of the sequence, as shown in Figure 8. The advantage of $\mathcal{B}$ against* HE *is*

$$\text{Adv}^{\text{mr}}_{\mathsf{HE}, p_m, p_k}(\mathcal{B}) = P[\text{MR}^{\mathcal{B}}_{\mathsf{HE}, p_m, p_k} \Rightarrow \mathsf{true}].$$

We emphasize that p_k, the password distribution, is non-uniform. We assume the most probable password has a probability w. Using Lemma 1 and Theorem 1, we can establish the following theorem.

Theorem 2. *Consider* $\mathsf{HE}[\mathsf{DTE}, H]$ *(the detailed definition is available in [13]) with H (the hash function) modeled as a random oracle and* DTE *using an hn-bit representation. Let p_m be the sequence distribution with maximum sequence probability γ, and p_k be a key distribution with maximum weight w. Let $\alpha = \lceil 1/w \rceil$. Then for any adversary $\mathcal{B}$,*

$$\text{Adv}^{\text{mr}}_{\mathsf{HE}, p_m, p_k}(\mathcal{B}) \leq w(1 + \delta) + \frac{3^n + \alpha}{2^{(h - \log_2 3)n}}, \qquad (5)$$

where $\delta = \frac{\overline{\alpha}^2}{2\overline{b}} + \frac{e\overline{\alpha}^4}{27\overline{b}^2}(1 - \frac{e\overline{\alpha}^2}{\overline{b}^2})^{-1}$ and $\overline{\alpha} = \lceil 3/w \rceil$ and $\overline{b} = \lfloor 2/\gamma \rfloor$.

Proof: The proof is similar to Corollary 1 in [13]. We omit the redundant details and specify the necessary modifications in the following.

p_m is a non-uniform sequence distribution and we assume $\gamma \leq 3 - \sqrt{5} \approx 0.76$, which is a requirement for Corollary 1 (in [13]). This assumption is reasonable considering the length of the sequence n (≥ 20000)[5]. To estimate γ, we can consider the sequence with all major alleles and pessimistically assume each major allele frequency is 0.995, large enough to give an upper bound for real datasets. Then, γ can be estimated by $0.995^{20000} \approx 2.89 \times 10^{-44} \ll 3 - \sqrt{5}$.

The term $\frac{3^n + \alpha}{2^{(h - \log_2 3)n}}$ is achieved by replacing $\text{Adv}^{\text{dte}}_{\mathsf{DTE}, p_m}(\mathcal{A}) \leq \frac{1}{2^l}$ with our Theorem 1, and $|p_m(\mathsf{M}) - p_d(\mathsf{M})| < \frac{1}{2^l}$ with our Lemma 1 in the proof

[5]We need to focus only on one chromosome because there is no LD between chromosomes. The number 20000 is based on the observation of chromosome 22 (one of the shortest chromosomes) in a real dataset from the International HapMap Project.

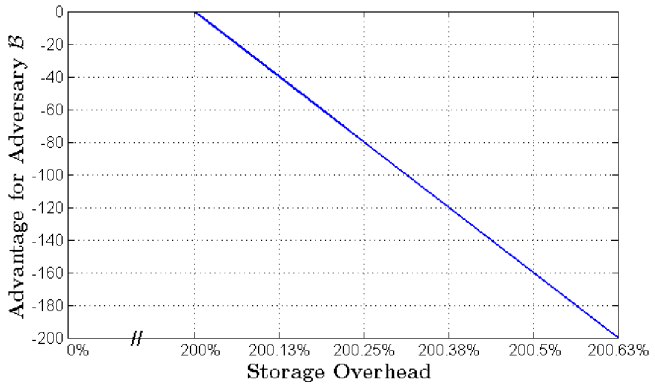

Fig. 9: Adversary advantage versus storage overhead. Without encryption, the minimum storage for a sequence of n SNVs is $n \cdot \log_2 3$ bits. The x-axis is the expansion ratio between the storage with GenoGuard and the storage without encryption, namely, $\frac{hn}{n \cdot \log_2 3} = \frac{h}{\log_2 3}$. The y-axis is logarithm of the security loss term, $\log_2 \Delta_{\text{Adv}}$, that is part of the advantage of the message recovery adversary $\mathcal{B}$ (Equation (5)). With GenoGuard, to ensure a security loss smaller than 2^{-200}, we only need a storage expansion ratio that is slightly larger than 2.

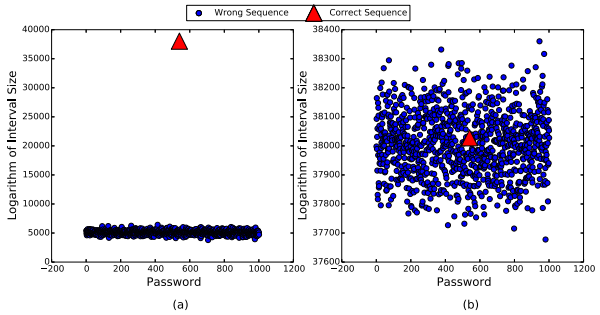

Fig. 10: Experimental security evaluation. We encrypted a genome with a given password from a pool of 1000 passwords (for simplicity, we assume that the passwords are integers from 1 to 1000). Each point represents one decryption result using an integer from the password pool (the x-axis). The y-axis is the logarithm[6] of the interval size of the decrypted sequence when encoded with the recombination model. (a) With a conventional PBE scheme [16], all the wrong passwords have been ruled out except the correct one; (b) Obviously, with GenoGuard, no password can be excluded.

of Corollary 1 (in [13]). Essentially, $\frac{3^n+\alpha}{2^{(h-\log_2 3)n}}$ is the security loss due to DTE imperfectness that causes the difference between p_m and p_d. ■

As mentioned in the proof, we denote $\Delta_{\text{Adv}} = \frac{3^n+\alpha}{2^{(h-\log_2 3)n}}$ as the security loss term. Consider a case where $n = 20000$, $h = 4$, and $\gamma = 2.89 \times 10^{-44}$. If p_k is a password distribution, then w can be estimated to be $1/100$ according to Bonneau's Yahoo! study [25], in which the most common password was selected by 1.08% of users. In this case, Δ_{Adv} is negligible ($\approx 2^{-16600}$), and $\delta \approx 0$, hence the upper bound on message recovery advantage is $w = 1/100$. If we consider an adversary who trivially decrypts the ciphertext with the most probable key and then outputs the resulting sequence, he can win the message recovery (MR) game with probability $1/100$. Hence, the bound is essentially tight. However, this case only happens if the patients choose weak passwords according to the previous password study.

