Icefish: Practical zk-SNARKs for Verifiable Genomics

Alexander Frolov
University of Maryland
sfrolov@umd.edu

Maurice Shih
University of Maryland
maurices@umd.edu

Rob Patro
University Of Maryland
rob@cs.umd.edu

Ian Miers
University Of Maryland
imiers@umd.edu

Abstract

Individual genomic data is a uniquely sensitive type of user data. While many papers have considered using Multi-Party Computation (MPC) or Fully Homomorphic Encryption (FHE) to allow collaborators to study combined genomic datasets they cannot share, few have considered verifying the results of genomic computations, either in research studies or in the emerging area of personalized genetic therapies.

In this paper, we initiate the first systematic study of zero-knowledge proofs for verifiable genomics, providing both building blocks for verifying common operations in computational genomics, such as sequence alignment, and exploring two end-to-end applications:

Verifiable Genome-Wide Association Studies: A Genome-Wide Association Study (GWAS) study operates over a repository of genomic data, identifying statistical correlations between genetic variations and observed traits or medical conditions. Our system enables third parties to verify that research was honestly computed over an authenticated, untampered database, ensuring both the integrity of the underlying data set and the correctness of the resulting science. We achieve practical performance (<40 minutes proving time) for studies of sizes equal to those in the existing genomics literature.

Verifiable CRISPR eligibility: We propose using zk-SNARKs in the context of gene engineering (e.g. CRISPR). To our knowledge, this is a new use case for zk-SNARKs. We implement and optimize models for detecting "on-target" and "off-target" sites for a CRISPR probe in zk-SNARKs, so users can, for example, demonstrate eligibility for a therapy or trial without having to reveal their own DNA sequence.

In support of these applications, we develop new building blocks, like zero-knowledge proofs of sequence alignment that are 30x faster than the prior state of the art, and storage-efficient indexes for Merkle trees for large scale genomic data that asymptotically reduce storage costs.

1 Introduction

Privacy of personal genomic data is a growing concern, driven by genetic testing and sequencing becoming cheaper and more widely used. The bankruptcy of 23andMe [1] highlights long standing concerns over the existence of large databases of individuals' genomic data potentially being sold to the highest bidder. Unlike a leaked password, which can be rotated, or other types of personal information whose relevance fades with time, leaked genomic information is permanent: an individual's DNA generally cannot be changed after it is leaked. Furthermore, an individual's DNA data also leaks information about that individual's close biological relatives.

Privacy concerns have historically motivated the use of confidential computing for genomics using techniques like Secure Multi-party Computation (MPC) [2] or Fully Homomorphic Encryption (FHE) [3]. The goal of using these approaches is typically collaborative computation: computing the results of some genomic study over multiple parties' data without requiring the parties to share the underlying sensitive material or handle the resulting regulatory issues. Consider a Genome-Wide Association Study (GWAS), a type of study conducted in genomics that takes genetic data and health outcomes from thousands of individuals to identify statistical correlations between variations in genes and patient outcomes like a specific disease or medical condition. In a collaborative setting, two hospitals could use these techniques to compute a GWAS across their combined sets of patient records, lending greater statistical sensitivity to the study, without either party learning the other's patient records.

While flexible and useful, these FHE and MPC protocols for collaborative computation are often costly. Beyond the high computational costs imposed by their cryptography, they present significant operational hurdles: they (in many configurations) require finding partners who are trusted not to collude, have the technical expertise to manage complex software services, and can afford the infrastructure requirements.

This paper focuses on a narrower problem: verifying computational claims made by a single party who holds private

genomic data. Verification serves two primary goals in this paper: it allows for public auditability and for computational claims to be proven without data ever leaving users' devices in any form. Public verifiability allows, for example, third parties to verify research results derived from restricted-access datasets that exist today without the raw data being released to verifiers. Data minimization, we propose, is useful for future applications in personalized genomics. One can consider a regime where an individual's personal genome never leaves their own device and they instead only reveal verifiable claims about it. In this verifiable genomics setting, because one party holds all the data, we can use comparatively lighter-weight cryptographic primitives, zero-knowledge proofs. Indeed, looking ahead, many of our results can be computed on consumer grade devices such as a laptop.

1.1 Our contributions

This paper initiates the first systematic study of zero-knowledge proofs for verifiable genomics, providing both building blocks for verifying common operations in computational genomics, such as sequence alignment, and exploring two end-to-end applications:

Verifiable Genome Wide Association Studies and genomic data depositories. Sensitive datasets containing thousands of individual genomes, or their ascertained genomic variants, are typically held by third party institutions such as biobanks or the National Institues of Health (NIH). Access to this information is tightly controlled, with researchers requesting access for specific studies that must be approved. Researchers who are granted access can publish the results of computational studies on that data, but must not reveal the underlying personal genetic records. Using our techniques for verifiable genomics, we can verify both (1) the integrity of these databases, to prove they have not been altered by the custodian and (2) statistics about the results of a Genome Wide Association Study (GWAS) performed by an academic or industrial group. This would allow a third party to, for example, verify a high profile claim linking a specific gene to a medical condition, while maintaining the privacy of every person in the dataset.

Towards verifiable CRISPR eligibility. Genetic therapies such as CRISPR [4] edit the DNA of some cells in a patient's body. For both clinical trials and commercial applications, it is necessary to check that a patients' genes are "compatible" with a given therapy, both that the therapy will have the proper *on target* effect, and that it won't have unintended negative *off target* effects. Likewise, in clinical trials, it is often necessary to verify that a patient's genome is compatible with the constraints proposed by a clinical trial. The goal in this setting is for a user to perform these checks locally on their genome, rather than sending it to the provider, thus relieving the provider of the burden of securing and analyzing the data themselves, and limiting the opportunity for this sensitive information to be compromised. We do not achieve this result entirely, instead we introduce the problem and setting and demonstrate feasibility. In particular, while our proof for on-target effects can scale to a human genome, we only demonstrate feasibility of our proofs for the absence of off-target effects for small eukaryotic genomes like yeast.

In more detail, in this paper:

- We show how to efficiently generate zk-SNARKs showing knowledge of a sequence alignment of a given cost. Our implementation can produce a proof verifying an alignment between two 2000 base pair sequences in 354ms, and is 30× faster than the previous state of the art [5]. We observe that our sequence alignment circuits have specific memory access patterns, allowing us to specialize techniques for zk-RAM [6], leading to performance improvements.

- We explore storage-efficient indexes for Merkle trees for large scale genomic data. When used to store a Merkle tree over a human genome-sized vector of data, at an example parameter setting, our data structure stores 32× less additional data than a naive Merkle tree, has the same proof verification time, and has modestly worse performance for generating Merkle inclusion proofs. We also prove that this data structure is in some sense optimal.

- We provide an end-to-end implementation of a Genome-Wide Association study implemented for an authenticated database using Lasso regression[1] [8] and show it can be used to verify the results of existing Genome Wide Association Studies from the genomics literature.

- For verifiable CRISPR eligibility, we implement and optimize two existing algorithms from the literature: Rule Set 1 [9] for on-target sites and MIT Score [10] for off-target sites, and show full verification is feasible for smaller eukaryotic genomes like yeast.

- We implement all of the above using the Noir [11] or Spartan2 [12] libraries, benchmark them on consumer hardware, and compare against relevant prior work.

2 Threat model

We assume there exist authenticated DNA sequences from a sequencing machine or laboratory[2] and we assume a PKI and trust in these sequencing parties.

A natural source of authentication in some settings is the provider of a dataset. For example, the National Institutes

[1] Note that Lasso regression is unrelated to the Lasso lookup argument [7].

[2] Sequencing is the process of determining a DNA sequence given a sample of biological material.

of Health (NIH) maintains many genomic data sets, with access restricted to approved researchers and studies. With our techniques, the NIH could publish a digest of their data sets, making any subsequent tampering (or accidental errors) detectable and allowing researchers, who do have access to the data set, to produce proofs of correctness for their computation that are verifiable by the general public or other researchers who only have the digest. This provides a specific mechanism to guard not only against potential fraud, but also against the likely more common scenario where unintentional mistakes or errors in pre-processing render the result of a complex analysis incorrect. Critically, this verification can both act as an extra safeguard against accidental errors by those performing the analyses, and can be done by parties who lack access to the protected data.

For personal genomics, our authentication assumption has subtler implications. We believe that having sequencing machines/laboratories output cryptographic signatures is a reasonable assumption: labs are already trusted for correctness in practice, and most prior literature on MPC or FHE for genomics implicitly assumes correct sequencing. We assume the existence of protocols to bind a patient's identity both to the physical sample and its resulting sequence, and subsequently to the individual undergoing the clinical procedure.[3] But what about confidentiality? After all, the lab sequencing a patient's DNA sees their genomic data. However, labs are already trusted with such data today, it is often protected by regulations, and users are free to select trustworthy labs. We focus not on the laboratories, but on downstream uses. Verifiable genomics ensures downstream parties need not learn such sequences, nor do labs need to retain the sequence data. As such, our threat model should be viewed as an exercise in data minimization: no one but the user retains their genomic data, and user's trust relationship with labs ends after the lab sequences and deletes the data. We note that there is interesting recent work on securing low-cost sequencing machines [13] which might allow users to sequence their genomes themselves.

Lastly, this paper focuses on verifying the operations performed on genomic data. This leaves open two major issues: whether the computation itself is privacy preserving, and whether it is correct. In our GWAS use case, we expect the coefficients of a model fitted on private data or other statistics about the model to be made public. These statistics may leak information about the underlying data. We consider this to be out of scope for this paper. Similarly, p-hacking/other statistical manipulations and issues in data collection/preparation are well known problems in research publications, and we do not focus on them here: our work is aimed at verifying what computations were performed, tools to detect computations meant to manipulate or skew results can be built on top of these primitives. Data sets for GWASs are often collected once and analyzed many times by other parties. In these cases, it is reasonable for there to be a different level of trust between preparing a data set and analyzing it.

[3] In the US, providers may already scan identity documents. While sufficient for medical billing, these procedures may need to be hardened.

3 Preliminaries

3.1 Cryptographic preliminaries

We first introduce the relevant cryptographic primitives.

Definition 3.0.1. *A **SNARK** (succinct non-interactive argument of knowledge) [14] is a triple of* PPT *algorithms* $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ *where:*

- $\mathsf{Setup}(1^\lambda, \mathcal{R}) \to \mathsf{pp}$ *takes the description of an NP relation* $\mathcal{R}$*, a security parameter, and outputs public parameters* pp *for the SNARK.*
- $\mathsf{Prove}(\mathsf{pp}, x, w) \to \pi$*, takes public parameters* pp*, public inputs* x*, private inputs* w *and returns a proof* π *that* $\mathcal{R}(x,w) = 1$*.*
- $\mathsf{Verify}(\mathsf{pp}, x, \pi) \to \{0,1\}$ *verifies that* π *is a valid proof for* $\mathcal{R}$ *and public inputs* x*.*

A SNARK is expected to be knowledge sound (that only true statements can be proved, and that a prover generating a valid proof "knows" a witness w) and complete (that a prover can prove all cases where $R(x,w) = 1$). A SNARK is called "zero-knowledge" (and a zk-SNARK) if it reveals no information besides what is implied by the statement $R(x,w) = 1$. We sometimes call SNARKs "zero-knowledge proofs" for diversity in writing. We call the computation executed inside of a zk-SNARK a "circuit".

A major difference between writing efficient circuits for zero-knowledge proofs and traditional programming is that zero-knowledge proofs can make use of *untrusted advice* (in the witness variable w) and verify it. Executing computations inside a zero-knowledge proof circuit is concretely much more expensive than executing them in plain computation, so having the circuit $\mathcal{R}$ verify precomputed advice from the prover often results in a faster overall system.

Definition 3.0.2. *A **Merkle tree** [15] is a labeled binary tree over a vector of values, instantiated with collision resistant hash functions* $\mathcal{H}_\ell : \{0,1\}^* \to \{0,1\}^\lambda$ *and* $\mathcal{H}_n : \{0,1\}^{2\lambda} \to \{0,1\}^\lambda$ *(the "leaf" and "node" hash functions, respectively). For a Merkle tree over a vector* d *of* i *elements, the* i^{th} *leaf of the Merkle Tree has label* $\mathcal{H}_\ell(d_i)$ *and each interior node's label is* $H_n(h_l || h_r)$*, where* h_1, h_r *are the hashes of the node's left and right neighbors respectively.*

The label of the highest node in a Merkle tree is its "Merkle root". A prover can generate a "Merkle inclusion proof" for a leaf d_i by providing the list of hashes that are the "siblings" of the nodes along the path from d_i to the root of the tree. A

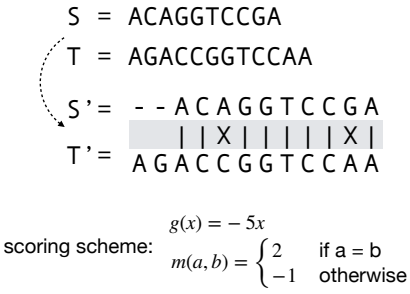

Figure 1: An example of a global pairwise sequence alignment between two strings S and T under a specific scoring scheme.

Merkle inclusion proof (for a dense balanced Merkle Tree with n elements) consists of $O(\log n)$ hashes, and can be verified in $O(\log n)$ time. The height of a Merkle tree is the length of the longest path from the root to a leaf node in the tree. The depth of a node in a Merkle tree is the length of the path from the Merkle tree's root to the node. A Merkle Tree is balanced if all leaves have depths which differ by at most 1, and complete, if each "layer" of the tree has the maximal number of nodes at each depth. We sketch a complete, balanced Merkle Tree with a highlighted inclusion proof in Figure 5 in Appendix A. We write the algorithms to build a Merkle Tree, generate an inclusion proof, and verify an inclusion proof as $\mathsf{MerkleTree}.\{\mathsf{Build}, \mathsf{IncProof}, \mathsf{VerifyProof}\}$.

A **lookup argument** [16] is a cryptographic argument that lets a prover demonstrate that the values in a committed-to vector are contained in a larger vector. Lookup arguments are often used in zk-SNARKs to speed up operations which are inefficient to express in constraints directly, like range checks, comparison operations, or boolean operations.

We also use standard definitions of hash functions and signatures, which are present in a reference like [15], and use standard definitions of asymptotic notation, which are present in a reference like [17].

3.2 Genomics preliminaries

We define the relevant genomics concepts for this paper.

Genomics is the study of the DNA of living organisms. DNA sequences consist of "bases" or "base pairs" which are one of the four letters A,C,G or T, corresponding to 4 possible types of molecules in DNA. In certain contexts, DNA sequences are studied in terms of *nucleotides*. A group of 3 bases is called a nucleotide, and these have a specific meaning when turning a DNA sequence into proteins.

Sequence alignment is a common problem in computational genomics. Concretely, given two strings S and T, which are finite sequences over some shared alphabet Σ (most frequently $\{A,C,G,T\}$), as well as a scoring system $(g(x), m(a,b))$, the sequence alignment problem asks to produce a new pair of strings S' and T', potentially containing a novel character –, called the gap character, that are of the same length and in one-to-one correspondence. In the scoring system, $g(x)$ gives the score for inserting a gap of length x (i.e. a sequence of x consecutive instances of –), and $m(a,b)$ gives the score associated with placing character a into correspondence with character b (generally positive if $a = b$ and negative otherwise). The alignment problem seeks to find a *highest scoring pair over all possible* S', T', under the given scoring function. See Fig 1 for an example. While such an optimal S', T' pair takes $O(S \cdot T)$ time to find, via dynamic programming, in the worst case [18], the alignment itself can be concisely encoded in $O(S+T)$ space.

