UNIVERSIDADE DE LISBOA

FACULDADE DE CIÊNCIAS

Efficient, Dependable Storage of Human Genome Sequencing Data

"Documento Definitivo"

Doutoramento em Informática

Especialidade de Engenharia Informática

Vinicius Vielmo Cogo

Tese orientada por:

Prof. Doutor Alysson Neves Bessani

Documento especialmente elaborado para a obtenção do grau de doutor

2020

UNIVERSIDADE DE LISBOA

FACULDADE DE CIÊNCIAS

Ciências ULisboa

Efficient, Dependable Storage of Human Genome Sequencing Data

Doutoramento em Informática

Especialidade de Engenharia Informática

Vinicius Vielmo Cogo

Tese orientada por:
Prof. Doutor Alysson Neves Bessani

Documento especialmente elaborado para a obtenção do grau de doutor

Este trabalho foi financiado pela Fundação para a Ciência e a Tecnologia (FCT, Portugal), através da unidade de investigação LASIGE (UIDB/00408/2020 e UIDP/00408/2020) e do projecto IRCoC (PTDC/EEI-SCR/6970/2014), e pela Comissão Europeia, através dos projectos BiobankCloud (FP7-ICT-317871), SUPERCLOUD (H2020-ICT-643964) e DiSIEM (H2020-IA-700692).

2020

Abstract

The understanding of human genome impacts several areas of human life. Data from human genomes is massive because there are millions of samples to be sequenced, and each sequenced human genome may size hundreds of gigabytes. Human genomes are critical because they are extremely valuable to research and may provide hints on individuals' health status, identify their donors, or reveal information about donors' relatives. Their size and criticality, plus the amount of data being produced by medical and life-sciences institutions, require systems to scale while being secure, dependable, auditable, and affordable. Current storage infrastructures are too expensive to ignore cost efficiency in storing human genomes, and they lack the proper knowledge and mechanisms to protect the privacy of sample donors. This thesis proposes an efficient storage system for human genomes that medical and life-sciences institutions may trust and afford. It enhances traditional storage ecosystems with privacy-aware, data-reduction, and auditability techniques to enable the efficient, dependable use of multi-tenant infrastructures to store human genomes. Contributions from this thesis include (1) a study on the privacy-sensitivity of human genomes; (2) to detect genomes' privacy-sensitive portions systematically; (3) specialised data reduction algorithms for sequencing data; (4) an independent auditability scheme for secure dispersed storage; and (5) a complete storage pipeline that obtains reasonable privacy protection, security, and dependability guarantees at modest costs (e.g., less than $1/Genome/Year) by integrating the proposed mechanisms with appropriate storage configurations.

Keywords: Data Storage; Genomic Data; Privacy; Deduplication; Auditability

Resumo

A compreensão do genoma humano impacta várias áreas da vida. Os dados oriundos do genoma humano são enormes pois existem milhões de amostras a espera de serem sequenciadas e cada genoma humano sequenciado pode ocupar centenas de gigabytes de espaço de armazenamento. Os genomas humanos são críticos porque são extremamente valiosos para a investigação e porque podem fornecer informações delicadas sobre o estado de saúde dos indivíduos, identificar os seus dadores ou até mesmo revelar informações sobre os parentes destes. O tamanho e a criticidade destes genomas, para além da quantidade de dados produzidos por instituições médicas e de ciências da vida, exigem que os sistemas informáticos sejam escaláveis, ao mesmo tempo que sejam seguros, confiáveis, auditáveis e com custos acessíveis. As infraestruturas de armazenamento existentes são tão caras que não nos permitem ignorar a eficiência de custos no armazenamento de genomas humanos, assim como em geral estas não possuem o conhecimento e os mecanismos adequados para proteger a privacidade dos dadores de amostras biológicas. Esta tese propõe um sistema de armazenamento de genomas humanos eficiente, seguro e auditável para instituições médicas e de ciências da vida. Ele aprimora os ecossistemas de armazenamento tradicionais com técnicas de privacidade, redução do tamanho dos dados e auditabilidade a fim de permitir o uso eficiente e confiável de infraestruturas públicas de computação em nuvem para armazenar genomas humanos. As contribuições desta tese incluem (1) um estudo sobre a sensibilidade à privacidade dos genomas humanos; (2) um método para detetar sistematicamente as porções dos genomas que são sensíveis à privacidade; (3) algoritmos de redução do tamanho de dados, especializados para dados de genomas sequenciados; (4) um esquema de auditoria independente para armazenamento disperso e seguro de dados; e (5) um fluxo de armazenamento completo que obtém garantias razoáveis de proteção, segurança e confiabilidade a custos modestos (por exemplo, menos de $1/Genoma/Ano), integrando os mecanismos propostos a configurações de armazenamento apropriadas.

Palavras-chave: Armazenamento de Dados; Dados Genómicos; Privacidade; Deduplicação; Auditoria

Resumo Alargado

A compreensão do genoma humano [243] impacta várias áreas da vida, como por exemplo as áreas médicas, jurídicas, sociais e históricas [144, 148]. Os genomas humanos são críticos porque a disponibilização de extensos conjuntos de amostras aos investigadores acelera o progresso em muitas áreas que aumentaram a nossa conscientização sobre a saúde e a nossa expectativa de vida [154, 251]. No entanto, a criticidade dos genomas verifica-se também porque eles podem fornecer informações sobre o estado de saúde dos indivíduos com níveis consideráveis de confiança e podem identificar de maneira exclusiva os seus proprietários ou revelar informações sobre os parentes dos mesmos [20, 191].

Simultaneamente, é difícil armazenar em larga escala os dados do sequenciamento inteiro do genoma porque os conjuntos de dados produzidos neste método são enormes. Primeiro, o advento do *Next Generation Sequencing* (NGS) [223] reduziu exponencialmente os custos do sequenciamento de ADN nos últimos anos, o que está aumentando consideravelmente a quantidade de dados a serem geridos e armazenados [181] Atualmente, o sequenciamento de todo o genoma de um indivíduo custa menos de $1000 [142, 253], e espera-se que os preços continuem a cair [44, 253]. Segundo, os repositórios de dados biológicos (por exemplo, os laboratórios de investigação, hospitais e biobancos) foram inicialmente responsáveis pelo armazenamento de milhares a milhões de amostras físicas e biológicas. No entanto, eles estão sob pressão para armazenar também os dados genômicos resultantes da sequenciação destas amostras. Terceiro, centenas de gigabytes de dados são gerados a partir de cada célula humana sequenciada (consulte a Seção 2.2 para obter

mais detalhes). Quarto, a medicina personalizada tem usado dados genómicos para levar decisões médicas ao nível individual, impulsionando o uso de procedimentos e tratamentos específicos para cada paciente [124, 148], o que também pode fazer com que os indivíduos tenham as suas células sequenciadas várias vezes durante a sua vida. Em quinto lugar, estudos modernos visam correlacionar milhares a milhões de amostras biológicas ou mesmo populações inteiras (por exemplo, o projeto FarGen [101, 147]), em vez de fazê-lo com apenas alguns poucos indivíduos.

O tamanho dos problemas que os repositórios de genomas estão a enfrentar pode ser ilustrado, por exemplo, pelos altos custos para criar e manter uma infraestrutura privada para armazenar e processar centenas de petabytes de dados advindos de um milhão de genomas sequenciados. Em 2012, Haussler *et al.* [127] estimaram que seriam necessários aproximadamente \$65M no primeiro ano e \$35M em cada ano subsequente para preparar dois data centers fisicamente independentes que suportem pelo menos 125PB cada. Este custo foi dividido em \$30M como despesas de capital inicial (CAPEX), \$25M em despesas operacionais (OPEX) por ano e outros \$10M por ano para atualizar completamente a plataforma a cada três anos [127]. As principais alternativas para o alto custo das soluções de armazenamento incluem o uso de infraestruturas mais baratas e a redução do tamanho dos dados. Mesmo iniciativas existentes (por exemplo, [127, 255]) exigem investimentos substanciais para manter conjuntos de dados tão grandes e expandir a sua capacidade, o que sobrecarrega as organizações e faz com que estas iniciativas possam ser interrompidas [50].

Em termos de eficiência de custo, a computação em nuvem pública é a alternativa econômica mais prominente às infraestruturas privadas, pois fornece escalabilidade praticamente infinita em um modelo de pagamento conforme o uso. Ela permite a implantação imediata de grandes sistemas, fornece elasticidade rápida e não requer manutenção ou atualização de hardware por parte dos utilizadores. Além disso, a variedade atual de serviços geridos pela nuvem (centrados em dados) reduz também a necessidade de manutenção de serviços na perspectiva do cliente. O custo para armazenar 125PB de dados em nuvens públicas é, por exemplo, \$6 milhões por ano no Amazon Glacier (ou seja, \$0,004/GB/mês [12] para o arquivamento de dados e armazenamento de longo prazo) ou $31,5 milhões por ano no Amazon S3 padrão (ou seja, $0,021/GB/mês [13]).

Infelizmente, muitas instituições médicas e de ciências da vida ainda estão reticentes em adotar serviços de nuvem pública por vários motivos. Primeiro, apesar de sistemas de bioinformática estarem cada vez mais integrados com as nuvens públicas [190], ainda existe uma lacuna a ser transposta por investigadores que muitas vezes não são especialistas em computação [17, 228]. Segundo, há preocupações em confiar dados críticos em serviços controlados externamente, os quais ocasionalmente sofrem de indisponibilidade e incidentes de segurança [42, 58, 84]. Terceiro, existem porções de dados genómicos que são mais sensíveis à privacidade do que outras, e não há método para detectá-las automaticamente. Quarto, a grande sensibilidade à privacidade de algumas partes dos dados genómicos não permite confiar em nenhuma entidade de terceiros (por exemplo, um provedor de nuvem pública). Quinto, o grande tamanho dos dados exige que mesmo as soluções baseadas na nuvem considerem técnicas adicionais de redução de custos de armazenamento. Finalmente, os controladores de dados confidenciais requerem meios de auditar o acesso aos dados para detectar, analisar e sancionar abusos.

Em termos de redução do tamanho dos dados, a compactação e deduplicação são as principais técnicas que permitem economia de custo. O primeiro substitui os dados por uma representação menor que a original [218]. De forma semelhante, o segundo substitui dados redundantes por apontadores para uma única cópia dos mesmos [202]. Técnicas de compactação e deduplicação de dados genéricas são ineficientes em dados genómicos [116] (consulte a Secção 4.3) e, portanto, algoritmos específicos devem ser usados [23, 87, 248]. Algoritmos baseados em compressão referencial são os mais adequados para comprimir genomas humanos [38, 105, 151, 213]. No entanto, eles alcançam os seus melhores resultados somente quando o genoma está em um formato de ADN contíguo (por exemplo, veja as diferentes representações de dados do genoma na Secção 2.2). Os dados brutos de sequenciamento são mais difíceis de compactar ou deduplicar, o que deixa oportunidades em aberto para explorarmos novas soluções. Além disso, muitas implementações existentes usam técnicas com perdas para alcançar os seus melhores resultados, o que significa que elas melhoram os ganhos de armazenamento reduzindo a utilidade dos dados [105]. Os algoritmos de deduplicação específicos são candidatos a substituir a necessidade de mecanismos com perdas na compactação. Até onde sabemos, não existe uma solução de deduplicação especializada para este tipo de dados.

Finalmente, a ampla prevalência de violações de dados amplia a importância da auditoria dos sistemas de armazenamento. Por exemplo, saber exatamente quem acedeu a um item de dados (por exemplo, um genoma humano) é um recurso que aumenta significativamente a confiança dos utilizadores em uma solução de armazenamento. Soluções simples que criptografam e armazenam todo o genoma em um único repositório podem facilmente suportar esse recurso, fornecendo aos auditores todos os registos de acesso do sistema. No entanto, este tipo de solução requer confiança na entidade que controla o repositório e incorre em um único ponto de falha no sistema. Os sistemas de armazenamento seguro (por exemplo, [28, 183]) geralmente dividem e convertem os dados originais em blocos codificados e armazenam cada bloco em um repositório diferente. A leitura de dados em um sistema como este requer acesso a vários repositórios para obter blocos distintos suficientes para recuperar os dados originais. Da mesma forma, os auditores precisam receber registos de acesso de vários repositórios e correlaciona-los antes de poder definir quem acedeu qual item de dados. De acordo com o nosso conhecimento, não há estudo sobre os requisitos teóricos exigidos pela auditoria para proteger os sistemas de armazenamento.

Em resumo, os sistemas de armazenamento para genomas humanos devem enfrentar três desafios principais para se tornarem eficientes: levar em consideração a privacidade ao armazenar os dados em infraestruturas externas, manter o alto desempenho dos sistemas e a utilidade dos dados ao reduzir o tamanho dos ficheiros e fornecer meios para os controladores de dados verificarem quem efetivamente leu os seus dados em sistemas de armazenamento seguro.

Esta tese propõe um sistema de armazenamento de genomas humanos eficiente, seguro e auditável para instituições médicas e de ciências da vida. Ele aprimora os ecossistemas de armazenamento tradicionais com técnicas de privacidade, redução do tamanho dos dados e

auditabilidade a fim de permitir o uso eficiente e confiável de infraestruturas públicas de computação em nuvem para armazenar genomas humanos. O sistema de armazenamento proposto nesta tese é um fluxo de armazenamento composto por quatro fases internas: DETECÇÃO, REDUÇÃO, ARMAZENAMENTO e GESTÃO. Este sistema recebe os dados de *sequenciadores de amostras* e as preferências de compartilhamento de dados de *dadores de amostras*. A etapa de DETECÇÃO identifica e separa as porções sensíveis à privacidade de genomas humanos das porções ditas não sensíveis (a primeira contribuição desta tese). A etapa de REDUÇÃO reduz o tamanho dos dados empregando deduplicação baseada em similaridade e codificação delta especializada para dados sequenciados (a segunda contribuição). A etapa de ARMAZENAMENTO não é uma contribuição direta desta tese, mas armazena os dados reduzidos em configurações apropriadas de acordo com a sensibilidade à privacidade das porções dos genomas. A etapa de GESTÃO contém dois componentes internos responsáveis por controlar o acesso aos dados com base nas preferências dos dadores (não é uma contribuição direta desta tese) e por fornecer meios de auditabilidade sobre quem acedeu aos dados (a terceira contribuição desta tese).

As contribuições desta tese constituem-se de:

(1) Um estudo sobre a sensibilidade à privacidade dos genomas humanos;

(2) Um novo método que detecta sistematicamente os segmentos de ADN sensíveis à privacidade vindos diretamente de um fluxo de entrada, usando como referência uma base de dados de conhecimento de sequências conhecidas de nucleótidos e aminoácidos considerados sensíveis à privacidade. A integração deste método de detecção com técnicas de segurança comuns fornece uma solução robusta e eficiente que reconhece a privacidade, que neutraliza ameaças relacionadas a ataques recentemente publicados à privacidade do genoma, com base em repetições curtas, genes relacionados a doenças e variações genômicas.

(3) Um novo método, chamado GenoDedup, para integrar uma deduplicação eficiente baseada em similaridade e uma codificação delta especializada para dados de sequenciamento de genoma. O objetivo específico do GenoDedup é equilibrar a economia

de espaço e o desempenho das leituras, aumentando os ganhos de redução de dados e restaurando os dados mais rapidamente do que os compressores genéricos usados na prática, enquanto obtém ganhos de redução próximos aos de ferramentas especializadas.

(4) Um estudo de emulações de armazenamento auditáveis, que fornecem a capacidade de um auditor descobrir as leituras executadas anteriormente em registos emulados. Ele define com precisão os registos auditáveis, as suas propriedades e estabelece limites estreitos e resultados de impossibilidade para as emulações de armazenamento auditáveis em cenários com objetos de armazenamento faltosos.

(5) A integração das contribuições anteriores em um fluxo de dados destinado a permitir o armazenamento eficiente e confiável de genomas humanos em núvens públicas. Este fluxo permite obter garantias razoáveis de proteção de privacidade, segurança e confiabilidade a custos modestos (menos de \$1/Genoma/Ano) usando configurações de armazenamento apropriadas.

Finalmente, esta tese pressupõe que os dados de seqüenciamento obtidos a partir de genomas humanos são críticos e massivos, o que impede que as infraestruturas de armazenamento existentes ignorem a segurança e a relação custo-benefício. Resumindo todas as contribuições mencionadas anteriormente, a hipótese desta tese é:

Aprimorar os ecossistemas de armazenamento tradicionais com técnicas de privacidade, redução do tamanho dos dados e auditoria permite o uso eficiente e confiável de infraestruturas públicas na nuvem para armazenar genomas humanos.

Palavras-chave: Armazenamento de Dados; Dados Genómicos; Privacidade; Deduplicação; Auditoria

Acknowledgements

I render my sincerest gratitude to every single person that has been part of this journey. Your example, love, and encouragement were determinant for me to reach this milestone.

To my advisor, Alysson Bessani for his immense support, guidance, and friendship. His rigour, ambition, and competitiveness will keep guiding many aspects of my career and life.

To my family for their unconditional support and encouragement. To my parents and their companions (Sandra, Marco, Juliana, and Luis) and my siblings (Vitor, Vitória, Mateus, João Pedro, and Maria Luisa) for being always present in my life and for enabling me to grab so many opportunities and follow my dreams. To Vanessa Lucas for boarding into my life in such an important moment, for supporting me and walking by my side full of kindness, positiveness, and love every day. To my parents-in-law, grandparents, uncles, and cousins for motivating me to be a better person and professional as they are.

To LASIGE and DI-FCUL for their culture of research excellence, openness, and collaboration. To the coordinators and directors for their availability and management, Luis Carriço, Paulo Veríssimo, Nuno Neves, Vasco Vasconcelos, Antónia Lopes, and Luis Correia. To co-authors of the thesis-related publications for their insights and dedication, in special to Francisco Couto, Paulo Veríssimo, João Paulo, Nuno Neves, Ricardo Mendes, and Tiago Oliveira. To my colleagues of the lab and PhD, for their companionship during all these years. To Mônica, Rudra, Giuliana, João A., Bruno Vavala, André, Patrícia, Miguel, Fernando Alves, Adriano, Telma, Eduardo V., Eduardo A., Luis Brandão, Max, Fernando Ramos, Diego Kreutz, Antonio Casimiro, Ibéria, Márcia, Soraia, Nuno, Diana, Sofia, Joana, Jośe C., Diogo L., Cajó, Bruno Valente, João Sousa, Henrique M., Cogumbreiro, Wilson, Alan, Marcel, Rui Caldeira, Túlio, and Pedro Costa. Many of them became close friends for life. To the administrative staff Sandra C., Pedro G., Alexandra E., Carla C., Márcio D., and Diogo F. To my previous advisors Andrea Charão and Marcelo Pasin, to INF-UFSM, PET-UFSM, LSC-UFSM, and all my professors.

To my friends for their constant encouragement and friendship. To Leticia for being an important part of this path, to Ana H., Teresa, Simão, Morgana, Maiara, Emanuel, Cristiane, Marcírio, Cristiane, Adriana, Rui, Janete, and Hugo Dias. To my friends from Brazil, Hugo, Francine, Harnye, Etiane, Barin, Leandro P., Márcia P., Daianne, Rissetti, Conrado, Mega, and Eduardo C. To my friends from ACEFL, Elda, Odete, Renet, Camila, Daniel, Patrícia, Teresa, and Dinis. To my friends from Nazaré and Constância (CP), from the soccer and padel.

From the bottom of my heart, thank you all!

Funding

The work from this thesis was partially supported by the Fundação para a Ciência e a Tecnologia (FCT, Portugal), through the LASIGE research unit (UIDB/00408/2020 and UIDP/00408/2020) and the IRCoC project (PTDC/EEI-SCR/6970/2014), and by the European Commission, through the BiobankCloud (FP7-ICT-317871), the SUPERCLOUD (H2020-ICT-643964), and the DiSIEM (H2020-IA-700692) projects.

Contents

List of Figures

List of Tables

Introduction 1

The understanding of human genome [243] impacts several areas of human life, such as medical, legal, social, and historical ones [144, 148]. Human genomes are critical because they provide extensive sample collections enabling researchers to accelerate breakthroughs in many areas that are increasing our health awareness and life expectancy [93, 154, 251]. However, genomes' criticality stands also because they may provide hints on individuals' health status with considerable confidence levels and can uniquely identify their owners or reveal information about donors' relatives [20, 21, 191].

Simultaneously, data from whole-genome sequencing (WGS) is difficult to store at scale because its datasets are massive [230]. First, the advent of the Next-Generation Sequencing (NGS) [223] reduced the costs of DNA sequencing exponentially in recent years, which has considerably increased the amount of data to be stored and managed [55, 181, 230]. Sequencing the whole genome of an individual costs less than $1000 [142, 253] in general, but prices are expected to continue falling [44, 211, 253]. Second, biological data repositories (e.g., research institutes, hospitals, and biobanks) were initially responsible for stocking thousands to millions of physical, biological samples. However, they have been under pressure to store also the resulting digitised genomic data. Third, hundreds of gigabytes of data are generated from each sequenced human cell (see Section 2.2 for more details). Fourth, personalised medicine has been using genomic data to bring medical decisions to the individual level propelling the use of specific procedures and treatments for each patient [124, 148], which also may cause individuals to have their cells sequenced multiple times during their lifespan. Fifth, modern studies aim to correlate thousands to millions of biological samples or even whole populations (e.g., FarGen project [101, 147]), instead of doing so with only a few individuals.

The size of the problem genome repositories are facing can be illustrated, for instance, by the high costs to create and maintain a private infrastructure to store and process hundreds of petabytes of data from a million sequenced genomes. In 2012, Haussler *et al.* [127] estimated that approximately $65M would be required in the first year and $35M in every subsequent year to prepare two physically independent data centres for at least 125PB each. This cost was divided in $30M as the initial capital expenses (CAPEX), $25M in operational expenses (OPEX) per year, and other $10M per year to completely upgrade the platform every three years [127]. The main alternatives to the high cost of storage solutions include using cheaper infrastructures and reducing the data size. Even existent initiatives (e.g., [127, 255]) require substantial investments for maintaining such large datasets and scaling out their capacity, which burden organisations and cause them to be discontinued [50].

In terms of cost-efficiency, public cloud computing is the most prominent economic alternative to private infrastructures since it provides virtually infinite scalability in a pay-as-you-go model. It allows the immediate deployment of large systems, provides fast elasticity, and requires no hardware maintenance or upgrade from users. Additionally, the current variety of (data-centric) cloud-managed services reduce also the need for service maintenance from the client perspective. The cost to store 125PB of data in public clouds is, for instance, $6M per year in Amazon Glacier (i.e., $0.004/GB/Month [12] for data archival and long-term storage) or $31.5M per year in standard Amazon S3 (i.e., $0.021/GB/Month [13]).

Unfortunately, many medical and life-sciences institutions are still reticent in adopting public cloud services for several reasons. First, despite the fact that bioinformatics systems are becoming increasingly integrated with the cloud [190], there is still a gap researchers that are non-computer experts need to transpose to run their workflows on the cloud seamlessly [17, 228]. Second, there are concerns in trusting critical data to externally-controlled services that occasionally suffer from unavailability and security incidents [42, 58, 84]. Third, there are portions of genomic data that are more privacy-sensitive than others, and there is no method to detect them automatically. Fourth, the higher privacy-sensitivity of some portions of genomic data disallows trusting any single third-party entity (e.g., a public cloud provider). Fifth, the large data size requires that even cloud-based solutions must consider additional cost-effective techniques to reduce storage costs. Finally, controllers of sensitive data require means of auditing data access to detect, analyse, and sanction misuses.

In terms of reducing the size of data, compression and deduplication are the main techniques enabling cost efficiency. The former replaces data by a representation smaller than the original one [218]. Similarly, the latter replaces redundant data by pointers to a single copy of it [202]. Standard compression and deduplication techniques are inefficient in genomic data [116] (see Sections 4.2 and 4.3.1), and thus application-based algorithms must be used [23, 87, 248]. Algorithms based on referential compression are the most adequate to compress human genomes [38, 105, 151, 213]. However, they reach their best results only when the genome is in a contiguous DNA format (e.g., see the different genome data representations in Section 2.2). Raw sequencing data is harder to compress or deduplicate, which leaves open opportunities to explore novel solutions. Additionally, many existent implementations use lossy techniques to reach their best results, which means they improve the storage gains by reducing data utility [105, 196]. Application-specific deduplication algorithms are a candidate to replace the need for lossy mechanisms in compression. To the best of our knowledge, there is no such a deduplication solution specialised for this type of data.

Finally, the widespread prevalence of data breaches amplifies the importance of auditing storage systems. For instance, being able to know exactly who has accessed a data item (e.g., a human genome) is a feature that significantly increases the trust of users in a storage solution. Simple solutions that encrypt and store the whole genome in a single repository can easily support this feature by providing to auditors all access records from the system.

However, this type of solution requires trust in the entity that controls the repository and incurs a single point of failure to the system. Secure storage systems (e.g., [28, 183]) usually split and convert the original data into coded blocks and store each block in a different repository. Reading data from a system like this requires accessing multiple repositories to obtain enough distinct blocks to recover the original data. Similarly, auditors need to receive access records from several repositories and merge them before being able to define who has accessed which data item. To the best of our knowledge, there is no study on the theoretical requirements auditability incurs to secure storage systems.

In summary, storage systems for whole human genomes must address three main challenges to become efficient and dependable: be privacy-aware when storing data in external infrastructures, keep systems' high performance and data utility when reducing the data size, and provide means to data controllers verify who has effectively read their data from secure storage systems.

1.1 Main Goal and Context

This thesis proposes an efficient, secure, auditable storage system for whole human genomes for medical and life-sciences institutions. It enhances traditional storage ecosystems with privacy-aware, data-reduction, and auditability techniques to enable the efficient use of multi-tenant public infrastructures to store human genomes. We start by placing our system proposal within the genomic research and clinical ecosystems. The upper part of Figure 1.1 depicts the main players and their interactions.

Initially, *sample donors* (1) donate biological material (i.e., physical samples) to research or clinical analyses and (2) inform their preferences for data sharing (if any). *Sample managers* receive, manipulate, and store these biological specimens on their facilities (e.g., using cryopreservation [80]) and (3) forward portions of these samples to sequencing. *Sample sequencers* are trusted entities that (3) receive parts of the physical, biological samples, sequence them, (4) send the sequenced data to storage, and securely discard

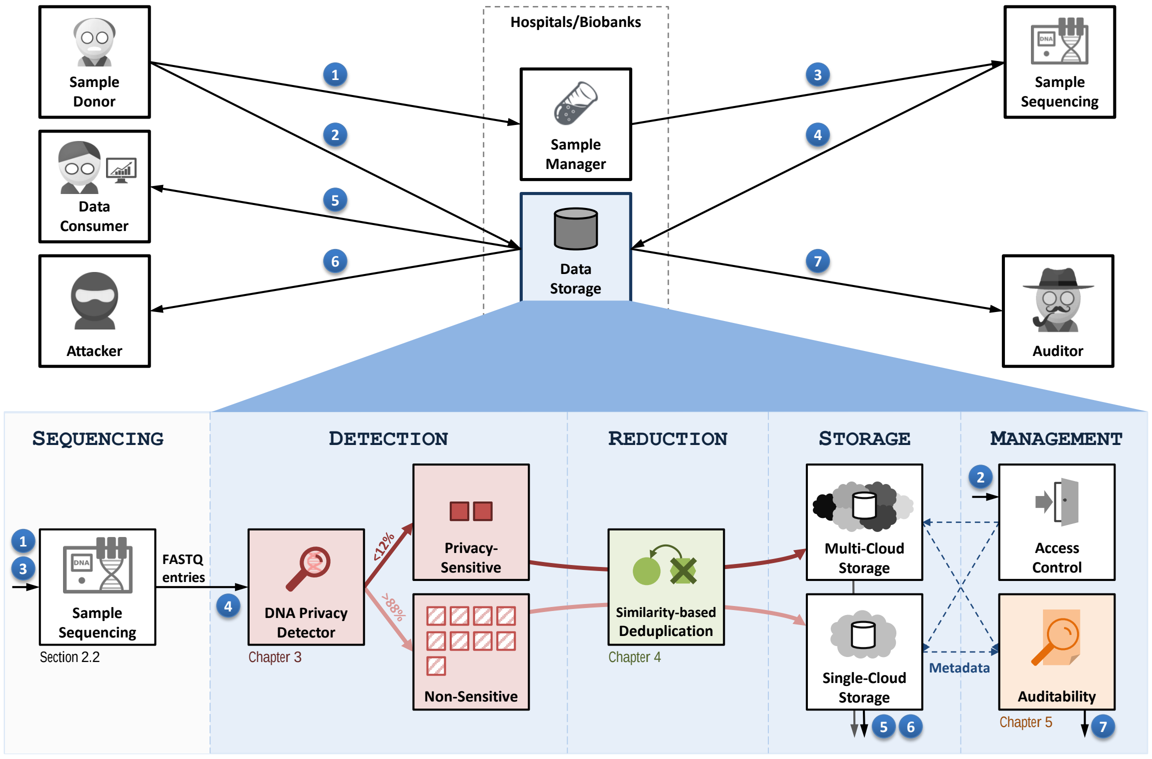


Figure 1.1: Main players of the storage ecosystem for human genomes, their interactions, and an overview of the enhanced storage pipeline proposed in this thesis.

the biological samples and the resulting data from their facilities. *Data consumers* are entities that (5) consume data from the storage system according to donors' preferences and other permission rules. *Attackers* are also entities, indistinguishable from data consumers, that (6) consume data from the storage system, but they do not necessarily respect all access rules. *Data storage* is the storage system proposed in this thesis. It provides the necessary scalability, performance, security, dependability, and cost-efficiency to store data from human genomes. Although the data storage belongs to biobanks' or hospitals' administrative domains, data does not necessarily reside in local private infrastructures (e.g., it can be stored in external public clouds). Finally, *auditors* are entities, controlled either by the data managers or by trusted third-party entities, that (7) request the records and logs (i.e., metadata only) evidencing who has accessed which portions of the stored data.