To choose the storage overhead parameter h in practice, we consider how it affects the security loss term Δ_{Adv}. Since α is negligible compared to 3^n, we have $\Delta_{\text{Adv}} \approx \frac{1}{2^{(h-2\log_2 3)n}}$. Taking the logarithm of Δ_{Adv}, we can observe that it has a linear relationship with h, as shown by Figure 9. For example, when $\frac{h}{\log_2 3} = 200.63\%$, we have $\Delta_{\text{Adv}} \approx 2^{-200}$. Hence, with a storage overhead slightly larger than two times (compared to the storage of a plaintext sequence), we achieve a negligible security loss.

Security under Brute-Force Attacks: To illustrate the security guarantee of GenoGuard, we conducted two experiments to compare GenoGuard with a simple (unauthenticated) PBE algorithm under brute-force attacks. For the simple PBE algorithm, we encoded the genome by assuming a uniform distribution in GenoGuard encoding, specifically by setting all edge weights in the tree to be equal (namely, $\frac{1}{3}$). Thus, its decryption under any key yields a valid genome ("valid" does not necessarily mean "plausible", as we will show). We show here that for this PBE scheme a very simple classifier suffices for identifying the correctly decrypted genome with high probability. We encrypted a victim's chromosome 22 (see Section VII-A for dataset description and implementation details) with a given password from a password pool of size 1000 (without loss of generality, we assume that the passwords are integers from 1 to 1000). We chose "539" as the correct password for both experiments; and we assumed that the adversary knows the correct password is a number from the password pool and that he performs a simple brute-force attack. In real life, brute-force attacks can be carried out if the adversary knows that the correct password has a limited number of characters (hence memorizable by users) or even a fixed length (e.g., six-digit PIN code). **In the first experiment**, we encrypted the victim's sequence directly with the PBE scheme in [16] (after encoding by assuming a uniform distribution). **In the second experiment**, we followed the same procedure except that we encrypted the victim's sequence by using the GenoGuard. Note that in our proposed DTE, the size of the interval of a leaf in the ternary tree is proportional to the probability of the corresponding sequence. In both experiments, to rule out wrong passwords, we computed the interval sizes of the decrypted sequences and observed the result. Figure 10 shows the result of the two experiments. We observe that if the sequence is protected by a direct application of the PBE scheme, the adversary can exclude most passwords in the attack because the corresponding decrypted sequences have much lower probabilities than that of the correct sequence. In this example, only the correct password is retained, as shown in Figure 10 (a). With GenoGuard, on the contrary, the correct sequence is buried among all the decrypted sequences, hence it is almost impossible to reject any wrong password.

[6]Note that hn is close to 80000, hence the interval size is a huge integer and is better expressed as its logarithm with base 2.

VI. Towards Phenotype-Compatible GenoGuard

An individual's physical traits (such as gender, ancestry and hair color) are highly correlated to his DNA sequence. Recently, researchers showed that it is even possible to model facial traits of an individual from his DNA [26]. Although such progress in human genetics is desirable for many applications (e.g., forensics), it can pose a threat to our proposed technique. In particular, such correlations could be used as side information by an adversary who tries to obtain the sequence of a specific victim (e.g., by trying various potential passwords). For instance, if the adversary knows that an encrypted sequence belongs to a victim of Asian ancestry, he might be able to eliminate a (wrong) password if the genetic sequence obtained using this password does not belong to an individual of Asian ancestry.

In genetics, gender and ancestry are the most well studied human genetic traits. These traits have deterministic genotype-phenotype associations, whereas other traits (such as hair color) have less certain (probabilistic) genotype-phenotype associations. In this section, we first show that the security of GenoGuard is not affected by traits with deterministic genotype-phenotype associations. Our main goal is to show that if an adversary knows a phenotype (physical trait) of a victim, he always retrieves a decrypted sequence that is consistent with the corresponding phenotype, even if he types a wrong password. Next, we quantify the privacy loss if an adversary has information about other traits (with probabilistic genotype-phenotype associations) of a victim via a privacy analysis.

A. Traits with Deterministic Genotype-Phenotype Associations

Gender: Gender is determined by sex chromosomes, namely, *X chromosome* and *Y chromosome*. Females have two copies of the X chromosome, whereas males have one X chromosome and one Y chromosome. Note however that X chromosome and Y chromosome have different lengths. Therefore, the adversary can immediately ascertain whether a ciphertext comes from an X chromosome or a Y chromosome because the latter is shorter than the former. As we mentioned in Section IV-C, (when implementing GenoGuard) the whole interval $[0, 2^{hn} - 1]$ is determined by the length n of the sequence. To deal with the gender problem, we use the length of X chromosome for both sex chromosomes. In other words, X chromosome and Y chromosome are encoded in the same interval $[0, 2^{hn} - 1]$, where n is the length of X chromosome.[7] In this way, the adversary cannot infer any information about the gender because the ciphertext is always of the same length, whether it belongs to a male sequence or a female sequence. Furthermore, if the adversary knows the gender of a victim, he will always get a consistent sequence (based on the gender) when he decodes the ciphertext by using the corresponding public knowledge of Y (or X) chromosome.