Many variants of sequence alignment exist. Local and semi-global sequence alignment are variants of sequence alignment in which the entirety of one sequence is aligned against a subsequence of the other (for instance when a short gene is being aligned against a much larger genome of an organism). We described sequence alignment with a simple "gap cost function" where each insertion and deletion costs the same amount, but more sophisticated cost functions exist. For instance, the affine gap cost function is a function where a gap of length n has a cost of $an+b$, where $b > a$. This expresses the restriction that many short gaps between two aligned sequences are less biologically likely than fewer long gaps. Lastly, multiple sequence alignment is a variant in which multiple sequences are aligned simultaneously, and the overall quality of alignment is scored.

A **succinct data structure** [19] is a type of data structure over a dataset of size Z, which enables efficient queries over the dataset while requiring $o(Z)$ additional storage. Succinct data structures are desirable in genomics, where datasets can be large, and databases must have small indices to be useful.

A **Genome-Wide Association Study (GWAS)** is a type of genomic study in which researchers seek to find new correlations between variations in an organism's genetic sequence (most commonly focusing on Single Nucleotide Polymorphisms (SNPs)) and observed traits (like having a disease).

GWASs often use algorithms like linear regression, logistic regression, or more complicated models, such as group regularized regression [20] or even deep learning [21, 22], to find correlations.

Lasso regression is a variant of regularized linear regression [8] which is commonly used in genomics. Given n data points in m dimensions $x : \mathbb{R}^{m \times n}, y : \mathbb{R}^n$, Lasso regression aims to find model coefficients $\beta : \mathbb{R}^{m+1}$ minimizing the quantity:

$$\sum_{i=0}^{n} (y_i - \beta_0 - x_i^T \beta_{1..n})^2$$

subject to the regularization constraint $\sum_{i=0}^{m+1} |\beta_i| \leq t$.[4] Lasso regression is unrelated to the Lasso lookup argument [7]. Lasso regression is well-suited for genomic studies because the regularization constraint forces the model to choose

[4] Note that the Lasso optimization objective is sometimes equivalently defined with the l_1 norm of the coefficient vector β being added to the objective function.

a small number of coefficients of high magnitude, meaning that a study is more likely to identify a small number of variables that explain a given trait.

CRISPR-Cas9 [4, 23, 24] is a technique for genome editing, derived from the adaptive immune mechanisms in the Streptococcus pyogenes (SpCas9) bacteria.

Abstractly, CRISPR involves **probes**, which are 20 base pair segments of DNA that are targeted for a given edit. Within this sequence, the target DNA must immediately follow a Protospacer Adjacent Motif (PAM), with the structure of "NGG" where N is a wild card. The more similar a 20 base pair sequence in the targeted organism is to the probe, the higher chance the edit will take place at that location.

Two main types of algorithms used for CRISPR are **on-target** and **off-target** scoring algorithms, which model the likelihood that a CRISPR edit will occur at a given location. Typically, the output of these algorithms is a decimal value, ranging from 0 to 1 where higher values indicate a higher chance of the edit occurring. On-target scoring algorithms rate the probability that a change will occur in the targeted gene, while off-target scoring algorithms rate the probability that an edit in a different part of the genome will occur. Examples of on-target scoring algorithms/models include Rule Set 1 [9], Rule Set 2 (Azimuth) [25], and Rule Set 3 [26, 27]. Examples of off-target scoring algorithms include the MIT score [10] and CFD (Cutting Frequency Determination) score [25]. For a specific CRISPR probe/edit to be useful for a patient/organism, it must have a high on-target score and a low off-target score.

4 Sequence alignment in zk-SNARKs

We first discuss generating proofs of knowledge of low-cost sequence alignments. Sequence alignment is a fundamental concept in bioinformatics. For example, a patient having a certain gene is generally equivalent to that gene's sequence and the patient's DNA having a low-cost sequence alignment. As an example application for zero-knowledge proofs of sequence alignment, consider a patient registering for a clinical trial which requires patients to have a certain gene. Patients could generate zk-SNARKs showing that a segment of their genome has a low-cost alignment with the relevant gene's sequence and privately show their eligibility for the study. All proofs in this section show knowledge of a sequence alignment below a given cost. They do not show the optimality of sequence alignments. However, in many contexts, like proving possession of a gene, there is a relevant cost threshold for alignments, and these approaches can be used.

4.1 Basic algorithm for sequence alignment

Naïvely, proving the presence of a low-cost sequence alignment between two strings in a zk-SNARK would require a circuit of quadratic size. Computing an optimal sequence alignment requires quadratic time under common complexity-theoretic assumptions [18], so directly computing an exact optimal sequence alignment in a SNARK would require quadratic circuit size.

However, zk-SNARK circuits have a different computational model: they can take precomputed advice as part of their untrusted inputs and use pseudorandom values. This section will describe how to prove knowledge of a sequence alignment with cost below some threshold faster than it can be directly computed in a zk-SNARK by using precomputed advice.

Genomics software commonly generates "CIGAR strings" [28], which are strings that describe an alignment of two sequences. CIGAR strings encode an alignment by describing the sequence of gaps that are inserted into two aligned sequences. For instance, the CIGAR string 3M1I3M1D5M describes an alignment of two strings of 12 characters with 1 gap inserted into the second sequence after 3 matching characters and 1 inserted into the first sequence after another 3 matching characters. We describe the basics of sequence alignment in more detail in Section 3.2.

We use the basic idea of the CIGAR format in our SNARK circuits. We slightly modify the existing format to be more SNARK-friendly. Numbered sequences like 3M are expanded into series of individual characters like MMM to avoid encoding parsing logic into the circuit. We also only implement the characters M, I, D for matches, insertions and deletions respectively, which are the only relevant characters for the basic variant of sequence alignment. We discuss other characters in the CIGAR standard in the next section and in Appendix C.

Verifying that a modified CIGAR string represents a valid alignment between two sequences and has a given cost can be done in linear time by iterating through the CIGAR string and the two aligned sequences. For initial presentation, we use a fixed cost $\mathsf{cost}_{\mathsf{gap}}$ for both insertions and deletions. The simple algorithm below can verify the alignment between sequences $\mathsf{seq}_1, \mathsf{seq}_2$ with a modified CIGAR string cigar:

```
1 : VerifyAlignment(seq_1, seq_2, cigar, threshold) :
2 : i_1, i_2, cost ← 0, 0, 0
3 : for char in cigar :
4 :   if char == 'M' :
5 :     assert seq_1[i_1] == seq_2[i_2]
6 :     i_1 = i_1 + 1
7 :     i_2 = i_2 + 1
8 :   if char == 'D' :
9 :     i_1 = i_1 + 1
10 :    cost = cost + cost_gap
11 :  if char == 'I' :
12 :    i_2 = i_2 + 1
13 :    cost = cost + cost_gap
14 : assert i_1 == seq_1.length
15 : assert i_2 == seq_2.length
16 : assert (cost < threshold)
```

At a high level, this algorithm iterates through the CIGAR string, checking that the characters of $\mathsf{seq}_1, \mathsf{seq}_2$ match when the CIGAR string requires them to, and updating indices into $\mathsf{seq}_1, \mathsf{seq}_2$ and the alignment cost when the CIGAR string specifies a gap in the alignment. VerifyAlignment asserts that the alignment's cost is below some threshold, though it can also return cost depending on privacy requirements.

To implement this algorithm efficiently in a SNARK circuit, the circuit must be able to access the entries of each sequence at data-dependent indices $\mathsf{i}_1, \mathsf{i}_2$ efficiently. This is nontrivial, seeing as the prior work [29] has a similar algorithm, but with linear per-access costs. We can achieve constant per-access costs using a standard "zk-RAM" construction, like [6]. However, observing that each index of $\mathsf{seq}_1, \mathsf{seq}_2$ will be read once in a successful execution of VerifyAlignment, we can further optimize a circuit for VerifyAlignment by customizing standard memory-checking techniques, which we explain in Appendix B.

In an end-to-end workflow to generate a zk-SNARK showing knowledge of a sequence alignment using VerifyAlignment, a user would first compute a sequence alignment in plain computation (which generally takes quadratic time). They would then obtain a CIGAR string or a variant thereof (which existing genomics software already produces [28]), and perform a linear amount of cryptographic work to produce a zero-knowledge proof that the alignment between the two sequences has a given cost. For example, seq_1 could be part of a user's DNA sequence (a private input), and seq_2 could be a public input with the sequence for a common gene.[5] Our approach for generating zero-knowledge proofs for the presence of a sequence alignment thus has the desirable property that generating a zero-knowledge proof of the computation takes asymptotically less time than the actual computation. This is a desirable property shared by common algorithms for zero-knowledge proofs for problems like matrix multiplication, discussed in [14].

[5] Of course, the SNARK circuit would also have to check a signature/otherwise authenticate on seq_1 for this to be secure.

4.2 Generalizing to other types of alignment

The previous section considers a simple variant of sequence alignment, commonly called the Levenshtein distance [30], where insertions and deletions all have the same cost, and only two sequences are aligned. In this section, we describe how to generalize the approach of the previous section to other common variants of sequence alignment. While we provide implementations of these variants, in this subsection we sketch the changes needed, since they are relatively simple.

Affine gap costs and other cost functions. In genomics applications, it is common to adopt sequence alignment variants that compute the cost of a sequence alignment in ways that are more complicated than having a fixed cost for each mismatched character or gap character. One common approach to assign a cost to an alignment is "affine gap costs." For a gap (a contiguous sequence of insertions or deletions in an alignment) of length n, affine gap costs assign the gap a cost of $an+b$, for $a, b > 0$. This it typically understood as having a constant "gap open" cost b, and separate constant "gap extension" cost a.

To modify VerifyAlignment to compute the cost of an alignment using affine gap costs, we can modify the parts of VerifyAlignment that increase cost. Specifically, these parts should be modified to add a or $a+b$ to cost depending on whether the previous character of the CIGAR was also 'I' or 'D'.

Furthermore, as long as it is possible to iterate through a CIGAR string and compute the cost of the alignment it represents in linear time (which is true of all practically used gap cost functions we are aware of), VerifyAlignment can be modified to also verify sequence alignments under other cost functions in linear time.

Semiglobal and local alignments. In semiglobal or local sequence alignment variants, subsequences of one or both sequences are aligned against each other instead of the entire sequences. Common variants of semiglobal alignment include "fitting" and "overlap" alignment [31], while in local alignment, one seeks the best alignment of any substring of one sequence against any substring of the other. For example, a user showing that a subsequence of their DNA contains a given gene is typically an instance of semiglobal sequence alignment — the entirety of the gene's sequence is aligned to a sub-string of the genome (at the locus containing the gene). If the characters of a seq_1 and/or seq_2 are stored in an authenticated Merkle tree (or other type of vector commitment), we can implement alignment of subsequences by simply having the circuit for VerifyAlignment check that seq_1 and/or seq_2 are contiguous sequences included in the relevant Merkle tree/accumulator. This approach is specifically useful for doing sequence alignment in a verifiable computation

setting, since this enables the circuit for VerifyAlignment to more efficiently authenticate a sequence by only checking a signature on a Merkle root/accumulator digest.

Multiple alignments and clipping. Multiple Sequence Alignment is a common variant of sequence alignment where multiple sequences are aligned against each other, trying to find the alignment that minimizes the overall cost of the observed mismatches or gaps. Clipping is a common feature of real-world alignment in which certain characters are ignored for the purpose of alignment. For space reasons, we defer a discussion of implementing Multiple Sequence Alignment and sequence alignment with clipping to Appendix C.

4.3 zk-SNARKs for dynamic programming

As explained in the previous two sections, sequence alignment and its variants have a nice property for zero-knowledge proofs: the cost of a solution to these problems can be verified asymptotically faster than computing one. This means that a prover can generate a zk-SNARK showing that they know a solution of a given cost to a problem by performing an amount of cryptographic work which is asymptotically smaller than actually computing the solution in the first place. Generally, dynamic programming algorithms are used to compute sequence alignments. In this section, we observe that many problems which are typically solved with dynamic programming algorithms also have this property. This observation generalizes some prior work, by showing that problems efficiently solvable by dynamic programming are well-suited to zk-SNARKs.

There is not a single "universal" dynamic programming algorithm [17], though broad frameworks that cover many uses and applications of dynamic programming, like modeling them as finding the best path in a DAG under a semi-ring or the best derivation in a hypergraph under a given weight function, have previously been explored and described [32]. Instead, we look at two general dynamic programming problems: parsing context-free grammars [33] and finding shortest paths in a graph without negative cost cycles (in the sense of the Bellman-Ford Algorithm [17]). These two problems are general in the sense that many other problems commonly solved with dynamic programming can also be reduced to instances of these problems and then solved using the relevant algorithms [34, 35].

The previous works [33, 36, 37] all describe how to generate zk-SNARKs for proving the parsing of context-free grammars (CFGs). They all make a similar observation: verifying that a "parse tree" for a string under some context free grammar is valid can be done more efficiently than directly parsing the string. It takes linear time to verify the parse tree for a string under a CFG of constant size (see [33]), while the fastest known algorithms for CFG parsing take $O(n^{2.37})$ time (i.e. the time taken by the fastest known matrix multiplication algorithms [38, 39]), and the fastest known dynamic programming-based algorithms take cubic time. These proofs do not show that the parse tree is optimal, though again, this is acceptable in their context.

In Appendix D we describe how to verify the solutions to shortest path problems more efficiently than directly computing them.

5 Succinct data structures for Merkle trees

In a full system using the previous section's approach to produce zero-knowledge proofs of sequence alignment, a user would likely need to show that a relevant segment of their DNA is actually contained in their genome (as discussed in Section 4.2). One common way of doing this would be for users to have a signed Merkle tree over their genome, and generate Merkle inclusion proofs showing that one of the sequences being aligned is contained in their genome. Furthermore, many genomic data sets have millions or billions of entries. If these data sets are stored in Merkle trees for use in verifiable computation, storing the internal hashes of these Merkle trees alone would potentially occupy gigabytes or terabytes of data.

This motivates our study of space-efficient indexes for storing Merkle trees such that Merkle inclusion proofs can be generated efficiently. The approach we design requires $O(\frac{n}{\log n})$ persistent storage in addition to the contents of the tree, while still being able to produce membership proofs for any of the tree's leaves in $O(\log n)$ time. As defined in [19], these are succinct data structures, since they can be used to efficiently answer queries over a data set of size N while using $o(N)$ additional space. We can also show that the data structure we design achieves asymptotically optimal storage usage in some sense in the random oracle model, and discuss a variant of it in Appendix E.2.

5.1 Split Merkle trees

Consider the problem of storing Merkle trees whose leaves have relatively small sizes (i.e. the leaves have $O(1)$ size). We require the leaves to be small because part of this indexing approach involves re-hashing leaves when generating a Merkle inclusion proof. We further assume that the leaf and node hash functions $\mathcal{H}_\ell, \mathcal{H}_n$ can both be evaluated in $O(1)$ time. We call this section's data structure a *Split Merkle tree* because it involves splitting a Merkle tree at some height and handling the hashes of the nodes above and below this height differently.

Consider a Merkle tree over a vector of length n, with leaf/node hash functions $\mathcal{H}_\ell, \mathcal{H}_n$ respectively. For ease of presentation/analysis, let n be a power of 2. Recalling some basic facts, a complete, balanced Merkle tree over n leaves consists of $2n-1$ hashes (including the leaf hashes), and building a Merkle tree involves evaluating $2n-1$ hashes. A complete binary tree of height h contains 2^h-1 entries.