Our work focuses specifically on the *data storage* component, which efficiently stores and provides human genomic data. The lower part of Figure 1.1 zooms into this component.

Numbers there represent the same interactions from the upper part of this figure. In the next section, we introduce the internal components of our data storage system and describe the thesis' contributions.

1.2 Contributions

The *data storage* component from Figure 1.1 is a storage pipeline composed of four internal phases: DETECTION, REDUCTION, STORAGE, and MANAGEMENT. This component receives data from *sample sequencers* (i.e., Interaction 4, in an external phase called SEQUENCING) and data sharing preferences from *sample donors* (i.e., Interaction 2, in the MANAGEMENT phase). The DETECTION step identifies and separates the privacy-sensitive portions of human genomes from the non-sensitive ones (the first contribution from this thesis). The REDUCTION step reduces the data size by employing similarity-based deduplication and delta-encoding specialised for sequencing data (the second contribution). The STORAGE step is not a direct contribution from this thesis, but it stores the reduced data in appropriate configurations according to the privacy-sensitivity of the genomes' portions. The MANAGEMENT step contains two internal components responsible for controlling the data access based on donors' preferences (it is not a direct contribution from this thesis) and for providing auditability means on who has effectively read data (the third contribution from this thesis).

This thesis results from specific tasks distributed across four research projects (BiobankCloud, SUPERCLOUD, DiSIEM, and IRCoC) conducted in the Large-Scale Informatics Systems Laboratory (LASIGE) in the Faculty of Sciences (Ciências) of the University of Lisbon (ULisboa). In the following, we introduce each of the three mentioned main contributions (they are also described in detail as individual chapters later), an integrated storage pipeline (the fourth contribution of this thesis), and other minor contributions related to the mentioned projects and topics of interest.

C1—Detection

The first contribution of this thesis is the proposal of a novel method that systematically detects privacy-sensitive DNA segments coming directly from an input stream, using as reference a knowledge database of known privacy-sensitive nucleic and amino acid sequences. Integrating this detection method with standard security techniques provides a robust, efficient privacy-aware solution that neutralises threats related to many published attacks on genome privacy based on short tandem repeats, disease-related genes, and genomic variations.

This method was developed during the BiobankCloud project and composes the DETECTION phase in the pipeline presented in the lower part of Figure 1.1. This contribution is detailed in Chapter 3, and it has directly resulted in two main publications [70, 71].

C2—Reduction

The second contribution of this thesis is the proposal of GenoDedup, which is the first method to integrate efficient similarity-based deduplication and specialised delta-encoding for genome sequencing data. The specific goal of GenoDedup is to balance space savings and restore performance by increasing data reduction gains and restoring data faster than the generic compressors used in practice, while approaching the reduction gains to the ones from specialised tools.

This solution composes the REDUCTION phase in the pipeline presented in the lower part of Figure 1.1 and was developed during the SUPERCLOUD project. This contribution is detailed in Chapter 4, and it has resulted in a journal publication [72].

C3—Auditability

The third contribution of this thesis is the proposal of auditable storage emulations, which provide the capability for an auditor to discover the previously executed reads in emulated registers. It precisely defines auditable registers, their properties, and establishes tight bounds and impossibility results for auditable storage emulations in the presence of faulty storage objects.

This work was developed during the DiSIEM and IRCoC projects and composes the Auditability component of the MANAGEMENT phase in the pipeline presented in the lower part of Figure 1.1. This contribution is detailed in Chapter 5, and it has directly resulted in a preprint article to be submitted [69].

C4—Pipeline

The fourth and last contribution of this thesis is the integration of the previous contributions into an end-to-end composite pipeline intended to enable the efficient, dependable cloud-based storage of human genomes. It allows one to obtain reasonable privacy protection, security, and dependability guarantees at modest costs (less than \$1/Genome/Year) using appropriate storage configurations.

We describe how data flows from one component to the other and present a feasibility evaluation of the pipeline in Chapter 6. A review of existent data integration initiatives for human genomes was conducted during the BiobankCloud project and published in an article [66]. The feasibility of the storage pipeline was evaluated during the DiSIEM and IRCoC projects, but it relates to all previously mentioned projects and contributions. Additionally, it has resulted in a workshop publication [68].

Other contributions

This thesis touches several areas of computer science, and its works have resulted in some secondary contributions, collaborations, and publications. Although related to the previous contributions, these works are presented here as additional achievements because they are not at the core of this thesis.

For instance, Contribution $C1$ was also presented during a PhD students session [63]. Works related to Contribution $C4$ were also presented as posters [64, 65]. Works related to Contributions $C2$ and $C4$ have been part of Deliverables $D1.4$ [171], $D2.2$ [30], $D4.1$ [29], $D4.2$ [31], and $D4.3$ [33] of the BiobankCloud project.

Some works, related to the Contribution $C4$, were co-authored by the candidate during his PhD period. For instance, the paper entitled "CHARON*: A Secure Cloud-of-Clouds System for Storing and Sharing Big Data*" [183] proposed a multi-cloud storage system for storing and sharing big data. The PhD candidate has contributed in the requirement specification to store human genomes, some design and motivation discussions, writing, and the evaluation of the system (more specifically in the development of the FS-Biobench macro benchmark [67]. CHARON is used in this thesis as a black box component providing all the identified requirements for the STORAGE phase of our pipeline. Previously, the paper entitled "*BiobankCloud: a Platform for the Secure Storage, Sharing, and Processing of Large Biomedical Data Sets*" [32] described the whole integrated platform of the BiobankCloud project, which also has CHARON [183] as an internal component. Finally, the candidate also collaborated in other two works related to Contribution $C2$ [10, 11].

1.3 Thesis Hypothesis

This thesis assumes that sequencing data obtained from human genomes are critical and massive, which prevents existing storage infrastructures from ignoring security and cost-efficiency. Summing up all previously introduced contributions, the hypothesis of this thesis is:

Enhancing the traditional storage ecosystem with privacy-aware, data-reduction, and auditability techniques enables the efficient, dependable storage of sequencing data from human genomes.

1.4 Structure of the Thesis

The remaining of this document is divided into six chapters.

Chapter 2. This chapter introduces several concepts and processes that are important for the remaining chapters of the thesis. Additionally, it presents general related works that compose the state-of-the-art in the topics of interest for this document. Some chapters can contain additional related works that are specific to the contribution being presented.

Chapter 3. This chapter describes a high-throughput method to detect the privacy-sensitive portions of human genomes, which is the first contribution of this thesis. This method allows one to separate these portions from the non-sensitive ones and treat them differently in the complete storage pipeline.

Chapter 4. This chapter presents a comparison of several tools for compressing sequencing data, as well as proposes and evaluates a similarity-based deduplication and delta encoding algorithm specialised for human genomic sequencing data.

Chapter 5. This chapter discusses the formal requirements and impossibility results, as well as proves the lower bounds of auditability in secure storage systems that use information dispersal across multiple repositories.

Chapter 6. This chapter presents the complete storage pipeline that integrates the contributions from Chapters 3–5 and evaluates its feasibility.

Chapter 7. This chapter concludes the thesis by remarking the main contributions and presenting possible future works deriving from this thesis.

Background and Related Work 2

This chapter presents the background and state-of-the-art in the research areas relevant to this thesis. It describes the context and related work on storage of genomic data and introduces mechanisms for privacy protection, data reduction, and auditability. More specifically, Section 2.1 contains a brief overview on human genomes; Section 2.2 describes the genome sequencing process and its files; Section 2.3 provides an overview of the existent initiatives for integrating genomic data from multiple data storages; Section 2.4 discusses the security and dependability aspects of data storage; Section 2.5 reviews privacy-aware storage techniques; Section 2.6 compares data reduction techniques; Section 2.7 overviews existent techniques for storage auditability; and Section 2.8 discusses the opportunities identified as open problems in the literature.

2.1 A Brief Background on Human Genomes

This section introduces several biology concepts that are important for this thesis. Their definitions were collected and adapted from introductory documents in this field (e.g., [78, 164, 200, 239]). Figure 2.1 depicts the relations between these concepts descending from human beings to DNA, RNA, genes, and proteins.

The body of a *human being* is composed of trillions of cells of many types (e.g., stem, bone, blood, muscle). Each of our *cells* contains a nucleus, which characterises humans

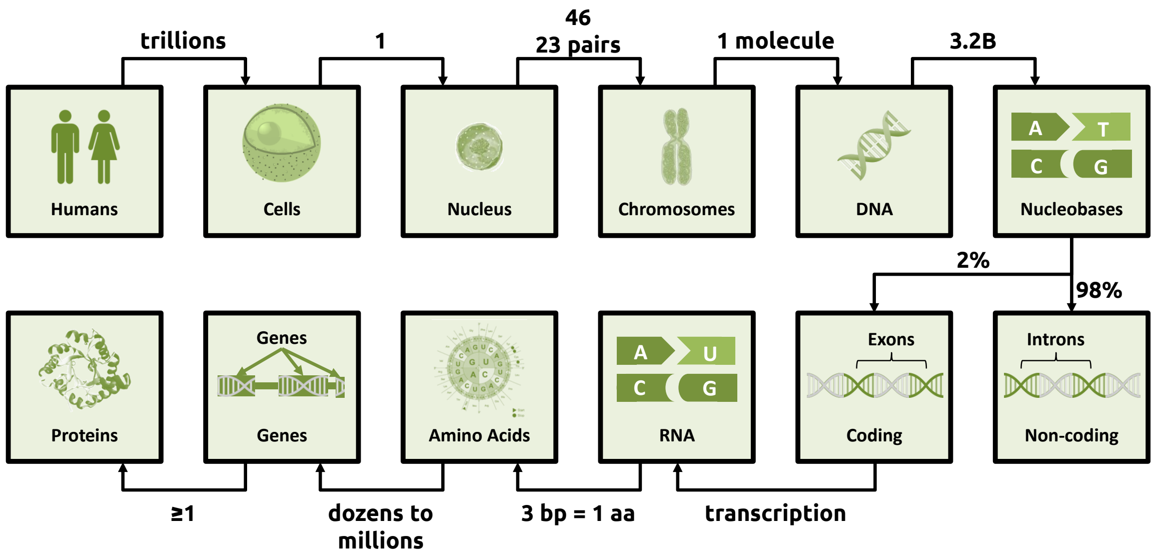


Figure 2.1: Biological diagram—from human beings to proteins.

as eukaryotic organisms. The *nucleus* of a human cell contains most of the individual's genetic material. Only a small part of this hereditary material is located in the mitochondria, outside the nucleus.

Cells can also be classified into haploid and diploid types, where the former contains only one complete set of chromosomes and are used in reproduction (i.e., gamete/ovum and sperm), and the latter comprises two sets and are somatic cells. A diploid human cell contains 46 chromosomes organised in 23 pairs, where 22 of these pairs are composed of autosomes (i.e., body chromosomes that are similar in males and females) plus a pair of allosomes (sex chromosomes)—XX for women or XY for men. A human *chromosome* is composed of a DNA (DeoxyriboNucleic Acid) molecule and histones (i.e., packaging proteins). The *DNA* molecule is composed of two chains of nucleotides that form the spiral ladder known as the double helix. The size of a human genome in a haploid cell is equivalent to approximately 3.2 billion nucleotides, which is the sum of all nucleotides of one chain from each human chromosome. This 3.2B number appears again in several sections of this document to describe the size of a human genome and help to measure the file size in several data representations.

A *nucleotide* is composed of a chemical base, a sugar, and a phosphate molecule. A *chemical base* or a *nucleobase* is a chemical compound that is represented as one of the basic four letters of DNA—“A” for adenine, “C” for cytosine, “G” for guanine, and “T” for thymine. Nucleobases pair with each other (i.e., A with T and C with G) to form base pairs (i.e., *bp*) bonding together the two nucleotide chains of the DNA.

The human genome contains several intercalated regions that can be classified into two groups: the *coding* and the *non-coding* regions. The former regions are the genome portions that have some known functional outcome (e.g., proteins). The latter ones are regions that contain structural DNA or DNA whose function is yet unknown. Less than 2% of a human genome is considered coding regions (or exons), while the remaining more 98% are non-coding regions (or introns).

Genes are functional units made up of DNA that can be transcribed to RNA (RiboNucleic Acid). *RNA* is a single strand also composed of a chain of nucleobases, where every thymine (“T”) from the DNA is replaced by uracil (“U”). *Alleles* are similar forms of genes with small differences in their sequences, which make one unique or may have an impact, for instance, in being more prone to develop a disease. Messenger RNA (mRNA) is the type of RNA that carries the needed information for making proteins, from the nucleus to the cytoplasm. In the cytoplasm, mRNA interacts with the ribosome to translate a *codon* (i.e., three bases) into an amino acid (aa). In humans, there are twenty *amino acids* that can be used as building blocks for proteins.

Another RNA type, known as transfer RNA (tRNA), assembles the protein, one amino acid at a time until it finds a codon that does not code for any amino acid (i.e., a “stop” codon). *Gene regulation* is an essential process because it determines which genes will be expressed (i.e., turned on) or repressed (i.e., turned off) in each cell. Finally, *proteins* are molecules that do most of the work in cells and contribute to determining the structure, function, and regulation of tissues and organs.

2.2 Genome Sequencing

Data from human genomes has three main representations: sequencing, aligned, and assembled data. *Sequencing data* is the most critical representation in genomics because it contains the purest version of genomic data and is unbiased from subsequent processing steps [61]. Sequencing machines digitise genomes by translating the chemical compounds from biological samples to digital information. Next-Generation Sequencing (NGS) [223] is the name given to the machines that sequence genomes at a high-throughput [172]. However, NGS machines do not provide the whole human genome in a single contiguous DNA sequence. They generate millions of small *DNA reads*, which are small pieces of DNA containing sequences with hundreds to thousands of nucleobases each. Additionally, every nucleotide from a human genome is sequenced many times and appear in several complimentary reads (e.g., 30–45$\times$) to improve the sequencing accuracy. Data obtained from sequencing genomes is stored in the FASTQ text format [61], which is usually written once and read many times later for processing. FASTQ is the standard format in both cold and hot storage systems for genomic sequencing data [61].

A FASTQ file contains many entries with four lines each—similar to the one presented at the top right corner of Figure 2.2. The first line is a *comment* about the entry starting with a "`@`" character. The second line contains the *DNA sequence* interpreted by the machine—e.g., `A` for adenine, `C` for cytosine, `G` for guanine, and `T` for thymine. The third line is another comment that starts with a "+" character to determine the end of the DNA sequence, and can optionally be followed by the same content as the first one. The fourth line contains *quality scores (QS)*, which measure the machine's confidence for each sequenced nucleobase.

The second (DNA) and fourth (QS) lines have the same length since one QS is attributed for each sequenced nucleobase. This length is configurable and may vary from file to file, but it is usually constant within the same file. We provide additional specific details about FASTQ entries in Section 4.1, where we enlist their aspects that influence data reduction the most.

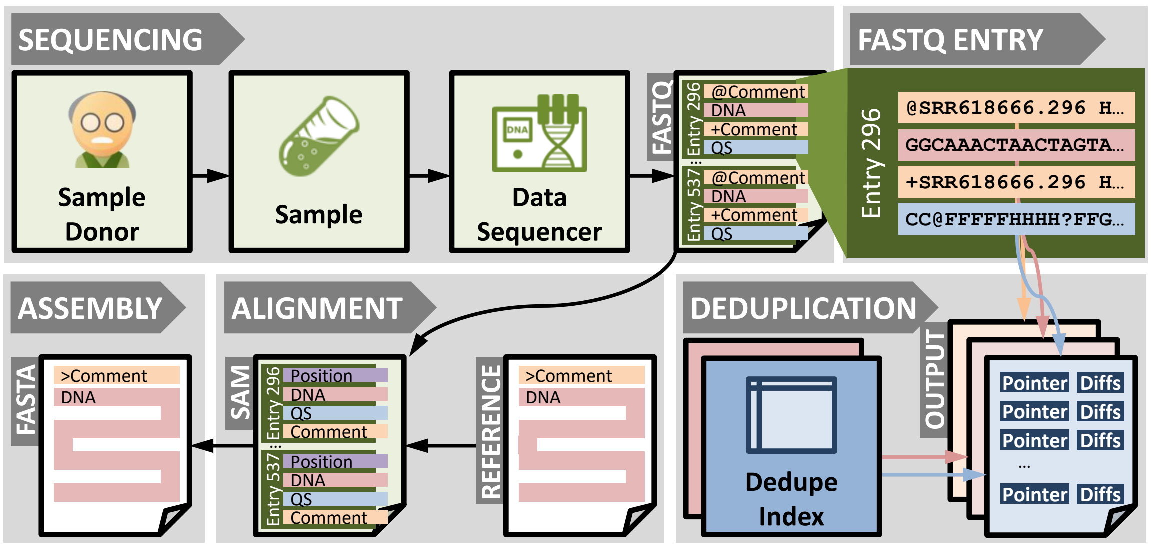


Figure 2.2: Genome sequencing overview, some subsequent workflows, and a FASTQ entry.

The *alignment* process maps each NGS read in a publicly available reference genome. The resulting data is usually stored in the SAM/BAM format [166], which contains one entry per read. Each aligned SAM/BAM entry is composed of the most probable chromosome and position where the sequence appears in the reference genome, a set of information describing the alignment run, the DNA sequence, the quality scores, and some optional user-defined tags about the alignment process. A drawback of storing aligned data is the fact that current alignment algorithms weight quality scores differently or are limited in terms of accuracy, which may lead to different alignment results [223]. This variance biases subsequent workflow steps and consequently may influence research conclusions. Thus, the SAM/BAM format is not the best alternative to store raw genomic data.

The *assembly* process forms the contiguous sequence of a genome by aligning and merging all sequenced reads. The assembled result is stored generally in the FASTA format [205], which contains only comment lines (started by the ">" character) and long sequences (e.g., the DNA of an entire chromosome). Similarly to the alignment process, the assembly of a genome may be influenced by the chosen algorithm. Additionally, the assembly process uses and then discards the quality scores, which prevents the use of FASTA files to store raw genomic data.

Although this thesis favours sequencing data from human genomes due to the potential impact of this data, the methods proposed in this document are generic enough to be adapted to sequencing data from other species, data representations (e.g., aligned data), and file formats (e.g., SAM). A discussion on other datasets and on why this work favours sequencing data rather than aligned or assembled representations is available in Section 4.6.

2.3 Storage of Genomic Data

Biological data repositories started by collecting and providing small public DNA sequences and related data to improve the scientific knowledge in genomics. However, they were isolated from each other and often overlapped. Their integration evolved over the years, and several paradigm changes took place. For instance, biological data repositories became responsible for storing whole genomes instead of only small sequences. Another example is biobanks, which are repositories that store biological, physical samples originally (e.g., in cryopreservation facilities [80]) and are also becoming responsible for storing sequencing data about these samples.

We surveyed a series of initiatives that integrate biological data repositories in an article [66], as well as detailed how do they manage data and identified their strengths and weaknesses. The analysed initiatives are the International Nucleotide Sequence Database Collaboration (INSDC) [60, 145], the UK Biobank [197, 237], the Biobanking and Biomolecular Research Infrastructure (BBMRI) [25, 26], ELIXIR [77, 95], the GenomEUtwin [170, 187], and the COMMIT project [74]. Additionally, we compared them to several cloud-based solutions existent at the time (e.g., DNAnexus [90], BaseSpace [141], and Galaxy [106]).

All compared initiatives employ either the data warehouse or the mediator approach for data integration [182]. All in all, some interesting trends were observable through the analysis of these systems. An integrated system for attributing accessioning numbers is critical, but at this point, it appears there was no evolved protection against duplicate entries, sequences, or individuals. All systems make their datasets available through web portals, that can be either freely accessible (e.g., if the data is public) or implement authentication and access control mechanisms to give access to specific datasets only to authorised users.

All mediator-based systems devise dependability mechanisms only in the endpoints (in-site). INSDC replicates all data in three globally distributed replicas, while UK Biobank uses two facilities to store only physical samples. Cloud-based solutions rely on the transparent replication and recovery mechanisms offered by the Amazon Web Services (AWS). All presented cloud-based solutions focus on approaching computation to where data is located, which can bring performance improvements since datasets upload, download, and even normalisation can be avoided.

2.4 Security and Dependability in Storage

Platforms that trust on a single external storage service (e.g., a public cloud provider) have the same strengths (e.g., transparent replication) and drawbacks (e.g., vendor lock-in) as the properties provided by the resources they use. Several storage solutions have been proposing the use of multiple infrastructures instead of a single one to increase the security and dependability guarantees of the system. Examples include libraries [3, 24, 28], file synchronisation services [54, 125, 233], and file systems [34, 183]. All these solutions employ several security and dependability techniques, such as data replication, encryption, and secret sharing [161].

Replicating all data across multiple clouds incurs in space and cost overheads proportional to the number of infrastructures where data is replicated. However, solutions employing erasure codes (e.g., CHARON [183]) reduce such space overheads since they store parity blocks instead of copies of the whole dataset.

Privacy perception of storing data in a single multi-tenant infrastructure is considered low even when data resides encrypted because attackers need to compromise only a single facility to obtain all data (still encrypted). Replicating all data in multiple sites worsen the privacy perception, being considered very low, because the attack surface is increased proportionally to the number of used infrastructures since every site stores all data. The privacy perception of distributing coded data blocks among repositories is high because no single site stores the whole datasets, and thus attackers must compromise more facilities to obtain enough data blocks (still encrypted) to recover the original data.

The platform-as-a-service proposed during the BiobankCloud project allows biobanks to use private clouds and extrapolate their capacity by securely storing data in a cloud-of-clouds [28, 183]. The BiobankCloud PaaS is the first solution for biological data that uses multiple clouds to store private data in these multi-tenant infrastructures securely. It integrates the data warehouse and mediator-based integration models, as well as provides data and function shipping to its users. Additionally, the platform can lookup and fetch data from other biobanks automatically on execution time (i.e., a mediator). It is secure and reliable because data is encrypted before its transfer and no single public cloud stores the whole dataset due to a secret sharing scheme used in this platform.

CHARON [183] is the only multi-site storage system that supports data as big as human genomes mainly because it splits data into small blocks (e.g., 16MB each), but also because it employs optimisations such as, prefetching, caching, and background writes. Additionally, it supports storing different data in diverse locations—private repositories, single public clouds, or multiple clouds. Portions of the BiobankCloud PaaS were converted into spinoffs: HopsWorks [192] processes data-intensive workflows in private repositories and it was transferred to Logical Clocks [173], whereas CHARON [183] was transferred to Vawlt [241].

2.5 Privacy-Aware Storage Techniques

Balancing privacy and utility is one of the main challenges in designing privacy-aware storage and sharing systems for genomic data [242]. On the one hand, open-access initiatives focus on sharing data indistinguishably, rely on improved laws and donor consents, and disregard privacy protection. On the other hand, privacy protection often limits access to data through encryption and restrictive access control.

Privacy and data sharing are not mutually exclusive. Properly discussing and defending privacy encourages the responsible data sharing and extends donors' engagement and trust in researches. Other publications corroborate with the ideas that clearly informing donors about the privacy risks of their choices does not affect negatively their willingness in donating samples [155] and that there is a need for balancing data access and privacy in genomics [242].

For instance, solutions for photo sharing (e.g., social networks like Facebook) have already faced several privacy-related conflicts and policy changes, and life sciences can learn from them. We published a paper [71] that makes an analogy between the privacy aspects of sharing photos and sharing genomes, which contributes to clarifying the privacy risks in the latter. It promotes the accessibility and sharing of human genomes, while advocates their responsible management considering the privacy of sample donors. This comparison is reasonable since sequenced genomes and modern photos are digitised records of real lives. Both contain private information that may compromise people's privacy, and most of the times, their highest value is only achieved when shared with others.

The human genome is privacy-sensitive because it contains personal information and researchers need to access extensive collections of genomes to accelerate medical breakthroughs. The ethical appeal for disclosure stimulates altruistic individuals to donate biological samples for medical and genomic research. However, this point of view must coexist with the ethical discussion on the risks to donors' privacy and encourage the development of secure models to share genomic data [8]. We defined an analogy by comparing the similarities and features of the processes of sharing photos and sharing genomes, which is based on the following aspects:

- Some portions of data are more privacy-sensitive than others.

- One's data may affect the privacy of others.

- Systematically detecting the privacy-sensitive portions of data is feasible.
- After classifying the portions, decide how to share them.
- The impact of data sharing is unpredictable.

Several privacy protection techniques have been explored in the context of genomic data [20, 191]. Three of the most prominent are de-anonymisation, differential privacy, and fully homomorphic encryption. The first method removes every annotation and metadata that may contain personally identifiable information (e.g., names and identification numbers). However, the effectiveness of this technique is limited since human genomic data can be used together with publicly-available information to re-identify their donors [118]. The second states that data resulting from studies must provide no hint on the participants or their individual contribution to the final result [92, 150]. The third studies encryption schemes that allow processing the encrypted data without requiring its decryption [109, 189]. The last two areas are evolving rapidly, but they still lack acceptable performance in practice, limit the utility and accessibility of data, or depend on configurations parameters that are ambiguous.

Additionally, trusted execution environment (TEE) provides an isolated execution space where encrypted data can be securely decrypted and processed, as well as its results are encrypted and can have its integrity and authentication validated [216]. This area is also promising and its utility will improve with the increase in its memory capacity. Nonetheless, recent studies (e.g., [115]) have been trying to integrate them with other techniques (e.g., multi-party computation and blockchain) to compose complete storage and processing solutions for genomic data.

Detecting privacy-sensitive genomic data as soon as it is generated is a long-term ambition from the research and clinical communities [97, 114]. Other works on privacy-preserving genome processing have advocated the partitioning of genomic data, but they assume this is done manually [19] or by a tool that is out of their scope and would be implemented by someone else [153].

Sedic [153] is a privacy-aware platform that modifies the Apache Hadoop MapReduce to work in a hybrid cloud environment. They compute all privacy-sensitive data in a private cloud or cluster and the non-sensitive portion in public infrastructure. However, they forward the burden of labelling sensitive data to the user-side and do not propose any solution on how to do that in genomic data.

In another work [19], the authors propose a privacy-preserving method to process mapped short reads in a scenario of personalised medicine. They encrypt all genomic data before storing it, mask a few genomic variations based on donor's preferences, and limit the access of medical units to portions of these aligned short reads. There are at least three main issues of their approach. First, they do not provide an automatic selection of the genomic variations that will be masked. Manually marking parts of the genomic data as private may lead to mistakes and misguided decisions that may cause leaks on already known privacy-sensitive genomic data. Second, they suggest the selection of only a few variations to be masked, while this protection is insufficient to preserve the donor's privacy completely [169]. Third, their masking mechanism roughly filters out the selected genomic variations, which makes several medical analyses impractical.

Portions of human genomic data are considered privacy-sensitive when they disclose non-consented personal information about an individual. By analysing many privacy attacks [118, 132, 191, 195, 224, 250] one can identify attackers' main goals and the threats they exploit. First, attackers' methods can be divided into two categories depending on their primary goal: obtaining the identity of genome donors [118]; or donors' sensitive personal information (e.g., health-related data) [132, 195, 224, 250]. Second, attacks can be divided into three categories regarding the threats they exploit: short tandem repeats [118], disease-related genes [195]; or genomic variations [132, 224, 250]. The studied attacks use only public data (including genomic sequences), but they can also be applied to any sequence obtained from private or external infrastructures.