Ancestry: Research has shown that ancestry information can be accurately inferred from DNA sequences. For example, the sequence of an individual of Asian ancestry usually has different combinations of SNVs compared to an individual of European origin. In genetics, ancestry can be inferred with a number of methods, e.g., *principal component analysis (PCA)* followed by *k-means clustering* [27]. In this method, a training set is comprised of a number of individuals, each of which is genotyped on a predefined set of SNVs (the most informative SNVs). This training set is then fed into PCA in order to find several principal components. After the dataset is projected on these principal components, k-means clustering is applied to cluster the individuals into different ethnicities.

What we want to achieve in GenoGuard is ethnic plausibility: the principal components of the decrypted genome-wide genotyping data should be broadly similar to those from a real genome. Hence, we argue that the decoding operation with knowledge of recombination rates and haploid genotype dataset from a specific population always yields a sequence belonging to that population. To verify this, we conducted an experimental analysis depicted in the following.

We used Phase III[8] data from the HapMap dataset [22]. In this dataset, we chose 3 populations for our evaluation:

(i) ASW (African ancestry in Southwest USA), with 90 samples;
(ii) CEU (Utah residents with Northern and Western European ancestry from the CEPH collection), with 165 samples;
(iii) CHB (Han Chinese in Beijing, China), with 90 samples.

We selected 100 SNVs to infer ancestry according to [28]. First, we applied PCA on the above dataset and selected the first two principal components. The projection of the dataset on the two principal components can be seen in Figure 11(a). We encrypted a sequence from a specific population (e.g., ASW) by using GenoGuard. Then, for each of the three aforementioned populations, we decrypted the ciphertext with randomly guessed passwords 100 times, generating 100 random sequences for each case (in total, we generated 300 sequences). Finally, we projected these 300 sequences on the principal components and observed the result, as shown in Figure 11(b), (c), and (d). We conclude that decoding with public knowledge from a population always produces a sequence of that population, which proves that ancestry inferred from a sequence does not pose a threat to our proposed technique. We leave the case for people with mixed blood for the future work, but a reasonable assumption is that corresponding public knowledge could be available for mixed-blood people in the future.

B. Traits with Probabilistic Genotype-Phenotype Associations

In theory, the idea we introduce for ancestry also works for other traits: incorporate phenotype-related data during encoding. For the case of ancestry, such data is provided as population-specific haploid genotype dataset. However, such data is not easily available for many other traits (e.g., those with probabilistic genotype-phenotype associations) and genotype-phenotype associations is ongoing research. In the following, we quantify the privacy loss when the phenotype of a victim is not taken into account during encoding, but is

[7] There is no LD between two different chromosomes, so each chromosome can be encrypted as an independent sequence.

[8] The third phase of the International HapMap project. This phase increases the number of DNA samples covered from 270 in phases I and II to 1,301 samples from a variety of human populations.

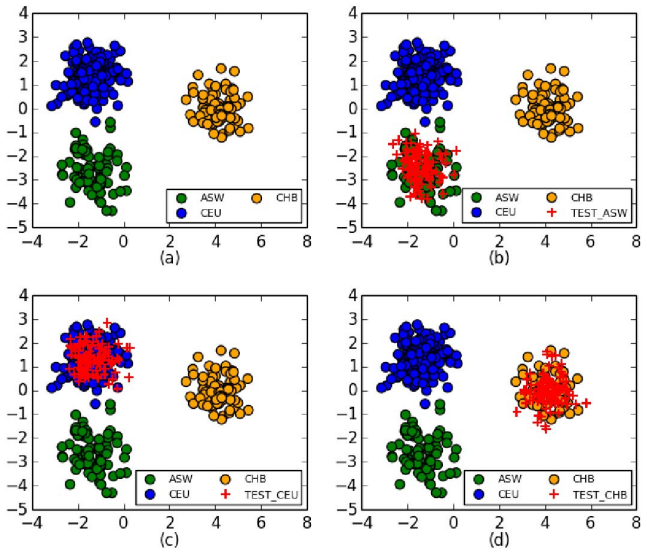

Fig. 11: Evaluation of ancestry compatibility on GenoGuard. (a) Ancestry inference with PCA on three populations: ASW (lower left cluster), CEU (upper left cluster), and CHB (right cluster). The red crosses are sequences decrypted from an ASW person with randomly guessed passwords, but with public haploid genotype dataset from different populations: (b) ASW; (c) CEU; (d) CHB. We can see that, regardless of the population which the original sequence belongs to, the ancestry of the decrypted sequence only depends on population-specific haploid genotype dataset used for the decoding.

exposed to the adversary as side information. For instance, the adversary could have access to a small number of phenotypical traits by observing a victim's photographs from online social networks.

Consider a genetic trait that has a set of possible phenotypes $\{T_1, T_2, \cdots, T_u\}$. For example, the trait "hair color" can have phenotype set $\{\text{Red}, \text{Blond}, \text{Brown}, \text{Black}\}$. Let P_{T_i} denote the prior probability of a phenotype T_i. Each phenotype T_i is also associated with a vector of prediction probabilities $A_{T_i}^{T_j}$: given a sequence with phenotype T_i, $A_{T_i}^{T_j}$ is the probability that the best classification algorithm will associate the sequence with phenotype T_j. Then, a brute-force attack proceeds as follows. For each password, the adversary uses it to decrypt the ciphertext, inputs the result sequence to the classifier, and excludes the password if the phenotype does not match; otherwise he retains the password. We assume that the adversary trusts the classifier and makes a binary decision on whether he should retain the password.

Suppose there are totally N unique passwords at the beginning, and they are in descending order regarding their probabilities: $P_1 \geq P_2 \geq \cdots \geq P_N$. The order of a password is usually called its rank. Note that $\sum_{i=1}^{N} P_i = 1$. It has been shown that the distribution of real-life passwords obeys Zipf's law [29], [30]. In other words, for a password dataset, the probability of password with rank i is

$$P_i = Wi^{-s}, \tag{6}$$

where W and s are constants depending on the dataset.