For the Split Merkle tree, we define an additional *split height* parameter $\mathsf{h}_{\mathsf{split}}$ (ultimately set to $\lceil \log \log n \rceil$). At a high level, we modify a complete binary Merkle tree with n leaves to only store the top $\log n - \mathsf{h}_{\mathsf{split}}$ layers of hashes in the tree. When generating a Merkle inclusion proof, the prover generates the bottom $\mathsf{h}_{\mathsf{split}}$ entries of the path by recomputing the internal hashes of the Merkle tree, and reads the top $\log n - \mathsf{h}_{\mathsf{split}}$ entries of the path from storage. The algorithm to verify an inclusion proof is unchanged. We give pseudocode in Figure 3, and a diagram in Figure 2.

To analyze the costs of this approach, storing a Split Merkle tree involves storing the top $\log n - \mathsf{h}_{\mathsf{split}}$ layers of a Merkle Tree. When $\mathsf{h}_{\mathsf{split}} = \lceil \log \log n \rceil$, this requires storing $2^{\log n - \lceil \log \log n \rceil} - 1 \leq 2^{\log n} / 2^{\log \log n} = \frac{n}{\log n} = O(\frac{n}{\log n})$ hashes. Computing a membership proof involves reading $\log n - \mathsf{h}_{\mathsf{split}}$ hashes from memory, which takes $O(\log n)$ time. Recomputing the internal hashes for the first $\lceil \log \log n \rceil$ entries of the path requires recomputing the internal hashes of a complete Merkle tree of height $\lceil \log \log n \rceil$, which takes $O(2^{\lceil \log \log n \rceil}) = O(\log n)$ time. Thus, this achieves the claimed space and time usage.

To give concrete numbers for this approach, consider a Merkle tree over a commonly used dataset like a synthetic version of the UK Biobank dataset [40]. A Merkle tree for each value in this dataset would have approximately 2^{34} entries, and thus $\sim 2^{35}$ hashes. $\mathsf{h}_{\mathsf{split}}$ would be 6 for this dataset. Thus, a Split Merkle tree for this dataset would store $64\times$ fewer hashes, at the cost of evaluating 64 hashes per inclusion proof. Assuming hashes occupy 32 bytes, this would reduce the storage requirements for a Merkle tree over this dataset from 1.1TB of hashes to about 17 GB. We implement benchmarks in Section 8.2.

We can show that Split Merkle trees achieve asymptotically optimal storage usage in the random oracle model. In particular, we show that any algorithm that can correctly generate Merkle inclusion proofs for an arbitrary Merkle tree in $O(\log n)$ time must use persistent storage of size $\Omega(\lambda \frac{n}{\log n})$. We use a compression argument [41, 42], and show that an algorithm that uses $o(\lambda \frac{n}{\log n})$ storage implies an information-theoretically impossible compression scheme for random data. We give a full proof in Appendix E.1. We also observe that Merkle trees with sufficiently large leaves have a similar "succinct data structure" property. We describe this in Appendix E.2.

6 Verifiable Genome-Wide Association Studies

As mentioned previously, a Genome-Wide Association Study (GWAS) is a type of study in which researchers find correlations between variations in DNA sequences and expressed traits in organisms (like a correlation between a disease and certain genes). GWASs typically involve fitting some statistical model to genetic data/expressed traits to find these correlations. We propose using SNARKs to prove that statistics about a GWAS were honestly computed to give a verifiable guarantee of the results of a study on private medical data.

We explore verifying the results of a simple type of linear regression. Lasso regression is a commonly used form of regression used in GWASs. As explained before, Lasso regression is a variant of standard linear regression, with a *regularization term*, which constraints the l_1 norm of model coefficients. Intuitively, this is useful for genomic studies where the goal is to find a small number of genetic variations that "explain" some trait. The regularization constraint forces a regressed model to choose a small number of coefficients of high magnitude.

Lasso regression does not have a closed-form solution, and is normally computed by some variant of gradient descent. We again make the simple observation that using precomputed advice allows us to substantially decrease the circuit size required for showing that the results of a Lasso regression achieve certain statistics. Specifically, again, the circuit can take the final model coefficients and evaluation data, and verify that they achieve a given loss and coefficient of determination (R^2). This same basic idea can be applied to computing other goodness-of-fit metrics for a linear regression, like p-values. We sketch this simple algorithm below:

1 : $\mathsf{VerifyModelPerformance}(\beta : \mathbb{R}^{m+1}, x : \mathbb{R}^{m \times n}, y : \mathbb{R}^n, t : \mathbb{R}) :$

2 : $L = \sum_{i=0}^{n} (y_i - \beta_0 - x_i^T \beta_{1..n})^2$

3 : $\mathsf{assert} \sum_{i=0}^{m+1} |\beta_i| \leq t$

4 : $\bar{y} = \frac{1}{n} \sum_{i=0}^{n} y_i$

5 : $R^2 = 1 - (\sum_{i=0}^{n} (y_i - \beta_0 - x_i^T \beta_{1..n})^2) / (\sum_{i=0}^{n} (y_i - \bar{y})^2)$

6 : $\mathsf{return}\ L, R^2$

To optimize the concrete costs of this algorithm, we propose using fixed point arithmetic [43], rather than floating point arithmetic. We review the details of fixed point arithmetic in Appendix F.1. Fixed point operations are concretely cheaper in zk-SNARKs, since they can generally be directly implemented as field operations. There is a small loss in precision from using fixed point arithmetic which we quantify in Section 8. By using lookup tables, we can further optimize the SNARK circuits verifying a Lasso regression. In particular, we can optimize the division step in fixed point multiplication and the regularization check for Lasso regression with lookup tables. We describe the details of optimizing these steps in Appendix F.1. We note that optimizing the regularization check step (checking that the sum of the coefficients β_i is at most t) is meaningful, since the vector β is often large relative to the size of the evaluation set in applications of Lasso regression. Lastly, to authenticate x, y, we use a variant of the approach for authenticating large files developed in

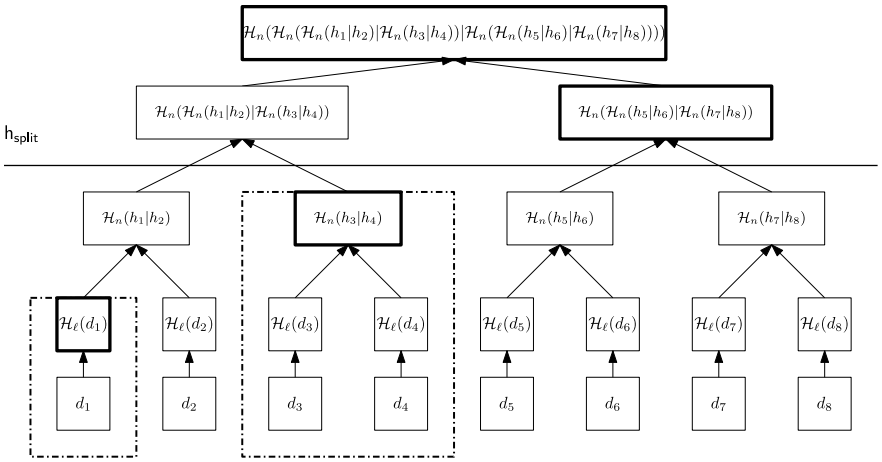

Figure 2: A Split Merkle Tree with n = 8, $\mathsf{h_{split}} = 2$. Only hashes above the horizontal line are stored. An inclusion proof for leaf d_2 is highlighted, and the vertical boxes indicate the hashes that need to be recomputed when generating an inclusion proof for d_2.

SplitMT.Build(leaves : D^n, $\mathsf{h_{split}} : \mathbb{N}$):

1. assert $\mathsf{h_{split}} \leq \log n \wedge \log_2 n \in \mathbb{Z}$.
2. Let vanillaTree = MerkleTree.Build(leaves)
3. Let splitTreeHashes be the list of hashes in the top $\log n - \mathsf{h_{split}}$ layers of vanillaTree.
4. Let root be the root of vanillaTree.
5. Return (splitTreeHashes, root).

SplitMT.IncProof(treeHashes, root, leaves, idx, $\mathsf{h_{split}}$):

1. Let $\mathsf{size_{chunk}} = 2^{\mathsf{h_{split}}-1}$.
2. Split leaves into equal chunks of size $\mathsf{size_{chunk}}$. Let chunk be the chunk at index $\lfloor \mathsf{idx}/\mathsf{size_{chunk}} \rfloor$.
3. Let subtree = MerkleTree.Build(chunk).
4. Let $\mathsf{proof_2}$ = MerkleTree.IncProof(subtree, $\mathsf{idx}\%\mathsf{size_{chunk}}$).
5. Let $\mathsf{proof_1}$ be the first $\log n - \mathsf{h_{split}}$ entries of the Merkle Proof for idx, as stored in treeHashes.
6. Return $\mathsf{proof_1}||\mathsf{proof_2}$.

SplitMT.VerifyProof(root, path, leaf,):

1. Return MerkleTree.VerifyProof(root, path, leaf).

Figure 3: Split Merkle tree pseudocode. We use the algorithms MerkleTree.{Build, IncProof, VerifyProof} for a regular Merkle tree, as discussed in Section 3.1.

VerITAS [44] for verifying images. We describe the approach in Appendix F.2.

This approach of proving that a set of coefficients achieves certain goodness-of-fit metrics does not show that a set of coefficients is optimal, or that the training process was honestly performed. This is a weaker notion than others in prior literature. Zero-Knowledge Proofs of Training [45] are are much more expensive than our approach, though they do provide a guarantee of how a model was trained. Proofs of optimimum vicinity [46] or optimal solutions to a numerical optimization problem [47] are also more expensive than our approach, but have more modest costs and prove that a computed solution is (near) optimal, assuming an optimization problem has certain structure. These approaches could be combined with ours, to prove optimality of coefficients in addition to proving that statistics were honestly computed.

While our approach does not generate proofs of optimality, prior notions of zero-knowledge proofs of training often do not cover hyperparameter selection or initial state selection [46], so these notions have some similar limitations.

7 Zero-knowledge proofs for CRISPR gene editing eligibility

Next we turn to our proposed use cases for zero-knowledge proofs in gene editing. As described in Section 3.2, CRISPR uses "probes" to define the DNA locations at which new edited DNA is spliced into the sequence. In general, for a patient to be eligible for a CRISPR-based therapy, their specific genome must meet certain requirements.

First, a patient must have the CRISPR probe bind to their DNA at the correct locations. These are called "on-target" sites. Patients must have an on-target site in their DNA sequence so that a CRISPR probe can splice in DNA at the correct location with high probability. Secondly, patients must not have sequences in their DNA which match a given CRISPR probe at incorrect locations in their genome. These locations are called "off-target sites". If a user has off-target sites in

their genome, a CRISPR therapy might edit their DNA in unintended locations, leading to potential negative health outcomes such as cancer.

We propose using zk-SNARKs in the context of CRISPR therapies for users to prove that they have effective on-target sites for a given therapy in their DNA, and do not have risky off-target sites elsewhere in their DNA. By using zk-SNARKs, users do not reveal their DNA sequences to the party offering CRISPR gene editing. On the other hand, a party offering gene editing services does not need to store patient's genetic data, and has a verifiable guarantee that they will not harm their patients.

In this section, we describe the details of implementing existing algorithms from the literature for detecting on-target and off-target sites for CRISPR probes. We pick Rule Set 1 [9] for the on-target algorithm and MIT Score [10] for the off-target algorithm. These algorithms are from the first generation of computational tools used to design CRISPR experiments and we choose them as lightweight example algorithms to demonstrate feasibility.

7.1 Overall implementation details

We first describe some details of the data used as inputs for our implementations of CRISPR-related algorithms. We expect users to provide two private inputs related to their DNA sequence. The first is the sequence of their genome, signed by a trusted DNA sequencer. Along with this, we use a signed annotation of the ranges where genes are found in the individual's DNA. This is a reasonable requirement, since annotation files for reference genomes exist and are commonly generated as part of DNA sequencing [48], and lightweight tools exist for coordinate conversion between a reference and individual genomes [49–51].

Additionally, DNA is double-stranded [52], and this affects how algorithms for these problems are designed. We run the scoring algorithms for measuring the quality/likelihood of a site on both strands of a DNA sequence: the forward (5'-3' direction) and the reverse-complement (3'-5' direction). As an additional note, it is more desirable for CRISPR edits to occur earlier in a gene, since these have a greater effect. Depending on whether the gene is on the forward strand or reverse strand, a lower or higher index of an on-target edit is more desirable.

7.2 Rule Set 1 algorithm implementation

Rule Set 1 [9] is an algorithm to compute on-target efficiency where a 30 base pair (bp) input sequence goes through a simple linear function and then a sigmoid function. The input is composed of (in order) a 4-bp upstream sequence, a 20-bp spacer (target of the CRISPR edit), a 3-bp PAM, and 3-bp downstream. As mentioned in Section 3.2, the PAM must be on the structure "NGG".

Given this input, a (floating-point) linear score is computed based on the intercept value (0.597636154) summed by $\texttt{coeff}_i$ if feature i exists. In Rule Set 1, there are 72 features. An example feature is "GT02," which checks if index 2 of a sequence is "G" and if index 3 is "T," with a coefficient value of -0.625778696.

Next, a GC score modification is added to the linear score base on how many G and C bases appear in the 30-bp input:

$$\begin{cases} (\text{GC}_{\text{count}} - 10) \cdot \text{GC}_{\text{high}} & \text{if } \text{GC}_{\text{count}} > 10 \\ (10 - \text{GC}_{\text{count}}) \cdot \text{GC}_{\text{low}} & \text{if } \text{GC}_{\text{count}} \leq 10 \end{cases}$$

For calculating the linear score, we used fixed point arithmetic , discussed in Appendix F.1. Finally, a sigmoid function is applied to the linear score as $1/(1+e^{-linear_score})$.

Calculating this sigmoid function would be expensive in a zero-knowledge circuit, so instead we employ a lookup table based on thresholds. If the score is above the thresholds $(0.2, 0.4, 0.5, 0.7)$ then it has (low, medium, good, high) effectiveness, respectively, based on work by experimental studies [53].

7.3 MIT Score algorithm implementation

The MIT Score off-target algorithm [10] takes as input a 20-bp spacer, the target sequence for editing, and a 20-bp protospacer, a sequence in the genome of the organism. The output score is how likely the sequence in the protospacer will be (un-intentionally) modified by the CRISPR edit.

The formula used is:

$$\left(\prod_{p \in M} w_p\right) \cdot \frac{1}{\frac{19-d}{19} \cdot 4 + 1} \cdot \frac{1}{m^2},$$

where M is the set of mismatched positions between the spacer and protospacer, w_p is the position based penalty for a mismatch at that position, m is the size of M, and d is the average distance between consecutive mismatches. If there is only 1 mismatch, then $d = 19$.

This computation uses floating point numbers. As mentioned before, these are expensive in zero-knowledge, so we use fixed point arithmetic, which is reviewed in Appendix F.1.

Like with the on-target scoring algorithm, after this score is computed, we use a lookup to determine if the edit has a low (≤ 0.5), medium (> 0.6), or high (> 0.8) probability of being an unintended change. Again, the thresholds are based on work by experimental studies [53].