2.6 Data-Reduction Techniques

Cost efficiency impacts the feasibility of storing datasets as big as collections of human genomes. Furthermore, storage solutions must benchmark their cost efficiency to not become a burden for institutions [50] and to make dependability and security affordable. In this section, we describe the most used data reduction techniques to increase the cost efficiency of storage solutions. Data reduction is comprised mostly of compression and deduplication, but erasure codes are an additional technique of interest to reduce the overhead of data that must be replicated or dispersed.

Data compression. This technique reduces the size of data by finding information that can be stored in a representation smaller than the original one. Standard compression techniques are inefficient in genomic data due to the specificity of this data [116] (see Sections 4.1 and 4.2). There are several proposals of application-specific algorithms for genomic data [23, 87, 248].

Referential compression is the state-of-the-art approach for highly similar genomes (e.g., humans have at least 99.9% of genetic similarity [243]) since it generates an output file containing only the differences between two input sequences (a to-be-compressed and a reference sequence) [248]. This technique, supported by Huffman and delta encodings [218], achieves high compression ratios (up to 1000$\times$) when reducing the size of contiguous assembled sequences in the FASTA format [11, 86, 247]. However, sequencing data from human genomes (i.e., the FASTQ format) is more complex than its assembled version (i.e., FASTA) and requires additional preprocessing (e.g., [76, 119, 246]) and compression techniques.

Many algorithms favour maximising compression ratio, which usually comes with penalties in (de)compression speed [177]. This decision is justifiable when data is intended to be archived. However, the decompression speed becomes a bottleneck in cases where compressed data is read from remote storage systems and needs to be decompressed and read several times. This issue justifies why many real-world solutions (e.g., 1000 Genomes

Project [96]) prefer generic compression algorithms that decompress fast (e.g., GZIP [88]) rather than those that only compress more.

Section 4.2 contains a detailed comparative evaluation of several generic and specialised compression tools, namely: GZIP [88], pigz [4], BSC [113], ZPAQ [178], SeqDB [133], DSRC2 [213], Quip [151], FQZcomp [38], FaStore [214], and SPRING [49]. From this analysis, it is possible to verify that there is an opportunity in exploring data deduplication to leverage inter-file similarities in sequencing genomic data while achieving a better restore performance than the one from compression.

Data deduplication. This method reduces the storage requirements by eliminating unrelated redundant data [202]. It roughly consists in: finding copies of identical data, storing it once, and replacing all data copies by pointers to that single instance.

FASTQ files contain unaligned entries, and each one of them is composed of unique blocks that include the identifiers of donors and the identifiers of every FASTQ entry (see Figure 2.2). This uniqueness from the entry identifiers makes standard identity-based deduplication inefficient in reducing the size of genomic data. Section 4.3 discusses in more detail why identity-based deduplication fails to reduce the size of files for human genomes and identify similarity-based deduplication and delta-encoding as an opportunity to reduce the size of FASTQ files. To the best of our knowledge, there is no deduplication solution targeting specifically this type of data.

Erasure codes. Erasure codes are a broadly used technique to make data replication affordable [207, 209]. While standard backup and redundancy techniques replicate whole datasets in several storage locations, erasure codes require the additional storage of only a few parity blocks. This approach results in reduced storage and network requirements [252] and incurs in overheads smaller than the doubling or tripling the size of data in replication.

2.7 Storage Auditability

Security, dependability, and privacy protection are preventive measures, while accountability and auditability act as deterrent measures—and complement preventive ones. Auditability and accountability are features employed to increase users' trust in the system. Auditability ensures events are "loggable" while accountability ensures that events deemed important are logged and not missed [157].

Accountability ensures all actors and actions performed on the data have been "accounted for", i.e., persistently recorded as evidence. It focuses on making systems' components accountable in a way their actions become non-repudiable [122, 257]. Works in the accountability literature have discussed generic scenarios for networked systems [123], described the necessary cryptographic building blocks [122, 257], or how pieces of evidence should be stored and protected [123].

Auditability is the relative ease of auditing a system or an environment. Storage systems must maintain logs and features that enable the efficient auditing of processes within it. In private infrastructures, one may provide auditability using existent solutions [18, 110] or implementing custom ones [43]. In single public clouds, users must trust in specific solutions from providers willing to make auditability easier (e.g., AWS provides the CloudTrail tool [14]). External trusted entities supervise public clouds, and providers try to comply with proposed standards and regulations [59]. Thus, public clouds have to provide accountability.

Several auditing schemes were proposed to verify the integrity of data stored in multi-tenant external infrastructures [159, 208]. They mainly focus on cryptographic techniques to produce retrievability proofs without the need to fetch all data from the system (e.g., [152]) or on providing public integrity verification (e.g., [249]). However, to the best of our knowledge, *there is no previous work on auditing who has effectively read data in a dispersed storage*.

Several other works have explored the space complexity of fault-tolerant register emulations (e.g., [5, 47, 51, 52]), including disintegrated storage [27] (e.g., [15, 28, 227]). However, *none of these works focuses on the requirements for auditing read accesses in a storage system despite the existence of faulty storage objects*.

2.8 Discussion

This thesis intends to enhance traditional storage ecosystems with privacy-aware, data-reduction, and auditability techniques to enable the efficient use of existent infrastructures to store human genomes. In this section, we summarise the open problems presented in this chapter and emphasise the opportunities arising from them.

The *privacy protection* open problem incites making storage systems aware of the privacy-sensitivity of each data portion and protecting it appropriately. Complementing the ideas from previous works [19, 153], there is an opportunity to detect and label privacy-sensitive genomic data automatically. It should use all the knowledge currently available about these sequences and be insusceptible to false negatives. Furthermore, the utility of data must be protected by allowing users to improve the security and control over the privacy-sensitive portion of genomes rather than throwing it away. Thus, the privacy protection open problem can be addressed by modelling and implementing a privacy-detector with the closest to complete knowledge on the privacy of genomic data as possible now, and redirecting the different data portions to the proper data flows.

The *cost-efficiency* open problem motivates the design and development of application-based algorithms to compress and deduplicate human genomic data. Algorithms should focus mostly on FASTQ files because otherwise a researcher that fetches aligned or assembled data from several repositories would obtain data processed with different algorithms and parameters and would have to reprocess all samples to normalise datasets. Additionally, they should focus more on lossless compression due to the completeness-related reasons previously mentioned [39]. The compression must be at least faster than sequencing machines since it is supposed to be done once on the repository side, and should be parallelised to be even faster. The decompression step must be fast and must use very few resources because it is supposed to be done several times on the client-side.

For the data deduplication, there is an opportunity to develop an application-based algorithm that parses compressed FASTQ files properly and deduplicate the internal components of FASTQ entries. The most efficient erasure codes techniques for storage must be considered in a storage system for genomic data when storing it in multiple infrastructures. Erasure codes reduce the overhead of storage redundancy but do not reduce the original size of data—which would be achieved with data compression and deduplication. This thesis uses CHARON [183] as a black-box solution for the STORAGE phase of our pipeline because it supports all the required interactions with storage infrastructures and provides drivers to many diverse data repositories, including a cloud-of-clouds.

The open problem of *auditing* data access in dispersed storage is an opportunity that remains unexplored in the literature. Additionally, the differentiated portions of data belong to a single file semantically. The interfaces must consider auditability over a complete genome file, or separately for a single portion of it (e.g., only the privacy-sensitive portion). In the case of multiple-site storage (e.g., a cloud-of-clouds), several blocks belong to the same file and must be transparently considered as a single file. Deduplication can increase even more the complexity for accounting data accesses. Thus, any storage system for genomic data must be aware of all these challenges.

The next three chapters address each one of the open challenges presented in this chapter. Additionally, Chapter 6 integrates all of them in a complete storage pipeline that enables the efficient, dependable storage of sequencing data from human genomes in public clouds.

3 Detection of Privacy-Sensitive Portions in Human Genomes

A human genome can uniquely identify its owner and reveal information about him and his relatives, even for some past and future generations [20, 191]. Additionally, portions of biological data may provide hints on an individual's health status with high confidence levels. Many studies identified attacks to individuals' privacy based on genomic data and other publicly-available information [118, 132, 195, 250]. They have the objective of non-consented disclosure of personal information of individuals from their genomic data, and it can be divided into two classes: threats leading to re-identify donors of anonymised DNA sequences, based on genetic genealogy profiling [118]; and threats leading to the inference of private and sensitive information (e.g., victim's health status) from (re-)identified DNA sequences, based on disease-related genes [195] and genomic variations [132, 250]. These attacks must be efficiently addressed to avoid a rollback on the trend to share DNA sequences, which would hurt genomic studies, or even harden regulations governing genomic data protection [40].

Detecting privacy-sensitive genomic data as soon as it is generated is a long-term ambition from the research and clinical communities [97, 114]. Previous works on privacy-preserving genome processing have advocated the partitioning of genomic data, but assume this must be done manually [19] or by a tool out of their scope [153]. To the best of our knowledge, our work is the first to provide a comprehensive privacy-aware detection method that enables users to implement such partitioning automatically. This chapter proposes a *privacy-sensitivity detection* scheme composed of:

- algorithmic solutions to retrieve privacy-sensitive nucleic and amino acid sequences automatically, with parametric and evolving sensitivity;

- a privacy-sensitivity detection architecture to systematically recognise privacy-sensitive genomic data from the source (e.g., sequencing machines).

Conceptually, the detector has a clear mission: given a DNA segment of predefined size s, detect whether this segment may contain known privacy-sensitive information or not (see Figure 3.1). It does so based on a database of published signatures or patterns of privacy-sensitive nucleic and amino acid sequences (a *knowledge database*). Although conceptually simple, this approach meets at least two feasibility challenges:

(i) *Accuracy:* how to automatically classify DNA segments as privacy-sensitive or non-sensitive with high sensitivity and specificity?

(ii) *Performance:* how to implement a scalable detection solution that supports the high-throughput of modern sequencing machines?

The output from our solution is divided into two subsets: one with the privacy-sensitive portion of input data and the other containing the non-sensitive part. We initially foresee the following scenarios where our detector can be employed, but others may arise in the future:

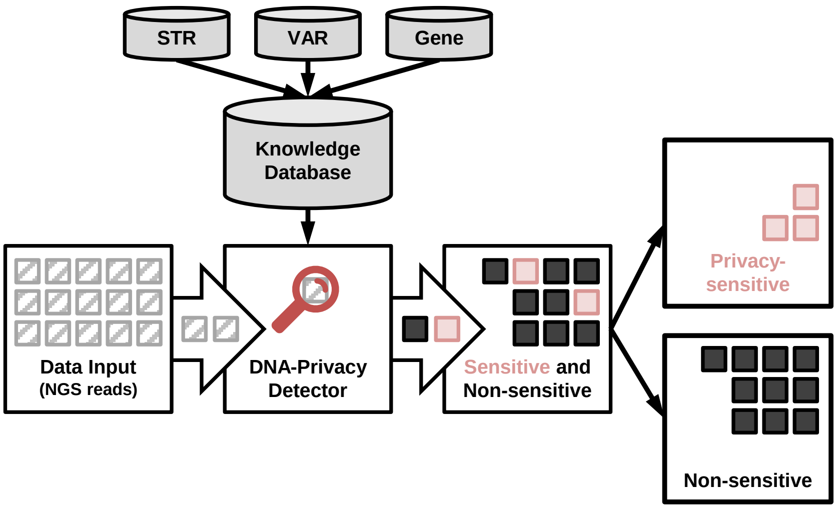


Figure 3.1: An overview of the method for detecting privacy-sensitive genomic data.

- *Data segregation*: one may store and analyse privacy-sensitive data in local, private infrastructure, while he/she may use external, multi-tenant infrastructures (e.g., public clouds) to work with the non-sensitive data.
- *Data outsourcing*: one may store the whole dataset in an external, multi-tenant infrastructure—if he/she applies strong, expensive security premises in the expectedly smaller privacy-sensitive portion, and applies more affordable security techniques in the non-sensitive portion. Additionally, homomorphic encryption [109, 189] or trusted execution environments [216] can be used in the smaller privacy-sensitive portion to compute it securely in the external facility (e.g., [240]).
- *Data masking*: One may filter out one of the portions. For example, filtering out the larger non-sensitive portion allows users to store and process only the smaller privacy-sensitive portion, which is enough for several important analyses (e.g., personalised medicine).

Note the method proposed here addresses the challenge of systematically detecting privacy-sensitive DNA sequences, whereas what should be done with the two output subsets is independent of our work and can have different implementations according to each use case.

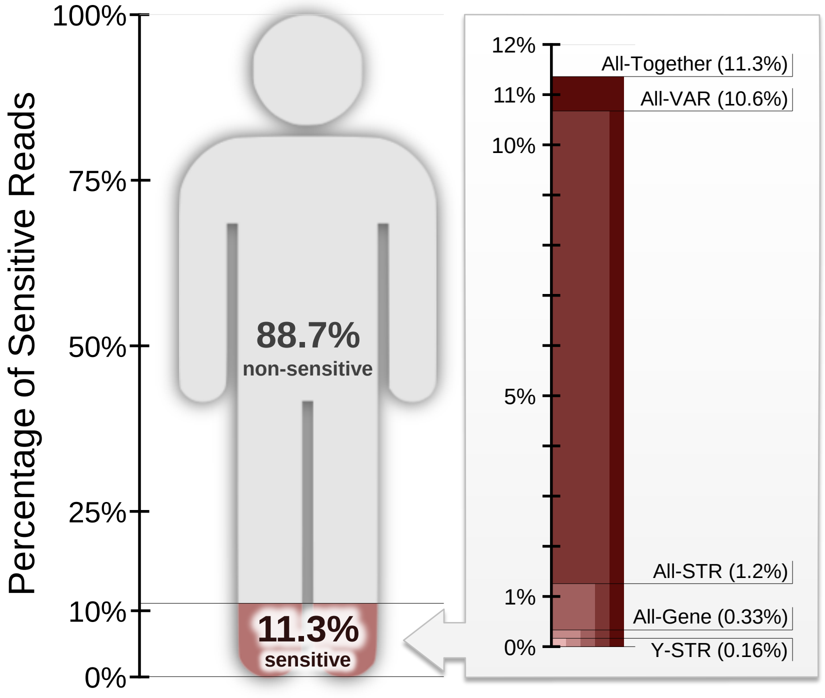


Figure 3.2: Percentage of privacy-sensitive sequences in entire human genomes considering different detector knowledge databases. The most complete dataset (All-Together) lead to the detection of only 11.3% of the sequences as privacy-sensitive.

As a proof of concept of the accuracy and completeness of our approach, we have built a knowledge database from known short tandem repeats (small repeated strings), disease-related genes, and genomic variations currently available in public databases. Using all this information and being conservative about what can be considered private, only 11.3% of the sequences of a human genome are detected as privacy-sensitive (see Figure 3.2).

The originality of our work is in adapting existent enterprise privacy-preserving solutions and intrusion detection techniques to the genomics area and combining them with different known privacy-sensitive information to protect individuals. By identifying the privacy-sensitive sequences using our solution and protecting them, one neutralises the existent threats of re-identifying individuals [118] and of inferring private information about them [132, 195, 250]. Moreover, by using the large body of knowledge about the privacy sensitivity of human genomes available today, it was possible to create a complete privacy detector. Despite this completeness, the detector knowledge database can be automatically updated to address future attacks as new privacy-sensitive sequences are identified, making it generic and evolvable [71]. It does not become outdated since public databases can be automatically tracked for updates as they evolve. From the performance viewpoint, we also show that our system can withstand the evolution of NGS and scale-out.

The remainder of this chapter is organised as follows. In Section 3.1, we describe our privacy-sensitive detection mechanism, its internal methods to construct a knowledge database, and the implementation details. Section 3.2 evaluates and shows the efficiency of the proposed method, Section 3.3 discusses the completeness of the used knowledge databases, and Section 3.4 presents the final remarks of this chapter.

3.1 The Detection Method

The detection of privacy-sensitive genomic sequences is important both for research and clinical communities. This section presents a mechanism to systematically detect privacy-sensitive genomic sequences and its internal methods for creating a knowledge database with this type of sequences.

3.1.1 Overview

This chapter proposes to enhance the NGS production cycle with a mechanism called DNA-Privacy Detector, which, taking short DNA sequences as input, automatically decides which ones represent privacy-sensitive information. An overview of our detector architecture appears in Figure 3.1.

The detector decides based on a database of published signatures or patterns of privacy-sensitive nucleic and amino acid sequences (a *knowledge database*), and forwards input DNA sequences alternatively to the privacy-sensitive output or the non-sensitive one. The knowledge is defined by statistical heuristics and previous data about a context or a population. Obtaining the privacy-sensitive sequences is not trivial since biological databases provide unharmonised formats and interfaces. Additionally, one needs to add all combinations of size *s* from a sequence to the database, including all their known DNA flanking sequences and mutations (as explained in the next section). The obtained sequences are considered privacy-sensitive and must never be sent to the non-sensitive output stream. Similarly to the signature lists of computer intrusion detection systems (IDSs) [81], the knowledge database can be updated as new patterns are discovered.

Our mechanism can analyse input datasets at any time, but the most powerful and effective configuration would be when input sequences come directly from NGS machines as they are generated. The detector can also be integrated into the NGS machine to automatically detect privacy-sensitive sequences and add a marker with this information in the comment line of a FASTQ entry (the NGS read) [61].

3.1.2 Privacy-Sensitive Human Genomic Data

Portions of human genomic data are considered privacy-sensitive when they disclose non-consented personal information about an individual. We analysed many privacy attacks [118, 132, 195, 250] to obtain the attackers' main goals and the threats they exploit. First, attackers' methods can be divided into two categories depending on their main goal: obtaining the identity of genome donors [118]; or donors' sensitive personal information (e.g., health-related data) [132, 195, 250]. Second, attacks can be divided into three categories regarding the threats they exploit: short tandem repeats [118], disease-related genes [195]; or genomic variations [132, 250]. The studied attacks use only public data (including genomic sequences). However, they can also be applied to any sequence obtained from private or external infrastructures.

The next three subsections introduce the threats, describe how they are exploited by attackers, and explain how our detection method, with the proper security techniques, prevents them from succeeding. We opted for a very conservative approach in our solution since it stores small sequences of size *s* in the knowledge database instead of single nucleotides, and detect them as privacy-sensitive sequences independently from their positions in the genome. Configuring the size *s* and using sequence alignment may interfere on the conservativeness and accuracy of the method.

3.1.2.1 Short Tandem Repeats

Short tandem repeats (STRs) are small repeated strings comprised of A, C, G, and T characters. For instance, the STR called DYS392 is represented by $[\mathrm{TAT}]_n$, and an individual who contains a sequence like *cgacTATTATTATTATcgca* in his DNA will score 4 for DYS392 in his profile. Genetic genealogy profiles are employed in forensic identification, paternity tests, missing people investigations, among others. A profile from paternal lineage is a set of counters of how many times each selected *short tandem repeat* from the Y chromosome (Y-STR) appears in an individual's DNA.

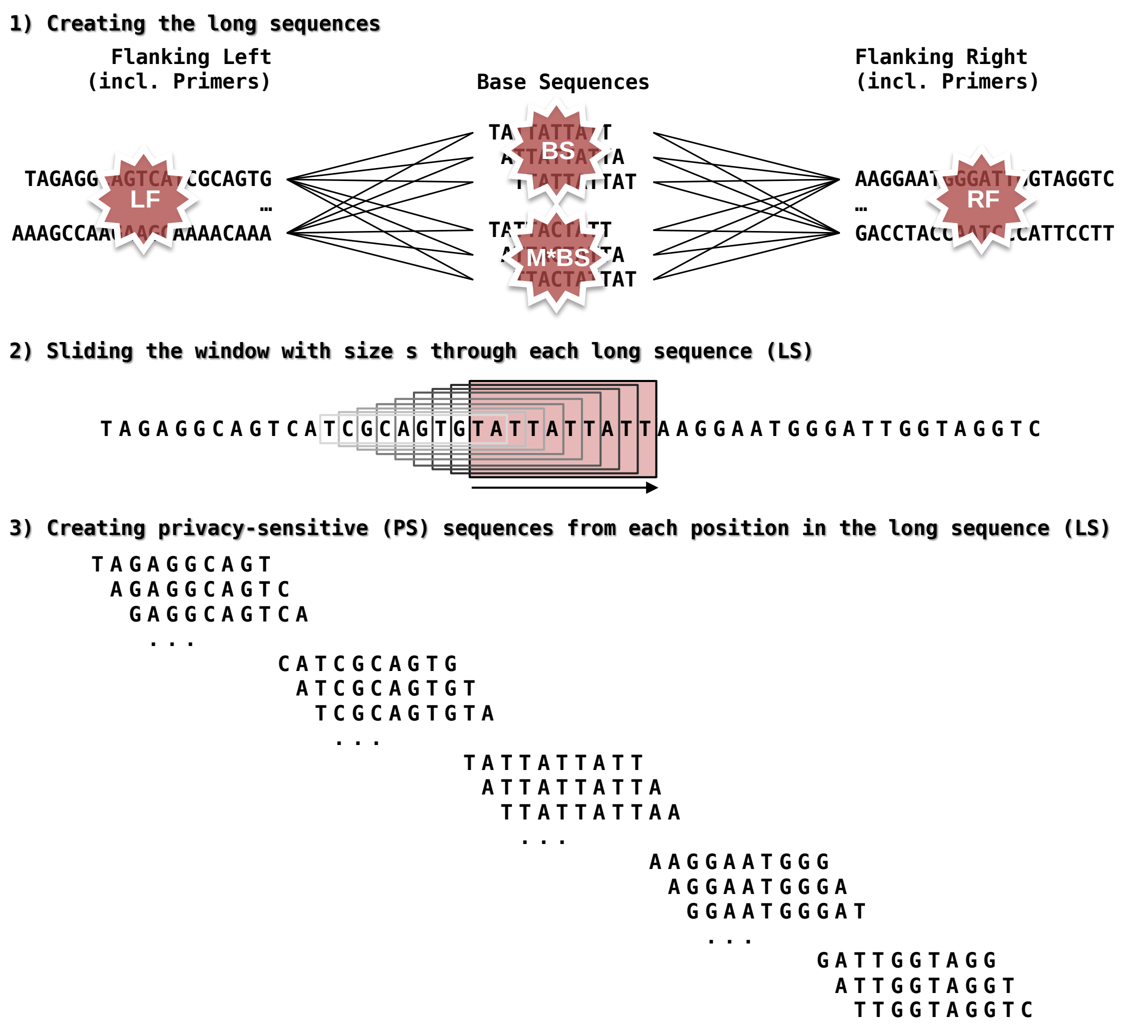
In the United States, a core set of thirteen STR markers are being used to generate a nationwide database for forensic identification [45], called FBI Combined DNA Index System (CODIS). Other countries and organisations such as EU, UK, DE, and Interpol also selected their sets of core STR markers to identify individuals. There are registers of several known STRs available in public databases, such as the STRBase [215] and the TRDB [107]. These databases have thousands of registered STRs, many more than those few core STRs. Since STRs can also contain mutations, the respective databases must store all known variations.

Attack. Gymrek *et al.* [118] described an attack that re-identifies participants of the 1000 Genomes project [96] in early 2013. The attack was based on two facts: surnames are paternally inherited in most human societies; and so are Y-STRs in male individuals [156]. It had two goals: obtain the surname of individuals and triangulate their identity. For surname inference, they profiled Y-STRs of individuals, queried them in public recreational genealogy databases, and obtained a list of possible surnames for the profiles in question. Each query contained registers of about thirty known Y-STRs in this case. Authors queried the Y-STR profiles in the YSearch [108] and the SMGF [226] databases, and recovered the correct surname in 12% of cases (with 82% of confidence). For triangulating identities, authors combined the obtained surnames with age and state, which were considered public information that did not need to be suppressed in anonymisation processes. A query on the US census by year of birth and state results in $60,000$ US males in 50% of cases. Aggregating the surname to the query shrinks the result to only twelve males on average. Each surname inference breached the privacy of nearly sixteen individuals. Although the result of 12% appears to be unimpressive, it means that from the $1,092$ participants of 1000 Genomes project, 131 of them will never recover their privacy, nearly $2,100$ participant's relatives had their privacy breached [118], and the disclosed data will continue available for their descendants.

Solution. One can protect genomic data against this attack with our approach by registering information in the knowledge database about all known Y-STRs, detecting privacy-sensitive sequences containing them, and segregating them from the non-sensitive output stream. We aggregate the following information for each known Y-STR:

1. The STR regular expression, e.g., $[\mathrm{TAT}]_n$ for DYS392.
2. The minimum and the maximum number of repetitions observed so far, e.g., 6-17 for DYS392.
3. All known mutations of the STR [100].
4. All observed left and right flanking sequences, which are commonly found either before or after the STR. For instance, 5'-TAGAGGCAGTCATCGCAGTG-3' is a primer sequence observed before DYS392 and 5'-AAGGAATGGGATTGGTAGGTC-3' after it.

Figure 3.3 presents the entire process of generating the privacy-sensitive entries for the knowledge database from the obtained information about each Y-STR. Since an STR is a repetition of a small string, a sequence can start with each different letter from that small string. For example, in the case of DYS392, one has three possibilities for base sequences,



1) Creating the long sequences
Flanking Left (incl. Primers)
Base Sequences
Flanking Right (incl. Primers)
LF
BS
M*BS
RF
2) Sliding the window with size s through each long sequence (LS)
T A G A G G C A G T C A T C G C A G T G T A T T A T T A T T A A G G A A T G G G A T T G G T A G G T C
3) Creating privacy-sensitive (PS) sequences from each position in the long sequence (LS)

Figure 3.3: Creating the blacklist sequences for the method based on short tandem repeats.

with strings starting with TAT, ATT, or TTA. Considering sequences with $s = 10$ base pairs, any read with this size matching this entry entirely should be only TATTATTATT, ATTATTATTA, or TTATTATTAT, which are called base sequences (*BS*). For each known mutation (*M*), one has to create the respective base sequences, i.e., three for each mutation of DYS392. Left and right flanking sequences (resp. *LF* and *RF*) are concatenated with each base sequence (*BS*), creating all possible combinations, which are called long sequences (*LS*). Each long sequence is composed of a *LF*, a *BS*, and a *RF* and gives place to small sequences of size *s* (the size of our entries) created by sliding a window of the same size, which are called privacy-sensitive sequences (*PS*).

While attacks based on paternal lineage use only Y-STRs, a hypothetical attack employing forensic identification methods may use STRs from all chromosomes, for example, when comparing the victim's genome from her blood sample to a database with identified genomes. Thus, one can also register STRs from all chromosomes in the detector's knowledge by using the same algorithm as for Y-STRs.

3.1.2.2 Disease-Related Genes

Some portions of a genome, called exons, are translated to RNA and later to amino acid sequences, which finally encode proteins (see Section 2.1). Proteins provide the functional elements of a biological system, which can have an essential role in many human diseases. The presence or absence of specific gene mutations may be indicative of the predisposition to or actual contraction of certain diseases. Masking disease-related genes is thus a viable approach to protect the privacy of individuals that had their genomes (re-)identified. This solution focuses mainly on protecting individuals' privacy when they have DNA sequences leaked by unauthorised disclosure (for example, through the first attack [118]) to preclude attackers from obtaining extra information about individuals, namely their health status.

This method is very important in cases where sample donors consent to analyse or store their genome in external infrastructures, but wish to mask or apply stronger protection on information about some specific diseases. One real example is Dr. Jim Watson's case. He is a co-discoverer of the double-helix structure of DNA and his complete genome was sequenced and published in 2008 [254]. However, Dr. Watson requested the retraction of all information about the *APOE*, a gene associated to the Alzheimer disease, before the public release of his genome.

Attack. The adversaries, after obtaining an identified genome or portions of it, may learn additional information about the victim's health by using the presence or absence of specific disease-related genes in it. The attack process resembles existent direct-to-consumer health-related genetic profiling [111], which informs about an individual's probabilities of contracting a disease or any other gene-related information. These tests do not provide diagnosis, but in the wrong hands, they still may cause harm to a victim's reputation or otherwise disadvantage her.

Solution. Detecting all known disease-related genes is a possible solution after obtaining the complete list of these genes and their sequences. There are dedicated databases that correlate genes and diseases (e.g., GeneCards [217]). These databases allow users to retrieve the names of all currently studied genes, or the few genes related to some specific disease.