Hair Color (T)	Prior (P_T)	$A_T^{Red}, A_T^{Blond}, A_T^{Brown}, A_T^{Black}$
Red	8.8%	60.7%, 28.6%, 7.1%, 3.6%
Blond	42.6%	0.8%, 93.9%, 3.8%, 1.5%
Brown	39.3%	0.8%, 56.7%, 20%, 22.5%
Black	9.3%	0%, 55.2%, 3.4%, 41.4%

TABLE II: Summary of the results from the HIrisPlex system [31]. The second column, *prior*, is the fraction of samples that have the corresponding hair color. The third column is the vector of prediction accuracies (of the classification algorithm) for all four hair colors, given that a person has hair color T^*.

This is actually the password distribution p_k. Suppose the victim's phenotype is T^*, which is known to the adversary. We assume that decryption under a given incorrect password yields phenotype T_i with probability P_{T_i}, and that such assignment is independent across passwords. Whether an incorrect password is retained then depends on the probability that the decrypted sequence is classified by the classifier as phenotype T^*. This event may be modeled as independent Bernoulli trials across passwords, each with retaining probability P_{ret} computed as

$$P_{ret} = \sum_{i=1}^{u} P_{T_i} \cdot A_{T_i}^{T} . \tag{7}$$

Note that for the correct password, the adversary retains it with probability A_T^T . From Theorem 2, we observe that the advantage of adversary $\mathcal{B}$ *without* side information is approximately equal to w, the maximum weight in the password distribution (equivalent to the above P_1). Let $\mathcal{B}'$ represent the adversary *with* side information T^*. $\mathcal{B}'$ first prunes passwords based on the classifier, and then executes the algorithm of adversary $\mathcal{B}$ in the MR game (Figure 8) on the resulting smaller password pool consisting of retained passwords. Let p'_k represent this new password distribution, with maximum weight w'. We can represent the password pruning procedure as a randomized function $f(p_k) \to p'_k$. Therefore, $\mathcal{B}'$ adheres to the procedure: i) $\mathcal{B}'$ uses f to compute p'_k; ii) $\mathcal{B}'$ gives p'_k to $\mathcal{B}$. Let $\text{Adv}(\mathcal{B}')$ represent the advantage of adversary $\mathcal{B}'$. We have

$$\begin{aligned} \text{Adv}(\mathcal{B}') &= A_T^T \cdot E_{p_k \leftarrow f(p_k)}[\text{Adv}^{\text{mr}}_{\mathsf{HE},p_m,p_k}(\mathcal{B})] \\ &\approx A_T^T \cdot E_{p_k \leftarrow f(p_k)}[w'], \end{aligned} \tag{8}$$

where E is the expectation over the randomized password pruning process, and we approximate $\text{Adv}^{\text{mr}}_{\mathsf{HE},p_m,p_k}(\mathcal{B})$ with the maximum weight w' in the password distribution p'_k. In the following, we quantify $\text{Adv}(\mathcal{B}')$ empirically with real data.

For this purpose, we study a recent work about predicting hair color from DNA (the HIrisPlex system [31]). The study collects DNA samples and hair color information from 1551 European subjects and builds a model to predict the hair color. The results are shown in Table II.

We use the Zipf's model in [30], where $N = 486118$, $W = 0.037871$ and $s = 0.905773$. For different hair colors known by adversary $\mathcal{B}'$, we perform the Bernoulli trials with corresponding P_{ret} on the password pool, and estimate $\text{Adv}(\mathcal{B}')$ in Equation (8). We repeat the whole experiment 1000 times for each hair color, and the average results are shown in Figure 12.

With the "Red" hair information, the adversary's advantage increases from 0.0379 to 0.0642, which is the worst among the

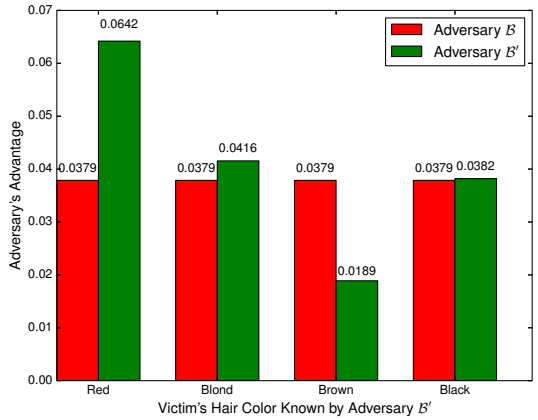

Fig. 12: Evaluation of adversary's advantage with the side information of hair color. Adversary $\mathcal{B}$ has no side information, and his advantage is approximately $w = 0.0379$, the maximum weight in the original password distribution p_k. The advantage of adversary $\mathcal{B}'$ depends on the prediction accuracy A^{T*}_{T*} and the retaining probability P_{ret} for the victim's hair color T^*.

four colors (for the victim). This is explained by the fact that "Red" hair has a very low prior probability that leads to a small P_{ret}, hence most wrong passwords are deleted. We observe that the empirical estimation of $E_{p_k \leftarrow f(p_k)}[w']$ is consistently larger for a smaller P_{ret}. On the contrary, because "blond" hair has a high prior probability, a larger number of passwords are retained, hence smaller $E_{p_k \leftarrow f(p_k)}[w']$ and smaller $\mathrm{Adv}(\mathcal{B}')$, compared to the case of "Red" hair. From Equation (8), the advantage is also positively correlated to the accuracy of the prediction algorithm. The low accuracies for "Brown" and "Black" hair (A^{Brown}_{Brown} and A^{Black}_{Black}) explain why the advantage of adversary $\mathcal{B}'$ barely increases, or even decreases[9], compared to adversary $\mathcal{B}$.

Even though side information is a common security concern in cryptography, we propose a general idea to avoid this problem for GenoGuard: incorporate the side information during the encoding phase. Nontrivial as this is, we will elaborate this idea in our future work, especially for traits with probabilistic genotype-phenotype associations.

VII. Discussion

In this section, we discuss the performance, application scenarios, some extensions, and limitations of the proposed scheme.