8 Implementation and evaluation

In this section, we detail the implementation and evaluation of our end-to-end verifiable genomics applications and supporting building blocks, i.e., proofs for sequence alignment and Split Merkle trees for large scale genomic data. We start with

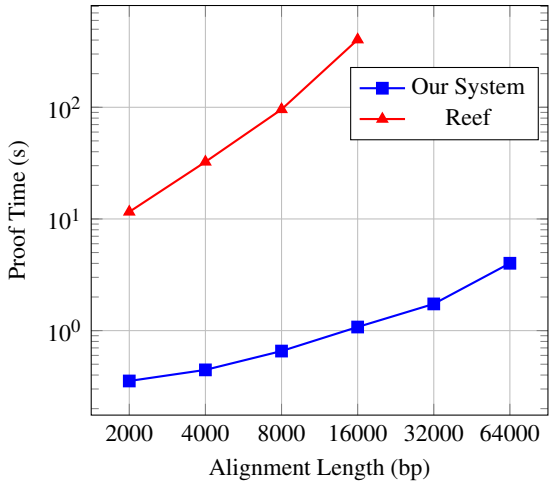

Figure 4: Alignment length vs. proving time for circuits implementing basic variant of sequence alignment.

the building blocks and then present our applications for (1) a verifiable Genome Wide Association Study (GWAS) on an authenticated data store and (2) verifiable CRISPR eligibility.

We used Microsoft's Spartan2 library [12] with the `bellpepper` frontend for our experiments with sequence alignment. For our Merkle tree experiments, we used the Plonky3 library's [54] implementation of the Poseidon hash function/Merkle trees. We used the Noir programming language [11] with the Barretenberg [55] backend to implement our experiments with GWAS and CRISPR. We used the UltraHonk proof system from Barretenberg. The source code for all experiments is available in our online repository.

All benchmarks were run on a laptop running Windows Subsystem for Linux (WSL2) with a 12-core AMD Ryzen AI 9 HX 370 series CPU and 96 GB of RAM.

8.1 Verifiable sequence alignment

In Figure 4, we plot performance results for our circuits proving knowledge of a solution for the basic variant of sequence alignment. "Alignment length" refers to the length of the advice string given as input to the sequence alignment circuit, since this determines the overall circuit size. To give real-world context for the sequence lengths we chose, the median length of a human gene is approximately 15,000 base pairs [56]. The time to verify these proofs ranges from 112ms to 399ms. Prover RAM usage for our circuits varies approximately linearly from 666.93MB to 36.01GB. In contrast, Reef uses up to 3.6GB of RAM in its largest circuit, where our construction uses 2.15 GB (1.67× less) for that size.

Compared to Reef [5], our approach is 30-130× faster. For example, when proving the alignment of 2 sequences of 2000 characters, our system requires 354 ms of prover time, while Reef requires 11.57s. We did not benchmark Reef for alignment lengths longer than 16,000 characters because building the circuits required over 1 GB of stack memory, exceeding our machine's resources. We note that Reef does not implement sequence alignment, but rather regular expression matching for their proofs about DNA matches. Regular expression matching is a different problem from sequence alignment, and Reef's approach of using regular expressions to represent alignments is not as expressive or very commonly used in genomics.

Compared to zkMSA [29], our circuits for verifying a pairwise alignment have linear size, while zkMSA's have quadratic size, because of zkMSA's usage of a memory scheme with linear per-access costs. zkMSA does not run their circuits and only evaluates constraint counts. For instance, to verify an alignment of two sequences of 200 characters, zkMSA requires about 100× more constraints than our approach.[6] This difference increases linearly for longer sequences.

SecureED [57], a recent work which implements sequence alignment in general multi-party computation,[7] requires 11.43 seconds to compute the alignment of 2 4000 base pair sequences, while our work can prove the alignment of 2 4000 base pair sequences in 445 milliseconds (around 25× faster). Our protocol is substantially faster than SecureED and requires much less communication (SecureED requires transmitting 3370 MB of data when computing an alignment of 2 sequences of length 4000, while our protocol's proofs have sizes of tens of kilobytes, and 39.996kB for this specific problem size). SecureED's circuits do compute an optimal sequence alignment and, as discussed in the introduction, MPC and zk-SNARKS are different primitives best suited to different use cases, so this is only an apple-to-apples in limited settings like verifiable genomics.

In addition to the algorithm above, we also implement zk-SNARKs for verifying sequence alignment under affine gap costs and verifying multiple sequence alignment, which are common variants of alignment in the genomics literature. These are not covered in previous work. The performance of these variants is largely comparable to that of the basic variant discussed above. We describe the performance of these variants of sequence alignment in Appendix G.1.

8.2 Storage-efficient Merkle trees

In Table 1, we show the performance of our Split Merkle tree over a vector of size comparable to a haploid human genome (2^{25} leaves of about 32 bytes each), represented as field elements. "Storage size" measures the storage space used by the given tree variant to store hashes. We chose this vector size so our benchmarks were over a piece of data of size relevant to genomics algorithms. In these benchmarks,

[6] We got this number by building their pairwise alignment circuit restricted to 2 sequences of 200 characters with an alignment length of 200 characters.

[7] We used SecureED's numbers with the fastest network configuration.

Tree Variant	Storage Size	Witness Gen. Time
Baseline: Vanilla	2048 MB	29.126µs
Split, $h_{split} = 3$	256 MB	34.52µs
Split, $h_{split} = 4$	128 MB	44.68µs
Split, $h_{split} = 5$	64 MB	66.83µs
Split, $h_{split} = 6$	32 MB	112.07µs
Split, $h_{split} = 10$	2047 kB	1.49ms
Split, $h_{split} = 15$	63 kB	46.40ms
Split, $h_{split} = 20$	1 kB	1.49s

Table 1: Split tree performance for a human-genome sized vector vs. a standard Merkle tree. Storage size is the additional storage needed to hold the hashes of the tree, while Witness Gen. Time is the time to compute an inclusion proof.

the internal hashes of the Merkle tree were stored in persistent storage to reflect a real-world deployment scenario.

As the "split height" is increased, the storage required for the Merkle tree decreases, while the time to generate inclusion proofs increases, and proof verification times stay approximately constant. When h_{split} is 3-6, the time to generate an inclusion proof only increases by tens of microseconds, while storage costs decrease by 8-64×, which is likely an acceptable cost to pay for the amount of storage saved. We note that the exact "sweet spot" value of h_{split} is highly dependent on hardware and implementation details (i.e. the relative performance between hashing data and random access reads to storage). For instance, when running these same benchmarks on an Apple M3 MacBook Air, inclusion proof generation time for h_{split} ranging from 3-7 was actually faster than for the vanilla tree.

As discussed in Section 5, we expect these savings to extend to larger databases: given a commonly used dataset [40], this would reduce storage costs of the full tree from 1.1TB of hashes to about 17 GB. We discuss another variant of storage-efficient Merkle trees in Appendix E.2, and we benchmark their performance in Appendix G.2.

8.3 Verifiable GWAS on authenticated data

In Table 2, we give the prover time required to prove the results of a Lasso regression for problems of different sizes. We generated synthetic data sets of sizes equal to those used by existing papers in the genomics literature that perform GWAS studies using Lasso regression for the first four rows. We generated random synthetic data because the relevant data sets were not openly available. The last row uses the QTL-MAS 2010 simulated dataset [58] [59] to evaluate quantitative trait locus (QTL) mapping and genomic prediction in livestock populations. Peak prover RAM usage for these experiments ranged from 5.04GB to 22.38GB. We briefly summarize the

# of variables × # of data points	Proof Time	Verif. Time
729 × 291 [60]	17.44s	13.67ms
30 × 10,000 [61]	31.66s	11.84ms
1000 × 867 [62]	72.03s	14.273ms
13,572 × 697* [63]	1042.82s	2.85s
10,031 × 2326* [58] [59]	2270.52s	3.07s

Table 2: Verifiable Genome Wide Association Study (GWAS) performance using Lasso regression. Data points marked with * had data sets broken into chunks to create separate proofs.

papers that perform these studies in Appendix G.3. In rows marked with an asterisk, we split the data set into multiple subsets, and generated proofs for the subsets individually.[8]

As can be seen, we achieve reasonable performance for verifying Lasso linear regression for smaller-scale studies in the genomics literature. There are recent studies in hundreds of thousands of variables and data points, which existing proof techniques would have difficulty scaling to. For example, [64] runs a variant of Lasso regression on a data set with more than 1 million variables and and 300,000 data points. To our knowledge, our work is the only work implementing Lasso regression in zero-knowledge proofs.

We compared the accuracy of our Lasso circuit to a ground truth from Pythons `scikit-learn` library. For the QTL-MAS 2010 simulated dataset [58] [59], using 64 bits of precision, our implementation achieves no difference in R^2 computed by `scikit-learn` and a a difference from the model loss computed by `scikit-learn` of $2.39 \times 10^{-14}\%$ (2.84×10^{-14} in magnitude).

[65] uses differential privacy (DP) techniques to build a system for verifiable GWASs with weaker security properties than ours (we discuss the differences in Section 9). Their results are not directly comparable to ours, since they implement a different type of regression. To give a high level comparison, their system is faster than ours because of their use of differential privacy. However, DP also leads to accuracy losses: [65] quotes that their system can catch deviations of more than 0.045 from correctly computed p-values with at least 90% probability. This is a much greater accuracy loss than the one achieved by our circuits.

8.4 CRISPR eligibility proofs

Recall from Section 7 that CRISPR gene therapy requires two types of computation to be performed before the procedure: determining the efficacy of the treatment via on-target scoring, and determining the risk of unintended changes via off-target scoring. In this section, we benchmark our implementations of

[8] This is mainly because of limitations of the Noir circuit compiler, not a performance limitation of the prover.

CRISPR-related algorithms. This includes running the respective on and off targeting algorithms, as well as authenticating the relevant data.

In Table 3, we show proof and verification times for the on-target scoring algorithm Rule Set 1 and the off-target scoring algorithm MIT Score. They roughly show a linear growth in proving time as the genome sequence length increases. In the tested configurations, on-target computation is up to 60% more expensive to run. Note that on-target scoring is only required to be run for one gene/location (to show the existence of a site), while off-target scoring must be done for the entirety of the genome (to show absence of sites).

In Table 4, we show computation times for the edge cases where part of the input segment is the targeted gene, and the rest is not. In this circuit, we keep the gene's location private. This computation is up to three times more expensive than the computation of on or off target scoring for equivalent lengths, but only has to be done up to two times at the boundary of the targeted gene. In Appendix G.4, we provide benchmarks for circuits where the location of the targeted gene is not hidden, and instead hard-coded into the circuit. Prover RAM usage for these experiments varied approximately linearly in the problem size from 349.66MB to 19.42GB.

Feasibility for human (and other) genomes. In order to prove these statements in practice, minor additional overhead is required. The examined segments must be authenticated as being part of a human's genome, for instance, by hashing and checking inclusion of the segment in a Merkle tree. We include these checks in our benchmarks, though they only use a few thousand gates, while the CRISPR scoring algorithms take millions over measured segment lengths.

Our implemented algorithms can be used to generate proofs for on-target scoring for a human genome, but do not scale to human genome size for off-target effects. In particular, on-target scoring only requires the prover to show that specific locations in the genome have a sufficiently high score. In contrast, off-target scoring requires the prover to iterate over the entire genome, showing that a probe does not match any other *unintended* candidate locations.

In particular, consider the HBB gene [66] [67], which is associated with sickle cell disease [68] [69]. Studies show that CRISPR gene therapy can cure sickle cell disease [70]. HBB is 1,608 base pairs [67], and a proof that a proposed therapy would have an on-target effect takes less a minute to compute and less than a second to verify (see Table 4).

In contrast, a complete CRISPR gene editing eligibility proof with on and off target scoring would need to iterate over the whole human genome and would take 220 days to prove on our benchmark machine. This is obviously not viable at present, but we note that advances in zk-SNARK systems, particular for GPU proving targeted at verifiable LLMs, could substantially increase performance.

At the same time, our techniques are viable to prove off-target effects (or the lack thereof) on an entire yeast genome.

	On-Target		Off-Target	
Segment Size	**Proving Time**	**Verif. Time**	**Proving Time**	**Verif. Time**
100bp	1.58s	11.2ms	1.59s	11.6ms
1000bp	15.13s	11.5ms	10.01s	11.7ms
2000bp	29.41s	11.7ms	18.26s	11.7ms
4000bp	-	-	36.15s	12.2ms

Table 3: Time to generate and verify a proof of on-target or off-target scoring over the entire input segment. The Noir compiler's RAM usage exceeded the limits of our setup for the 4000bp segment size.

Segment Size	Proving Time	Verification Time
100bp	2.80s	12.0ms
1000bp	30.75s	12.1ms
2000bp	56.32s	12.3ms

Table 4: Time to generate and verify a proof where the input segment partially contains the targeted gene.

A common target for CRISPR edits on yeast is the ADE2 gene which changes the yeast's color [71]. ADE2 is 1,716 base pairs, and since the yeast genome is only 9,040,256 base pairs, off-target computation for the rest of the genome would take an estimated 1.71 days on a laptop.

8.4.1 Accuracy estimates

Our circuits have some accuracy loss because they use fixed-point arithmetic to implement algorithms which normally use floating point arithmetic. We quantify this accuracy loss.

"On-Target" model: Rule Set 1. Rule Set 1 considers a 30-bp window, where two bases in the PAM are expected to be 'G'. Since this algorithm is dependent on the exact values at the other 28 positions, there exist 4^{28} possible inputs. As it is computationally infeasible to run all possibilities, we instead run 2^{20} random instances. We measure the accuracy loss before the sigmoid function, as we use a lookup table based on the value before applying it. We find that the zero-knowledge circuit has a median error of 0.00016% and a mean of 0.00071%, equal to an error of less than 10^{-5}.

"Off-Target" model: MIT Score. We evaluate accuracy of our MIT Score implementation over all possible 2^{20} mismatch combinations. Median and mean accuracy losses are 6% and 16.5%, respectively, equating to a negligible difference less than 10^{-6} on the actual score, especially when the threshold boundaries are $(0.5, 0.6, 0.8)$. Indeed, we observe much better accuracy when filtering out negligible scores. For example, when filtering out scores <0.001 we see a median percentage error of 0.0273% and mean percentage error of 0.0327%.

9 Related work

Because of the sensitivity of genetic data, genomic computations are a frequently proposed use case for cryptographic primitives. Prior work has primarily focused on applying Multi-Party Computation or Fully Homomorphic Encryption to genomic computations. Since 2014, the annual iDASH workshop [72] has been running competitions where researchers compete to optimize various types of secure genomics computations. This has been an impetus for a large amount of research in secure computation and genomics.

There are many works using Fully Homomorphic Encryption and simpler homomorphic encryption schemes for performing secure computation on genomic data in the literature, with many papers related to iDASH. Many papers focus on the use case of a Genome-Wide Association Study (GWAS) [3, 73–78]. There is also work on using implementing sequence alignment [79, 80].

There is also a great deal of work designing Multi-Party Computation protocols for genomic computations. This also includes related primitives, like Private Set Intersection [81]. Protocols have been designed for different numbers of parties and different trust assumptions. These protocols solve problems like edit distance between strings [2, 57, 82, 83], analyzing genomics data sets for GWAS or other contexts [84–86], and securely implementing genomics-related queries over databases [87–89].

There is also some work on applying other secure computation techniques to genomic computations, like ORAM [90], trusted hardware [91] or functional encryption [92].

Lastly, as far as we are aware, the following papers have explored genomics inside of zero-knowledge proofs: As mentioned before, Reef [5] proposes users proving that they have a gene as a use-case for zero-knowledge proofs. One main limitation of Reef for genomics is the use of regular expressions rather than alignments to express sequence similarity. Suwannik [29] designs circuits for verifying multiple sequence alignment inside of a zero-knowledge proof, but they only focus on global alignments and their circuits have quadratic rather than linear size for each pairwise alignment.