Again, the detector is capable of handling datasets obtained from any set of genes present on any database. We use the GeneCards database [217] only as a proof-of-concept. From the estimate of existing 20k–25k human genes (the exact number is not yet know [56]), the GeneCards database contained the name and data about disease correlation of $19,231$ genes at the time of this study. A more conservative approach might detect all known human genes as privacy-sensitive independent if they were already correlated to a disease or not. For example, one may use the OMIM database [198] to get the sequences of all known human genes (approximately 23k). In this work, we retrieve the genes from GeneCards, obtain their accession numbers, and retrieve their sequences from UniProt [238]. For each gene sequence, one must break it into smaller sequences of size $\frac{s}{3}$ (e.g., ten amino acids) and insert them into the knowledge database.

Masking disease-related genes is sensitive to imputation methods based on linkage disequilibrium between genomic variations. In 2009, Nyholt *et al.* [195] described the method that allowed them to recover Dr. Watson's masked *APOE* status. They respected Dr. Watson's request for *APOE* anonymity in the public manuscript. Still, the main goal of the authors was to highlight the challenges concerning the privacy and the complexities of informed consents. Note the next method is based on genomic variations, the very same information type used in Nyholt *et al.* [195], and thus one will be able to prevent also that attack from succeeding.

3.1.2.3 Genomic Variations

Humans are 99.9% genetically similar to one another. However, small portions of the remaining 0.1% can uniquely identify whom a DNA belongs to [20, 243]. There are numerous studies about *genomic variations* present on individuals. Allele-frequency analysis [235] roughly identifies how common or rare the sequence variants of an individual are, in comparison to a specific population. Genome-wide association studies (GWAS) correlate several traits with these genetic variants common in a population [130].

Attack. In 2009, Wang *et al.* [250] showed that it is possible to acquire knowledge about targeted individuals from statistical results publicly released by GWAS studies. More precisely, the attacker is assumed to have a physical sample of the victim (e.g., blood) and genotyped as few as a couple of hundreds of her variations, for example, single nucleotide polymorphisms (SNPs). Then, the attacker goes on to determine the victim's presence in the GWAS' case group, which indicates her contraction of a disease. This result extended another work [132], published one year before, which shows a similar attack to other conventional techniques employed in genetic studies (e.g., microarrays). Another study [169] states that obtaining 30 to 80 statistically independent SNP positions is enough to identify a single person uniquely Finally, the study from Nyholt [195] described a genetic imputation method based on linkage disequilibrium between genomic variations, which allowed them to infer a masked gene of a genome by interpreting neighbouring variations present on it.

Solution. Detecting all known genomic variations of an individual is a feasible approach to prevent such attacks. The knowledge of all genomic variations (e.g., SNPs, indels, substitutions) within a population is as complete as the allele frequency (AF) analysis performed in this population. A file in the Variation Call Format (VCF) contains a table with all variations resulting from the AF analysis and the occurrence of them on each individual from the studied population.

We employed the AF analysis of 1000 Genomes project [235], since it is one of the most important AF studies freely available on the internet. It contains 39.7 million variations of $1,092$ individuals from 14 populations. We employed it as a considerable use case, though it still does not cover 100% of variations of all those samples [235]—there are several extremely rare mutations that are yet to be documented. However, our methodology is generic and evolvable: the detector is again not limited to this specific dataset since it supports any other AF analysis of any population, independent of coverage of variations. Additionally, our framework could be regularly used in biobanks and hospitals, which could create AF analyses of their private sample collections and use the resulting data to generate knowledge for the detector. It is important to remark that the more complete the sample collection and AF analysis, the better privacy protection our solution provides.

The knowledge construction in this algorithm starts with obtaining a VCF file. For each genomic variation present in this file, one extracts some data fields about it, for example, the chromosome and position in which it appears, as well as the reference and variation alleles. After extracting this information, one searches the chromosome position in the reference genome used by the AF study and concatenate the $s-1$ left flanking nucleotides with the variation allele and the $s-1$ right flanking neighbours. Finally, one adds each sequence of s base pairs to the knowledge database, executing a process similar to the sliding window from the bottom part of Figure 3.3.

Detecting and protecting all known genomic variations with this method also neutralises attacks by genetic imputation using those variations neighbouring the masked genes [195], as Dr. Watson's case, allowing the safe use of the previous method: detecting disease-related genes.

3.1.3 Implementation

After obtaining the knowledge database of nucleic and amino acid sequences for the privacy-sensitivity detector, a second challenge that needs to be addressed is how to implement this component efficiently and effectively. The technical challenge is three-fold:

- Questions such as *"Is GCTAGCTAGCTAGCGGGGCCCTAGCTAGCT privacy-sensitive?"* cannot be immediately answered as there is no available data label or pattern to detect privacy-sensitive DNA sequences, which differs genomic data from enterprise data (e.g., `SocialSecurityNumber = 123-45-6789`).

- Obtaining the privacy-sensitive sequences is not trivial, and one also needs to lookup large amounts of data—including DNA flanking regions and all combinations of size *s* from a sequence.

- Useful solutions in the genomics area must support the high throughput of NGS machines and scale-out while searching input sequences in the whole database (i.e., tens of GBs).

Several data structures can address the mentioned challenges, and Bloom filters [36] presented the best results. Thus, our solution uses a Bloom filter data structure to store the knowledge database of privacy-sensitive sequences. Bloom filters (BF) are space-efficient probabilistic data structures that can be used to represent sets compactly. In a nutshell, when performing lookups, BFs return `false` if an entry *definitely does not belong to the set* or `true` if it *probably belongs to the set*. Notice that false-positives do not affect the detector's effectiveness (its privacy guarantees), only its efficiency (wrongly classifying a sequence as privacy-sensitive overloads the respective output stream, versus the cheaper and more available non-sensitive).

Our knowledge database may contain both nucleic sequences (i.e., consisting of As, Cs, Gs, and Ts) and amino acid sequences (i.e., composed of letters from the IUPAC nomenclature [89]), but lookups in the detector receive only DNA sequences of size *s* as input (30 nucleotides in our case). One first searches the original DNA sequence, and if it is not found in the Bloom filter, then he translates the 30-bp sequence to the correspondent ten amino acids sequence and lookup again, returning the respective result. If the knowledge contains only DNA sequences, then one only does the first lookup and return. If the knowledge contains only amino acid sequences, then one directly translates each received DNA sequence to the correspondent amino acid sequence, lookups, and returns.

An interesting aspect of our Bloom filter implementation is that, due to the large number of entries in the knowledge database, it will be using an unconventionally big Bloom filter, sizing several gigabytes. We chose and improved a BF implementation called Java-LongFastBloomFilter [174], which is a bigger and faster Java solution than most BF implementations. Bigger because it uses numbers of `long` type (64-bits) to index the bit set of BFs, while others still use numbers of `int` type (32-bits). This implementation is faster for two reasons. One, it uses a 64-bit Murmur hash, which is one of the fastest non-cryptographic hash functions with a good random distribution of regular keys. Two, it has an algorithmic optimisation that allows reducing (by configuration) the number of hash keys needed to index an entry by increasing the BF size up to a configurable size (in terms of percentages). We modified two aspects from the mentioned BF implementation: we made the *add* and *contain* methods thread-safe and added an argument in the constructor to configure the value considered on the performance optimisation.

The source code of our implementation is available under the Apache License (v2.0) in GitHub [62] together with additional descriptions for reproducibility on the steps performed to obtain the privacy-sensitive datasets to the knowledge database. Additionally, we may provide the complete datasets (more than 60GB) used in this chapter upon request.

3.2 Experimental Evaluation

This section presents results showing that the proposed detection method is efficient in terms of *privacy-sensitivity of genomes*, *memory space required for our Bloom filter*, and *throughput performance*.

3.2.1 Experimental Setup

The implementations of the previously described methods, henceforward referred as STR-, Gene-, and VAR-based, generate different datasets of sequences. The STR-based method generates two datasets containing sequences of 30 nucleic acids each, namely:

Dataset	Method	Acid type	Entry size	Entries	Text file size
Y-STR	STR-based	Nucleic	30	0.5 M	15MB
All-STR	STR-based	Nucleic	30	22 M	660MB
All-Gene	Gene-based	Amino	10	8.7 M	87MB
All-VAR	VAR-based	Nucleic	30	1,147 M	34.4GB
All-Together	All	Both	10 and 30	1,178 M	35.1GB

Table 3.1: The different privacy-sensitive datasets considered in this study. Our partitioning method uses them as a knowledge database to decide if a DNA segment is privacy-sensitive or not. For each dataset, we present the approach used to obtain it, the acid type, size and number of entries, and the dataset size when using a text-based representation.

the Y-STR and All-STR, which contain short tandem repeats from Y chromosome and all chromosomes respectively. The Gene-based method contains one dataset (All-Gene) of sequences composed of ten amino acids each, where all published disease-related genes are added to the knowledge database. The VAR-based method generates one dataset (All-VAR) of sequences also composed of 30 nucleic acids each, where it contains all genomic variations available in the AF analysis of 1000 Genomes project [235]. Table 3.1 contains the number of entries from each dataset to the privacy-sensitivity detector, and the respective size as a text-based input file with one entry per line (considering one byte per character, e.g., UTF-8).

The knowledge data is a set of small sequence entries comprised of 30 base pairs or ten amino acids each. The number of entries in Table 3.1 can be directly translated to the amount of storage space needed for them. For example, if each base pair requires one byte to be stored, the Y-STR dataset would require $1 \times 30 \times 0.5 \times 10^6$ bytes, or 15MB. The Gene-based would require $1 \times 10 \times 8.7 \times 10^6$ bytes, or 87MB. The All-Together would require $1 \times (30 \times 1169 + 10 \times 8.7) \times 10^6$ bytes, or 35.1GB.

We have randomly selected ten donors identifiers from the 1000 Genomes project, which compose the input data to our experiments. The resulting donors were those identified by the numbers NA19788, HG00173, NA20810, HG00339, HG00619, NA20339, HG00475, HG01390, NA19449, and NA12546. We describe the steps performed to obtain the entire genome sequences from these donors in the detector's GitHub page. Each input genome of 3GB is the contiguous genomic sequence in the FASTA format, which is the resulting data from assembling a 30$\times$ coverage sequencing data (e.g., FASTQ files with 180GB).

Our experimental environment is one physical machine that runs all components of our system architecture. This machine is a Dell PowerEdge R410 server, equipped with two Intel Xeon E5520 (quad-core, HT, 2.27Ghz), 32GB of RAM, and a hard disk with 146GB (15k RPM). The operating system is an Ubuntu Server Lucid Lynx (10.04 LTS, 64-bits), running with a kernel 2.6.32-21-server, and the Java version is the 1.7.0_25 (64-bits).

3.2.2 Privacy-Sensitivity of Human Genomes

Our first analysis calculates how much of an assembled genome is considered privacy-sensitive for each knowledge dataset and false-positive rate. We picked the ten mentioned genomes to execute the test, where each genome was split in approximately 103 million sequences of 30-bp each, the equivalent to 3GB. Figure 3.4 shows the average percentages of privacy-sensitive entries from these genomes for different false-positive rates in our Bloom filter.

There is a minimal percentage of privacy-sensitive reads that is independent of the false-positive rate of the Bloom filter, which is in the similar results from probabilities 10^{-6} (i.e., 0.0001%) to 10^{-3} (i.e., 0.1%). It means that one needs to enforce strong security premises in at least 0.16% (4.8MB—without compression) of each assembled genome if using the Y-STR knowledge database, and in at least 11.3% (345MB) if the All-Together dataset is used instead. Segregating 11.3% of each human genome to the privacy-sensitive portion leads to the reduction of almost 90% of data that must be maintained under strong security premises. Due to the high similarity in human genome sequences (more than 99.9%), increasing the number of input samples will not affect the obtained privacy-sensitive percentage significantly.

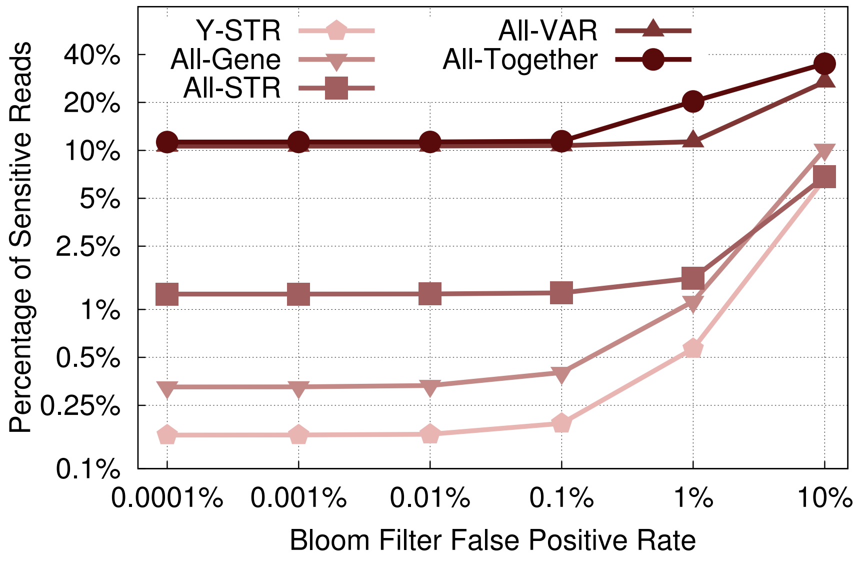


Figure 3.4: Percentage of privacy-sensitive reads for different false-positive rates and knowledge datasets. Both axes are in logarithmic scale.

3.2.3 Space Efficiency

This second analysis estimates the size of the knowledge database in main memory when using the different datasets from Table 3.1. Since our solution uses a Bloom filter, theoretically, the filter size depends only on the expected number of entries and the expected false-positive rate [36]. The expected number of entries is a constant for each configuration/dataset (from All-Together to Y-STR), which appears in Table 3.1. The false-positive rate can be defined by the system administrator to fit the Bloom filter size in the server's memory capacity. Figure 3.5 presents the resulting database size (in megabytes) based on these two properties.

The Bloom filter size using input from STR- or Gene-based methods is 50- to 130-fold smaller than the size if using the VAR-based method. The largest size corresponds to the use of the largest dataset (All-Together) and a false-positive rate of 10^{-6} (one in a million), which leads to a data structure as big as 5.6GB. This size easily fits on the main memory of current commodity servers and is 6× smaller than the original size of the entries of this data structure (35.1GB).

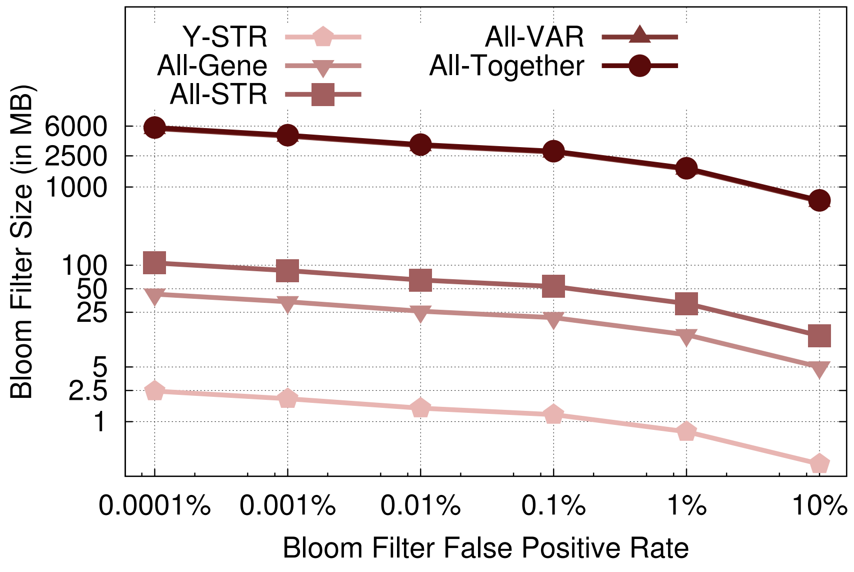


Figure 3.5: Bloom filter size for different false-positive rates and knowledge datasets. Both axes are in logarithmic scale.

3.2.4 Throughput Performance

Our final experiment aims at identifying how many base pairs per second our detector can analyse. Figure 3.6 shows the results considering different false-positive rates in a single core deployment, whereas Figure 3.7 considers different numbers of threads in our multi-core system.

As expected, the higher the false-positive rate, the higher the throughput (Figure 3.6). This relation happens because higher rates lead to smaller Bloom filters, which require fewer hash operations to test whether or not a sequence belongs to the set. There is a difference in terms of performance between methods that use knowledge about nucleic acids and those that use amino acid sequences as entries. This difference occurs because one needs to translate each 30-bp sequences from the testing genome to 10-aa sequences when the knowledge database contains entries composed of amino acids.

Notably, even with our largest and most complete dataset (All-Together with a false-positive probability of 10^{-6}), the detector is still able to analyse more than 13.2 million bp

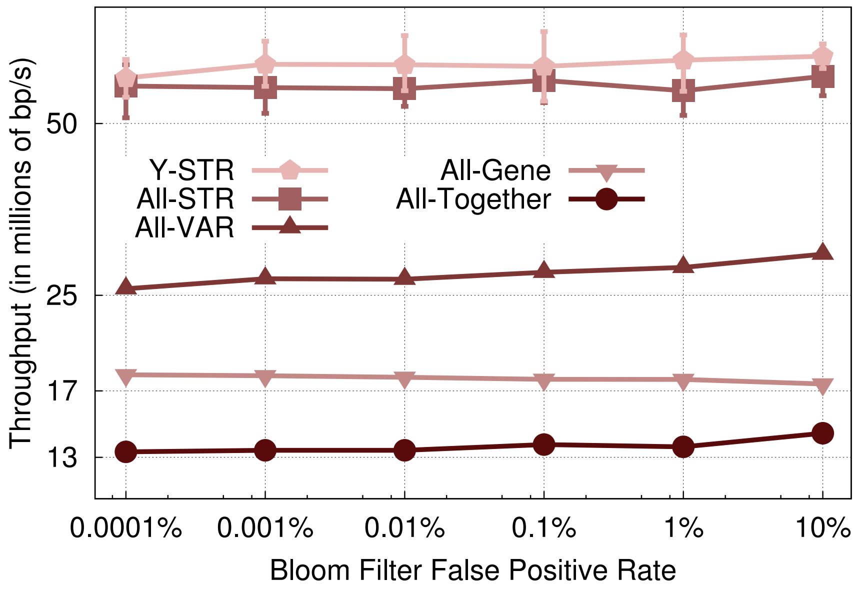


Figure 3.6: Throughput of our detection method considering different false-positive rates (single-core). Both axes are in logarithmic scale.

(i.e., 0.44 million operations) per second *with a single core*. Additionally, the detector evaluates 60 million bp (i.e., two million operations) per second using the other dataset of interest (Y-STR). Considering that high-throughput NGS machines produce 0.3 million bp/sec [172], our solution works 44× to 200× faster. It means the detector could be integrated directly into NGS machines with the addition of minimal hardware. In this way, the machine could generate different FASTQ files containing either privacy- or non-privacy-sensitive reads, or, just add one character to the comment line of the FASTQ entry with its privacy sensitivity.

Obtaining higher throughput is possible when parallelising the detection process. Figure 3.7 shows a scalability test of up to 32 threads in our test machine, which is equipped with two quad-core processors with hyper-threading, i.e., 16 hardware threads. This scalability test considers a false-positive rate of 0.1% (i.e., 10^{-3}), which we consider to be the sweet spot of our design since it minimises the BF size at the same time that it maintains similar percentages of privacy-sensitive reads (see Figures 3.4 and 3.5). The throughput scales up to 480 million bp per second when testing only nucleic acids, up to 110 million for amino acids, and up to 66 million bp per second when using a knowledge database with

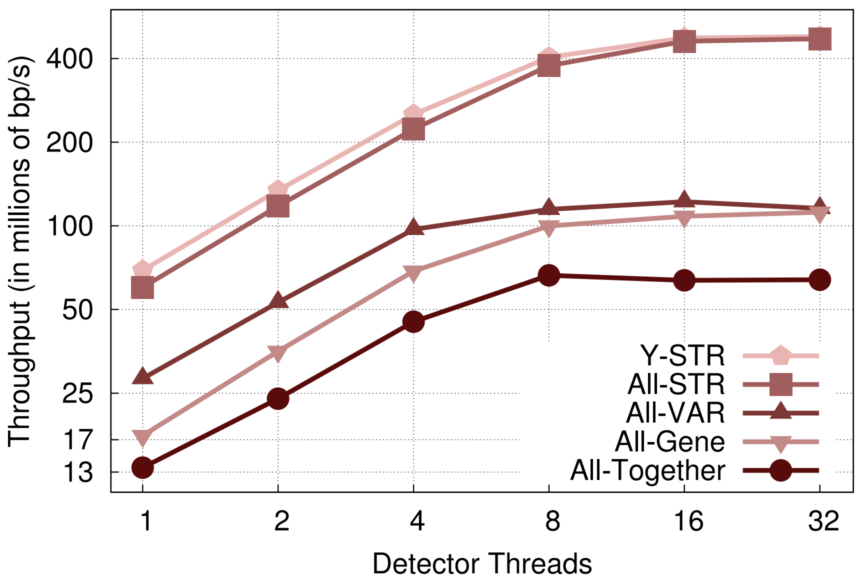


Figure 3.7: Throughput of our detection method considering multiple threads (with 0.1% false-positive rate). Both axes are in logarithmic scale.

both acids. This result corresponds to a speedup of up to eight times, which is the number of cores in use. Our solution works 200× to 1600× faster than current high-throughput NGS machines (0.3 million bp/sec) [172]. Therefore, the detector is not a bottleneck for current or near-future machines.

3.3 Completeness of the Method

The severity of threats to the privacy of genomic information will be amplified by the explosive growth in DNA sequences resulting from NGS, bound to be stored and analysed in external multi-tenant infrastructures [181]. If little is done, a severe leak may reverse the public opinion trend to make DNA sequences public or shareable and hinder genomic studies, or even harden state laws for genomic data protection. Researchers have been alerting about the inevitability of a significant leak of genome information in the near future, and that we should start defining the steps that need to be taken to avoid a public outcry a genome breach might incite [40]. The proposed systematic detection, with the proper protection of DNA sequences as they are generated, can dramatically reduce the risk of such leaks. Thus, it is an essential step towards a sound semantic data ecology by improving privacy protection while enabling adequate mechanisms to promote trusted and secure data sharing [244].

The approach used in our detection method adapts the blacklisting from passive knowledge-based intrusion detection systems (IDS) [81], where network message sequences are continuously filtered by comparison with entries in a database of known attack signatures. Our knowledge database can be considered a blacklist because it stores all known privacy-sensitive sequences (dangerous in the sense of IDSs) that one wants to prevent from being sent to the non-sensitive output stream.

The detector's effectiveness is complete for all "signatures" existent in the database, but only for them. Sequences made vulnerable by new, previously unknown attacks, will not be recognised without updating the knowledge database, pretty much like what happens in IDSs with the notorious zero-day vulnerabilities. In the following, we discuss the risks and implications of discoveries related to the three knowledge sources employed by our technique.

New STR Sequences. As discussed before, STR is a prime method to identify individuals in forensic analysis. Our detector uses the database containing all know STR sequences, but it might fail to detect a sequence containing a yet to be discovered STR. To understand the window of vulnerability posed by newly appeared STRs, we analysed the annual evolution of the number of entries in TRDB [107], the main database for short tandem repeats. From 2003 to 2014, this database evolved from 237k to 238k STRs. This increase means that in eleven years, the TRDB registered 1k novel STRs, which represent a growth of only 0.42% in its entries. It suggests that finding many novel STRs is unlikely, even with the explosive growth on the number of whole-genome studies due to the introduction of NGS methods. Additionally, the genetic genealogy databases employed in the re-identification attack [118] use a small static set of STRs (i.e., few dozens) to profile individuals. They probably will not increase the number of profiled STRs since it would require the reanalysis of all participants. Finally, if an attacker discovers novel STRs, he will be able to harm victims only after these databases reanalyse the victim's DNA or profile victim's relatives with the newly identified STRs, which is also unlikely.

New Disease-Related Genes. The main risk of detecting privacy-sensitive sequences using the disease-related genes knowledge database is that it might not identify genes that were not yet linked to some disease. As with STRs, we also analysed the annual evolution on the number of entries in the GeneCards database [217] (the database we used in this chapter to obtain the known disease-related genes). Our analysis shows an impressive evolution from 3k disease-related genes in 2005, to more than 19k in 2014, which corresponds to a growth of 83.3% in nine years (mostly due to NGS). Given that researchers estimate that humans have 20–25k genes in their genomes [56], we have that there are at most 4–23% of genes to be correlated with diseases yet (and thus are not included in our detector database). It means that a database such as GeneCards can only grow up to 25k entries. Additionally, the remaining genes do not determine alone the contraction of a disease, may have no relation with any disease, or are related to rare diseases that affect very few people. Finally, assuming discoveries directly translate into novel privacy-sensitive sequences is a misconception. Most discoveries correlate diseases with already known genes and genomic variations, not new ones. Our method detects a sequence as privacy-sensitive independently on how many diseases it is correlated with (i.e., one correlation is enough).

New Rare Genomic Variations. The sequences generated by identifying variations are as complete as the size and coverage of the population considered in the allele frequency study over that population. Currently, there are already countries storing and conducting studies on the genome of its whole population [147]. A detector using a database like this will be able to detect most (if not all) privacy-sensitive sequences since the VAR-based detection is by far the method that generates more entries in our detection database (see Table 3.1). As the size and representativeness of the population on the database increase, the detector accuracy increases, accounting for the exact frequency of a given variant in a population, being thus capable of detecting any privacy-sensitive sequence.

These discussions suggest that there is already a large body of knowledge about the privacy sensitiveness of the human genome. More specifically, it is possible to have a *reasonably complete* privacy detector since (1) the rate of discovery of new STRs was extremely low in the last decade—suggesting we probably know most of them, (2) disease-related genes are being discovered at an incredible pace—exhausting the maximum number of human genes, and (3) rare variations can be accurately covered by increasing population samples in allele frequency studies. This result vouches for the usefulness of an evolvable tool that can be used together with standard security techniques to dramatically improve the *status quo* of the robustness of genomic data repositories, much in the lines of what intrusion detection has achieved in the protection of IT data servers.

Evolution Module. Our solution is reproducible and evolving: the same database may be reused with different datasets, and the knowledge database can continually and transparently be updated as new privacy-sensitive sequences are revealed, without affecting the work-flow [71]. In essence, a protection ecosystem built in accordance to the principles proposed here would exhibit similar effectiveness as the continuously updatable industry from the intrusion detection systems. The knowledge database from the DNA privacy detector can be automatically updated to address future attacks as new privacy-sensitive sequences are identified. An *evolution module* integrated into the system architecture presented in Figure 1.1 allows the stored datasets to be reanalysed at any moment and attested again for their privacy-sensitivity. As soon as a new privacy-sensitive sequence is identified, the datasets are updated, access rules are adapted accordingly, and the access history is logged for future inquiries. The *access control* component complements the evolution module by automatically updating the lists and rules according to the datasets' version. The *auditability* component (Chapter 5) complements the evolution module by allowing the verification of who has read previous versions of a dataset that was reanalysed because it could contain previously unknown privacy-sensitive sequences.

3.4 Final Remarks

This chapter proposed a novel method to systematically detect the privacy-sensitive DNA portions of human genomes. Data resulting from this chapter (i.e., the DETECTION phase of our storage pipeline) is divided into two sets: a small privacy-sensitive portion and a large non-sensitive one. The privacy-sensitive portion of human genomes contains only 11.3% of the NGS reads from each genome, which already contributes to the cost efficiency of any storage solution by reducing the size of data that requires stronger security and dependability premises. Both portions are sent to the `Reduction` phase, which deduplicates and efficiently encodes this data to make it even more cost-efficient.

4 Sequencing Data Reduction with Similarity-based Deduplication and Delta-Encoding

Datasets produced in human genomics are massive since its studies compare thousands to millions of biological samples, where hundreds of gigabytes of data are generated from each sequenced body cell [181]. This data deluge must be efficiently stored, transferred, and processed to avoid stagnating medical breakthroughs [203]. Cutting costs in storage space and achieving a high-throughput in restoring data are paramount for this domain [127]. Our primary goal in this chapter is to *increase data reduction gains and restore it faster than the generic compressors* used in practice (e.g., GZIP [88]), while approaching the reduction gains to the ones from specialised tools.

As described in Section 2.2, genomic data has three main representations: sequencing, aligned, and assembled data. Humans have 99.9% of DNA sequence similarity since the *assembled genome* of any two individuals differ in less than 0.1% [243]. Additionally, this representation has a public blueprint (i.e., a reference genome) for humans [219] (e.g., hg38 [188]). It sizes ∼3GB of data from its 3.2 billion contiguous sequence of nucleobases. Assembled human genomes can be reduced ∼700× from ∼3GB to ∼4.2MB in 40 seconds [11] by storing only the genome differences to the mentioned blueprint in a process called referential compression [248]. However, *sequencing data* is much bigger than *assembled data* and has particularities that prevent such compression ratio.