A. Performance

The time complexity of the encoding phase is $O(n)$ where n is the length of the sequence. Moreover, the storage overhead of the encrypted seeds is low as shown in Figure 9. Note that the ternary tree does not need to be stored. The encoding and decoding process are completely executed based on public knowledge; not on a pre-stored tree.

We implemented GenoGuard in Python. It includes mainly four steps: *encode*, *decode*, *PBE_encrypt*, *PBE_decrypt*. As before, we used the Phase 3 data in International HapMap Project, for the CEU population [22]. We set the storage overhead parameter $h = 4$. As for password-based encryption (decryption), we followed the standard PKCS #5 [16]. That is, using HMAC-SHA-1 as the underlying pseudorandom function, given a password P, we first applied a key derivation function

$$DK = KDF(P, S),$$

where DK is a 128-bit derived key and S is a 64-bit random salt. DK is used as the key for an AES block cipher that encrypts the seed in CBC mode. We ran the algorithm on a cluster of 22 nodes, each with 3.40GHz Intel Xeon CPU E31270 and 64-bit Linux Debian systems. In other words, the task of encrypting the whole genome was parallelized in 22 nodes that independently encrypt 22 chromosomes. We evaluated GenoGuard on 165 CEU samples, and the average performance is shown in Figure 13. Although encoding (decoding) is more costly than PBE, it is still acceptable considering the size of a full genome. Moreover, encoding different chromosomes was run in parallel, hence the running time depends only on the longest chromosome.

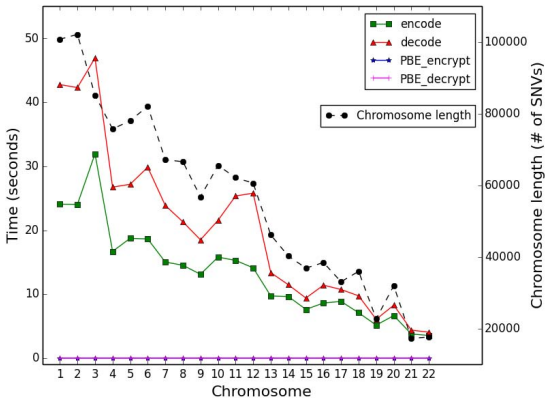

Fig. 13: Performance of GenoGuard on 22 chromosomes, averaging over 165 CEU samples. The dashed line shows the length of each chromosome, whereas the solid lines show the running time of the four procedures: *encode*, *decode*, *PBE_encrypt*, *PBE_decrypt*. The number of SNVs roughly decreases from chromosome 1 to chromosome 22. We can see that the running time of password-based encryption (decryption) is negligible compared to encoding (decoding), whose running time increases almost linearly with the length of a chromosome).

B. Application Scenarios

GenoGuard can be applied to various scenarios, including healthcare and recreational genomics, for the protection of genomic data. The general protocol in Figure 3 can work in a healthcare scenario without any major changes. In this scenario, a patient wants a medical unit (e.g., his doctor) to access his genome and perform medical tests. The medical unit can request for the encrypted seed on behalf of (and with consent from) the patient. Hence, there is a negotiation phase that provides the password to the medical unit. Such a phase

[9]When the prediction is unreliable, it's better for the adversary to ignore the side information.

can be completed automatically via the patient's smart card (or smart phone), or the patient can type his password himself. In this setup, the biobank can be a public centralized database that is semi-trusted. Such a centralized database would be convenient for the storage and retrieval of the genomes by several medical units.

For direct-to-customer (DTC) services, the protocol needs some adjustments. For instance, Counsyl[10] and 23andMe[11] provide their customers various DTC genetic tests. In such scenarios, the biobank is the private database of these service providers. Thus, such service providers have the obligation to protect customers' genomic data in case of a data breach. In order to perform various genetic tests, the service providers should be granted permission to decrypt the sequences on their side, which is a reasonable relaxation of the threat model because customers share their sequences with the service providers. Therefore, steps 8 and 9 in Figure 3 should be moved to the biobank. A user (customer) who requests a genetic test result logs into the biobank system, provides the password for password-based decryption and asks for a genetic test on his sequence. The plaintext sequence is deleted after the test.

C. Typos

Providing an incorrect password yields a fake but valid-looking sequence. This is a good security characteristic of honey encryption, but can be bad for usability if a legitimate patient or doctor does not realize she has made a mistake when typing the password. To solve this problem, we propose several solutions, as discussed below.

The first idea is to append to the plaintext some information that is unique and verifiable by the patient but meaningless for the adversary. We propose encoding such information (such as a 4-digit PIN chosen by the patient) as a string of bits similar to the seed. Such a PIN can be appended to the seed and encrypted together. In other words, the third encryption step in Figure 1 can be replaced with $C \leftarrow_{\$} \mathsf{encrypt}(K, S||\mathsf{PIN}, r)$. This option works well if the PIN is a uniformly random string; otherwise it will cause some security degradation because $S||\mathsf{PIN}$ is no longer uniform. Moreover, it requires the PIN to be kept secret and that the adversary cannot link it to a patient.

Another approach might be to leverage the distinction between recall memory and recognition memory [32]. The latter is shown to be more robust than the former. For instance, the system can provide a pool of N confirmation images, and the user can choose one before encryption. The confirmation images do not themselves have to be part of the ciphertext. The system can hash the genome sequence into $Z_N = \{0, 1, 2, \cdots, N-1\}$ to obtain a confirmation index, for security parameter N. The user might confirm correct decryption simply by indicating that a displayed image is familiar. A similar idea has been proposed in previous work where the authors apply it to anti-phishing techniques [33].

Another idea is based upon concealment of a biometric template among decoys. For instance, the user can provide his fingerprint template that is stored with some honey templates (e.g., synthetic fingerprint images [34], or other users' templates). These templates can also be indexed as what we propose for the confirmation images above. During retrieval, only the user can verify whether the decryption is correct or not using his own fingerprint.

[10]https://www.counsyl.com/.