Efficiently storing large Merkle Trees has been explored in the trusted hardware and blockchain communities. For instance, [93] designs a data structure called a "Merkle Bonsai Tree," which is specific to the trusted hardware domain, but has similar ideas about optimizing Merkle Trees for storage consumption. The Zcash codebase [94] contains an approach for storing a large Merkle Tree to improve the efficiency of "fast-forwarding" a client's stored state of the blockchain. We believe that our formal analysis of approaches for compactly storing Merkle Trees is our main contribution in this direction.

There is a prior literature on representing linear regression, logistic regression, and other variants of optimization problems in zk-SNARKs [45–47]. [45] builds (non-succinct) zero-knowledge proofs of training for logistic regression. [46, 47] design approaches for proving that solutions to certain types of optimization/gradient descent problems are (near) optimal. The industry codebase [95] has an approach for computing zero-knowledge proofs of linear regression similar to our algorithm in Section 6.

Two prior works [65, 96] build verifiable GWASs, though in different security models. [96] targets a use case where a verifier outsources the computation of a GWAS to a server and verifies the computed results. Their system does not provide data privacy, and provides a somewhat informal notion of computational integrity. [65] uses differential privacy techniques to build a verifiable GWAS. It provides data privacy, but assumes honest provers. Their use of differential privacy also leads to precision losses. Our work is the first to build verifiable GWASs which both keep data private and guarantee integrity of the computation against a dishonest prover, while achieving cryptographic security guarantees.

Lastly, there is some work on the intersection of CRISPR/gene editing and cryptography, such as as the line of work starting with [97], though this work focuses on authenticating edited genomes, rather than the use case that we explore. Multiple works [98, 99] exist for the computational side of CRISPR editing, but aim to create efficient user-friendly tools, rather than verifiable and privacy enhancing computations like the contributions of this paper.

10 Conclusion

To conclude, we have proposed new use cases for zk-SNARKs in genomics, and optimized various common genomics algorithms for zk-SNARKs. We believe that this work shows that there are interesting and valuable use cases for zk-SNARKs in bioinformatics.

The code for this paper is available at https://github.com/sashafrolov/zk-genomics/tree/main. We used synthetic datasets when possible to improve the reproducibility of the paper, and did not use private datasets.

Acknowledgments. This work was supported in part by DARPA, NSF under Grant 2504578, and US National Institutes of Health grant R01HG009937, and by grants 2022-311195 and 2024-342821 from the Chan Zuckerberg Initiative DAF, an advised fund of the Chan Zuckerberg Initiative Foundation. Opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of DARPA. We thank Miranda Christ, Kanav Gupta, Daniel Apon, Faraz Ghahremani, Jess Woods, Eli Margolin and Haohan Wang for their helpful discussions about parts of this paper.

Disclosure. RP is a co-founder of Ocean Genomics Inc.

References

[1] A. McElhaney, "23andMe site went down as customers struggled to delete data," 2025, https://www.wsj.com/business/23andme-delete-data-bankruptcy-5778341f.

[2] S. Jha, L. Kruger, and V. Shmatikov, "Towards practical privacy for genomic computation," in *Proceedings of the 2008 IEEE Symposium on Security and Privacy*, ser. SP '08. USA: IEEE Computer Society, 2008, p. 216–230. [Online]. Available: https://doi.org/10.1109/SP.2008.34

[3] M. Kim and K. Lauter, "Private genome analysis through homomorphic encryption," *BMC Medical Informatics and Decision Making*, vol. 15, no. 5, p. S3, 2015. [Online]. Available: https://doi.org/10.1186/1472-6947-15-S5-S3

[4] M. Jinek, K. Chylinski, I. Fonfara, M. Hauer, J. A. Doudna, and E. Charpentier, "A programmable dual-RNA–guided DNA endonuclease in adaptive bacterial immunity," *Science*, vol. 337, no. 6096, pp. 816–821, 2012. [Online]. Available: https://www.science.org/doi/abs/10.1126/science.1225829

[5] S. Angel, E. Ioannidis, E. Margolin, S. Setty, and J. Woods, "Reef: Fast succinct non-interactive zero-knowledge regex proofs," USENIX Security 2024, 2023. [Online]. Available: https://eprint.iacr.org/2023/1886

[6] Y. Yang and D. Heath, "Two shuffles make a RAM: Improved constant overhead zero knowledge RAM," Cryptology ePrint Archive, Paper 2023/1115, 2023. [Online]. Available: https://eprint.iacr.org/2023/1115

[7] S. Setty, J. Thaler, and R. Wahby, "Unlocking the lookup singularity with lasso," Cryptology ePrint Archive, Paper 2023/1216, 2023. [Online]. Available: https://eprint.iacr.org/2023/1216

[8] R. Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 58, no. 1, pp. 267–288, 12 2018. [Online]. Available: https://doi.org/10.1111/j.2517-6161.1996.tb02080.x

[9] J. G. Doench, E. Hartenian, D. B. Graham, Z. Tothova, M. Hegde, I. Smith, M. E. Sullender, B. L. Ebert, R. J. Xavier, and D. E. Root, "Rational design of highly active sgRNAs for CRISPR-Cas9-mediated gene inactivation," *Nature Biotechnology*, vol. 32, no. 12, pp. 1262–1267, 2014.

[10] P. D. Hsu, D. A. Scott, J. A. Weinstein, F. A. Ran, S. Konermann, V. Agarwala, Y. Li, E. J. Fine, X. Wu, O. Shalem *et al.*, "DNA targeting specificity of RNA-guided Cas9 nucleases," *Nature biotechnology*, vol. 31, no. 9, pp. 827–832, 2013.

[11] noir-lang, "noir: A domain specific language for zero knowledge proofs," https://github.com/noir-lang/noir, 2026.

[12] S. Setty, "Spartan2: High-speed zkSNARKs without trusted setup," https://github.com/microsoft/Spartan2, 2025.

[13] C. Stillman, J. E. Bravo, C. Boucher, and S. Rampazzi, "Toward security-aware portable sequencing," *Nature Communications*, vol. 16, no. 1, p. 9829, 2025. [Online]. Available: https://doi.org/10.1038/s41467-025-66024-z

[14] J. Thaler, *Proofs, Arguments, and Zero-Knowledge*, 2022. [Online]. Available: https://people.cs.georgetown.edu/jthaler/ProofsArgsAndZK.pdf

[15] D. Boneh and V. Shoup, *A Graduate Course in Applied Cryptography*, 2015. [Online]. Available: https://toc.cryptobook.us/book.pdf

[16] M. Campanelli, A. Faonio, D. Fiore, T. Li, and H. Lipmaa, "Lookup arguments: Improvements, extensions and applications to zero-knowledge decision trees," Cryptology ePrint Archive, Paper 2023/1518, 2023. [Online]. Available: https://eprint.iacr.org/2023/1518

[17] J. Kleinberg and E. Tardos, *Algorithm Design*. USA: Addison-Wesley Longman Publishing Co., Inc., 2005.

[18] A. Backurs and P. Indyk, "Edit distance cannot be computed in strongly subquadratic time (unless SETH is false)," in *Proceedings of the Forty-Seventh Annual ACM Symposium on Theory of Computing*, ser. STOC '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 51–58. [Online]. Available: https://doi.org/10.1145/2746539.2746612

[19] G. J. Jacobson, *Succinct static data structures*. Carnegie Mellon University, 1988.

[20] A. Nouira and C.-A. Azencott, "Multitask group Lasso for Genome Wide association Studies in diverse populations," in *Biocomputing 2022*. WORLD SCIENTIFIC, Nov. 2021, p. 163–174. [Online]. Available: http://dx.doi.org/10.1142/9789811250477_0016

[21] B. Mieth, A. Rozier, J. A. Rodriguez, M. M. C. Höhne, N. Görnitz, and K.-R. Müller, "DeepCOMBI: explainable artificial intelligence for the analysis and

discovery in genome-wide association studies," *NAR Genomics and Bioinformatics*, vol. 3, no. 3, Jun. 2021. [Online]. Available: http://dx.doi.org/10.1093/nargab/lqab065

[22] A. van Hilten, S. A. Kushner, M. Kayser, M. A. Ikram, H. H. H. Adams, C. C. W. Klaver, W. J. Niessen, and G. V. Roshchupkin, "GenNet framework: interpretable deep learning for predicting phenotypes from genetic data," *Communications Biology*, vol. 4, no. 1, Sep. 2021. [Online]. Available: http://dx.doi.org/10.1038/s42003-021-02622-z

[23] L. Cong, F. A. Ran, D. Cox, S. Lin, R. Barretto, N. Habib, P. D. Hsu, X. Wu, W. Jiang, L. A. Marraffini *et al.*, "Multiplex genome engineering using CRISPR/Cas systems," *Science*, vol. 339, no. 6121, pp. 819–823, 2013.

[24] P. Mali, L. Yang, K. M. Esvelt, J. Aach, M. Guell, J. E. DiCarlo, J. E. Norville, and G. M. Church, "RNA-guided human genome engineering via Cas9," *Science*, vol. 339, no. 6121, pp. 823–826, 2013.

[25] J. G. Doench, N. Fusi, M. Sullender, M. Hegde, E. W. Vaimberg, K. F. Donovan, I. Smith, Z. Tothova, C. Wilen, R. Orchard *et al.*, "Optimized sgRNA design to maximize activity and minimize off-target effects of CRISPR-Cas9," *Nature biotechnology*, vol. 34, no. 2, pp. 184–191, 2016.

[26] P. C. DeWeirdt, K. R. Sanson, A. K. Sangree, M. Hegde, R. E. Hanna, M. N. Feeley, A. L. Griffith, T. Teng, S. M. Borys, C. Strand *et al.*, "Optimization of AsCas12a for combinatorial genetic screens in human cells," *Nature biotechnology*, vol. 39, no. 1, pp. 94–104, 2021.

[27] P. C. DeWeirdt, A. V. McGee, F. Zheng, I. Nwolah, M. Hegde, and J. G. Doench, "Accounting for small variations in the tracrRNA sequence improves sgRNA activity predictions for CRISPR screening," *Nature Communications*, vol. 13, no. 1, p. 5255, 2022.

[28] H. Li, B. Handsaker, A. Wysoker, T. Fennell, J. Ruan, N. Homer, G. Marth, G. Abecasis, R. Durbin, and . G. P. D. P. Subgroup, "The sequence alignment/map format and SAMtools," *Bioinformatics*, vol. 25, no. 16, pp. 2078–2079, Aug. 2009, epub 2009 Jun 8.

[29] W. Suwannik, "Zero knowledge proof for multiple sequence alignment," 2024. [Online]. Available: https://arxiv.org/abs/2404.19064

[30] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," *Soviet physics. Doklady*, vol. 10, pp. 707–710, 1965. [Online]. Available: https://api.semanticscholar.org/CorpusID:60827152

[31] P. Compeau and P. Pevzner, *Bioinformatics Algorithms: An Active Learning Approach*, 6 2018. [Online]. Available: https://books.google.com/books/about/Bioinformatics_Algorithms.html?hl=&id=RdzLtgEACAAJ

[32] L. Huang, "Advanced Dynamic Programming in Semiring and Hypergraph Frameworks," in *Coling 2008: Advanced Dynamic Programming in Computational Linguistics: Theory, Algorithms and Applications - Tutorial notes*, L. Huang, Ed. Manchester, UK: Coling 2008 Organizing Committee, Aug. 2008, pp. 1–18. [Online]. Available: https://aclanthology.org/C08-5001/

[33] S. Angel, S. Celi, E. Margolin, P. Mishra, M. Sander, and J. Woods, "Coral: Fast succinct non-interactive zero-knowledge CFG proofs," Cryptology ePrint Archive, Paper 2025/1420, 2025. [Online]. Available: https://eprint.iacr.org/2025/1420

[34] A. Abboud, A. Backurs, and V. V. Williams, "If the current clique algorithms are optimal, so is Valiant's parser," in *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*, 2015, pp. 98–117.

[35] S. Dasgupta, C. H. Papadimitriou, and U. Vazirani, "Dynamic programming," in *Algorithms*. Draft / Lecture Notes, University of California, Berkeley and UCSD, 2006, pp. 169–199, available online. [Online]. Available: https://people.eecs.berkeley.edu/~vazirani/algorithms/chap6.pdf

[36] E. Laufer, A. Ozdemir, and D. Boneh, "zkPi: Proving lean theorems in zero-knowledge," Cryptology ePrint Archive, Paper 2024/267, 2024. [Online]. Available: https://eprint.iacr.org/2024/267

[37] H. Malvai, S. Hussain, G. Neven, and A. Miller, "Practical proofs of parsing for context-free grammars," Cryptology ePrint Archive, Paper 2024/562, 2024. [Online]. Available: https://eprint.iacr.org/2024/562

[38] R. Duan, H. Wu, and R. Zhou, "Faster matrix multiplication via asymmetric hashing," 2023. [Online]. Available: https://arxiv.org/abs/2210.10173

[39] L. Lee, "Fast context-free grammar parsing requires fast Boolean matrix multiplication," *Journal of the ACM*, vol. 49, no. 1, pp. 1–15, 2002.

[40] UK Biobank / Nuffield Department of Population Health, "UK biobank synthetic dataset," https://biobank.ndph.ox.ac.uk/synthetic_dataset/, n.d., accessed: 2026-02-05.

[41] J. Bonneau, J. Chen, M. Christ, and I. Karantaidou, "Merkle mountain ranges are optimal: On witness update frequency for cryptographic accumulators," CRYPTO 2025, 2025. [Online]. Available: https://eprint.iacr.org/2025/234

[42] C. Dwork, M. Naor, G. N. Rothblum, and V. Vaikuntanathan, "How efficient can memory checking be?" in *Theory of Cryptography Conference*. Springer, 2009, pp. 503–520.

[43] V. H. Adams, "Fixed-point arithmetic," 2021, link.

[44] T. Datta, B. Chen, and D. Boneh, "VerITAS: Verifying image transformations at scale," IEEE S& P 2025, 2024. [Online]. Available: https://eprint.iacr.org/2024/1066

[45] S. Garg, A. Goel, S. Jha, S. Mahloujifar, M. Mahmoody, G.-V. Policharla, and M. Wang, "Experimenting with zero-knowledge proofs of training," Cryptology ePrint Archive, Paper 2023/1345, 2023. [Online]. Available: https://eprint.iacr.org/2023/1345

[46] G. Tan, A. Gascón, S. Meiklejohn, M. Raykova, X. Wang, and N. Luo, "Founding zero-knowledge proofs of training on optimum vicinity," Cryptology ePrint Archive, Paper 2025/053, 2025. [Online]. Available: https://eprint.iacr.org/2025/053

[47] S. Angel, A. J. Blumberg, E. Ioannidis, and J. Woods, "Efficient representation of numerical optimization problems for SNARKs," Cryptology ePrint Archive, Paper 2021/1436, 2021. [Online]. Available: https://eprint.iacr.org/2021/1436

[48] A. S. Hinrichs, D. Karolchik, R. Baertsch, G. P. Barber, G. Bejerano, H. Clawson, M. Diekhans, T. S. Furey, R. A. Harte, F. Hsu *et al.*, "The UCSC genome browser database: update 2006," *Nucleic Acids Research*, vol. 34, no. suppl_1, pp. D590–D598, 2006.

[49] A. Shumate and S. L. Salzberg, "Liftoff: accurate mapping of gene annotations," *Bioinformatics*, vol. 37, no. 12, pp. 1639–1643, 2021.

[50] R. M. Kuhn, D. Haussler, and W. J. Kent, "The UCSC genome browser and associated tools," *Briefings in bioinformatics*, vol. 14, no. 2, pp. 144–161, 2013.