Sequencing data is the most critical representation in genomics because it contains the purest version of genomic data and is unbiased from subsequent processing steps [61]. On the contrary, the output from alignment and assembly is imprecise, lossy, and algorithm-dependent [165]. For instance, using *aligned data* from multiple sources means they presumably were aligned with different algorithms and reference genomes. It precludes subsequent analyses, except if one first converts data back to *sequencing data* and realigns it with the same algorithm and reference (see Section 2.2).

The main reasons *sequencing data* is harder to compress than *assembled data* are (i) the randomness on entries' locality (small data chunks sequenced in no specific order [180]); and (ii) the lack of a stable reference for quality scores [203] (e.g., a similar blueprint as the one available for human DNA [188]). Corroborating these observations, specialised algorithms usually compress *sequencing data* no more than 7× (see Section 4.2 for details on FASTQ compression).

Many algorithms favour maximising compression ratio, which usually comes with penalties in (de)compression speed. This decision is justifiable when data is intended to be archived. However, the decompression speed becomes a bottleneck in cases where compressed data is read from remote storage systems and needs to be decompressed and read several times. This threshold justifies why many real-world solutions (e.g., 1000 Genomes Project [57]) prefer generic compression algorithms that decompress fast (e.g., GZIP [88]) rather than those that only compress more.

Storage of sequencing data is an important, challenging mostly unexplored domain for the systems community [203]. It presents an excellent opportunity for deduplication and its assets: leveraging inter-file similarity and achieving high-performance in reading data. However, traditional identity-based deduplication fails to provide a satisfactory reduction in the storage requirements of genomes (see Section 4.3.1).

Solutions for similarity-based deduplication commonly cluster similar entries into buckets and use identity-based deduplication within them [202], or they focus mostly on the delta-encoding problem [91] while employing inefficient global indexes [256]. In this chapter, we balance space savings and restore performance by proposing GenoDedup, the first method that integrates scalable, efficient similarity-based deduplication and specialised delta-encoding for genome sequencing data.

Novelty in our approach encompasses (i) the proposal (Section 4.3.2) and implementation (Section 4.4.3.2) of GenoDedup, a similarity-based deduplication solution that integrates scalable, efficient Locality-Sensitive Hashing (LSH) with delta-encoding; and (ii) specialisations on delta-encoding for genome sequencing data, namely:

- Circular deltas (Section 2.2);
- Delta-Hamming (Section 4.4.3.1);
- A scalable modelling of generic indexes for multiple genomes (Section 4.4.2).

Additionally, we introduce a converged characterisation of aspects from sequencing data important to deduplication (Section 2.2) and justify why identity-based deduplication fails on it (Section 4.3.1). Our experimental results (Section 4.5) attest the feasibility of GenoDedup since it currently achieves 67.8% of the reduction gains of SPRING [49] (i.e., the best specialised tool in this metric) and restores data 1.62× faster than SeqDB [133] (i.e., the fastest competitor). Additionally, GenoDedup restores data 9.96× faster than SPRING and compresses files 2.05× more than SeqDB.

The remainder of this chapter is organised as follows. Section 4.1 details the different portions of FASTQ entries, which can be explored to reduce the size of FASTQ file. Section 4.2 reviews several tools for compressing sequencing data, Section 4.3 discusses the effectiveness of (identity- and similarity-based) deduplication on human genomes. Section 4.4 presents the GenoDedup implementation and Section 4.5 evaluates it. Section 4.6 discusses how our solution can work with or be adapted to support other datasets and methods, and Section 4.7 presents some final remarks of the chapter.

4.1 Anatomy of a FASTQ Entry

As mentioned in Section 2.2, data obtained from sequencing genomes is stored in the FASTQ text format [61]. A FASTQ file contains many entries with four lines each: a comment line, the DNA sequence, a second comment line, and the sequence of quality scores. In this section, we detail each of these portions and enlist their aspects that impact data reduction the most.

4.1.1 Comment Lines

The first and third lines of each FASTQ entry are comments that start with a "`@`" character in the former and a "+" in the latter. These lines usually contain: a sample identifier (e.g., `SRR618666` in Figure 2.2, the entry identifier (e.g., `296`), and some information about the sequencing run (e.g., `HWI-ST483:151:C08KDACXX:7:1101:21215:2070/1`). Comments follow a similar structure through the file, which can be determined if it contains numeric or alphanumeric fields, and if they are constant, incremental, or variable among entries [35].

4.1.2 DNA

The second line of each FASTQ entry contains the DNA sequence interpreted by the sequencing machine. This sequence is composed of ℓ characters, where this length ℓ can be configured on each sequencing job. Nucleobases can be represented using different sets of characters, where the most commonly used is the {A, C, G, T, N}. It considers the four nucleobases (i.e., adenine, cytosine, guanine, and thymine) and a special character “N” to represent any of them when the machine is unsure on the sequenced nucleobase [75].

A contiguous human genome sizes 3.2 billion nucleobases and results in more than 3GB of data in text mode (e.g., UTF-8 encodes each character in 1 byte). However, NGS machines do not provide the whole genome in a single contiguous DNA sequence [180]. They generate millions of randomly-dispersed reads, which contain small pieces of DNA sequences with hundreds to thousands of nucleobases each [61].

A configurable sequencing parameter determines the coverage in which a genome is sequenced. It is equivalent to the average number of different entries in which every nucleobase position from a genome appears in. Common configurations consider coverage of 30–45$\times$ to increase accuracy. This redundancy results, for instance, in 96 to 144GB of DNA characters per whole sequenced human genome in the FASTQ format.

4.1.3 Quality Scores (QS)

The fourth line of each FASTQ entry contains the sequence of quality scores asserting the confidence level for each sequenced nucleobase. *Phred* quality score [99] is the typical notation in FASTQ files. QS values usually range from 0 to 93 (the higher, the better) and are encoded in ASCII (requiring seven bits per QS) [61]. QS roughly occupy the same storage space as DNA in FASTQ since there is one QS for each nucleobase and standard text encoding (e.g., UTF-8) use eight bits per character.

Quality score sequences are the most challenging portion of FASTQ entries to compress, and as such, we concentrate most of our efforts on it. There is no reference sequence for quality scores [203], but they do have patterns that can boost data reduction [160]. In this thesis, we take into consideration three of them. The first pattern is that many NGS machines have limited precision and generate QS only in the range between 0 and 40 [49, 61], which allows one to describe them using six bits instead of seven. Second, the longer the read

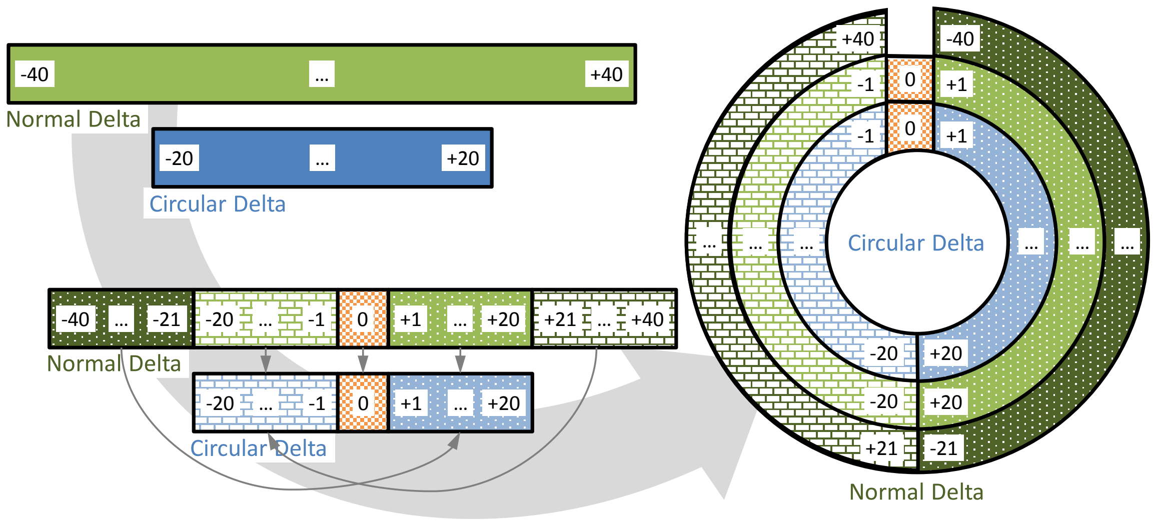


Figure 4.1: Using modular arithmetic to convert normal deltas into circular deltas.

DNA sequence is, the bigger the uncertainty at the end of the QS sequence [232]. For instance, a practical implication from this pattern is that, in FASTQ files from Illumina HiSeq 2000 [136] (the most common NGS machine in the world [120]), several QS sequences finish with a chain of "#" characters—i.e., a low *Phred* value equivalent to 0.

The third pattern is the fact that subsequent QS tend to vary little from one to the other [119]. It means that one may replace subsequent QS by a delta value, which results in the zero value most of the times [160], and convert data to a normal distribution between -40 and $+40$.

However, using delta values naively increases the number of bits required to describe a QS to seven bits again since there are eighty-one options between -40 and $+40$. With this in mind, we propose to use modular arithmetic to convert them to *circular deltas*, which distributes the mentioned range into a circular array from -20 to $+20$. Figure 4.1 illustrates this conversion, where each circular delta can be translated into two different normal delta values. For instance, the circular delta -1 is equivalent to both -1 and $+40$ normal deltas. When solving circular deltas to restore the original QS sequence, the correct alternative can unambiguously be distinguished because only it results in a valid QS between 0 and 40. This transformation reduces the QS encoding back to six bits.

Tool	Version	Compression	Decompression
GZIP [88]	1.6	$F1	-d $F2
pigz [4]	2.3.1	$F1	-d $F2
BSC [113]	3.1.0	e $F1 $F2	d $F2 $F3
ZPAQ [178]	7.15	a $F2 $F1 -m5 -t24	x $F2 -t24 -force
SeqDB [133]	0.2.1	120 120 $F1 $F2	$F2 > $F3
DSRC2 [213]	2.00	c -t24 $F1 $F2	d -t24 $F2 $F3
Quip [151]	1.1.8	$F1	-d $F2
Fqzcomp [38]	4.6	$F1 $F2	-d -X $F2 $F3
FaStore [214]	0.8.0	--lossless --in $F1 --out $F2 --threads 24	--in $F2 --out $F3 --threads 24
SPRING [49]	1.0	-c -i $F1 -o $F2 -t 24	-d -i $F2 -o $F3 -t 24

Table 4.1: Compression tools, versions, and parameters used in our comparisons. $F1 = original FASTQ, $F2 = compressed file, $F3 = decompressed file.

4.2 Sequencing Data Compression

Before discussing the challenges of deduplicating genomic sequencing data, we discuss the state-of-the-art on the compression of sequencing data, its limitations, and the opportunities it leaves open for deduplication. There is a well-known trade-off in data compression between compression ratio and throughput [177]. We selected ten relevant compression algorithms that achieve the best results in these properties [38, 85]: GZIP [88], pigz [4], BSC [113], ZPAQ [178], SeqDB [133], DSRC2 [213], Quip [151], FQZcomp [38], FaStore [214], and SPRING [49]. We evaluate these tools in our experimental environment, which is described in Section 4.5. Table 4.1 presents the version of the tool used in our tests and the arguments passed to compress and decompress data. In this table, $F1 is the path to the original FASTQ file (usually passed as input to the compressor), $F2 is the path to the resulting compressed file, and $F3 is the path to the decompressed file (usually different from $F1 to compare their content hashes later).

Our analyses use five representative FASTQ files of human genomes from the 1000 Genomes Project [57]: SRR400039, SRR618664, SRR618666, SRR618669, and SRR622458. They are human genomes sequenced with the Illumina HiSeq 2000 platform [136]. To the best of our knowledge, this machine was the most used NGS machine in sequencing laboratories around the world when we started this work [120]. Additionally, some of the selected genomes were also used in other papers on FASTQ compression (e.g., SRR400039

Genome	Size (MB)	Size (GB)	Comments	DNA	QS	Entries	Length	Coverage (×)
SRR400039_1	33712.2	33.7	8.3	12.7	12.7	124331027	101	3.91
SRR618664_1	64619.2	64.8	16.2	24.3	24.3	239715311	100	7.46
SRR618666_1	62342.7	62.3	15.7	23.4	23.4	231285558	100	7.20
SRR618669_1	79617.5	79.6	20.0	29.8	29.8	295256611	100	9.20
SRR622458_1†	23617.7	23.6	4.0	9.8	9.8	96097046	101	3.02
Total	**263909.2**	**264.2**	**64.2**	**100**	**100**	**986685553**	—	—

Table 4.2: The genome datasets used in this chapter, their size (in MB and GB), the size (in GB) of the comment, DNA, and QS portions separately, the number of FASTQ entries, the length of each DNA and QS sequence within an entry, and the sequencing coverage (in ×). † We used only portions of this file to complete 100GB of DNA and of QS lines in our experiments.

in Quip's paper [151]). Only the FASTQ file from the first end of these genomes are considered in our analyses, but they sum up 265GB of data and result in almost one billion FASTQ entries.

Table 4.2 provides additional details about the selected datasets. It contains the size of the selected genomes (in B, MB, and GB), the size (in GB) of the comment, DNA, and QS portions separately, the number of FASTQ entries, the length of each DNA and QS sequence within an entry, and the sequencing coverage [138].

Table 4.3 presents these files and the resulting compression ratio and restore throughput of each algorithm on them. At the end of this section, we present the complete tables comparing the ten selected tools using the five selected datasets in terms of compressed size (Table 4.4), compression throughput (Table 4.5), and decompression throughput (Table 4.6).

GZIP [88] is a generic compression tool employed in several application domains, including the storage of human genome sequencing data. For instance, the 1000 Genomes Project [57] stores their FASTQ files compressed with GZIP. Even recent frameworks for bioinformatics (e.g., Persona [46]) use GZIP to compress data. The main strength of GZIP is its decompression/restore throughput, which reaches 41MB/s on average in our files and

Genome	GZIP$^{\otimes}$	pigz$^{\otimes}$	BSC$^{\otimes}$	ZPAQ$^{\otimes}$	SeqDB	DSRC2	Quip	Fqzcomp	FaStore	SPRING	GenoDedup
400039	2.80	2.80	3.99	4.43	2.01	3.88	4.55	4.52	4.69	**5.18**	4.11
618664	3.01	3.00	4.33	4.84	2.01	4.24	4.98	4.93	N/A	**6.04**	4.42
618666	2.93	2.93	4.20	4.69	2.00	4.12	4.82	4.78	N/A	**5.84**	4.35
618669	3.03	3.03	4.36	4.89	2.01	4.29	5.03	4.97	N/A	**6.19**	4.52
622458†	4.37	4.37	5.83	7.37	1.92	4.21	4.81	5.02	6.17	**6.87**	3.05
Avg. C.R.	3.22	3.23	4.54	5.24	1,99	4.15	4.84	4.84	5.43	**6.02**	4.09
Write (MB/s)	15.5	281.1	159.9	5.3	415.6	**1375.9**	28.7	60.5	25.5	43.1	0.3*
Read (MB/s)	41.4	66.1	46.2	1.1	127.9	125.3	3.4	9.6	45.2	20.9	**208.2***
Version	1.6	3.1.0	7.15	2.00	0.2.1	1.1.8	4.6	1.0	0.8.0	9.22	0.1

Table 4.3: Genomes and compression tools. Per algorithm: compression ratio (i.e., C.R., the ratio between the *original_size* and the *compressed_size*) on each genome, write and read throughput (in MB/s), and its version. $^{\otimes}$ Generic compression algorithm. * See Section 4.5 for the complete analysis.

66MB/s in its parallel version (i.e., pigz), while ZPAQ, Quip, and Fqzcomp reach less than 10MB/s and SPRING reaches 20MB/s. FaStore and BSC reach a similar throughput as GZIP, but DSRC2 and SeqDB are the fastest (specialised) tools to decompress FASTQ files, reaching a throughput of approximately 125MB/s. We use GZIP and pigz as the baseline generic tools and SeqDB and DSRC2 as the baseline specialised tools in experiments that evaluate throughput.

Many specialised tools for FASTQ files focus on maximising compression ratio. For instance, SPRING [49] is the specialised tool that reaches the best compression ratio in our files (6.023× on average). It is followed up by FaStore [214] (5.4×) and by the generic tool ZPAQ (5.2×). We use ZPAQ as the baseline generic tool (together with GZIP and pigz due to their importance and restore throughput) and SPRING as the baseline specialised tool in experiments that evaluate FASTQ compression ratio.

We have evaluated other specialised (e.g., G-SQZ [234] and KIC [258]) and generic compression algorithms (e.g., BZIP2 [221] and LZMA2 [204]). However, they compress data less than SPRING [49] and restore data slower than pigz and SeqDB [133] in our

Genome	GZIP	pigz	BSC	ZPAQ	SeqDB	DSRC	Quip	Fqzcomp	FaStore	SPRING	GenoDedup
400039	12041	12036	8441	7617	16728	8693	7410	7454	7180	**6509**	8202
618664	21498	21511	14930	13355	32204	15239	12971	13094	N/A	**10702**	14624
618666	21298	21277	14849	13299	31117	15132	12920	13054	N/A	**10674**	14318
618669	26304	26302	18253	162958	39580	18572	15833	16025	N/A	**12869**	17627
622458	5408	5401	4051	3206	12274	5607	4909	4706	3826	**3438**	7751

Table 4.4: Compressed size of the selected genomes per tool, in MB.

experiments. Additionally, we have evaluated LFQC [194] and discarded its results because it uses LPAQ8 to compress the quality score sequences and LPAQ8 does not support files bigger than 2GB.

LZMA2 is an improved multi-threaded version of the generic compressor entitled Lempel-Ziv-Markov chain algorithm (LZMA), which was implemented in the 7-Zip compressor [204]. We experimented this tool with the FASTQ file of the SRR400039 genome. However, it has compressed this file 3.26× at a throughput of 1.36MB/s and decompressed it at 14.1MB/s.

PPM (Prediction by partial matching) is another interesting generic compression algorithm to be considered [185]. It is also one of the possible compression methods used by 7-Zip. We experimented a tool called PPMd with the FASTQ file of the SRR400039 genome, and it has compressed this file 3.95× at a throughput of 17.1MB/s and decompressed it at 3.86MB/s.

These two tools were discarded from our complete comparison because they are slower than GZIP and have a lower compression ratio than ZPAQ. Other tools fall in the same scenario since these do not surpass the performance of GZIP nor compress more than ZPAQ, namely: BZIP2 [221], G-SQZ [234], and KIC [258]. ZPAQ achieving a better compression ratio than PPMd is usually justified by the fact that ZPAQ (and similar context mixing algorithms) use many prediction models from different contexts while PPMd uses a single one [6].

Genome	GZIP	pigz	BSC	ZPAQ	SeqDB	DSRC	Quip	Fqzcomp	FaStore	SPRING
400039	12.80	251.58	161.30	5.42	474.82	**636.08**	28.89	59.46	14.05	35.30
618664	13.46	255.41	167.84	5.35	425.13	**1576.08**	28.11	61.25	N/A	41.45
618666	13.55	259.76	133.21	5.30	418.41	**1558.57**	29.20	61.06	N/A	40.72
618669	13.77	201.56	136.80	5.41	277.41	**1421.74**	28.40	61.67	N/A	35.93
622458	24.03	437.36	200.15	5.09	481.99	**1686.975**	28.98	59.04	36.96	62.32
Average	15.52	281.14	159.86	5.31	415.55	**1375.89**	28.72	60.50	25.51	43.14

Table 4.5: Compression throughput (*original_size/compression_time*) per tool in MB/s.

Genome	GZIP	pigz	BSC	ZPAQ	SeqDB	DSRC	Quip	Fqzcomp	FaStore	SPRING
400039	44.60	69.57	45.38	1.21	**164.01**	140.20	3.60	9.85	47.87	22.52
618664	44.60	70.53	43.53	1.11	**82.58**	75.81	3.29	9.58	N/A	19.49
618666	44.65	69.76	35.69	1.15	**92.61**	76.04	3.43	9.64	N/A	21.39
618669	38.01	58.58	36.73	1.11	**102.28**	54.30	3.23	9.56	N/A	18.95
622458	34.89	62.07	69.84	0.69	197.97	**280.34**	3.60	9.23	42.51	22.18
Average	41.35	66.10	46.23	1.05	**127.89**	125.34	3.43	9.57	45.19	20.91

Table 4.6: Decompression throughput (*compressed_size/decompression_time*) per tool in MB/s.

The idea of context mixing brings another tool, called LPAQ8 [176], into the discussion. LPAQ8 is a slow compressor that reaches the best compression ratio in many benchmarks and was developed by the same person that developed ZPAQ. Other FASTQ compression tools (e.g., LFQC [194] and LFastqC [6]) separate the different portions of FASTQ files (i.e., comments, DNA, and QS) and compress them separately.

Both LFQC [194] and LFastqC [6] use LPAQ8 to compress the quality score sequences, while for the comments and DNA sequences the former uses ZPAQ and the latter uses Mfcompress [206]. However, LPAQ8 uses signed integers to track the file length, which makes it crash if its input files have more than 2GB. As presented in the sixth column of Table 4.2, all QS portions of our datasets have more than 2GB, which prevents us from using any tool that employs LPAQ8 for compressing these portions.

Developers from LFQC suggested, in an issue report [193], to replace LPAQ8 by ZPAQ to compress the QS sequences when they are bigger than 2GB. However, it would make

LFQC's results very similar to the ones from ZPAQ, which vouched for discarding it from the complete comparison.

Another tool that separates the portions of FASTQ files and uses the Mfcompress [206] for the DNA sequences is the MZPAQ [94]. As the name suggests, it uses the ZPAQ algorithm for the QS sequences. Unfortunately, we were not able to find a publicly available source-code or executable of MZPAQ, which prevents us from adding it to the complete evaluation. We even considered running its underlying software separately, but Mfcompress has compressed a human genome only 34.3% more than GZIP [206] and using the ZPAQ to compress the QS sequences would result in ZPAQ's low restore throughput, incurring in the same limitations as the previously mentioned generic tools.

Algorithms that align the DNA data before compressing it (e.g., SlimGene [160]) can reduce the DNA portion alone up to 20×, but they take considerable time (e.g., 8 hours per human genome) and consequently reduce the compression throughput. Nonetheless, as will be described in Section 4.6, the methods presented in this chapter can also work with aligned data.

Finally, Zhou *et al.* [260] propose a similarity-based compression algorithm for quality scores from genome sequencing data. However, they use a non-scalable memetic algorithm to create a small codebook for each FASTQ file they want to compress and inefficiently compare each QS sequence to all base chunks in the codebook to calculate the best delta-encoding. Additionally, we cannot compare the performance of our solution to theirs empirically because they provide no implementation, but our work surpasses theirs in several other theoretical aspects, which are detailed in Section 4.4.

4.3 Human Genome Deduplication

Deduplication reduces the storage requirements by eliminating unrelated redundant data [37, 103]. Additionally, deduplication has two advantages when compared to compression algorithms: it may leverage the inter-file similarities, while most compression algorithms consider only intra-file data or use a single generic contiguous reference; and it usually achieves a better restore performance than compression.

There are many deduplication approaches and systems available [202], and several of them rely on *index data structures* to lookup exact copies of data already stored in the system. This indexing mechanism maps the content of stored chunks to their actual storage location to efficiently find duplicate instances.

4.3.1 Identity-based Deduplication

In this section, we discuss the strengths and limitations of conventional approaches for identity-based deduplication and present examples confronting them with FASTQ files. Given the particularities of FASTQ files (Section 2.2), this discussion is of extreme importance to clarify and caution the general deduplication community in the search for efficient solutions to the problem of interest. The discussion that follows encompasses three identity-based approaches: file deduplication, block deduplication, and application-aware deduplication.

4.3.1.1 File deduplication

This approach identifies exact copies of the same file by comparing their content hashes (e.g., SHA-2) and replaces the redundant data with pointers to a single instance [37]. It is ineffective in genome repositories because these facilities store data mostly from their unique samples [71] or because even sequencing the same sample results in files with different content [180].

Example. The 1000 Genomes Project [57] contains half a million files, in which more than 200k are FASTQ. We downloaded its current directory tree [1] and compared the content hashes (MD5) of all FASTQ files to obtain the duplicate ratio. These MD5 hashes are available in the last column of this directory tree, which means one does not need to download all FASTQ files to perform the present comparison. The result indicates that less than 0.007% of the FASTQ collection is composed of duplicate files, which validates the low interest for file deduplication in sequencing data.

4.3.1.2 Block deduplication

This approach splits files into fixed- or variable-size blocks, calculates their content hashes, and compares them to find duplicates. Systems with fixed-size block deduplication commonly adopt blocks of 4KiB for historical and compatibility reasons—e.g., this is the size of virtual memory pages in several computer architectures and of blocks in many filesystems. For variable-length blocks, the most common algorithms are the Rabin fingerprinting [210] and the Two-Threshold Two-Divisor (TTTD) [98].

Block deduplication fails to identify copies of FASTQ data chunks because they are unlikely to happen. Reasons for that include the fact FASTQ files contain the unique sample and entry identifiers; the DNA sequences contain mutations, transformations, and are sequenced in no specific order; and the distribution of QS varies from run to run.

Example. We have split three FASTQ files (SRR400039, SRR618664, and SRR618666) into 40 million fixed-size blocks of 4KiB, calculated the MD5 hash of each block, and verified that there are no duplicates on it. We executed the same experiment with variable-size chunks using the Rabin fingerprinting [79] (with blocks between 1–8KiB) to generate more than 23 million hashes, where no duplicates were found.

4.3.1.3 Application-Aware deduplication

A final strategy is to consider the files' structure and content to increase the chances of deduplication. One may write each line type of FASTQ entries into different files—each one containing only (1) the "@" sequencing comments, (2) the DNA sequences, (3) the

"+" comments, or (4) the quality scores—and deduplicate them separately. Both fixed- and variable-size block deduplication can be employed in this approach.

Example (Comment lines). Comment lines have an identifiable structure that can be parsed into fields. For instance, lines from the SRR618666 genome have ten fields each. Five of them are constant across the whole file, two are incremental numbers, and three are variable. One may replace the constant and incremental fields by a small encoding at the beginning of a compressed file. Then, the remaining variable fields can be placed in a file to be deduplicated separately. In SRR618666, the 231 million lines, with three variable fields each, can be replaced by pointers to only 48 unique values in the first field, 20k in the second, and 199k in the third. Bhola *et al.* [35] compresses comments $\sim 17\times$ with this approach.

Example (DNA and QS blocks). We separate the lines from the three FASTQ files as previously mentioned, removed the newline character and performed the block deduplication previously presented. We split the DNA and the QS files into 4KiB blocks and separately compared their content hashes, which results in no duplicates. Similarly, executing the same workflow with Rabin fingerprinting does not find any redundant blocks.

We execute the same block deduplication with the block size as 100 characters in SRR618664 and SRR618666 (i.e., the sequence length in these files). This approach is the first to provide a considerable number of duplicates. From the 471 million entries in these genomes, 44 million DNA lines (9.42%) are exact duplicates, as well as 468 thousand QS lines (0.01%). However, these values are unsatisfactory since spatial deduplication usually requires gains of 20–40% to be considered worth the invested cost and time [103].

4.3.1.4 Summary

The three selected FASTQ files used in these examples are enough to illustrate the inefficiency of traditional identity-based deduplication methods, whereas considering more genomes here leads to similar conclusions. Identity-based deduplication provided significant gains only in comment lines in our analyses. Based on the descriptions from the present section and the characteristics of FASTQ files, there are excellent opportunities for similarity-based deduplication, which we discuss in the next section.

4.3.2 Similarity-based Deduplication

Similarity-based deduplication matches resembling objects of any size using similarity search to deduplicate them [91]. We integrate similarity-based deduplication with delta-encoding, which stores (1) a *pointer* to the most similar entry together with (2) the *minimal list of modifications* to restore the original object from this entry. This most similar entry is known as the *base chunk* [202].

Associating this approach with the application-aware deduplication is intuitively a promising solution to deduplicate genomes. However, there are at least three challenges that need to be addressed: (1) *choosing a distance metric and encoding*, (2) *modelling the deduplication index*, and (3) *reducing the number of candidate comparisons*.

A *distance metric* is critical as it defines what makes entries similar and determines how to choose the best deduplication candidates. In this work, we consider three metrics [175] and present experiments using them in Section 4.5.