[11]https://www.23andme.com/.

VIII. Related Work

Privacy concerns around genomic data have been extensively investigated by researchers in recent years. Homer *et al.* [2] show the possibility of inferring the participation of an individual in a genotype database with the help of public allele frequencies. Wang *et al.* [3] give similar results of inference power based on p-values released in genome-wide association studies. As a response to the above privacy breach in published genomic statistics, Fienberg *et al.* [4] propose to apply Laplacian noise to the released data to achieve differential privacy. Another approach to achieving differential privacy in genome-wide association study is proposed by Johnson and Shmatikov [5]. Yu *et al.* [6] present scalable privacy-preserving methods in genome-wide association studies based on Laplace mechanism and exponential mechanism. In spite of these works about differentially private genomic data, Fredrikson *et al.* [35] demonstrate an unsatisfactory tradeoff between privacy and utility in an end-to-end case study of personalized warfarin dosing. A similar unsatisfactory result in an association study is also mentioned by Erlich and Narayanan [36].

Jha *et al.* [37] design several privacy-preserving protocols for some fundamental genomic computations (edit distance and Smith-Waterman score) that use oblivious transfer and oblivious circuit evaluation. Kantarcioglu *et al.* [9] propose the use of homomorphic encryption to store encrypted genomic sequence records in a centralized repository, such that queries can be executed without decryption and thus without violating participants' privacy. Baldi *et al.* [10] propose a set of techniques based on private set operations to address genomic privacy in several important applications, namely, paternity tests, personalized medicine, and genetic compatibility tests. Ayday *et al.* [8] introduce a framework that integrates stream ciphers and order-preserving encryption to store and retrieve raw genomic data in a privacy-preserving manner. Researchers also propose to protect privacy in genomic computation by partitioning the computation through program specialization, according to the sensitivity levels of different parts of the genome data [7]. Naveed *et al.* [38] provide a comprehensive survey that discusses the latest considerations about the privacy issues and countermeasures in the genomic era.

However, no existing cryptographic solution in this domain addresses the challenge of long-term threats to encryption, such as quantum computing [39], or of the common short-term threat of brute-force cracking of PBE ciphertexts [40], [41], [42].

There have been a number of practices of applying deception and decoys in the literature of computer security. Honeypots [43] are fake computer systems intended to bait malicious actions that will be tracked and studied once these systems are probed or compromised. Honeypots are widely used in intrusion detection system [44], [45], [46]. Similarly, a honeynet [47] is proposed to assist the system administrator

in identifying malicious traffic on the enterprise network. The Kamouflage system [48] and honeywords [49] are designed to protect a password vault by constructing plausible decoy passwords. Juels and Ristenpart [13] formalize such a construction process with the concept of DTE, and propose honey encryption that provides security beyond the brute-force bound of password-based encryption.

IX. Conclusion and Future Work

The long-term sensitivity of genomic data gives rise to a need for especially strong protective mechanisms. Brute-force attacks on standard encryption schemes under strong passwords should not be considered infeasible in the long term, given the rapid evolution of computing technology and potential algorithmic advances. In the short term, the use of low-entropy keys, such as passwords, poses serious risks to password-based encryption of genomic data.

We propose GenoGuard, a cryptographic system that offers long-term protection for genomic data against even computationally unbounded adversaries. Decryption attempts against a GenoGuard ciphertext under an incorrect key yield a genome sequence that appears statistically plausible even to a sophisticated adversary. To achieve this guarantee, GenoGuard introduces a novel DTE scheme that efficiently encodes a genome sequence on a ternary tree with sensitivity to genetic recombination and mutation, thereby capturing the highly non-uniform probability distribution and special structure of genomic data. GenoGuard additionally provides security against adversaries with phenotypic side information (physical traits of victims). We provide a parallelized software implementation of GenoGuard and demonstrate its efficiency and scalability on a cluster of nodes. GenoGuard thus offers an appealing approach to the increasingly important challenge of protection of genomic data.

Acknowledgements

We thank Jean Louis Raisaro, Mathias Humbert, Huang Lin, Florian Tramèr and Kévin Huguenin for their feedback on the paper. We are grateful to Zoltán Kutalik for his suggestion on the genomic background and models. We also thank Sahel Shariati Samani for her work on part of the code in the system.

References

[1] https://cloud.google.com/genomics/, [Online; accessed 13-November-2014].

[2] N. Homer, S. Szelinger, M. Redman, D. Duggan, W. Tembe, J. Muehling, J. V. Pearson, D. A. Stephan, S. F. Nelson, and D. W. Craig, "Resolving individuals contributing trace amounts of DNA to highly complex mixtures using high-density SNP genotyping microarrays," *PLoS genetics*, August 29, 2008.

[3] R. Wang, Y. F. Li, X. Wang, H. Tang, and X. Zhou, "Learning your identity and disease from research papers: Information leaks in genome wide association study," in *Proceedings of the 16th ACM conference on Computer and communications security*, 2009, pp. 534–544.

[4] S. E. Fienberg, A. Slavkovic, and C. Uhler, "Privacy preserving GWAS data sharing," in *IEEE 11th International Conference on Data Mining Workshops (ICDMW)*, 2011, pp. 628–635.

[5] A. Johnson and V. Shmatikov, "Privacy-preserving data exploration in genome-wide association studies," in *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2013, pp. 1079–1087.

[6] F. Yu, S. E. Fienberg, A. B. Slavković, and C. Uhler, "Scalable privacy-preserving data sharing methodology for genome-wide association studies," *Journal of biomedical informatics*, 2014.

[7] R. Wang, X. Wang, Z. Li, H. Tang, M. K. Reiter, and Z. Dong, "Privacy-preserving genomic computation through program specialization," in *Proceedings of the 16th ACM conference on Computer and communications security*. ACM, 2009, pp. 338–347.

[8] E. Ayday, J. L. Raisaro, J.-P. Hubaux, and J. Rougemont, "Protecting and evaluating genomic privacy in medical tests and personalized medicine," in *Proceedings of the 12th ACM workshop on Workshop on privacy in the electronic society*, 2013, pp. 95–106.