[51] H. Zhao, Z. Sun, J. Wang, H. Huang, J.-P. Kocher, and L. Wang, "CrossMap: a versatile tool for coordinate conversion between genome assemblies," *Bioinformatics*, vol. 30, no. 7, pp. 1006–1007, 2014.

[52] L. A. Urry, M. L. Cain, S. A. Wasserman, P. V. Minorsky, and R. B. Orr, *Campbell Biology*, 12th ed. Pearson, 2020.

[53] M. Haeussler, K. Schönig, H. Eckert, A. Eschstruth, J. Mianné, J.-B. Renaud, S. Schneider-Maunoury, A. Shkumatava, L. Teboul, J. Kent *et al.*, "Evaluation of off-target and on-target scoring algorithms and integration into the guide RNA selection tool CRISPOR," *Genome biology*, vol. 17, no. 1, p. 148, 2016.

[54] Polygon, "Plonky3: A toolkit for polynomial iops (piops)," https://github.com/Plonky3/Plonky3, 2025.

[55] AztecProtocol, "barretenberg: An optimized elliptic-curve library and PLONK SNARK prover," https://github.com/AztecProtocol/barretenberg, 2025. [Online]. Available: https://github.com/AztecProtocol/barretenberg

[56] G. Fuchs, Y. Voichek, S. Benjamin, S. Gilad, I. Amit, and M. Oren, "4sUDRB-seq: measuring genomewide transcriptional elongation rates and initiation frequencies within cells," *Genome Biology*, vol. 15, no. 5, p. R69, 2014. [Online]. Available: https://doi.org/10.1186/gb-2014-15-5-r69

[57] J. Gao, Y. Palanikuma, D. Mouris, D. T. Nguyen, and N. Trieu, "SecurED: Secure multiparty edit distance for genomic sequences," Cryptology ePrint Archive, Paper 2025/500, 2025. [Online]. Available: https://eprint.iacr.org/2025/500

[58] M. Szydlowski and P. Paczyńska, "QTLMAS 2010: simulated dataset," in *BMC proceedings*, vol. 5, no. Suppl 3. Springer, 2011, p. S3.

[59] M. Szydlowski and P. Paczynska, "QTL-MAS 2010 simulated dataset," https://jay.up.poznan.pl/qtlmas2010/dataset.html, 2010, accessed: 2026-05-20.

[60] O. Kohannim, D. Hibar, J. Stein, N. Jahansha, X. Hua, P. Rajagopalan, A. Toga, C. Jack, M. Weiner, G. {de Zubicaray}, K. McMahon, N. Hansell, N. Martin, M. Wright, and P. Thompson, "Discovery and replication of gene influences on brain structure using LASSO regression," *Frontiers in Neuroscience*, no. AUG, 2012.

[61] M. B. van Egmond, G. Spini, O. van der Galien, A. IJpma, T. Veugen, W. Kraaij, A. Sangers, T. Rooijakkers, P. Langenkamp, B. Kamphorst, N. van de L'Isle, and M. Kooij-Janic, "Privacy-preserving dataset combination and Lasso regression for healthcare predictions," *BMC Medical Informatics and Decision Making*, vol. 21, no. 1, p. 266, 2021. [Online]. Available: https://doi.org/10.1186/s12911-021-01582-y

[62] S. Srivastava and L. Chen, "Comparison between the stochastic search variable selection and the least absolute shrinkage and selection operator for genome-wide association studies of rheumatoid arthritis,"

BMC Proceedings, vol. 3, no. Suppl 7, p. S21, 2009. [Online]. Available: https://doi.org/10.1186/1753-6561-3-S7-S21

[63] J. B. Fontanarosa and Y. Dai, "Using LASSO regression to detect predictive aggregate effects in genetic studies," *BMC Proceedings*, vol. 5, no. 9, p. S69, 2011. [Online]. Available: https://doi.org/10.1186/1753-6561-5-S9-S69

[64] J. Richland, T. Kiiskinen, W. Wang, S. Lu, B. Narasimhan, T. Hastie, M. Rivas, and R. Tibshirani, "Univariate-guided sparse regression for biobank-scale high-dimensional omics data," 2026. [Online]. Available: https://arxiv.org/abs/2511.22049

[65] A. Halimi, L. Dervishi, E. Ayday, A. Pyrgelis, J. R. Troncoso-Pastoriza, J.-P. Hubaux, X. Jiang, and J. Vaidya, "Privacy-preserving and efficient verification of the outcome in genome-wide association studies," *PETS 2022*, vol. 2022, no. 3, pp. 732–753, 2022.

[66] V. A. Schneider, T. Graves-Lindsay, K. Howe, N. Bouk, H.-C. Chen, P. A. Kitts, T. D. Murphy, K. D. Pruitt, F. Thibaud-Nissen, D. Albracht *et al.*, "Evaluation of GRCh38 and de novo haploid genome assemblies demonstrates the enduring quality of the reference assembly," *Genome Research*, vol. 27, no. 5, pp. 849–864, 2017.

[67] NCBI RefSeq Assembly, "Homo sapiens genome assembly GRCh38.p14," Accession GCF_000001405.40, 2022, accessed: 2026-02-01. [Online]. Available: https://ftp.ncbi.nlm.nih.gov/genomes/all/GCF/000/001/405/GCF_000001405.40_GRCh38.p14/

[68] V. M. Ingram, "Gene mutations in human haemoglobin: The chemical difference between normal and sickle cell haemoglobin," *Nature*, vol. 180, no. 4581, pp. 326–328, 1957.

[69] S. L. Thein, "The molecular basis of β-thalassemia," *Cold Spring Harbor Perspectives in Medicine*, vol. 3, no. 5, p. a011700, 2013.

[70] H. Frangoul, D. Altshuler, M. D. Cappellini, Y.-S. Chen, J. Domm, B. K. Eustace, J. Foell, J. de la Fuente, S. Grupp, R. Handgretinger, T. W. Ho, A. Kattamis, A. Kernytsky, J. Lekstrom-Himes, A. M. Li, F. Locatelli, M. Y. Mapara, M. de Montalembert, D. Rondelli, A. Sharma, S. Sheth, S. Soni, M. H. Steinberg, D. Wall, A. Yen, and S. Corbacioglu, "CRISPR-Cas9 gene editing for sickle cell disease and β-thalassemia," *The New England Journal of Medicine*, vol. 384, no. 3, pp. 252–260, 2021.

[71] J. E. DiCarlo, J. E. Norville, P. Mali, X. Rios, J. Aach, and G. M. Church, "Genome engineering in Saccharomyces cerevisiae using CRISPR-Cas systems," *Nucleic Acids Research*, vol. 41, no. 7, pp. 4336–4343, apr 2013.

[72] "iDASH Privacy & Security Workshop 2025 - Secure Genome Analysis Competition," https://humangenomeprivacy.org/2025/, National Institutes of Health, 2025.

[73] M. Blatt, A. Gusev, Y. Polyakov, and S. Goldwasser, "Secure large-scale genome-wide association studies using homomorphic encryption," Cryptology ePrint Archive, Paper 2020/563, 2020. [Online]. Available: https://eprint.iacr.org/2020/563

[74] M. Kim and K. Lauter, "Private genome analysis through homomorphic encryption," Cryptology ePrint Archive, Paper 2015/965, 2015. [Online]. Available: https://eprint.iacr.org/2015/965

[75] M. Blatt, A. Gusev, Y. Polyakov, and S. Goldwasser, "Secure large-scale genome-wide association studies using homomorphic encryption," Proceedings of the National Academies of Sciences, 2020. [Online]. Available: https://eprint.iacr.org/2020/563

[76] M. Blatt, A. Gusev, Y. Polyakov, K. Rohloff, and V. Vaikuntanathan, "Optimized homomorphic encryption solution for secure genome-wide association studies," BMC Medical Genomics, 2019. [Online]. Available: https://eprint.iacr.org/2019/223

[77] S. Carpov, N. Gama, M. Georgieva, and J. R. Troncoso-Pastoriza, "Privacy-preserving semi-parallel logistic regression training with fully homomorphic encryption," *BMC Medical Genomics*, vol. 13, no. Suppl 7, p. 88, 2020. [Online]. Available: https://doi.org/10.1186/s12920-020-0723-0

[78] H. Chen, R. Gilad-Bachrach, K. Han, Z. Huang, A. Jalali, K. Laine, and K. Lauter, "Logistic regression over encrypted data from fully homomorphic encryption," BMC Medical Genomics, 2018. [Online]. Available: https://eprint.iacr.org/2018/462

[79] S. Hwang, E. Ozturk, and G. Tsudik, "Balancing security and privacy in genomic range queries," *ACM Trans. Priv. Secur.*, vol. 26, no. 3, Mar. 2023. [Online]. Available: https://doi.org/10.1145/3575796

[80] W. Legiest, J.-P. D'Anvers, B. Spasic, N.-L. Tran, and I. Verbauwhede, "Leuvenshtein: Efficient FHE-based edit distance computation with single bootstrap per cell," USENIX Security 2025, 2025. [Online]. Available: https://eprint.iacr.org/2025/012

[81] 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*, ser. CCS '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 691–702. [Online]. Available: https://doi.org/10.1145/2046707.2046785

[82] S. Madabushi and P. Ramanathan, "Two-cloud private read alignment to a public reference genome," *Proceedings on Privacy Enhancing Technologies*, vol. 2023, no. 2, pp. 449–463, 2023. [Online]. Available: https://petsymposium.org/popets/2023/popets-2023-0062.php

[83] H. D. V. Madrigal, D. C. Jaramillo, and D. F. Aranha, "Privacy-preserving edit distance computation using secret-sharing two-party computation," LATINCRYPT 2023, 2023. [Online]. Available: https://eprint.iacr.org/2023/1201

[84] C. Hong, Z. Huang, W. jie Lu, H. Qu, L. Ma, M. Dahl, and J. Mancuso, "Privacy-preserving collaborative machine learning on genomic data using TensorFlow," Cryptology ePrint Archive, Paper 2020/159, 2020. [Online]. Available: https://eprint.iacr.org/2020/159

[85] K. Hamacher, T. Kussel, T. Schneider, and O. Tkachenko, "PEA: Practical private epistasis analysis using MPC," ESORICS 2022, 2022. [Online]. Available: https://eprint.iacr.org/2022/1185

[86] S. Pentyala, D. Railsback, R. Maia, R. Dowsley, D. Melanson, A. C. A. Nascimento, and M. D. Cock, "Training differentially private models with secure multiparty computation," *CoRR*, vol. abs/2202.02625, 2022. [Online]. Available: https://arxiv.org/abs/2202.02625

[87] M. Akgün, N. Pfeifer, and O. Kohlbacher, "Efficient privacy-preserving whole-genome variant queries," *Bioinformatics*, vol. 38, no. 8, pp. 2202–2210, 02 2022. [Online]. Available: https://doi.org/10.1093/bioinformatics/btac070

[88] G. Asharov, S. Halevi, Y. Lindell, and T. Rabin, "Privacy-preserving search of similar patients in genomic data," *Proceedings on Privacy Enhancing Technologies*, vol. 2018, no. 4, pp. 104–124, 2018. [Online]. Available: https://petsymposium.org/popets/2018/popets-2018-0034.php

[89] T. Schneider and O. Tkachenko, "EPISODE: Efficient privacy-PreservIng similar sequence queries on outsourced genomic DatabasEs," ASIACCS 2019, 2021. [Online]. Available: https://eprint.iacr.org/2021/029

[90] C. Liu, X. S. Wang, K. Nayak, Y. Huang, and E. Shi, "(oblivm): A programming framework for secure computation," in *2015 IEEE Symposium on Security and Privacy*, 2015, pp. 359–376.

[91] A. Mandal, J. C. Mitchell, H. Montgomery, and A. Roy, "Data oblivious genome variants search on intel SGX," Cryptology ePrint Archive, Paper 2018/732, 2018. [Online]. Available: https://eprint.iacr.org/2018/732

[92] C. Nuoskala, H. Abdinasibfar, and A. Michalas, "SPADE: Digging into selective and PArtial DEcryption using functional encryption," Cryptology ePrint Archive, Paper 2024/1387, 2024. [Online]. Available: https://eprint.iacr.org/2024/1387

[93] B. Rogers, S. Chhabra, M. Prvulovic, and Y. Solihin, "Using address independent seed encryption and bonsai merkle trees to make secure processors OS- and performance-friendly," in *40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2007)*, 2007, pp. 183–196.

[94] Zcash, "Zcash: Secure decentralized cryptocurrency," https://github.com/zcash/zcash, 2016.

[95] zk-ml, "linear-regression-demo: Private quantized linear regression demo using zk-snarks on ethereum," https://github.com/zk-ml/linear-regression-demo, 2022.

[96] X. Wang, X. Jiang, and J. Vaidya, "Efficient verification for outsourced genome-wide association studies," *Journal of Biomedical Informatics*, vol. 117, p. 103714, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1532046421000435

[97] S. Mueller, "On DNA signatures, their dual-use potential for GMO counterfeiting, and a cyber-based security solution," *Frontiers in Bioengineering and Biotechnology*, vol. 7, p. 189, 2019. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fbioe.2019.00189

[98] C. Li, W. Chu, R. A. Gill, S. Sang, Y. Shi, X. Hu, Y. Yang, Q. U. Zaman, and B. Zhang, "Computational tools and resources for CRISPR/Cas genome editing," *Genomics, proteomics & bioinformatics*, vol. 21, no. 1, pp. 108–126, 2023.

[99] L. Hoberecht, P. Perampalam, A. Lun, and J.-P. Fortin, "A comprehensive Bioconductor ecosystem for the design of CRISPR guide RNAs across nucleases and technologies," *Nature Communications*, vol. 13, no. 1, p. 6568, 2022.

[100] A. Frolov, "Zero knowledge memory-checking techniques for stacks and queues," Cryptology ePrint Archive, Paper 2024/2084, 2024. [Online]. Available: https://eprint.iacr.org/2024/2084

[101] L. Wang and T. Jiang, "On the complexity of multiple sequence alignment," *Journal of computational biology*, vol. 1, no. 4, pp. 337–348, 1994.

[102] K. Bringmann, A. Cassis, and N. Fischer, "Negative-weight single-source shortest paths in near-linear time: Now faster!" in *2023 IEEE 64th Annual Symposium on Foundations of Computer Science (FOCS)*, 2023, pp. 515–538.

[103] A. Bernstein, D. Nanongkai, and C. Wulff-Nilsen, "Negative-weight single-source shortest paths in near-linear time," *Commun. ACM*, vol. 68, no. 2, p. 87–94, Jan. 2025. [Online]. Available: https://doi.org/10.1145/3631536

[104] V. Guruswami, A. Rudra, and M. Sudan, *Essential Coding Theory*, 2025, draft published online, latest version dated August 26, 2025. [Online]. Available: https://cse.buffalo.edu/faculty/atri/courses/coding-theory/book/

[105] Noir Lang Documentation, "Explainer: Writing noir," https://noir-lang.org/docs/explainers/explainer-writing-noir, 2026, accessed: 2026-01-11, version v1.0.0-beta.18, technical documentation.