- HAMMING: Counts the number of positions with different characters in two strings of the same size. The resulting list of edit operations is composed of only `UNMODIFIED` and `SUBSTITUTION` operations.
- LEVENSHTEIN: Calculates the minimal number of modifications to convert a string into another. It considers `UNMODIFIED`, `DELETE`, `INSERT`, and `SUBSTITUTION` operations.
- JACCARD: Calculates the ratio between the intersection and the union of N-grams from the strings.

The first two metrics return the distance value and a list of edit operations to restore the original data from the base chunk, whereas the last one provides only the distance.

After choosing the distance metric, one may *model the deduplication index* based on it. It is an optimisation process that selects a subset of (real or synthetic) entries and results, for example, in the smallest distance sum to a known sample of sequences. As previously mentioned, human DNA sequences have a comprehensive reference (e.g., *hg38* [188]) that can be used to create such index, *but there is no such reference for QS sequences* [203]. To create the index for quality score sequences, one may resort to optimisation, memetic (e.g., [260]), or clustering (e.g., K-Means [146]) algorithms to find the best codebooks to the deduplication task.

Another option is to choose the most frequent sequences from each file empirically. However, naively creating the index with entries exactly as they appear in FASTQ files is inefficient due to a combinatorial explosion. Finally, one may initiate the system with an empty deduplication index and dynamically insert every queried entry that has not found a similar enough neighbour (i.e., under a predefined threshold). However, the index may grow indefinitely if the threshold is too hard to achieve, or it will result in low reduction gains if the threshold is too easy to reach.

After obtaining a deduplication index that achieves satisfactory compression results, one may decide how to improve the scalability and performance of the system [261]. The human DNA reference provides nearly 3.2 billion base chunks. As mentioned before, QS sequences do not have a reference, and thus one may define the limits of the index size according to capacity of its system. For instance, storing 1 billion entries of 100 characters each in a simple key-value store, indexed by integers of 32-bits, results in at least 100GB of data. Keeping all data in main memory in a single node may become a burden, and thus partitioning data across several nodes [104] or using sparse indexes [168] emerge as desirable alternatives.

Finally, *reducing the number of candidate comparisons* is another crucial performance improvement to the system. One may achieve this goal through other auxiliary data structures such as K-mer tables [259], indexes for Locality-Sensitive Hashing (LSH) [143], or cluster deduplication [104]. However, these structures may interfere with the recall of the best deduplication entries, producing suboptimal search results depending on their configuration. It means that there is a trade-off in improving the performance that may compromise the deduplication gains.

4.4 GenoDedup

In this section, we describe GenoDedup, which integrates scalable, efficient similarity-based deduplication and specialised delta-encoding for sequencing data. In Section 4.4.1, we present the main components of GenoDedup and how data flows among them. Sections 4.4.2 and 4.4.3 detail how we solve the three main challenges from Section 4.3.2.

4.4.1 Overview

The main components of GenoDedup can be seen in Figure 4.2. The similarity-based deduplication selects the nearest base chunk for each sequence in FASTQ entries using two auxiliary data structures. The first is a Locality-Sensitive Hashing (LSH) index, which enables the similarity search when the number of deduplication candidates is too big to perform optimal searches. Entries are blocks with a variable size similar to the length of the DNA and QS lines in the FASTQ files used in this work. The second data structure is a key-value store (KVS) indexing unique entries that are used in optimal similarity searches and to retrieve the value of deduplication candidates using their content hashes as keys. A data storage component is used to store the deduplicated files and provide them to readers. Readers use a restore module, which reads the pointers and delta-encoding from the deduplicated file and queries the deduplication index of unique entries to restore the original FASTQ file from it.

An offline setup phase, described in Section 4.4.2 but not shown in Figure 4.2, prepares the environment where the deduplication will take place. This phase populates the auxiliary

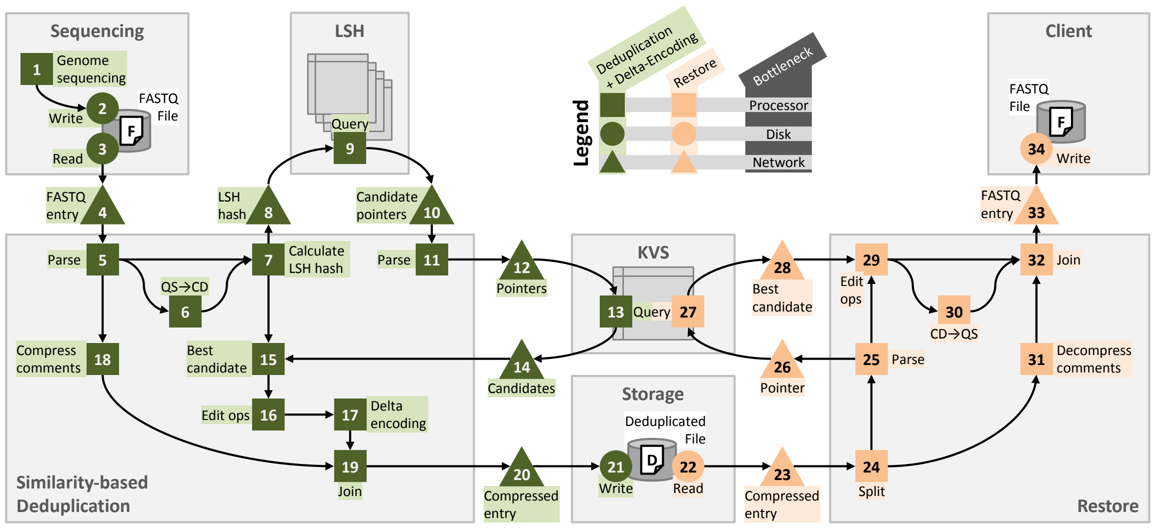


Figure 4.2: Overview of the architecture of GenoDedup.

data structures (i.e., LSH and KVS) with the previously generated list of deduplication candidates. For instance, the human reference genome (e.g., *hg38*) can be loaded to the LSH and KVS during this phase. At the end of this offline phase, data has been loaded to the appropriate data structures in a way that the similarity search can be executed efficiently.

Data flow during a deduplication execution is composed of the numbered steps present in Figure 4.2. Steps 1–21 represent the deduplication process, while steps 22–34 represent the FASTQ restore process. Squared steps are processor-bounded tasks, circular steps are disk-bounded, and triangular ones are network-bounded.

When sequencing a genome, (1) NGS machines generate data at 0.3MB/s [172], which is (2) stored in a disk that supports this throughput. Similarity-based deduplication receives the sequenced data by (3) reading it from the disk and (4) transferring it through the network to the deduplication component. Then, (5) each FASTQ entry is parsed into the different line types, where comments are sent to Step 18 (see below), DNA to Step 7, and QS to Step 6. QS sequences are (6) converted to circular deltas, and QS and DNA sequences are used to (7) calculate the hashes that will be used to query the LSH. These hashes are (8) sent to the LSH component, which will (9) obtain the internal LSH keys from these hashes, query them in the respective LSH indexes, and join the lists of pointers to the candidates in a bigger list, which is (10) returned to the deduplication component.

The deduplication component (11) receives this list of pointers to candidates and (12) sends it to the KVS to obtain their content. The KVS (13) obtains the candidate value using each pointer as a key and (14) returns the list of candidates (their content, not the pointers). The deduplication algorithm (15) calculates the edit distance only (not the edit operations) between each candidate from the received list and the sequence from the FASTQ file and keeps track only on the pointer and value of the best candidate (i.e., the one with the smallest edit distance). After identifying the best candidate, it (16) calculates the edit operations between the sequence from the FASTQ file and the best candidate and (17) converts the edit operations to the delta encoding using Huffman codes. In parallel to this process, the deduplication component (18) compresses the comment lines with an external algorithm (e.g.,Bhola *et al.* [35]). In the end, the component (19) joins the deduplicated and compressed version of the comment, DNA, and QS lines and (20) sends the reduced entry to a storage component, which (21) writes the entry in a deduplicated file.

When a client intends to read a deduplicated FASTQ file, he (22) reads the file from the disk and (23) transfers it to the FASTQ restore component. The restore module (24) converts, both for the DNA and QS sequences, the bytecode to the pointer to the best candidate, to the first character of the original QS sequence and the delta encoding. For each sequence (25), the restore module (26) sends the pointer to the KVS, which (27) obtains the respective value indexed by the pointer as a key and (28) returns the value of the best candidate to the restore module. The restore module then (29) applies the edit operations from the delta-encoding to the returned candidate and (30) converts from circular delta QS to normal QS if it is a QS sequence. Finally, it (31) decompresses the comment lines using an external algorithm (e.g., Bhola *et al.* [35]) and (32) joins the restored comment, DNA, and QS lines. The restored entry is (33) sent to the client, which (34) stores it in a FASTQ file on disk.

Steps 12 and 14 can be avoided if the LSH index stores and returns the list of the actual content of the deduplication candidates instead of their content hashes. These content hashes are used as pointers to retrieve the candidate content from the KVS index with unique entries. We opted by LSH storing only the content hashes of candidates because it makes the size of LSH index smaller and linearly proportional to the number of entries, independent on the candidate's size.

4.4.2 Offline Phase

Modelling the group of base chunks that will be inserted in the deduplication index is paramount to achieve satisfactory reduction gains. As previously mentioned, we suggest the use of the available human reference genome *hg38* [188] as the deduplication index for DNA sequences. For QS sequences, we resort to representative entries that result in the smallest sum of distances to a group of real entries, e.g., from the SRR618666 genome.

GenoDedup converts the original input QS sequences to circular delta values (see Section 2.2) and employs clustering algorithms to distribute them into a predefined number of clusters. Centroids from the resulting clusters are stored in a file that is loaded to the deduplication index during an offline setup phase. GenoDedup employs the Bisecting K-means [229] from Apache Spark, which is a faster and more scalable hierarchical divisive version of K-means [146]. Additionally, our solution can generate three orders of magnitude more base chunks than the memetic algorithm from Zhou *et al.* [260] in useful time. This scaling-up allows us to create generic deduplication indexes from many genomes instead of generating one small codebook for each FASTQ file. In this chapter, we deliberately select specific numbers k of clusters in the form of $k = 2^i$, where i ranges from 0 to 20, four by four.

At this point of the process, the base chunks that compose the deduplication index have already been defined and are placed in the proper data structures for the execution of the online phase.

4.4.3 Optimisations of the Online Phase

In this section, we describe two optimisations that balance storage space and performance in the similarity-based deduplication described in Section 4.4.1. The first one describes how the distance metric and its encoding are implemented, whereas the second discusses how do we reduce the number of candidate comparisons.

4.4.3.1 Distance Metric and Encoding

Choosing a distance metric determines what makes entries similar while designing an optimal encoding provides reduction gains when describing entries as the differences to previously known sequences. GenoDedup includes all the three string distances (Hamming, Levenshtein, and Jaccard) mentioned in Section 4.3.2. Our implementation uses the java-string-similarity library [82], which provides implementations for these distances. We employ Huffman codes [134] to describe the divergent characters in all metrics. The Hamming and Levenshtein schemes are used to create the encodings presented in the following descriptions.

In the Hamming algorithm, the list of edit operations between two strings with the same size considers only `UNMODIFIED` and `SUBSTITUTION` operations. One may encode the result using 1 bit per edit operation (e.g., 0 for `UNMODIFIED` or 1 for `SUBSTITUTION`), plus the symbol encoding when characters do not match. We employ the mentioned Hamming encoding with the use of Huffman codes [134] to describe the divergent characters. The resulting size (in bits) is given by Equation (4.1).

$$SizeH = M + C_0 + \ell + (S * huf(\bullet)) \tag{4.1}$$

The size of the candidate pointer M (in bits) corresponds to $M = log2(N)$, where N is the expected number of entries in the deduplication index. C_0 describes the first character in the original sequence, which allows one to initiate a chain of conversions from circular delta values to the original quality score sequence. In the case of DNA sequences, the C_0 element is unnecessary. It can be a fixed-size value or a Huffman code based on the distribution of the first character observed from several FASTQ files. ℓ is the length of the entry sequence, S is the number of characters that differ from the base chunk, and $huf(\bullet)$ is the function that returns the size of the Huffman code for each divergent character. The best-case scenario for the Hamming encoding results in ℓ bits per sequence, an exact match, which leads to a compression ratio upper bounded by $8\times$.

We propose to extend the Hamming algorithm to aggregate subsequent matching characters—in an encoding dubbed *Delta-Hamming*—and replace them by a delta number that informs how many characters should be skipped before finding the next substitution. For example, applying this comparison algorithm between the strings "ABCDEFGH" and "AXCDEYZH" results in the following operations: "1X3YZ", while the result in the original Hamming is: "USXUUUSYSZU". This algorithm results in the encoding size presented in Equation (4.2).

$$SizeDH = M + C_0 + (len(\diamond) * (5)) + (len(\circ) * (1 + huf(\bullet))) \qquad (4.2)$$

The function $len(\diamond)$ is the quantity of delta numeric characters (i.e., [0–9]) in the string and $len(\circ)$ is the number of differing characters in the string. One of the main advantages of this approach is that its encoding size is not lower bounded by the length of the sequences ℓ. For instance, if two strings are identical, this encoding results only in a special code of five bits to inform that there is no additional edit operation in the comparison. Contrarily, the encoding of Hamming (already presented) and Levenshtein (the next description) algorithms are lower bounded by ℓ bits informing that there are ℓ `UNMODIFIED` operations.

The Levenshtein algorithm verifies four different edit operations (`UNMODIFIED`, `SUBSTITUTION`, `DELETE`, and `INSERT`), which requires 2 bits per operation. Similarly to the previous encoding, we use Huffman codes to describe characters in SUBSTITUTION and INSERT operations. The resulting size (in bits) is given by Equation (4.3).

$$SizeL = M + C_0 + 2 * \ell + ((S + I) * huf(\bullet)) \tag{4.3}$$

S and I denote the number of SUBSTITUTION and INSERT operations. In this Levenshtein encoding, the best-case scenario results in $2 * \ell$ bits when all characters match, which is the equivalent to an upper bound of $4\times$ in compression ratio. We also evaluated a *Delta-Levenshtein* encoding, similar to the proposed *Delta-Hamming*. However, it has a limited impact because it requires more bits to represent its additional operations, and it reduces very little the total number of edit operations of deduplicated sequences.

4.4.3.2 Number of Candidate Comparisons

The number of candidate comparisons executed on each query influences the search performance and directly depends on the employed algorithm and configuration. We implement two forms of similarity search: optimal and probabilistic.

In the former, the system loads all modelled base chunks to a list in main memory and compares each queried sequence to all entries in this list. This process is inefficient when the number of candidates is very large. However, it always finds the best candidate (i.e., the nearest neighbour) in the index and is a feasible solution for small indexes.

In the latter, the system inserts all base chunks into an efficient data structure, called Locality-Sensitive Hashing (LSH) index, and compares each queried sequence only to entries that belong to the same buckets as the queried sequence. It effectively reduces the number of candidates to be compared.

LSH is an algorithm that, given an entry, returns a content hash that has a high probability of colliding with the hash of similar objects—and a low probability of colliding with distinct ones [143]. This idea is the opposite of cryptographic hashes, where even very similar objects should generate very distinct content hashes [22].

The resulting hash from LSH is composed of a group of *k* smaller hashes (e.g., integers). The LSH index is composed of *k* multimaps—i.e., a KVS where each key maps to a list of values. Each smaller hash from the LSH hash is the key to one of these multimaps. In an insert operation, the LSH hash from the received sequence is obtained, and the object is appended in the list of values mapped by each small hash in the respective multimap.

In a query operation, the LSH hash is also obtained, and the result is the joint set of values mapped by the small hashes in the respective multimaps. The best candidate sequence is obtained by calculating the string distance of choice between the queried sequence and all base chunks present in the returned small joint set, which becomes an optimal search with less candidates. Finally, the chosen base chunk is used to calculate the delta-encoding, which is the minimal list of edit operations necessary to restore the queried sequence from the base chunk.

In GenoDedup, we implement the LSH hash as a MinHash [41], which is proportional to the Jaccard distance—i.e., the ratio between the intersection and union of two sets. It means that sequences that are more similar than a given threshold will have a higher probability of being placed in the same bucket in at least one multimap. We also implement bit sampling techniques [167] in our LSH hash to reduce its size and to become even more efficient in space and time.

To implement the LSH index, we extended the Chronicle-Map library to provide a multimap instead of their original key-value store [53]. GenoDedup benefits from Chronicle's principles and results in a well-engineered solution that provides: off-heap techniques to avoid garbage collection; efficient persistent storage to support data bigger than the available main memory; multi-threads and fine-grain locks to support multiple writers and readers; collections of objects as small as Java primitives to avoid space overhead; among others. Finally, our deduplication index supports four orders of magnitude more base chunks than the values reported by other solutions for similarity-based deduplication [91, 256].

4.5 Evaluation

We evaluate our Java prototype of GenoDedup to illustrate the strengths and limitations of similarity-based deduplication in genome sequencing data. It is open-source and publicly available on GitHub [73].

Experiments are divided into three parts: the encoding size of deduplicated entries; a performance evaluation; and an end-to-end scenario with a large workload. In the first two experiments (Section 4.5.1 and Section 4.5.2), our testing dataset is the first two hundred and fifty thousand FASTQ entries from the SRR618666 genome, which properly represents the diversity of its entries. We use this subset, instead of all portions of this genome, because we intend to evaluate an optimal (exhaustive) search algorithm that compares every queried sequence to all candidates in a deduplication index. Testing this optimal search with the whole genome (instead of using only these 250k reads) would make it infeasible to complete these tests in practical time when using indexes with more than 2^{16} deduplication entries. This optimal search is also important to identify the expected performance of the system given different number of candidates returned by the LSH optimisation (Section 4.4.3.2).

Tests with DNA sequences are directly executed using the original FASTQ file and the human reference genome *hg38* [188]. Tests with QS sequences first convert them to circular delta values (Section 2.2) and compare them to entries in the deduplication index, which also are encoded as circular deltas.

Experimental Setup

The experimental setup is composed of a Dell PowerEdge R430 server, equipped with 2 Intel Xeon E5-2670v3 processors (2.3GHz), 128GB of RAM (DIMM 2133MHz) and a 300GB disk of 15k RPM with an average sequential write and read throughput of 215MB/s. The operating system used was an Ubuntu 16.04.2 LTS x86_64.

4.5.1 Encoding Gains

In this experiment, we compare the average size (in bits) of delta-encoded entries using Hamming edit operations, Levenshtein, and Delta-Hamming ones. They include a pointer to the most similar deduplication entry and the encoded edit operations to transform it back into the original sequence.

To validate the differences in the data entropy of each portion of FASTQ entries, we separate and compress them individually with ZPAQ. For instance, the file with the first comment line of every FASTQ entry from the SRR618666 sizes 15.1GB. ZPAQ compresses it 6.43× to 2.3GB. The DNA and QS portions of this genome size 23.3GB each. ZPAQ compresses the former to 5.4GB (4.33×) and the latter to 7.4GB (3.14×).

Every queried DNA and QS sequence has 100 characters, which means that each one of them occupies 800 bits in text mode originally. When using the human reference genome *hg38* as the deduplication index, DNA sequences are compressed 13.43× with our *Delta-Hamming* encoding, whereas ZPAQ compresses it only 4.33×. We used only the Delta-Hamming encoding for DNA sequences because the encoding of Hamming and Levenshtein are bounded up to 8× and 4×, respectively (see Section 4.4.3.1 for more details on their encoding).

For QS sequences, the results from Figure 4.3 show that Hamming encoding achieves a smaller output size than Delta-Hamming, which is smaller than Levenshtein. Their best case (i.e., 2.46× considering the Hamming distance and 2^{20} index entries) already achieves nearly 80% of the reduction gains from the ZPAQ algorithm when considering only quality score sequences—3.14× for SRR618666. Our solution can obtain even better reduction gains with bigger indexes.

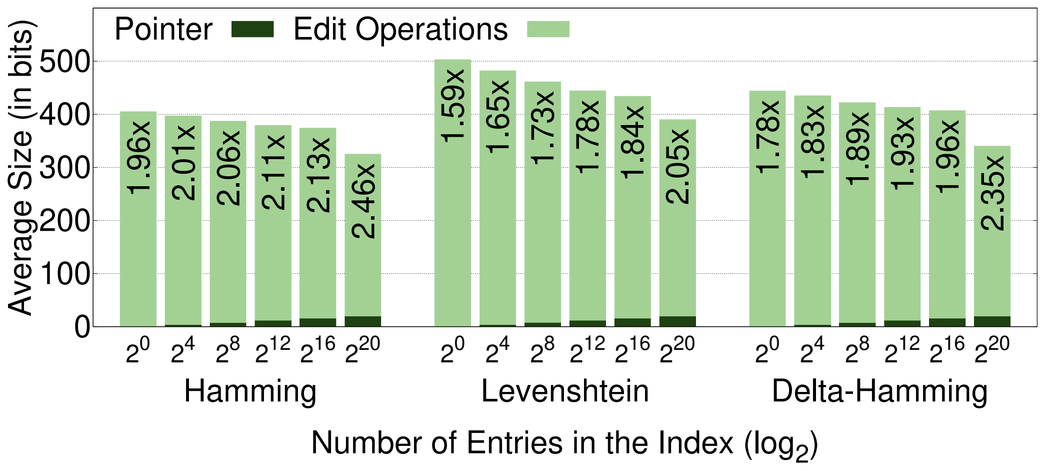


Figure 4.3: Average encoding size of deduplicated QS sequences (in bits) and its reduction ratio.

4.5.2 Performance

In this section, we evaluate the read and write performance of GenoDedup both for DNA and QS sequences. We discuss the performance of the deduplication and restoring processes only in the aspects that our algorithm and implementation may have a bigger impact or may represent a bottleneck to the workflow. More specifically, GenoDedup is compute-bound, mostly by finding the best candidate, which requires calculating the distance metric between the query and all returned candidates. For this reason, we *do not* evaluate in this section:

- Processing bottlenecks on services (e.g., LSH and KVS), because they can be placed in local memory if they are small enough or they can horizontally scale by using multiple nodes [48];

- Bottlenecks from parsing and direct data conversion, because they usually are significantly faster than the main processing steps we evaluate here;

- Disk bottleneck, because it is specific to the hardware used in the experimental environment and it can be avoided by processing entries from multiple files on different disks up to the point the processing becomes the main bottleneck again;

- Network bottleneck, because it is also specific to the experimental environment and can be avoided with faster networks (e.g., 10Gbps instead of 1Gbps).

4.5.2.1 Read Operations

Our next experiment aims to evaluate the performance of applying the list of edit operations (i.e., the delta-encoding) in a base chunk to restore the original sequences from FASTQ entries (i.e., steps 29 and 30 in Figure 4.2). In theory, it is directly related to the number of differences between the two sequences, where the fewer differences they have, the faster it is to recover.

Figure 4.4 presents the throughput (in MB/s) that GenoDedup reaches in applying the identified edit operations with different parallelism configurations (from 1 to 48 threads). The length of the analysed sequences is 100 characters. As expected, the more threads processing requests up to the number of physical cores, the bigger the throughput. For instance, GenoDedup restores entries in a single thread at a pace of 30.8MB/s with the Hamming encoding, 21.3MB/s with Delta-Hamming, and 9.5MB/s with Levenshtein. Since the machine in our experimental environment has 24 physical cores (i.e., two processors with 12 cores each), using 24 threads obtains the best results: 188MB/s with Hamming, 108.1MB/s with Delta-Hamming, and 65.3MB/s with Levenshtein.

The Hamming algorithm results in the best throughput because it is the simplest encoding to be restored. The Hamming and Levenshtein algorithms have the number of operations directly proportional to the length of the used sequences since they store `UNMODIFIED` operations when characters from both sequences match. However, the Delta-Hamming has potential in obtaining a higher throughput as the modelled index becomes better. If the best base chunk for each queried sequence results in less `SUBSTITUTION` operations, then the Delta-Hamming becomes proportionally smaller and reduces the restore time.

As mentioned in Section 4.2, the restore throughput from the ten selected compression algorithms range as follows: ZPAQ, Quip, and Fqzcomp reach less than 10MB/s; SPRING

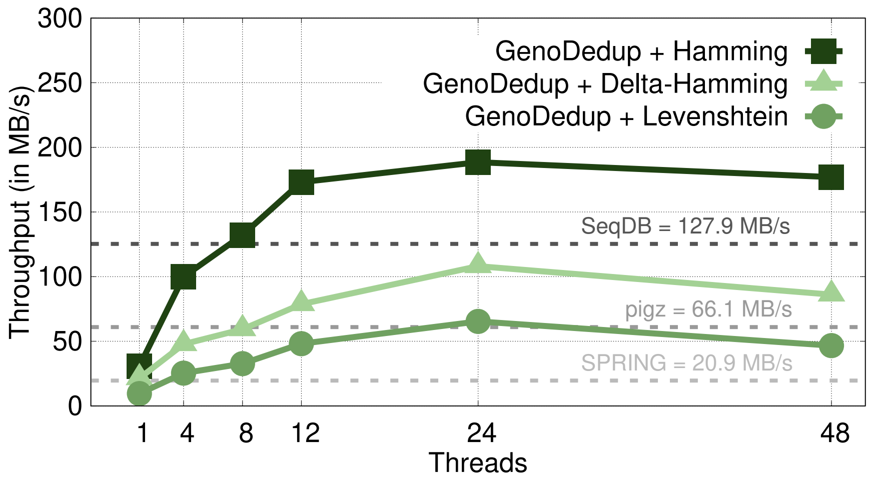


Figure 4.4: Average throughput of reads.

reaches 20MB/s; GZIP, BSC, and FaStore reach 40–50MB/s, pigz reaches 66MB/s, and DSRC2 and SeqDB reach 125MB/s. These values refer to decompressing the whole FASTQ file in the specialised tools, not only quality scores as in the results from Figure 4.4. Restoring only the QS data from GenoDedup is up to 2.84× faster than pigz, the fastest generic competitor and up to 170× faster than ZPAQ, the generic algorithm with the best compression ratio for QS.

4.5.2.2 Write Operations

We evaluate the performance of string comparisons using different encoding algorithms and how do they interfere with the performance of GenoDedup (i.e., mainly steps 15–17 in Figure 4.2). Similarity-based deduplication directly depends on the number of comparisons necessary to find the nearest neighbour (i.e., the best candidate) of a queried sequence. Genome sequencing data is usually written once and read many times later for processing. Systems for genome sequencing data should support a write throughput of at least 0.3MB/s—the average write throughput from an NGS machine [172]—to not become a bottleneck in an NGS pipeline.

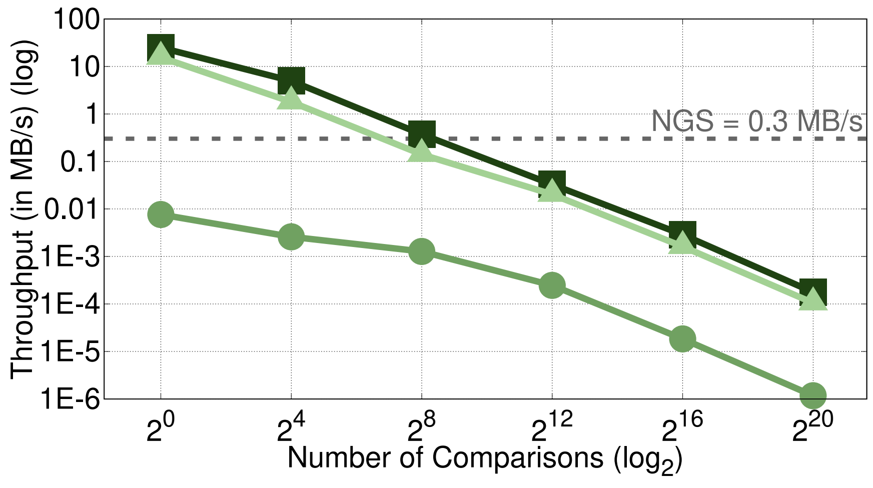


Figure 4.5: Average throughput of writes.

Figure 4.5 presents the throughput (in MB/s) obtained when comparing a single queried sequence to *all entries* in deduplication indexes of different sizes (from 2^0–2^{20}) using a single thread. As expected, the more entries to compare in the index, the smaller the throughput is. More specifically, GenoDedup reaches 25MB/s when comparing the queried sequence to a single candidate using Hamming encoding (i.e., 0.004ms per comparison), 15.5MB/s using Delta-Hamming (i.e., 0.0064ms), and 0.0076MB/s using Levenshtein (i.e., 13.11ms). From these results, Levenshtein is two to three orders of magnitude slower than Hamming and Delta-Hamming algorithms.