[9] M. Kantarcioglu, W. Jiang, Y. Liu, and B. Malin, "A cryptographic approach to securely share and query genomic sequences," *IEEE Transactions on Information Technology in Biomedicine*, vol. 12, no. 5, pp. 606–617, 2008.

[10] P. Baldi, R. Baronio, E. De Cristofaro, P. Gasti, and G. Tsudik, "Countering GATTACA: Efficient and secure testing of fully-sequenced human genomes," in *Proceedings of the 18th ACM conference on Computer and communications security*, 2011, pp. 691–702.

[11] M. Humbert, E. Ayday, J.-P. Hubaux, and A. Telenti, "Addressing the concerns of the Lacks family: Quantification of kin genomic privacy," in *Proceedings of the ACM SIGSAC conference on Computer & communications security*, 2013, pp. 1141–1152.

[12] D. Florencio and C. Herley, "A large-scale study of web password habits," in *Proceedings of the 16th International Conference on World Wide Web*, ser. WWW '07. New York, NY, USA: ACM, 2007, pp. 657–666. [Online]. Available: http://doi.acm.org/10.1145/1242572.1242661

[13] A. Juels and T. Ristenpart, "Honey encryption: Security beyond the brute-force bound," in *Advances in Cryptology–EUROCRYPT*, 2014, pp. 293–310.

[14] J. M. VanLiere and N. A. Rosenberg, "Mathematical properties of the r^2 measure of linkage disequilibrium," *Theoretical population biology*, vol. 74, no. 1, pp. 130–137, 2008.

[15] M. Benantar, *Access control systems: Security, identity management and trust models*. Springer, 2006.

[16] B. Kaliski, *PKCS# 5: Password-based cryptography specification version 2.0*, RSA Laboratories, September, 2000.

[17] M. S. McPeek and A. Strahs, "Assessment of linkage disequilibrium by the decay of haplotype sharing, with application to fine-scale genetic mapping," *The American Journal of Human Genetics*, vol. 65, pp. 858–875, 1999.

[18] S. L. Salzberg, A. L. Delcher, S. Kasif, and O. White, "Microbial gene identification using interpolated Markov models," *Nucleic acids research*, vol. 26, pp. 544–548, 1998.

[19] N. Li and M. Stephens, "Modeling linkage disequilibrium and identifying recombination hotspots using single-nucleotide polymorphism data," *Genetics*, vol. 165, pp. 2213–2233, 2003.

[20] L. Rabiner, "A tutorial on hidden Markov models and selected applications in speech recognition," *Proceedings of the IEEE*, vol. 77, pp. 257–286, 1989.

[21] J. Marchini, B. Howie, S. Myers, G. McVean, and P. Donnelly, "A new multipoint method for genome-wide association studies by imputation of genotypes," *Nature genetics*, vol. 39, pp. 906–913, 2007.

[22] http://hapmap.ncbi.nlm.nih.gov/downloads/index.html.en, [Online; accessed 11-November-2014].

[23] R. B. DiAgostino and J. M. Massaro, "Goodness-of-fit tests," *Handbook of the Logistic Distribution*, p. 327, 2013.

[24] Z. Huang, E. Ayday, J. Fellay, J.-P. Hubaux, and A. Juels, "GenoGuard: Protecting genomic data against brute-force attacks," https://infoscience.epfl.ch/record/205068, EPFL, Tech. Rep., 2015.

[25] J. Bonneau, "The science of guessing: Analyzing an anonymized corpus of 70 million passwords," in *IEEE Symposium on Security and Privacy*, 2012, pp. 538–552.

[26] P. Claes, D. K. Liberton, K. Daniels, K. M. Rosana, E. E. Quillen, L. N. Pearson, B. McEvoy, M. Bauchet, A. A. Zaidi, W. Yao *et al.*, "Modeling 3D facial shape from DNA," *PLoS genetics*, March 20, 2014.

[27] A. L. Price, N. J. Patterson, R. M. Plenge, M. E. Weinblatt, N. A. Shadick, and D. Reich, "Principal components analysis corrects for

stratification in genome-wide association studies," *Nature genetics*, vol. 38, no. 8, pp. 904–909, 2006.

[28] J. N. Sampson, K. K. Kidd, J. R. Kidd, and H. Zhao, "Selecting SNPs to identify ancestry," *Annals of human genetics*, vol. 75, no. 4, pp. 539–553, 2011.

[29] D. Malone and K. Maher, "Investigating the distribution of password choices," in *Proceedings of the 21st international conference on World Wide Web*. ACM, 2012, pp. 301–310.

[30] D. Wang, G. Jian, H. Cheng, Q. Gu, C. Zhu, and P. Wang, "Zipfs law in passwords," Cryptology ePrint Archive, Report 2014/631, Tech. Rep., 2014.

[31] S. Walsh, F. Liu, A. Wollstein, L. Kovatsi, A. Ralf, A. Kosiniak-Kamysz, W. Branicki, and M. Kayser, "The HIrisPlex system for simultaneous prediction of hair and eye colour from DNA," *Forensic Science International: Genetics*, vol. 7, no. 1, pp. 98–115, 2013.

[32] F. Haist, A. P. Shimamura, and L. R. Squire, "On the relationship between recall and recognition memory," *Journal of Experimental Psychology: Learning, Memory, and Cognition*, vol. 18, no. 4, p. 691, 1992.

[33] R. Dhamija and J. D. Tygar, "The battle against phishing: Dynamic security skins," in *Proceedings of Symposium on Usable Privacy and Security*, 2005, pp. 77–88.

[34] R. Cappelli, A. Erol, D. Maio, and D. Maltoni, "Synthetic fingerprint-image generation," in *Proceedings of the 15th International Conference on Pattern Recognition*, vol. 3, 2000, pp. 471–474.