[106] A. Frolov, J. Guo, X. Zhao, T. Datta, D. Boneh, and I. Miers, "zk-cinema: Proving video provenance in zero knowledge," Cryptology ePrint Archive, Paper 2026/1598, 2026. [Online]. Available: https://eprint.iacr.org/2026/1598

[107] A. Ozdemir, E. Laufer, and D. Boneh, "Volatile and persistent memory for zkSNARKs via algebraic interactive proofs," Cryptology ePrint Archive, Paper 2024/979, 2024. [Online]. Available: https://eprint.iacr.org/2024/979

[108] M. Campanelli, D. Fiore, and A. Querol, "LegoSNARK: Modular design and composition of succinct zero-knowledge proofs," Cryptology ePrint Archive, Paper 2019/142, 2019. [Online]. Available: https://eprint.iacr.org/2019/142

A Deferred figures

In Figure 5, we sketch a Merkle Tree with an example inclusion proof highlighted.

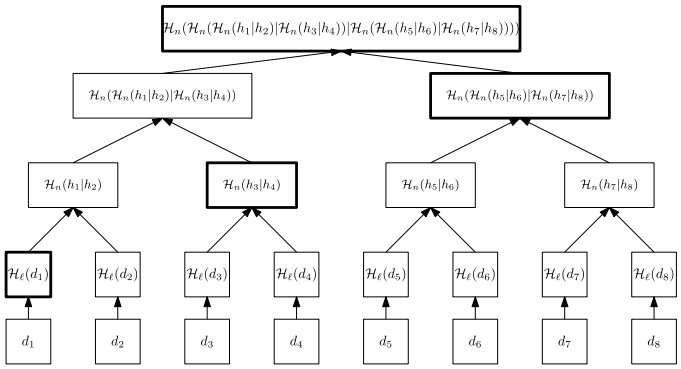

Figure 5: A Merkle Tree over 8 elements with the hashes required to create an inclusion proof for d_2 highlighted. For succinctness, we write h_i instead of $\mathcal{H}_\ell(d_i)$.

B Optimizing zk-RAM schemes

To efficiently verify an alignment between two sequences in linear time in a zk-SNARK circuit, it must be possible to access entries of both sequences at data-dependent indices in sublinear time. In a computational model like FHE or MPC, this is not possible in general, but with the combination of randomness and precomputed advice available in zk-SNARKs, existing "zk-RAM" schemes like [6] allow for SNARK circuits to achieve constant time accesses.

The algorithm of Section 4.1 could be implemented using a standard zk-RAM scheme, and be reasonably efficient (for instance, the scheme of [6] could be implemented at a cost of 10 constraints per access in a zk-SNARK using R1CS arithmetization[9]).

However, we can optimize standard zk-RAM schemes for this application domain. We implemented such an optimized scheme in our experiments. Observe that the access pattern of the algorithm in Section 4.1 has heavily structured accesses to the sequences $\mathsf{seq}_1, \mathsf{seq}_2$. In a successful execution of VerifyAlignment, each entry of seq_1 and seq_2 is accessed exactly once. Furthermore, the access indices to both sequences are strictly increasing. [100] builds optimized stack and queue data structures for zero-knowledge proofs. We can implement the accesses to $\mathsf{seq}_1, \mathsf{seq}_2$ by building a queue for each sequence, inserting the characters of each sequence into the corresponding queue, and then conditionally removing an entry of each sequence on each loop iteration. An unconditional queue enqueue operation costs 1 constraint for the queue data structure in [100], while a conditional queue dequeue operation costs 3 constraints per operation, meaning this adds a cost of 4 constraints per character of the CIGAR string in the circuit, a substantial optimization over a compa-

[9]The authors of [6] quote a per-access cost of 6 multiplications with private inputs, though their cost model is specific to their VOLE setting, and the per-access costs are higher in the setting of zk-SNARKs with fixed circuits.

rable generic zk-RAM data structure.

For the overall system, we note that this is a relatively small optimization, and gives a savings on the order of 10% in circuit size, since other parts of the circuit for verifying an alignment have higher costs.

To briefly describe how the optimized queue for zk-SNARKs of [100] works, it uses the "vector" or "coefficient" universal hash function of [36]. Consider 2 vectors $u, v \in \mathbb{F}^n$ of field elements. Define the polynomials $f_u(x) = u_0 + u_1 x + \ldots + u_{n-1}x^{n-1}, f_v(x) = v_0 + v_1 + \ldots + v_{n-1}x^{n-1}$. Given a random point x', $f_u(x') = f_v(x')$ when $u \neq v$ with probability at most $\frac{n-1}{|\mathbb{F}|}$ by the Schwarz-Zippel lemma.

This hash function can be used to verify accesses to a queue in a SNARK circuit efficiently. For a queue, the ordered vector of enqueued values enqueue must be equal to the vector of dequeued values dequeue, with all dequeued values removed from the queue strictly after being added. The equality of these vectors can be verified using the universal hash function mentioned previously. The vector hash of "enqueues"/"dequeues" can be evaluated in one constraint per entry using Horner's rule. Furthermore, maintaining a counter that checks that the queue is non-empty before removing a value requires 1 constraint per dequeue operation, and conditionally performing operations requires 1 constraint for both types of operations. This data structure is described in detail in Section 3 of [100].

C Verifying multiple sequence alignment and alignments with clipping

C.1 Multiple Sequence Alignment

Multiple sequence alignment is a variant of sequence alignment in which multiple sequences are simultaneously aligned against each other. In a multiple sequence alignment problem for m strings of n characters each, the goal is to determine a sequence of gaps to insert into each of the m strings such that an alignment score is minimized. There are multiple ways to score an instance of Multiple Sequence Alignment. We describe a simple, common variant called "sum-of-pairs alignment". In sum-of-pairs alignment, the cost of an alignment of m sequences is computed by summing the alignment cost for each possible pair of sequences. Concretely, for each of the $\binom{m}{2}$ sequences, the cost of alignment is computed by adding a fixed cost for each mismatching character (i.e. when two characters do not agree, or when one sequence has a gap and the other does not).

We briefly summarize how VerifyAlignment from Section 4.1 and the format of cigar would need to be modified to support multiple sequence alignment. A variant of VerifyAlignment for multiple sequence alignment would take in m strings, as well as m strings of $O(n)$ characters specifying where gaps should be inserted into each string (for example, such strings could be sequences of Boolean values specifying where to insert gaps). Next, VerifyAlignment would need to iterate over all $\binom{m}{2}$ possible pairs of sequences. For each sequence, it would perform a similar linear pass to that of Section 4.1 to determine the amount this pair of sequences contributes to the alignment cost. The linear pass should compute the alignment cost with gaps inserted as specified by the cigar string. This computation can be implemented in linear time by iterating through both sequences.

In total, this variant of VerifyAlignment takes $O(m^2n)$ time to verify an instance of multiple sequence alignment, when alignments are well-conditioned (i.e. only $O(n)$ gaps are inserted), and this is implemented in a zk-SNARK using memory-checking techniques with constant per-access costs. In comparison, the approach of [29] would take $O(m^2n^3)$ time to verify an alignment because of their usage of a memory scheme with linear per-access costs. With no restriction on the size of cigar, the running times of these algorithms would be $O(m^3n)$ and $O(m^4n^2)$ respectively. Extending the discussion of Section 4.3, Multiple Sequence Alignment with sum-of-pairs costs is an NP-complete problem in general [101].

C.2 Clipping

In addition to semiglobal and local alignment, one other common aspect of alignments as they are used in genomics is *soft clipping* and *hard clipping*, as in the CIGAR string standard [28]. Clipping means that a character in one of the two sequences being aligned should be ignored for the purposes of alignment. One common cause for these is that a sequenced read may sometimes contain an "adapter" (a technical sequence, unrelated to the underlying biological sample, that should be removed prior to alignment but which is sometimes incompletely handled). Hard clipping specifies that nucleotides were actually removed from a string prior to alignment, while soft clipping specifies that the nucleotide was retained, but it should not be included as either a match or a mismatch in the alignment scores (sometimes, softclipped bases are given a score strictly less than the mismatch cost). Adding characters for soft-clipping and hard-clipping to the modified CIGAR strings for the previous section's algorithms enables this. Specifically, we can add cases to the inner loop of VerifyAlignment for the characters 'S' and 'H' for soft and hard clipping respectively. These cases would advance index i_1 or i_2 by 1, and not change the alignment cost (for hard clipping) or change the alignment cost concordant with the soft clipping penalty for the soft clipping case. VerifyAlignment should also check that soft/hard clipping characters appear in contiguous locations at allowed locations.

D Verifying shortest paths

We continue our discussion of how it is possible to verify solutions to common dynamic programs more efficiently than

computing an optimal solutions directly. As before, we discuss verifying the validity/cost of a solution rather than its optimality. We now discuss negative weight, single source shortest paths, the problem solved by the Bellman-Ford algorithm.

To formally specify the shortest paths problem, the problem takes a graph $G = (V, E)$ with edges costs $c : E \to \mathbb{R}$, such that there are not negative cost cycles in the graph. A shortest path in the graph is a sequence of edges. Verifying that a path has a certain cost can be done in $O(|E|)$ time. Specifically, the path P has at most E edges (since otherwise, it would necessarily have a cycle, and would not be the minimum cost path). In checking that the path is valid, a verifier must check that each edge is in the graph, that the edges form a continuous path, compute the path cost, and check that the start and ends of a path are at the desired indices. This can be done in $O(|E|)$ time assuming the graph G and edge cost function c is stored using proper random-access data structures. $O(|E|)$ running time is strictly less than the $O(|V||E|)$ time it takes to execute the Bellman-Ford algorithm directly.

When comparing to the fastest known algorithms for Negative-Weight Single-Source Shortest Paths, which is the algorithmic problem solved by the Bellman-Ford algorithm, the fastest known algorithm in the literature takes $O(|E|\log^2(|V|)\log(|V|W)\log\log|V|)$ time (from [102], which improves on a breakthrough in [103]), where W is the magnitude of the largest negative edge weight in the graph. Thus, while this "verification" approach is strictly asymptotically faster than the fastest algorithm, the gap between verifying a solution and directly computing an optimal solution is, interestingly, much smaller for single source shortest paths than for parsing a CFG or computing a sequence alignment. We also note that the fastest algorithm in the theory literature that we cited is likely not practical to implement in practice.

E Extended Merkle tree results

E.1 Proof of optimality for Split Merkle trees

We analyze the optimality of our Split Merkle tree construction in the random oracle model. Formally, the theorem we show is:

Theorem E.1. *Consider a complete binary Merkle tree* tree *with distinct leaves that uses hash functions H_ℓ, H_n which output λ bits. Let $n(\lambda)$ be the number of leaves in* tree *and assume that $n^2/2^\lambda$ is negligible.*[10] *Consider a pair of deterministic algorithms* Index, Prove. Index *is an algorithm which runs in polynomial time, and generates an advice string of size $T(n)$ given* tree *and* leaves. Prove *is an algorithm which runs in time $O(\log n)$ and has access to the advice string generated by* Index, leaves*, and a designated leaf. If H_ℓ, H_n are modeled as random oracles, and* Prove *can correctly generate Merkle inclusion proofs for all leaves of* tree *with high probability ($\Omega(1)$), with probability taken over the choice of random oracle, then $T(n)$ cannot be $o(\lambda\frac{n}{\log n})$.*

At a high level, we use a variant of a "compression argument" (inspired by [41, 42]), and show that any correct pair of algorithms Index, Prove which use asymptotically less than $\lambda\frac{n}{\log n}$ space for the advice string would lead to a coding scheme (equivalently, a compression algorithm) whose efficiency contradicts Shannon's source coding theorem [104]. Note that for notational convenience, we assume that evaluating H_ℓ, H_n takes $O(1)$ time, as does reading values of hashes from storage and returning them as output. Note that n, the number of leaves in the tree, is a function of λ, and all asymptotic statements in this proof are in terms of λ. For notational convenience, and following the conventions of the prior work [41], we sometimes do not explicitly write n's dependence on λ.[11]

Assume toward a contradiction that $T(n) = o(\lambda\frac{n}{\log n})$. Let c, λ_0' be constants such that $c\log n(\lambda)$ is greater than the worst-case running time in steps of Prove for any leaf in any tree when $\lambda > \lambda_0'$. Furthermore, let λ_0'' be a value such that $T(n(\lambda)) \leq \lambda\frac{n(\lambda)}{32c\log(n(\lambda))}$ for all $\lambda > \lambda_0''$, and let $\lambda_0 = \max(\lambda_0', \lambda_0'')$. Consider a Merkle tree tree with $\lambda > \lambda_0$, and $n = n(\lambda)$ leaves (slightly overloading notation).

Consider the height of a node in a complete binary Merkle tree such that it has at least $c\log(n)$ descendants. A node at height $\lceil \log c\log(n)\rceil$ has at least $c\log(n)$ descendants. Consider the set of nodes in the Merkle tree at height $\lceil \log c\log(n)\rceil + 2$, which necessarily have at least $4c\log(n)$ descendants. There are at least:

$$2^{\log n - \lceil \log c\log(n)\rceil - 2} = \frac{n}{2^{\lceil \log c\log(n)\rceil + 2}} \geq \frac{n}{2^{\log c\log(n)+3}} = \frac{n}{8c\log n}$$

nodes at this height.

Now, for each node in tree at height $\lceil \log c\log n\rceil + 2$, choose a random leaf which is a descendant of that node, forming the set $\mathsf{r}_{\mathsf{leaves}}$. $\mathsf{r}_{\mathsf{leaves}}$ has at least $\frac{n}{8cT(n)}$ leaves. Now, consider running Prove to create a Merkle inclusion proof for each leaf in $\mathsf{r}_{\mathsf{leaves}}$, creating a set of Merkle inclusion proofs which we call proofs.

Given the structure of Merkle inclusion proofs, proofs must contain the hash of the left or right descendant of each node at height $\lceil \log c\log n\rceil + 2$. There are at least $\frac{n}{8cT(n)}$ such hashes. Call the set of node hashes at height $\lceil \log c\log n\rceil + 1$ which are included in the Merkle proofs for $\mathsf{r}_{\mathsf{leaves}}$ $\mathsf{r}_{\mathsf{desc}}$.

We now reason about the set of queries made to the random oracles for H_ℓ, H_n made by Prove in all executions required

[10] We sometimes suppress the dependence of n on λ for notational convenience. We explain this in the next paragraph.

[11] See, for instance, the proof of Lemma 1 in [41], where n's dependence on the security parameter is suppressed after the theorem statement

to generate proofs. For notational convenience, we reinterpret $H_n : \{0,1\}^{2\lambda} \to \{0,1\}^{\lambda}$ as $H_n : \{0,1\}^{\lambda} \times \{0,1\}^{\lambda} \to \{0,1\}^{\lambda}$ in the natural way.

First, consider the probability that the random oracles for H_n, H_ℓ have a hash collision among the set of internal nodes of tree, queries made by Index and the queries made to the oracles when executing Prove to generate proofs. By assumption, the leaves of the tree are distinct and the probability of a collision among this polynomially-sized set of hash evaluations is negligible, so we assume that there are no hash collisions in the relevant set of random oracle evaluations used by Prove in generating proofs for the rest of this proof.

Let hash_1 be the set of hash values in $\mathsf{r}_{\mathsf{desc}}$ which are not the result of a random oracle query made by Prove while generating proofs.

Next, consider $\mathsf{r}_{\mathsf{desc}} \setminus \mathsf{hash}_1$. Necessarily, each hash in this set is the result of a random oracle query made by Prove while generating proofs. Given a hash $h_d \in \mathsf{r}_{\mathsf{desc}} \setminus \mathsf{hash}_1$, let it be the result of oracle query $H_n(h_{\mathsf{left}}, h_{\mathsf{right}}) = h_d$. As mentioned previously, we assume H_ℓ, H_n have no hash collisions, so $h_{\mathsf{left}}, h_{\mathsf{right}}$ are either the result of a random oracle query to H_ℓ or H_n, or were generated by Prove.