GenoDedup must process at least 3000 queries (of 100 characters each) per second to support the 0.3MB/s throughput from NGS machines [172]. The maximum number of comparisons on each query, to reach at least 0.3MB/s, is 422 for Hamming and 113 for Delta-Hamming. Levenshtein requires a speed-up of 40 to reach 0.3MB/s when comparing the queried sequenced to a single entry, which makes this algorithm unappealing for our solution. Parallelisation can improve GenoDedup to support higher write throughput in the future. Currently, it means that queries should return fewer deduplication candidates than these numbers; otherwise, the system does not support the required throughput of 0.3MB/s.

Conveniently, reducing the number of candidates returned in a query is the exact benefit LSH brings to GenoDedup. It makes Hamming and Delta-Hamming algorithms even more feasible. For instance, in an LSH with a similarity threshold of 0.95 and 128 permutations on the MinHash, it can reduce the number of candidate comparisons from 350 million entries (333$\times$ bigger than 2^{20}) to only 50. Such reduction contributes, for instance, to approximate the write throughput of GenoDedup with the one of the ZPAQ (i.e., 5.3MB/s). GenoDedup has the potential of reaching higher write throughput with parallelism since FASTQ entries are processed unrelated to each other.

These results consider only the string comparison. Usually, there are other steps (e.g., communication and parsing) that need to be considered. However, 0.3MB/s per second is an achievable throughput for most modern service solutions.

4.5.3 Large End-to-End Workload

Our last experiment evaluates GenoDedup with a large workload in an end-to-end scenario. The evaluation considers approximately 265GB from all files in Table 4.3. We compare our results with the most prominent competitors in terms of compression ratio (i.e., SPRING [49]) and read throughput (i.e., SeqDB [133]). We also add the DSRC2 [213] and pigz to the comparison. Table 4.7 presents the observed results. The complete set of components in our end-to-end solution encompasses: (1) the algorithm from Bhola *et al.* [35] to compress FASTQ comments with a ratio of 17.26$\times$ on average; and our similarity-based deduplication both (2) for DNA sequences, which compresses them 13.43$\times$; and (3) for QS sequences (with 2^8 entries), which compresses them 1.88$\times$.

SPRING compresses the mentioned FASTQ files 6.023$\times$, which is the biggest compression ratio observed in our experiments. For the same dataset, SeqDB achieves a compression ratio of 1.992$\times$, DSRC2 4.148$\times$, and pigz 3.227$\times$. Our end-to-end deduplication solution achieves a compression ratio of 4.089$\times$ using a deduplication index with the human reference genome *hg38* for the DNA sequences and with 2^8 entries for the QS sequences. This result corresponds to 67% the compression ratio of SPRING, 98.6% the ratio of DSRC2, while we compress 2.05× more than SeqDB and 1.26× more than pigz.

Our solution compresses data at 0.3MB/s with an index of 2^8 candidates for the QS sequences in a single thread. The other solutions perform better than us in terms of the compression speed. However, GenoDedup can reach higher speeds with multi-threading, with better deduplication indexes, and with the use of LSH to reduce the number of candidate comparisons (e.g., it reaches almost 10MB/s when comparing 2^4 entries).

In terms of restoring throughput, SeqDB decompressed the selected FASTQ files at 127.9MB/s (i.e., the fastest competitor), DSRC2 at 125.3MB/s, pigz at 66.1MB/s, and SPRING at 20.9MB/s. Our end-to-end solution achieves a read throughput of 208.25MB/s in the same dataset. This result makes GenoDedup 1.62× faster than SeqDB, 1.66× faster than DSRC2, 3.15× faster than pigz, and 9.96× faster than SPRING.

An important aspect of our solution is that achieving better compression ratios is possible in the future, and it does not compromise our read throughput since it is independent of the number of candidates in the deduplication index. Furthermore, a better compression ratio implies more `UNMODIFIED` edit operations, which accelerates the restore process even more.

Applying these results to a repository like the 1000 Genomes Project [57] provides a better figure of the savings GenoDedup can bring to big genome data warehouses. The project currently stores approximately 115TB of FASTQ files compressed with GZIP (i.e., the equivalent to 370TB of uncompressed FASTQ files). By using our deduplication strategy, it would be able to store such files using only 90TB, which corresponds to 78% of the 115TB used today with GZIP. Perhaps even more importantly, GenoDedup would also improve their data sharing ecosystem by allowing data consumers to restore FASTQ files 5× faster than today (i.e., GZIP decompressed our files at 41.4MB/s).

Algorithm	C.R.	Write	Read
SPRING	{1.47×}	{143.6×}	9.96×
DSRC2	{1.01×}	{4586.3×}	1.66×
SeqDB	2.05×	{1385.3×}	1.62×
pigz	1.26×	{937×}	3.14×

Table 4.7: Comparison (i.e., the ratio *best/worst*) of the compression ratio (C.R.) and the (write and read) throughput between GenoDedup with 2^8 QS candidates and its main competitors. Brackets ({ }) indicate where competitors are better than GenoDedup.

4.6 Discussion

The methods proposed in this chapter are generic enough to support sequencing data from other species and NGS machines, as well as other data representations (e.g., aligned data) and file formats (e.g., SAM). Additionally, it can be explored in other highly-dimensional data where identity-based deduplication fails [91]. In this section, we discuss how our solution can work with or be adapted to support other datasets and methods.

4.6.1 Other Data Representations

As previously mentioned, *sequencing data* is considered the purest unbiased version of genomic data coming from Next-Generation Sequencing (NGS) machines [61]. Contrarily, the output from alignment and assembly processes is imprecise, lossy, and depend on the employed algorithm and reference [165].

For instance, aligned data in the 1000 Genomes Project [57] was generated using different algorithms and references in distinct phases of the project. Studies that use data from multiple of these phases must first reconvert the aligned data into sequencing data, and then realign all the data of interest using the same reference and algorithm before analysing it. This rework in converting from aligned to sequencing data takes considerable additional time (e.g., 50–200 minutes for each 100GB [236]) and is even more likely to be required in studies that involve large quantities of samples and data from multiple sources. It is no surprise that the 1000 Genomes Project stores the original sequencing data (i.e., FASTQ) of every aligned data (i.e., SAM/BAM) they have.

Notwithstanding, our methods can be used with *aligned data* (e.g., in the SAM/BAM format [166]). The only difference is that this data representation already contains the pointer to the best candidate of DNA sequences in the aligned file. It only eliminates the need to execute the similarity-based deduplication for the matched DNA sequences. However, it still requires (1) a compression algorithm for the QS sequences, in which our similarity-based deduplication has shown its potential, (2) a delta-encoding (e.g., our Delta-Hamming) for the matched DNA sequences, and (3) a similarity-based solution like ours to the unmatched ones.

Another important data representation is the *assembled data*. However, recovering the original sequencing data file from assembled data is impossible because the resulting assembled genome file (i.e., FASTA) does not contain details such as: How many FASTQ entries were used to create the assembled genome? In which genome position each one of them has started and ended? Which was the quality score for each sequenced nucleobase? Additionally, we consider the compression of assembled data a challenge that has been mostly addressed by different approaches. For instance, we devised a tool that reduces assembled human genomes $\sim 700\times$ in 40 seconds [11].

4.6.2 Paired-end Sequencing

Paired-end sequencing digitises both ends of DNA fragments to increase accuracy and help in detecting repetitive sequences and rearrangements. It produces two FASTQ files, one for each end, where the entry order matches among them (i.e., entry 15 from the second file is the reverse complement of the DNA sequence of entry 15 from the first file). GenoDedup may explore this additional redundancy in the future to eliminate the similarity-search for the DNA sequences of the second FASTQ file. In this case, GenoDedup may use the reverse complement of the best candidate of the DNA sequence from the first file also as the best candidate for the DNA sequence in the second file. It will reduce the size of our encodings and speed-up the deduplication and restore since only one pointer is used as the best candidate of two sequences.

4.6.3 Other Species

In this work, we favour sequencing data from human genomes due to the availability of a comprehensive reference genome and the potential impact of this data domain. However, the methods proposed in this work can easily be adapted to work with sequencing data from other species. In an extreme case, one will end up with one deduplication index for DNA of each species. Moreover, species with a representative reference genome have the advantage of using it as the deduplication index for DNA, but it is not a requirement since the same method of modelling the index of QS sequences can be used for DNA.

4.6.4 Other Sequencing Machines

Many related works for FASTQ compression select genomes from several species and sequencing machines. This choice usually results in selecting only a few genomes per species or selecting small FASTQ files with very low coverage. We intended to select more and bigger genomes from the same species and the same sequencing machine to reduce the influence of these two variables in our results.

Our datasets include only human genomes (due to the previously mentioned reasons) sequenced with the Illumina HiSeq 2000 platform. To the best of our knowledge, this machine was the most used NGS machine in sequencing laboratories around the world when we started this work [120]. Additionally, some of the selected datasets were also used in other papers on FASTQ compression (e.g., SRR400039 in Quip's paper [151]).

Datasets in our work had an expected alphabet of forty possible QS [137]. Newer Illumina platforms have been binning QS into groups with reduced alphabets (e.g., seven groups in HiSeqX10 [139] and four in NovaSeq [140]). This binning is similar to the initial approach of many lossy FASTQ compression algorithms [196]. These smaller alphabets reduce the size of our encoding and may benefit our index modelling since it also reduces the possible combinations. Notwithstanding, our methods can work with data from most of the modern NGS machines. The differences in QS distribution patterns and alphabet may require one to model new deduplication indexes, and one may end up with one index per machine in an extreme case.

4.6.5 Other Sequence Lengths

Our methods support sequences of any length while working even with indexes containing sequences with multiple lengths. The length influences several aspects in our solution (e.g., index size, string comparison time, chances of finding more differences). More specifically, LSH supports entries of different sizes when using MinHash. MinHash converts entries of any size into hashes of fixed size proportional to Jaccard distance (i.e., also independent on the entry size). Furthermore, Levenshtein edit operations can compare strings of different sizes since it includes insert and delete operations. In an extreme case, one may model a few different deduplication indexes for different entry sizes. However, big differences in the size of the query sequences and the modelled ones may reduce the compression ratio.

Additionally, the bigger the sequence length is, the bigger the chances of having more edit operations, which tends to reduce the compression ratio and throughput on the selection of the best candidate. The impact of this length in GenoDedup is proportional to the impact of the sequence length in the string distance calculation.

4.6.6 Reordering FASTQ entries

Reordering FASTQ entries to group similar entries is another pattern explored in the literature [49, 245]. The current version of our methods works entry by entry, without correlating them or their order. GenoDedup compresses the DNA and QS sequences independently and reordering them separately would reduce its compression ratio and performance since it requires storing and working with additional correlating metadata.

4.7 Final Remarks

This chapter proposed the GenoDedup, which is the first method to integrate efficient similarity-based deduplication and specialised delta-encoding for genome sequencing data. Data resulting from this chapter (i.e., the `Reduction` phase of our storage pipeline) is composed of two compressed deduplicated files per genome, one for the privacy-sensitive portion and another for the non-sensitive one. The obtained reduction in this phase (i.e., $4.09\times$) enables the efficient storage of whole human genomes in the `Storage` phase of our pipeline, which stores each genome portion in the most appropriate location.

5 Auditability of Effective Reads in Register Emulations

Characteristics like cost-effectiveness, high scalability, and ease of use promoted the migration from private storage infrastructures to public multi-tenant clouds in the last decade. Security and privacy concerns were the main deterrents for this migration since the beginning [231]. Numerous secure storage systems have been proposing the use of advanced cryptographic primitives to securely disperse critical data (e.g., human genomes) across multiple clouds (i.e., independent administrative domains [3]) to reduce the risk of data breaches [225] (e.g., DepSky [28] and AONT-RS [212]).

Given a secure storage system (e.g., we use CHARON [183] in this thesis) composed of n storage objects, these *information dispersal* techniques split and convert the original data item into n coded blocks [161, 207, 209, 225]. Each coded block is stored in a different base object, and clients need to obtain only τ out of n coded blocks to effectively recover the original data. In this type of solution, no base object stores the whole data item, which differentiates information dispersal from fully-replicated storage systems (e.g., [2, 16, 179])—where each object retains a full copy of the data.

Despite the advances in secure storage systems, the increasing severity of data breaches (e.g., [220]) and the tightening of privacy-related regulations (e.g., GDPR [201]) have been driving the demand for further improvements on this topic. For instance, *auditability* [158] enables the systematic verification of who has effectively read data (e.g., the privacy-sensitive portion of a human genome) in secure storage systems. Notably, this verification allows one to separate these users from the whole set of clients that are authorised to read data but have never done so. It is an important step to detect data breaches (including those caused by authorised users, e.g., Snowden's case [126]), analyse leakages, and sanction misuses.

Problem. In this chapter, we address the following question: *How to extend resilient storage emulations with the capability of auditing who has effectively read data?* More specifically, we address the problem of protecting storage systems from readers trying to obtain critical data without being detected (i.e., audit completeness) and protecting correct readers from faulty storage objects trying to incriminate them (i.e., audit accuracy). The answer must encompass the techniques used in these emulations, such as information dispersal [161, 209, 225] and available quorum systems [179], for providing a R/W register abstraction [162]. We consider information dispersal the primary form of auditable register emulations because alternative solutions that replicate the whole data (even if encrypted) can suffer from faulty base objects leaking data (and encryption keys) to readers without logging this operation.

Contributions. This chapter initiates the study of auditing in resilient storage by presenting lower bounds and impossibility results related with the implementation of an *auditable register* on top of n base objects that log read attempts despite the existence of f faulty ones. Our results show that, given a minimum number τ of data blocks required to recover a data item from information dispersal schemes (e.g., CHARON [183]) and a maximum number f of faulty storage objects (1) auditability is impossible with $\tau \leq 2f$; (2) when fast reads (reads executed in a single communication round-trip [117]) are supported, $\tau \geq 3f+1$ is required for implementing a weak form of auditability, while a stronger form of auditability is impossible; (3) signing read requests overcomes the lower bound of weak auditability; and (4) totally ordering operations or using non-fast reads can still provide strong auditability. Most discussions in this chapter are generic for any critical data, however they can be adapted directly to the data of interest of this thesis, i.e., human genome sequencing data. See Chapter 6 for a more detailed integration of this auditability into our storage pipeline.

The remainder of this chapter is organised as follows. Section 5.1 contains the theoretical preliminaries, Section 5.2 introduces the idea of auditable register emulations, Section 5.3 presents some preliminary results, and Section 5.4 contains impossibility results for the audit properties. Section 5.5 describes our audit algorithm and proves the lower bounds of audit properties, and Section 5.6 presents alternative models to reduce the requirements of weak auditability or to achieve stronger auditability. Finally, Section 5.7 concludes the chapter.

5.1 Preliminaries

System Model. Our system is composed of an arbitrary number of *client processes* $\Pi = \{p_1, p_2, ...\}$, which interact with a set of *n storage objects* $\mathbb{O} = \{o_1, o_2, ..., o_n\}$. Clients can be subdivided into three main classes: *writers* $\Pi_W = \{p_{w1}, p_{w2}, ...\}$, *readers* $\Pi_R = \{p_{r1}, p_{r2}, ...\}$, and *auditors* $\Pi_A = \{p_{a1}, p_{a2}, ...\}$. These different roles do not necessarily mean they have to be performed by different processes.

A *configuration C* is a vector of the states of all entities (i.e., processes and objects) in the system. An *initial configuration* is a specific configuration where all entities of the system are in their initial state. An *algorithm Alg* defines the behaviour of processes in Π and objects in $\mathbb{O}$ as deterministic state machines, which can modify the system's states through *actions* (e.g., invoke and response). Clients *invoke* operations in the storage objects and wait for *responses*. An *execution segment* is a (finite or infinite) sequence of alternated configurations and actions. An *execution* ρ is an execution segment that begins in an initial configuration. An *event* is the occurrence of an action in an execution.

A finite sequence of invocations and responses compose a *history* σ of the system. We consider the following relationships between operations within a history. First, if a history σ contains the invocation of an operation op_1 and its response, then op_1 is *complete* in σ. Otherwise, if σ contains only the former, then operation op_1 is *incomplete*. Second, if the response of an operation op_1 precedes the invocation of another operation op_2 in σ, then op_1 *precedes* op_2. Third, if operations op_1 and op_2 do not precede each other in σ, then they are *concurrent*. Fourth, a history σ that does not contain concurrent operations is considered *sequential*.

Information Dispersal. We consider a high-level shared storage object that stores a value v from domain $\mathbb{V}$ using information dispersal schemes (i.e., a special case of disintegrated storage [27]). These schemes provide efficient redundancy since they disperse pieces of information across multiple locations instead of replicating the whole value on each one of them. Moreover, they allow clients to recover the original stored value only after obtaining a subset of these pieces of information. Examples of information dispersal schemes include erasure codes [207, 209] and secret sharing [161, 222].

This object provides two high-level operations: *a-write(v)* and *a-read()*. A high-level *a-write(v)* operation converts a value $v \in \mathbb{V}$, passed as an argument, into n coded blocks $b_{v_1}, b_{v_2}, ..., b_{v_n}$ from domain $\mathbb{B}$, and each coded block b_{v_k} is stored in the k^{th} base object in $\mathbb{O}$ (as described bellow). The employed techniques guarantee that no untrusted base object o_k stores an entire copy of value v and no reader recovers v by obtaining less than a certain fraction of these blocks.

A high-level *a-read()* operation recovers the original value v from any subset of a specific number (τ) of blocks b_{v_k}. It means readers do not need to execute low-level reads in all n base objects to obtain these τ blocks. Base objects in this work are *loggable R/W registers*.

Loggable R/W Register Specification. A *loggable read-write (R/W) register* is an object o_k that stores a data block b_{v_k} from domain $\mathbb{B}$ and has a log L_k to store records of every read operation that this base object has responded. This object o_k provides three low-level operations [162]:

- *rw-write(b_{v_k})*: writes the data block $b_{v_k} \in \mathbb{B}$, passed as an argument, in this base object o_k and returns an *ack* to confirm that the operation has succeeded.
- *rw-read()*: returns the data block $b_{v_k} \in \mathbb{B}$ currently stored in this base object o_k (or $\perp \notin \mathbb{B}$ if no block has been written on it).
- *rw-getLog()*: returns the log L_k of this base object o_k.

The behaviour of a loggable R/W register is given by its sequential specification. A low-level *rw-write* operation always writes the block passed as an argument and returns an *ack*. A low-level *rw-read* operation returns the block written by the last preceding *rw-write* operation on this object or $\perp \notin \mathbb{B}$ if no write operation has been executed in this object. Additionally, the low-level *rw-read* operation also creates a record $\langle p_r, label(b_{v_k})\rangle$ about this read operation in its log L_k, where p_r is the identifier of the reader that invoked the *rw-read* operation and $label(b_{v_k})$ is an auxiliary function that, given a block b_{v_k}, returns a label associated with the value v (from which the block b_{v_k} was derived). A low-level *rw-getLog* returns the log L_k containing records of all preceding *rw-read* executed in this object o_k. A history σ of low-level operations in this object is linearisable [129] if invocations and responses of concurrent operations can be reordered to form a sequential history that is correct according to its sequential specification. This object is linearisable if all of its possible histories are also linearisable.

We define a *providing set* based on the notion introduced by Lamport [163] to abstract the access to multiple base objects. A providing set is the set of base objects that have both stored a block associated with value v and responded this block to a reader p_r in history σ.

Definition 1 (Providing Set)**.** The set of objects $P^{\sigma}_{p_r,v}$ is providing blocks associated to value v for the reader p_r in a history σ iff σ contains, for every object $o_k \in P^{\sigma}_{p_r,v}$, an event in which o_k receives a write request to store b_{v_k} and an event in which o_k responds b_{v_k} to a read request from p_r.

Given a finite history σ being audited, there is only one possible providing set $P^{\sigma}_{p_r,v}$ for each pair $\langle p_r, v \rangle$ of reader p_r and value v. For the sake of simplicity, this set is presented only as $P_{p_r,v}$ from now on. Every single base object o_k that stored block b_{v_k} and returned this block to p_r belongs to the providing set $P_{p_r,v}$, independent on how many times o_k has returned b_{v_k} to p_r. There might exist cases where providing sets for different pairs of readers and values are composed of the same objects.

We introduce the notion of an *effective read*, which characterises a providing set large enough that a reader p_r is able to effectively read value v from the received blocks.

Definition 2 (Effective read)**.** A value $v \in \mathbb{V}$, written in w_v, is *effectively read* by a reader p_r in history σ iff $|P^{\sigma}_{p_r,v}| \geq \tau$, i.e., p_r has received at least τ distinct blocks b_{v_k} associated with value v from different base objects $o_k \in P^{\sigma}_{p_r,v}$ in history σ.

An effective read depends only on the number of different coded blocks (derived from the same value v) that a reader p_r has already obtained from different base objects in the same history. Reader p_r does not necessarily obtain all these blocks in a single high-level *a-read* operation. Due to unlimited concurrency, there might exist cases where it is accomplished only after receiving responses from many subsequent *a-read* operations.

High-level operations in our system have two additional particularities. First, high-level read operations (i.e., *a-read*) are *fast reads*.

Definition 3 (Fast read [117])**.** An *a-read* operation is *fast* if it completes in a single communication round-trip between the reader and the storage objects.

Second, we consider multi-writer multi-reader (MWMR) shared objects where multiple writers from Π_w execute high-level *a-write* operations and multiple readers from Π_R execute high-level *a-read* operations.

Fault Model. Clients (i.e., writers, readers, and auditors) and storage objects that obey their specifications are said to be *correct*. *Faulty writers* and *faulty auditors* are honest and can only fail by crashing. *Faulty readers* may crash or request data to only a subset of objects, which may characterise an attack where it attempts to read data without being detected and reported by the audit.

Writers are trusted entities because they are the data owners, who are the most interested party in the auditable register we are proposing in this work. This assumption is common (e.g., [28, 179]) since malicious writers can always write invalid values, compromising the application's state. Furthermore, auditors are also trusted because they are controlled either by the same entity as writers or by third-party entities writers trust.

Faulty storage objects can be nonresponsive (NR) [149] in an execution since they may crash, omit their values to readers, omit read records to auditors, or record nonexistent read operations. More specifically, they can fail by NR-omission when accessing their values and NR-arbitrary when accessing their log records [149]. Omitting records to auditors means a faulty object may be helping a reader to evict being detected by the auditor. Producing records for nonexistent reads characterises an attack where a faulty object may be trying to incriminate a reader. Furthermore, we assume no more than f storage objects are faulty.

5.2 Auditable Register Emulations

We extend the aforementioned register emulation with a high-level operation *a-audit()*. It uses the fail-prone logs obtained from the *rw-getLog* operation in loggable R/W registers to define an *auditable register emulation*. This emulation has access to a *virtual log* $L \subseteq \bigcup_{k \in \{1..n\}} L_k$ from which we can infer who has effectively read a value from the register.

The *a-audit* operation obtains records from L and produces a *set of evidences* E_A about the effectively read values. We define a threshold t as the required *number of collected records* to create an evidence $\mathcal{E}_{p_r,v}$ of an effective read. Each *evidence* $\mathcal{E}_{p_r,v}$ contains at least t records from different storage objects o_k proving that v was *effectively read* by reader p_r

in history σ. This threshold t is a configurable parameter that depends on the guarantees *a-audit* operations provide (defined bellow).

We define an *auditing quorum* A as the set of objects from which the *a-audit* collects fail-prone individual logs to compose the set of evidences. To ensure audit operations are available in our asynchronous system model, we consider $|A| = n - f$.

We are interested in auditing effective reads because we aim to audit who has actually read a data value v—including faulty readers that do not follow the read protocol and leave operations incomplete. A correct auditor receives E_A and reports all evidenced reads.

An *auditable register* provides an *a-audit()* operation that guarantees *completeness* (i.e., Definition 4) and at least one form of *accuracy* (i.e., Definitions 5 and 6).

Definition 4 (Completeness)**.** Every value v *effectively read* by a reader p_r before the invocation of an *a-audit* in history σ is reported in the E_A from this audit operation, i.e., $\forall p_r \in \Pi_R, \forall v \in \mathbb{V}, |P_{p_r,v}| \geq \tau \implies \mathscr{E}_{p_r,v} \in E_A$.

Definition 5 (Weak Accuracy)**.** A correct reader p_r that has never invoked an *a-read* (i.e., never tried to read any value v) before the invocation of an *a-audit* in history σ will not be reported in the E_A from this audit operation, i.e., $\forall p_r \in \Pi_R, \forall v \in \mathbb{V}, P_{p_r,v} = \varnothing \implies \mathscr{E}_{p_r,v} \notin E_A$.

Definition 6 (Strong Accuracy)**.** A correct reader p_r that has never effectively read a value v before the invocation of an *a-audit* in history σ will not be reported in the E_A from this audit operation as having read v, i.e., $\forall p_r \in \Pi_R, \forall v \in \mathbb{V}, |P_{p_r,v}| < \tau \implies \mathscr{E}_{p_r,v} \notin E_A$.

While completeness is intended to protect the storage system from readers trying to obtain data without being detected, accuracy focuses on protecting *correct readers* from faulty storage objects incriminating them.

Both variants of accuracy guarantee that auditors report only readers that have invoked read operations. The difference between them is that strong accuracy guarantees that auditors report only readers that have effectively read some value, while weak accuracy may report readers that have not effectively read any value. Strong accuracy implies weak accuracy. Additionally, the accuracy property provides guarantees to correct readers only, which means that auditors may report faulty (honest or malicious) readers in incomplete or partial reads because they had the intention to read the data and aborted it or crashed while doing so.

We consider *weak auditability* when the storage system provides completeness and weak accuracy in audit operations and *strong auditability* when it provides completeness and strong accuracy.

5.3 Preliminary Results

In the presence of faulty base objects, it is impossible to audit systems in which readers can recover values by accessing a single object (e.g., *non-replicated* and *fully-replicated* systems). The reason is that, in these solutions, every base object stores a full copy of data and can give it to readers without returning the log record of this operation to auditors. Therefore, we consider *information dispersal* techniques (e.g., [161, 209, 225]) as the primary form of auditable register emulations. Moreover, we assume these information dispersal schemes require $\tau > f$ blocks to recover the data because, otherwise, the f faulty objects may also deliver their blocks to readers and omit the records of these operations from auditors.

Records Available for Auditing Registers. As a starting point, we identify the minimum number of records from each preceding effective read that will be available for any *a-audit* operation. It relates to the minimum number of correct objects that are present at the smallest intersection of any providing set $P_{p_r,v}$ and an auditing quorum A.

Lemma 1. *Any available a-audit operation obtains at least $\tau - 2f$ records of every preceding effective read in history σ.*

Proof. Let us assume a system configuration composed of n storage objects divided into four groups G_{1-4}, as depicted in Figure 5.1. Objects within the same group initially contain the same value v and an empty log $L_k = \varnothing$. Without loss of generality, group G_1 contains the f faulty objects of the system, G_2 contains $\tau - 2f$ correct objects, G_3 contains f correct objects, and G_4 contains the remaining $n - \tau$ correct objects of the system.

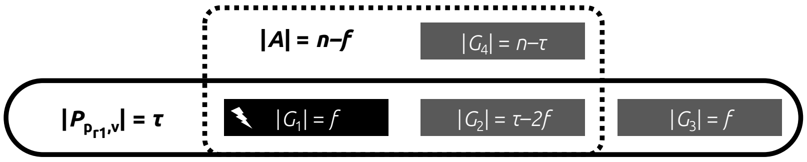


Figure 5.1: A configuration with a providing set $P_{p_{r1},v}$ of an effective read and an auditing quorum A.

A reader p_{r1} effectively reads a value v after obtaining exactly τ blocks b_{v_k} from objects in a providing set $P_{p_{r1},v}$ composed of the f faulty objects in G_1 (that do not record the read operation) and $\tau - f$ correct objects in $G_2 \cup G_3$ (that do record it). In the worst-case scenario, an available *a-audit* operation receives $n - f$ *rw-getLog* responses from an auditing quorum A composed of the f faulty objects in G_1 (that belong to the providing set $P_{p_{r1},v}$ but return empty logs L_k to auditors), $\tau - 2f$ correct objects in G_2 (that belong to the providing set $P_{p_{r1},v}$ and return the correct records of this read operation), and $n - \tau$ correct objects in G_4 (that do not belong to $P_{p_{r1},v}$).

In this case, auditors receive only the $\tau - 2f$ records from the objects in G_2, which are the only correct objects at the intersection of $P_{p_{r1},v}$ and A. This is the minimum number of records of an effective read available to an auditing quorum since any other configuration makes auditors receive more records of this effective read, i.e., in any alternative scenario, the faulty objects in G_1 may correctly record the read operation and return these records to auditors or the auditing quorum A may also include up to f correct objects from G_3 (that belong to $P_{p_{r1},v}$) instead of objects from G_4. □

Receiving this minimum number $\tau - 2f$ of records from an effective read must be enough to trigger the creation of an evidence for that read in *a-audit* operations. As a consequence, the value of t must be defined considering this constraint.