[35] M. Fredrikson, E. Lantz, S. Jha, S. Lin, D. Page, and T. Ristenpart, "Privacy in pharmacogenetics: An end-to-end case study of personalized warfarin dosing," in *USENIX Security*, 2014.

[36] Y. Erlich and A. Narayanan, "Routes for breaching and protecting genetic privacy," *Nature Reviews Genetics*, vol. 15, no. 6, pp. 409–421, 2014.

[37] S. Jha, L. Kruger, and V. Shmatikov, "Towards practical privacy for genomic computation," in *IEEE Symposium on Security and Privacy*, 2008, pp. 216–230.

[38] M. Naveed, E. Ayday, E. W. Clayton, J. Fellay, C. A. Gunter, J.-P. Hubaux, B. A. Malin, and X. Wang, "Privacy and security in the genomic era," *arXiv preprint arXiv:1405.1891*, 2014.

[39] M. A. Nielsen and I. L. Chuang, *Quantum computation and quantum information*. Cambridge University Press, 2010.

[40] L. S. Clair, L. Johansen, W. Enck, M. Pirretti, P. Traynor, P. McDaniel, and T. Jaeger, "Password exhaustion: Predicting the end of password usefulness," in *Information Systems Security*, 2006, pp. 37–55.

[41] M. L. Mazurek, S. Komanduri, T. Vidas, L. Bauer, N. Christin, L. F. Cranor, P. G. Kelley, R. Shay, and B. Ur, "Measuring password guessability for an entire university," in *Proceedings of the ACM SIGSAC conference on Computer & communications security*, 2013, pp. 173–186.

[42] A. Das, J. Bonneau, M. Caesar, N. Borisov, and X. Wang, "The tangled web of password reuse," in *Proceedings of Network and Distributed System Security Symposium*, 2014.

[43] L. Spitzner, *Honeypots: Tracking hackers*. Addison-Wesley Reading, 2003.

[44] C. Kreibich and J. Crowcroft, "Honeycomb: Creating intrusion detection signatures using honeypots," *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 1, pp. 51–56, 2004.

[45] D. Dagon, X. Qin, G. Gu, W. Lee, J. Grizzard, J. Levine, and H. Owen, "Honeystat: Local worm detection using honeypots," in *Recent Advances in Intrusion Detection*, 2004, pp. 39–58.

[46] K. G. Anagnostakis, S. Sidiroglou, P. Akritidis, K. Xinidis, E. P. Markatos, and A. D. Keromytis, "Detecting targeted attacks using shadow honeypots." in *Usenix Security*, 2005.

[47] J. Levine, R. LaBella, H. Owen, D. Contis, and B. Culver, "The use of honeynets to detect exploited systems across large enterprise networks," in *IEEE Systems, Man and Cybernetics Society Information Assurance Workshop*, 2003, pp. 92–99.

[48] H. Bojinov, E. Bursztein, X. Boyen, and D. Boneh, "Kamouflage: Loss-resistant password management," in *ESORICS*, 2010.

[49] A. Juels and R. L. Rivest, "Honeywords: Making password-cracking detectable," in *Proceedings of the ACM SIGSAC conference on Computer & communications security*, 2013, pp. 145–160.

APPENDIX
THE RECOMBINATION MODEL

Here we describe how to compute the probability of a haploid genotype h_{k+1}, given k sampled haploid genotypes $\{h_1, \cdots, h_k\}$ with the forward-backward algorithm for hidden Markov models. The model is borrowed from genetic research, but we try to avoid using too much genetic terminology in this work; for details about some variables and constants we will use here, we kindly ask the readers to refer to the original paper [19].

Initially, at state 1, we have $P(X_1 = x) = \frac{1}{k}(x \in \{1, \cdots, k\})$. The transition probability from state j to $j+1$ is characterized by

$$\begin{aligned} &P(X_{j+1} = x' | X_j = x) \\ &= \begin{cases} exp(-\frac{\rho_j}{k}) + \frac{1-exp(-\frac{\rho_j}{k})}{k} & \text{if } x' = x; \\ \frac{1-exp(-\frac{\rho_j}{k})}{k} & \text{otherwise,} \end{cases} \end{aligned} \tag{9}$$

where ρ_j is the genetic distance between locus j and $j+1$. It is computed based on the recombination rate between these two loci. Intuitively, a smaller genetic distance will make the two states more likely to take the same value, meaning that they are more likely to come from the same haploid genotype.

At state j, an allele (0 or 1) will be emitted. To mimic the effects of mutation, the emitting probability is characterized by

$$P(h_{k+1,j} = a | X_j = x) = \begin{cases} 1 - \lambda & \text{if } h_{x,j} = a; \\ \lambda & \text{otherwise,} \end{cases} \tag{10}$$

where a is 0 or 1, and λ is the mutation rate.

Let the forward variable $\alpha_j(x) = P(h_{k+1,\leq j}, X_j = x)$. Then $\alpha_1(x) = P(h_{k+1,1}|X_1 = x)P(X_1 = x)$. And $\alpha_2(x), \cdots, \alpha_n(x)$ can be computed recursively using

$$\begin{aligned} \alpha_{j+1}(x) =& P(h_{k+1,j+1}|X_{j+1} = x) \\ & \sum_{x'=1}^{k} \alpha_j(x') P(X_{j+1} = x | X_j = x'), \end{aligned} \tag{11}$$

The probability of a complete haploid genotype is then computed using

$$P(h_{k+1}|h_1, \cdots, h_k) = \sum_{x=1}^{k} \alpha_n(x). \tag{12}$$

The conditional probability for allele $j+1$, given all preceding alleles, is computed using

$$P(h_{k+1,j+1}|h_{k+1,\leq j}, h_1, \cdots, h_k) = \frac{\sum_{x=1}^{k} \alpha_{j+1}(x)}{\sum_{x=1}^{k} \alpha_j(x)}. \tag{13}$$

For a genome sequence that couples two haploid genotypes, all the above quantities can be computed similarly by an extension of this hidden Markov model [21].