Continuing this logic, the node with hash h_d has at least $4c\log n$ descendants because of its height, which is strictly greater than $c\log n$, the bound on the number of steps taken by Prove. Given that h_d only appears in one proof in proofs, one specific execution of Prove must output h_d. Consequently, for some node which is a descendant of h_d (call its hash $h_{d'}$), there must be a random oracle query where $H_\ell(x) = h_{d'}$ or $H_n(y,z) = h_{d'}$, but x or one of y or z respectively are not the result of a query to the random oracles for H_ℓ, H_n made by the relevant instance of Prove. Thus, each execution of Prove must generate the hash of at least one descendant of h_d which is not the output of a random oracle query made by that execution (since there are more hashes that are descendants of h_d than execution steps of Prove). Let hash_2 be the set of hashes of descendants of entries of $\mathsf{r}_{\mathsf{desc}} \setminus \mathsf{hash}_1$ which are produced by Prove, but are not the outputs of random oracle queries made by Prove.

Now, $\mathsf{hash}_1 \cup \mathsf{hash}_2$ has at least $\frac{n}{8c\log n}$ distinct elements, since we assumed there are no hash collisions, and there is at one such hash in the set for each member of $\mathsf{r}_{\mathsf{desc}}$.

Now, we argue that Index, Prove imply a scheme for compressing random data whose efficiency contradicts Shannon's source coding theorem. Repeated evaluations of Prove must be able to output the values of at least $\frac{n}{8c\log n}$ outputs of the random oracles H_ℓ, H_n ($\lambda\frac{n}{8c\log n}$ bits) with high probability, based on at most $\lambda\frac{n}{32c\log n}$ storage space and unrelated evaluations of the random oracle. It remains to apply Shannon's Source Coding theorem [104] to imply a contradiction.

There is one more remaining complexity: elements of $\mathsf{hash}_1 \cup \mathsf{hash}_2$ are chosen from a set of $4c\log n$ hashes according to some deterministic function, so we bound the Shannon entropy of each entry of $\mathsf{hash}_1 \cup \mathsf{hash}_2$.

Specifically, we bound the entropy of the following random variables: X, I. X is computed by an arbitrary deterministic function f, which takes n uniformly random blocks of m bits $B_1, B_2, .., B_n$, and according to f, outputs one of the blocks. Let I be the index of the block chosen. Now, where H is Shannon entropy [104], $H(B_1, B_2, ..., B_n) = nm$, since these are nm uniformly random bits. $nm = H(B_1, B_2, ..., B_n) = H(B_1, B_2, ..., B_n, I, X)$ because I, X are computed from a deterministic function of the blocks. Next, by the chain rule for Shannon entropy, $H(B_1, B_2, ..., B_n, I, X) = H(I, X) + H(B_1, ..., B_n | I, X)$. $H(B_1, B_2, ..., B_n | I, X) \leq (n-1)m$ because I, X fully determine one of the blocks. Substituting this inequality into the previous equations, we get $nm \leq H(I,X) + (n-1)m$, implying $m \leq H(I,X)$. Finally, $m \leq H(I,X) = H(X) + H(I|X) \leq H(X) + H(I) \leq H(X) + \log n$. These intermediate steps are justified using standard properties of Shannon entropy. This implies that $m - \log n \leq H(x)$. Now, considering our situation, each hash compressed by Index, Prove is a block of λ uniformly random bits chosen from a set of $4c\log n$ blocks, which have at least $\lambda - \log 4c\log n$ entropy.

Now, applying Shannon's theorem, Index and repeated applications of Prove imply deterministic functions for compressing at least $\frac{n}{8c\log n}$ i.i.d. random variables with entropy at least $\lambda - \log 4c\log n$ into less than $\lambda\frac{n}{32c\log n}$ bits. Now, the set of hashes chosen by this scheme have entropy at least $(\lambda - \log 4c\log n) * \frac{n}{8c\log n}$. Since $n^2/2^{\lambda}$ is negligible by assumption, $\lambda - \log 4c\log n < \frac{\lambda}{2}$. $(\lambda - \log 4c\log n) * \frac{n}{8c\log n} < \frac{n}{16c\log n}$ is a non-constant factor less than $\lambda\frac{n}{32c\log n}$ bits, contradicting Shannon's Source Coding Theorem.

The contradiction of Shannon's Source Coding Theorem completes the proof. We believe this theorem can be extended fairly directly to the case of average case complexity, but leave this to future work. We also assumed that the leaves of the tree are all unique to ensure that the tree's internal hashes are distinct. While this is a somewhat strong assumption, some variant of this assumption is required for this theorem, since the internal hashes of a Merkle tree with all identical leaves can trivially be stored in small space. We believe this assumption can be somewhat relaxed as well. Lastly, this can be fairly directly extended to a more general time-space tradeoff for storing Merkle trees (e.g. by allowing $O(\log^2 n)$ proof generation time but using proportionally less storage), though we leave this to future work.

E.2 Merkle trees with large leaves

The approach of Section 5.1 applies to Merkle Trees whose leaves have size $O(1)$. In this section, we make an observation about Merkle Trees with larger leaves. Namely, for a Merkle Tree over a vector of $O(n)$ data points split into leaves of size $\Theta(\log n)$, the Merkle Tree has $O(\frac{n}{\log n})$ leaves. Consequently, the Merkle Tree has $O(\frac{n}{\log n})$ internal hashes, which means that the hashes of the Merkle Tree can be stored

in $O(\frac{n}{\log n})$ additional space. Furthermore, it is still clearly possible to generate inclusion proofs in $O(\log n)$ time. Thus, in this parameter regime, when leaves have large sizes, it is also possible to have a "succinct data structure" construction for storing a Merkle tree.

F Extended GWAS discussion

F.1 Review of fixed point arithmetic

Fixed point arithmetic [43] is a common approach for optimizing computations that perform arithmetic on numbers with fractional components in zk-SNARKs. Field operations are directly used to emulate arithmetic with fractional components. We describe a variant of fixed point arithmetic that can represent negative integers up to a certain magnitude. Given a real number $x \in [-2^n, 2^n)$, fixed point arithmetic represents x as the integer $\lfloor x \cdot 2^n \rfloor + 2^{2n}$. This is said to be *n bits of precision*. When used in SNARKs, this value $\lfloor x \cdot 2^n \rfloor + 2^{2n}$ is further interpreted as a field element in the natural way.

To add two fixed-point numbers, they can simply be added together while subtracting 2^n from the sum. To multiply two fixed-point numbers, x, y, we multiply $(x - 2^{2n}) \cdot (y - 2^{2n})$, compute $(x - 2^n) \cdot (y - 2^n) \cdot 2^{-n}$ (as real numbers), and then return $\lfloor (x - 2^n) \cdot (y - 2^n) \cdot 2^{-n} \rfloor + 2^{2n}$ as an integer or field element. It can be easily checked that this new value is a fixed point approximation of xy.

A major concern for using fixed point arithmetic in zk-SNARKs is ensuring that this field arithmetic does not "wrap around", where the magnitudes of fixed point values exceed the field modulus, or that the fixed point numbers exceed their maximum magnitudes. Arithmetic wrap-around causes fixed point arithmetic to no longer approximate floating point arithmetic. When implementing our GWAS experiments, we ensured that this would not happen. We used 32 bits of precision, and allowed our integers to have a maximum integer magnitude of 32 bits. The field used by our Noir circuits is about 254 bits, so this was not an issue.

When implementing fixed point multiplication in SNARK circuits, there are a few simple optimizations to decrease the associated costs. The most expensive step in fixed point arithmetic is computing the "right shift" operations needed for multiplication which are represented by multiplication by 2^{-n}. This right shift step can be batched and deferred if computations take a certain structure. For instance, when computing a sum like $\sum_{i=0}^{m} |\beta_i|^2$, we can defer the "right shift" step to happen after the additions (assuming there is a guarantee of no overflow), meaning only one right shift step has to be performed, rather than m such steps.

Furthermore, right shifting can be accelerated via lookup tables. Specifically, when computing $x \cdot 2^{-n}$, the fastest way to implement this in a zk-SNARK is for a prover to provide values q, r satisfying $q * 2^n + r$, with $0 \leq r < 2^n$. The main expensive step in this computation is checking that r is in $[0, 2^n)$, which can be made faster using standard lookup table approaches. Noir [105] implements these optimizations. For range checking that a value x satisfies $0 \leq x \leq 2^n$ when 2^n is large, Noir decomposes the value x into limbs, and range checks the chunks of bits separately.

Optimizing the absolute value step for Lasso regression can be done similarly. In our representation, 0 is represented as 2^{2n}, so checking that a fixed point value x is non-negative involves checking that it is greater than 0, which involves computing $x - 2^{2n}$ and checking that this value is in the range $[0, 2^{2n})$. Again, this range check can be accelerated using lookup tables. To implement an absolute value computation for a value x, our circuits can take a fixed point $|x|$ as advice, check that $|x| - 2^{2n}$ is in $[0, 2^{2n})$, and that $x = |x|$ or $x + |x| = 2 * 2^{2n}$.

F.2 VerITAS authentication variant

To authenticate data in our GWAS experiments, we use a variant of the "polynomial commitment hash" from Section 4.3 of VerITAS [44]. The variant is described in greater depth in Section 7.1 of [106]. This authentication scheme is also related to ideas in [107]. We summarize it here.

At a high level, VerITAS uses a signed polynomial commitment to a polynomial representing authenticated data. VerITAS's approach uses details of the Plonk proof system to include the data represented by this polynomial commitment in a Plonk circuit. VerITAS also expects a proof's verifier to separately verify a signature on this signed polynomial commitment. This yields an approach for authenticating data in zk-SNARKs which is very low cost: it leads to no increase in circuit size, and mild increases in other costs. VerITAS contains an in-depth discussion of this approach.

The basic idea of VerITAS can be straightforwardly extended to arbitrary proof systems which support random challenges in their SNARK circuits,[12] At a high level, we augment a SNARK circuit to evaluate the polynomial representing a piece of data at a random point in-circuit, and have verifiers check a polynomial commitment opening proof to ensure that this random evaluation matches the authenticated polynomial commitment via an opening proof. This is secure by the Schwarz-Zippel lemma. Many proof systems, like Microsoft's Spartan2 [12] allow for circuits to use pseudorandom challenges based on the circuit's public instance and witness. Given a polynomial commitment com representing a vector v as polynomial f, the proof circuit for a zk-SNARK can sample a challenge x, evaluate $f(x)$, and output $f(x)$ as a public output. The prover then also creates an opening proof for com at point x. The verifier is then expected to check com, and that this evaluation agrees with the one produced by the

[12] For instance, circuits using the Groth16 proof system can have wires taking pseudorandom values derived from all circuit inputs by using "commit-and-prove" techniques [108]. Spartan supports a similar approach, which is implemented in the Spartan2 [12] library, for example (see this code location).

Sequence Length	Proof Time	Verification
2000bp	404ms	135 ms
4000bp	508ms	150 ms
8000bp	706ms	186 ms
16000bp	1109ms	195 ms
32000bp	2205ms	360 ms
64000bp	4236ms	488 ms

Table 5: Sequence length vs. proof system performance for affine gap cost variant of sequence alignment.

Sequence Len. $\times$ # of Sequences	Proof Time
2000bp $\times$ 2	372ms
2000bp $\times$ 4	1129ms
2000bp $\times$ 8	3289ms
4000bp $\times$ 2	486ms
4000bp $\times$ 4	1454ms
4000bp $\times$ 8	6043ms
8000bp $\times$ 2	1020ms
8000bp $\times$ 4	3580ms
8000bp $\times$ 8	11371ms

Table 6: Sequence length vs. proof system performance for our circuit for multiple sequence alignment.

circuit. This leads to only modest increases in circuit size, prover time, and verifier time.

G More experimental data

G.1 Performance evaluation for sequence alignment variants

In Table 5, we give performance numbers for the circuits that we implemented for sequence alignment with affine gap costs. As can be seen, the costs of these circuits is comparable to those for the basic sequence alignment variant, reflecting the similar circuit complexity.

In Table 6, we give performance numbers for the circuits we implemented for multiple sequence alignment. The costs scale quadratically with the number of sequences aligned and linearly in the length of the sequences, as expected.

G.2 Performance evaluation for Merkle Tree with large leaves

.

Leaf size	Tree Size	Witness Gen.	Verification
120bp	775.05MB	553ns	18.50µs
240bp	387.53MB	586ns	8.76µs
480bp	193.76MB	583ns	19.27µs
960bp	96.88MB	554ns	21.35µs
1,920bp	48.44MB	623ns	26.22µs
3,840bp	24.22MB	764ns	37.01µs
7,680bp	12.11MB	921ns	58.80µs
15,360bp	6.06MB	1.09µs	103.57µs
30,720bp	3.03MB	1.42µs	193.56µs
61,440bp	1.51MB	2.09µs	374.34µs
122,880bp	775.75kB	3.10µs	736.14µs
245,760bp	388.19kB	5.17µs	1.46ms
491,520bp	194.38kb	8.95µs	2.91ms
983,040bp	97.44kb	16.40µs	5.81ms
1,966,080bp	48.94kb	27.80µs	11.64ms
3,932,160bp	24.66kb	48.97µs	23.67ms

Table 7: Merkle Tree leaf size vs. performance. Witness Gen. is the time to generate a Merkle inclusion proof, and verification time is the time to verify such a proof in computation.

In Table 7, we give performance numbers for the variant of Merkle trees with large leaves discussed in Appendix E.2. The costs scale as expected, with the size of the Merkle Tree decreasing proportionately to the size of the leaves. Interestingly, the proof verification time first decreases and then increases as leaf size increases, because the length of the inclusion proof decreases, but a larger leaf must be hashed as part of verifying an inclusion proof. Witness generation time decreases with increasing leaf size, because fewer hashes must be read from memory to generate an inclusion proof.

G.3 More information about GWAS experiments

We briefly summarize the different existing genomics studies which we ran experiments of comparable size to.

- [60], a paper which uses Lasso to find variables which predict Alzheimer's disease. The largest regression performed is in 729 data points, with 291 variables.
- [61], a paper which uses MPC to perform a GWAS on real data held by health organizations in the Netherlands to find predictive factors of heart failure. The largest regression performed in the study is in 10,000 data points, with 30 variables each.

Segment Size	On-Target Segment Size	Proving Time	Verification Time
100bp	50bp	1.70s	11.2ms
1000bp	200bp	9.86s	11.2ms
1000bp	500bp	9.98s	11.3ms
2000bp	200bp	18.57s	11.3ms
2000bp	500bp	18.77s	11.5ms
2000bp	1000bp	19.29s	11.6ms

Table 8: Time to generate and verify a proof where on-target scoring was done on hardcoded indices and off-target elsewhere.

- [62], a paper which evaluates Lasso against a different type of regression. As an example data set, they perform a regression to find predictors of rheumatoid arthritis. The largest regression performed is a regression in 1000 variables for 867 patients.
- [63], a paper which evaluates Lasso regression to correlate different types of diseases with SNPs and demographic features. The largest regression is in 13,572 variables with 697 individuals.
- [58, 59], a dataset related to cattle breeding used to evaluate various GWAS approaches.

G.4 CRISPR computations with public gene locations

In Table G.4, we show proving and verification times for proofs of segments where on-target scoring was done on hardcoded indices and off-target elsewhere. These benchmarks show that keeping the ranges of the targeted gene of an individual private costs up to 3 times more in computation for the boundary segments. It may be possible to do so without loss of privacy for the individual, but requires separate proving and verification keys per hardcoded index.