5.4 Resilience Lower Bounds

This section identifies impossibility results for the properties of auditable registers: completeness (Definition 4), weak accuracy (Definition 5), and strong accuracy (Definition 6).

Lemma 2. *It is impossible to satisfy the* **completeness** *of auditable registers with $\tau \leq 2f$.*

Proof. Consider an auditable register implemented using the same four subset groups of objects G_{1-4} from Lemma 1 (depicted in Figure 5.1). Without loss of generality, let us assume that $\tau = 2f$. As a consequence, group G_2 will be empty because $|G_2| = \tau - 2f = 0$.

A reader p_{r1} obtains τ coded blocks for value v from a providing set $P_{p_{r1},v}$ composed of f faulty objects in G_1 (which do not log the operation) and $\tau - f = f$ correct nodes in G_3 (that log the operation). This reader can decode the original value v after receiving these τ blocks, performing thus an effective read. Consequently, $|G_2| = 0$ and any auditing quorum A that does not include at least one object from G_3 will not receive any record for the read operation from reader p_{r1}. For instance, an auditing quorum $A = G_1 \cup G_2 \cup G_4$ (e.g., from Figure 5.1) receives no record of this read in this history, which violates the completeness property since an evidence cannot be created without records (i.e., $t \geq 1$). □

Lemma 3. *It is impossible to satisfy the* **weak accuracy** *of auditable registers with $t \leq f$.*

Proof. Consider an auditable register implemented using $n - f$ correct objects and f faulty objects. A read has never been invoked in this history of the system, but the f faulty objects create records for a nonexistent read operation from a reader p_r. Any auditing quorum A that includes these faulty objects will receive f records for a read operation that has never been invoked. If the threshold $t \leq f$ is enough to create an evidence and report an effective read, then a correct auditor must report it, violating the weak accuracy property. □

The next theorem shows in which conditions these two properties cannot be supported simultaneously.

Theorem 1. *It is impossible to satisfy both* **completeness** *and* **weak accuracy** *of auditable registers with $\tau \leq 3f$.*

Proof. Consider an auditable register implemented using the same four subset groups of objects G_{1-4} from Lemma 1 (depicted in Figure 5.1). Without loss of generality, let us assume that $\tau = 3f$ is enough to satisfy both the completeness and weak accuracy properties. As a consequence, group G_2 will contain $|G_2| = \tau - 2f = f$ objects.

A reader p_{r1} obtains τ blocks for value v from a providing set $P_{p_{r1},v}$ composed of f blocks from the faulty objects in G_1 (which do not log the operation) and other $\tau - f$ correct objects in $G_2 \cup G_3$ (that log the operation). The auditing quorum A will produce $\tau - 2f = f$ records (Lemma 1). Evidences must be created using these f records to report the effective read and satisfy completeness. However, as proved in Lemma 3, it is impossible to guarantee weak accuracy with $t \leq f$. □

Now, we turn our attention to strong accuracy, i.e., the capability of an auditor to report exactly which value v each reader p_r has effectively read.

Lemma 4. *It is impossible to satisfy the* **strong accuracy** *of auditable registers with $t < \tau + f$.*

Proof. Consider an auditable register implemented using five subset groups of objects G_{1-5}, as depicted in Figure 5.2. The main difference to the scenario from Figure 5.1 is that the group G_4 was subdivided into two groups: G_4 with $2f - 1$ objects and G_5 with $n - \tau - 2f + 1$ objects.

This configuration has an incomplete write operation[1] of a value $x \in \mathbb{V}$ that has arrived to objects from all groups but G_2 and G_4. A reader p_{r2} obtains blocks of value v from objects from groups G_2 and G_4, which log the record for the read, and form a providing set with size $|P_{p_{r2},v}| = \tau - 2f + 2f - 1 = \tau - 1$. In this history, due to the incomplete write

[1]An incomplete write operation may be caused, for instance, by network delays in clients' requests or a writer that has crashed in the middle of a write operation.

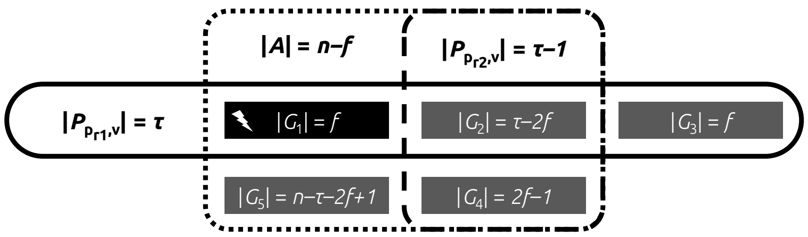


Figure 5.2: A configuration with two providing sets and an auditing quorum A. Only $P_{p_{r1},v}$ represents an effective read.

operation of value x, reader p_{r2} has not received τ blocks for value v, which means it cannot recover v.

However, the f faulty objects in G_1 decide to mimic groups G_2 and G_4 and log the record for the read of value v by reader p_{r2}. Any auditing quorum A that includes the objects from groups G_1, G_2, and G_4 will return $\tau + f - 1$ records. If $t < \tau + f$ is enough to produce an evidence of an effective read, then the auditor will report this read from reader p_{r2} (i.e., not an effective read), violating the strong accuracy property.

□

Theorem 2. *It is impossible to satisfy both* **completeness** *and* **strong accuracy** *of auditable registers.*

Proof. Consider an auditable register implemented using the same five subset groups of objects G_{1-5} from Lemma 4 (depicted in Figure 5.2). Without loss of generality, let us assume $\tau \geq 2f + 1$ is required to guarantee the completeness of audit operations (Lemma 2). As a consequence, G_2 contains $\tau - 2f \geq 1$ object and G_5 contains $n - \tau - 2f + 1 \leq n - 4f$ objects.

A reader p_{r1} executes a read operation in groups G_{1-3}. Faulty objects from G_1 return their blocks for value v (but do not log the read operation). Reader p_{r1} receives exactly τ correct data blocks and obtains the original value v (i.e., an effective read).

A writer p_w leaves incomplete an operation to write a value x, similarly to the configuration of Lemma 4, which has been received by objects in all groups but G_2 and G_4. A second reader p_{r2} executes a read operation that receives $\tau - 1$ blocks for value v from a providing set $P_{p_{r2},v}$. These $\tau - 1$ blocks are insufficient to recover the original value v (i.e., it does not represent an effective read). Faulty objects from G_1 decide to mimic groups G_2 and G_4 and log the read of v by p_{r2}.

After these two reads, any auditing quorum A that includes G_4 receives the same number or more records of the read from p_{r2} (i.e., not an effective read) than records of the effective read from p_{r1}. As a consequence, it is impossible to define a single reporting threshold t to be used to produce evidences without violating either completeness or strong accuracy.

For instance, the auditing quorum $A = G_1 \cup G_2 \cup G_4 \cup G_5$ receives $\tau + f - 1$ records $\langle p_{r2}, label(b_{v_k}) \rangle_k$ and $\tau - 2f$ records $\langle p_{r1}, label(b_{v_k}) \rangle_k$. Defining $t \leq \tau - 2f$ is enough to report the effective read from p_{r1}, but it violates the strong accuracy by also reporting the (not effective) read from p_{r2}. Alternatively, defining $t \geq \tau + f$ satisfies the strong accuracy because it does not report the read from p_{r2}, but it violates the completeness because it does not report the effective read from p_{r1}. $\square$

Remark: Adding any number of objects to the scenario of Figure 5.2 does not change the impossibility result of Theorem 2. The reason is that with the unlimited concurrency in our model, each additional object may have a value different from all values stored in the other objects.

5.5 Audit Algorithm

We present a generic auditability algorithm for register emulations and prove that all bounds from the previous section (Lemmata 2–4 and Theorem 1) are tight in our system model when using this algorithm. Concretely, we prove that this algorithm satisfies the completeness property with $t \geq 1$ and $\tau \geq 2f + 1$ (Lemma 5) and the weak accuracy with $t \geq f + 1$ (Lemma 6). Then, we prove that it supports both completeness and weak accuracy with $t \geq f + 1$ and $\tau \geq 3f + 1$ (Theorem 3). Finally, we prove it supports the strong accuracy property alone with $t \geq \tau + f$ (Lemma 7).

Algorithm 1 The *a-audit()* algorithm.

```
function a-audit( )
    $E_A \leftarrow \varnothing, L[1..n] \leftarrow \varnothing$
    parallel for $1 \le k \le n$ do $L[k] \leftarrow o_k.getLog()$
    wait $|\{k : L[k] \neq \varnothing\}| \geq n - f$
    for all $\langle p_r, label(b_v)\rangle \in \bigcup_{k\in\{1..n\}} L[k]$ do
        for $1 \le k \le n$ do if $\langle p_r, label(b_v)\rangle \in L[k]$ then $\mathscr{E}_{p_r,v} \leftarrow \mathscr{E}_{p_r,v} \cup \{k\}$
        if $|\mathscr{E}_{p_r,v}| \geq t$ then $E_A \leftarrow E_A \cup \{\mathscr{E}_{p_r,v}\}$
    return $E_A$
```

The implementation for *a-audit()* is presented in Algorithm 1. It starts with an empty set E_A that will be used to store the evidences attesting the effective reads and an array of empty logs to store the logs it will receive from objects (Line 2). It then queries n storage objects to obtain the list of records on objects' logs (i.e., o_k*.getLog()*), waits the response from at least $n-f$ of them, and stores these responses in the array of logs L (Lines 3–4).

For each previously seen record (Line 5), it adds the identifier of every object containing this record to an evidence $\mathscr{E}_{p_r,v}$ (Line 6). If this evidence contains more than t identifiers, then it refers to an effective read and is added to the reporting set of evidences E_A (Line 7). After verifying all records, the audit operation returns the set E_A (Line 8), which is used by auditors to report that the detected readers have effectively read the mentioned data values.

Lemma 5. *Algorithm 1 satisfies the* ***completeness*** *property of auditable registers with* $t \geq 1$ *and* $\tau \geq 2f+1$.

Proof. Based on Lemma 1 if $\tau \geq 2f+1$, any auditing quorum A receives, for every effective read, at least one record from a correct storage object that has also participated on the providing set of this read (i.e., $\tau - 2f \geq 1$). Consequently, Algorithm 1 satisfies completeness by obtaining some record from any effective read. □

Lemma 6. *Algorithm 1 satisfies the* ***weak accuracy*** *of auditable registers with* $t \geq f+1$.

Proof. To support the weak accuracy of auditable registers, Algorithm 1 simply needs to make the f records from faulty objects insufficient to create an evidence reporting an effective read from a reader p_r. Therefore, $t \geq f+1$ ensures that any auditing quorum A includes at least one correct object that received a read request from this reader and participates in the providing set P_{p_r}, v. □

Theorem 3. *Algorithm 1 satisfies both* ***completeness*** *and* ***weak accuracy*** *of auditable registers with* $\tau \geq 3f+1$.

Proof. Assuming $\tau \geq 3f+1$ directly satisfies completeness (Lemma 5). Based on Lemma 1, any auditing quorum A receives at least $\tau - 2f \geq f+1$ records from correct storage objects that have participated on the providing sets of each effective read. As proved in Lemma 6, $t \geq f+1$ is enough for Algorithm 1 to satisfy also weak accuracy. □

A practical consequence of Theorem 3 is that existing information dispersal schemes that support fast reads using $n \geq \tau + 2f$ objects, such as Hendricks *et. al* [128], would require at least $n \geq 5f+1$ to support this weak auditability.

Lemma 7. *Algorithm 1 satisfies the* ***strong accuracy*** *of auditable registers with* $t \geq \tau + f$.

Proof. To support the strong accuracy property of auditable registers, evidences must be created using at least τ records attesting the read of the same value from correct storage objects that have participated in the providing sets of each effective read. Since f faulty objects can mimic these objects and also participate in an auditing quorum A, this number f must be added to the threshold number t of records required to create an evidence. These two requirements make strong accuracy satisfiable with $t \geq \tau + f$ by accessing any auditing quorum A because, in this case, the records from the f faulty objects make no difference when reporting a value effectively read by a reader. □

5.6 Alternative Models for the Algorithm

As proved in Theorems 1 and 2, respectively, weak auditability is impossible with $\tau \leq 3f$ and strong auditability is impossible in our system model. In the following, we

Model	**Completeness**	**Weak Accuracy**	**Completeness + Weak Accuracy**	**Strong Accuracy**	**Completeness + Strong Accuracy**
Our Model	$\tau \geq 2f+1$	$t \geq f+1$	$\tau \geq 3f+1$	$t \geq \tau + f$	Impossible
Signed Reads/SR		$t \geq 1$	$\tau \geq 2f+1$		
Total Order/TO		$t \geq f+1$	$\tau \geq 3f+1$	$t \geq f+1$	$\tau \geq 3f+1$
Non-Fast Reads/NF					
TO+SR		$t \geq 1$	$\tau \geq 2f+1$		
NF+SR				$t \geq 1$	$\tau \geq 2f+1$

Table 5.1: Threshold number t and number of blocks τ in each model for each audit property.

present three modifications to our system model that allow Algorithm 1 to overcome these negative results. More specifically, signing read requests makes weak auditability easier (Section 5.6.1), while totally ordering operations (Section 5.6.2) or using non-fast reads (Section 5.6.3) enables strong auditability. Table 5.1 presents an overview of the results in our system model and models with these three modifications.

5.6.1 Signed Read Requests

Digital signatures are tamper-proof mechanisms that enable attesting the authenticity and integrity of received messages. Correct readers may sign their read requests to mitigate being incriminated by faulty objects.

In general, readers signing requests prevents a faulty object from creating correct records about them if this object has never received a signed request from them before. It does not modify the number of records from an effective read available to auditors (Lemma 1) nor does it modify the lower bound on the completeness (Lemma 5). However, it directly relates to the weak accuracy property since auditors receiving any correct record with a signed read request is enough for them to attest that a reader has definitely tried to read data in the storage system.

Lemma 8. *Correct readers signing read requests in our model allows Algorithm 1 to satisfy* ***weak accuracy*** *with* $t \geq 1$.

Proof. If read requests from correct readers are signed and these readers have never sent signed requests to the storage system, faulty objects will never create correct records about these readers. As a result, $t \geq 1$ ensures that auditors will only create evidences for the readers that have tried to read data by sending signed requests to the storage objects. □

Theorem 4. *Correct readers signing read requests in our model allows Algorithm 1 to satisfy both* ***completeness*** *and* ***weak accuracy*** *with* $\tau \geq 2f+1$.

Proof. With $\tau \geq 2f+1$ and based on Lemma 1, any auditing quorum A receives at least $\tau - 2f = 1$ record from a correct storage object that has participated on each effective read. As proved in Lemma 8, $t \geq 1$ is enough to satisfy weak accuracy in our model when read requests are signed. Moreover, this limitation does not change the lower bound of Lemma 5, which means that $\tau \geq 2f+1$ is enough for Algorithm 1 to also satisfy completeness in our model. □

Faulty objects can still repurpose generic signed read requests to create correct records of an arbitrary value. For that reason, signing generic read requests does not modify the lower bounds of strong accuracy (Lemma 7) and supporting it in conjunction with completeness remains impossible (Theorem 2).

Alternatively, correct readers can sign the read request with the label of the exact value they intend to read (e.g., a timestamp). However, additional modifications are required to enable readers learning this label before sending the read request, as will be explained in Section 5.6.3.

5.6.2 Total Order

Serialising operations using total order broadcast [121] allows our system to execute operations sequentially. By doing so, we limit the number of different values in storage objects to only one value since high-level *a-read* and *a-write* operations are executed in total order. In the worst case, the system will have f faulty objects with incorrect read records. With this limitation, strong accuracy becomes easier and can be satisfied together with completeness.

Lemma 9. *Totally ordering operations in our model allows Algorithm 1 to satisfy* ***strong accuracy*** *with* $t \geq f+1$.

Proof. If operations are totally ordered in the system, all objects will store blocks of the same value v, and no object returns blocks of other values. No read operation will result in correct records for more than one data value, and every read operation results in the creation of records in at least $n-f$ objects. The worst case is when the f faulty objects log incorrect read records for a value other than v. As a result, any auditing quorum A that includes the faulty objects will receive f records attesting the read of a value other than v. Requiring $t \geq f+1$ to create an evidence guarantees that at least one correct object has also participated in the providing set $P_{p_r,v}$, which guarantees that $n-f$ objects will also return their blocks, allowing p_r to effectively read the value v. □

Theorem 5. *Totally ordering operations in our model allows Algorithm 1 to satisfy both* ***completeness*** *and* ***strong accuracy*** *with* $\tau \geq 3f+1$.

Proof. With $\tau \geq 3f+1$ and based on Lemma 1, any auditing quorum A receives at least $\tau - 2f = f+1$ records from correct storage objects that have participated on each effective read of value v. As proved in Lemma 9, $t \geq f+1$ is enough to satisfy strong accuracy in our model when operations are totally ordered. Moreover, this limitation does not change the lower bound of Lemma 5, which means that $\tau \geq 3f+1$ is enough for Algorithm 1 to also satisfy completeness in our model. □

As proved in Lemma 8, signing generic read requests reduces the lower bound of weak accuracy, which also benefits the model with total ordering (see Table 5.1). However, as previously mentioned, these generic signatures do not modify the lower bounds of strong accuracy.

5.6.3 Non-fast Reads

There are read algorithms that use more than one communication round (i.e., a *non-fast read* [117]) to ensure correct readers will only fetch the blocks of the most up-to-date value stored in the register. For instance, DepSky-CA [28] is a register emulation in which the first round of a read obtains the label of the most up-to-date value available in $n-f$ objects, while the second round actually reads only the coded blocks for that specific value. As a consequence, correct objects log reads from readers only if they hold the most up-to-date value available in at least $n-f$ objects.

Lemma 10. *Applying Algorithm 1 to DepSky-CA protocol satisfies* ***strong accuracy*** *with* $t \geq f+1$.

Proof. Let us assume registers follow DepSky-CA protocol [28] with two communication rounds in read operations. No read operation will result in correct records for more than one data value, and every read operation results in the creation of records in at least $n-f$ objects. In the worst-case scenario, the f faulty storage objects can only create at most f incorrect records for an arbitrary value. Requiring $t \geq f+1$ to create an evidence and report an effective read guarantees that it is achieved only when the reported value was effectively read from $n-f$ objects. □

Theorem 6. *Applying Algorithm 1 to DepSky-CA protocol satisfies both* ***completeness*** *and* ***strong accuracy*** *with* $\tau \geq 3f+1$.

Proof. Let us assume registers follow DepSky-CA protocol [28] with two communication rounds in read operations. Assuming $\tau \geq 3f+1$ and based on Lemma 1, any auditing quorum A receives at least $\tau - 2f = f + 1$ records from correct storage objects that have participated on the providing set of each effective read. As proved in Lemma 10, $t \geq f + 1$ is enough to satisfy strong accuracy when using DepSky-CA algorithm. Moreover, this limitation does not change the lower bound of Lemma 5, which means that $\tau \geq 3f + 1$ is also enough to satisfy completeness in our model. □

As mentioned in Section 5.6.1, readers can sign their read requests for any value (i.e., generic signatures) or for specific values. Non-fast reads can leverage the latter to allow the system to protect against faulty objects using signed read requests to record reads from other values than the one intended by the reader. It contributes to reduce the lower bound of strong accuracy and makes easier to support it with completeness.

Lemma 11. *Applying Algorithm 1 to DepSky-CA protocol with specific signed read requests satisfies* ***strong accuracy*** *with $t \geq 1$.*

Proof. Let us assume registers follow DepSky-CA protocol [28] with two communication rounds in read operations. If readers learn, in the first communication round, the label of the value they can effectively read and use it to sign the read request specifically for that value, faulty objects will never create correct records of these readers and that value. Additionally, no read operation will result in correct records for more than one data value, and every read operation results in the creation of correct records in at least $n - f$ objects. As a result, $t \geq 1$ ensures that auditors will only create evidences for the readers that have effectively read that value by sending specific signed requests to the storage objects. □

Theorem 7. *Applying Algorithm 1 to DepSky-CA protocol with specific signed read requests satisfies both* ***completeness*** *and* ***strong accuracy*** *with $\tau \geq 2f + 1$.*

Proof. Let us assume registers follow DepSky-CA protocol [28] with two communication rounds in read operations and signed read requests for specific values. Assuming $\tau \geq 2f + 1$ and based on Lemma 1, any auditing quorum A receives at least $\tau - 2f = 1$ record from correct storage objects that have participated on the providing set of each effective read. As proved in Lemma 11, $t \geq 1$ is enough to satisfy strong accuracy when using DepSky-CA algorithm with specific signatures. Moreover, this limitation does not change the lower bound of Lemma 5, which means that $\tau \geq 2f+1$ is also enough to satisfy completeness in our model. □

A practical consequence of Theorems 6 and 7 is that DepSky-CA protocol [28] would require at least $n \geq 5f+1$ (without signed reads) and $n \geq 4f+1$ (with signed reads for specific values) to support strong auditability.

5.7 Final Remarks

In this chapter (i.e., the auditability component of the MANAGEMENT phase of our storage pipeline), we have identified the formal requirements of auditing who has effectively read critical data (e.g., the privacy-sensitive portion of human genomes) from secure storage systems (e.g., CHARON [183]). This auditability requires $n \geq 5f+1$ (i.e., $\tau \geq 3f+1$) to provide weak auditability in systems supporting fast reads or strong auditability in systems with slow reads (i.e., reads with more than one communication round) or with total order. Signing read requests enables both forms of auditability to be supported with $\tau \geq 2f+1$.

6 An End-to-End Storage Pipeline for Human Genomes

In this chapter, we present and evaluate an end-to-end composite pipeline intended to enable the secure, dependable cloud-based storage of human genomes by integrating the three mechanisms we proposed in the previous chapters. These mechanisms encompass (1) a privacy-sensitivity detector for human genomes (Chapter 3) [70], (2) a similarity-based deduplication and delta-encoding algorithm for sequencing data (Chapter 4) [72], and (3) an auditability scheme to verify who has effectively read data in storage systems that use secure information dispersal (Chapter 5) [69]. The first mechanism identifies the privacy-sensitive portions of human genomes and allows the portions associated with different privacy risk levels to follow different privacy-related paths in the pipeline. The second mechanism focuses on balancing reduction ratio and read performance better than existent genome compression algorithms. Finally, the third mechanism identifies and enforces the additional requirements for auditing who has effectively read data from secure dispersed storage systems.

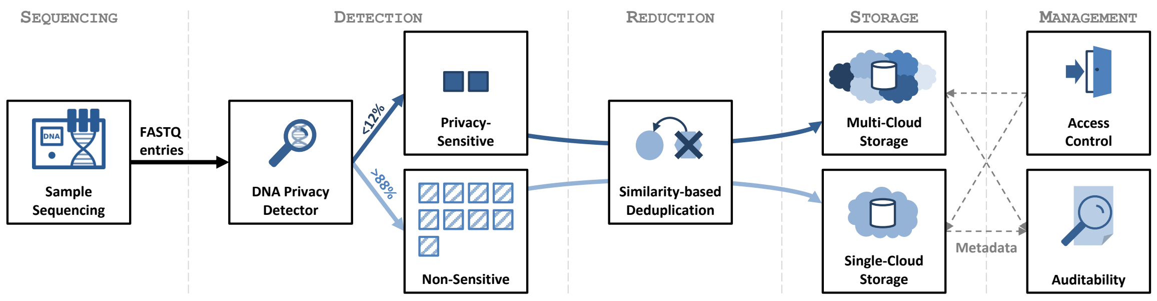


Figure 6.1: Overview of our pipeline intended to enable the efficient, dependable storage human genomes in public clouds.

We advocate that one can obtain reasonable privacy protection, security, and dependability guarantees at modest costs (e.g., less than \$1/Genome/Year) by integrating the mentioned mechanisms with appropriate storage configurations. Our pipeline presents a small storage overhead of 3% compared to non-replicated systems, but it costs 48% less than fully-replicating all data and 31% less than secure information dispersal schemes.

The remainder of this chapter is organised as follows. Section 6.1 contains an integrated description of the pipeline, Section 6.2 discusses its feasibility, and Section 6.3 presents the final remarks.

6.1 The Pipeline

In this section, we present an end-to-end composite pipeline intended to enable the efficient, dependable cloud-based storage of human genomes. This pipeline inserts privacy-awareness, cost-efficiency, and auditability into the storage ecosystem focused on human genomes. It is composed of five phases, as presented in Figure 6.1: SEQUENCING, DETECTION, REDUCTION, STORAGE, and MANAGEMENT.

The first phase (SEQUENCING) obtains the digitised genome from biological samples. The second (DETECTION) separates genomes' portions according to their privacy sensitivity. The third (REDUCTION) applies data reduction techniques to improve data density and reduce storage costs. The fourth (STORAGE) retains the genome's portions in appropriate repositories and provides data to clients. Finally, the fifth (MANAGEMENT) provides tools for controlling and monitoring the system. These phases are described in the remaining of this section.

6.1.1 Sequencing

Sequencing machines digitise genomes by translating the chemical compounds from biological samples to digital information. Next-Generation Sequencing (NGS) [223] is the name given to the machines that sequence genomes at a high-throughput [172]. However, NGS machines do not provide the whole human genome in a single contiguous DNA sequence. They generate millions of small *DNA reads*, which are small pieces of DNA containing sequences with hundreds to thousands of nucleotides each. Additionally, every nucleobase from a human genome is sequenced many times and appear in several complementary reads (e.g., 30–45$\times$) to improve the sequencing accuracy.

Data obtained from this process is usually stored in the FASTQ text format [61], in which every *entry* contains four lines. The first line of every FASTQ entry is a comment about the read sequence and starts with a "`@`" character. The second line contains the DNA sequence read by the machine. The third line starts with a "+" character to determine the end of the nucleotide sequence and can optionally be followed by the same content of the first line. The fourth line contains quality scores, which measure the confidence of the machine on each read nucleotide. Each sequenced FASTQ entry (i.e., four lines) is sent separately to the next step in our storage pipeline.

6.1.2 Detection

Previous works on privacy-preserving genome processing advocated the partitioning of genomic data [97, 114], but assumed it would be done manually [19] or by a tool out of their scope [153]. We closed this gap by proposing a DNA Privacy Detector [70], which was the first comprehensive privacy-aware detection method that enabled users to implement such partitioning automatically.

Given a DNA segment of a predefined size, our method detects whether this segment may contain known privacy-sensitive information or not. It does so based on a knowledge database of published signatures or patterns of privacy-sensitive nucleic and amino acid sequences. The detector decides based on the information present in the knowledge database, and forwards each received FASTQ entry alternatively to a privacy-sensitive output or a non-sensitive one. Recent works have upgraded this detection method to: evolve the knowledge database to detect previously unknown privacy-sensitive sequences [71]; support FASTQ entries with larger DNA sequences [83]; and support additional privacy-sensitivity levels according to different risk classifications [102].

In this work, data from the DETECTION phase results in two subsets: a small privacy-sensitive portion (i.e., 12% of the FASTQ entries from each human genome) and a large non-sensitive one (i.e., 88%). This 12/88 ratio between these two portions comes from the employed knowledge database, which contains the currently known privacy-sensitive sequences [70]. Reducing the data that requires stronger security and dependability premises (to less than 12%) naturally contributes to the cost-efficiency of any storage solution.

By identifying the privacy-sensitive sequences using our solution and protecting them, one neutralises the existent threats of re-identifying individuals [118] and of inferring private information about them [191]. Finally, FASTQ entries from both portions (i.e., the privacy-sensitive and the non-sensitive) are sent to the REDUCTION phase, which deduplicates this data to make it even more cost-efficient.

6.1.3 Reduction

Reducing the size of data from genomes is imperative to enable the efficient storage of large data sets of human genomes. Without a considerable data reduction, most hospitals and biobanks cannot store this data, which may delay advances in medical research and diagnosis [203].

We have selected portions of five representative human genomes (SRR400039, SRR618664, SRR618666, SRR618669, SRR622458) from the 1000 Genomes project [57]. They sum up approximately 265GB of data in FASTQ files. We evaluated and compared several generic and specialised compression tools in Section 4.2 to better depict the state-of-the-art in the reduction of human genomes. However, for this chapter, we have selected only two of them: GZIP [88] and SPRING [49] for the reasons that follow. GZIP was chosen because it is the fastest generic compressor used in practice and compresses the selected genomes, on average, 3.225×, which results in a reduction ratio $r = 0.31$. SPRING was selected because it is the specialised tool with the best reduction ratio and compresses the mentioned genomes, on average, 6.02×, which results in a reduction ratio $r = 0.166$.

Storage of sequencing data is an important, challenging domain mostly unexplored in the systems community [203]. Deduplication is a technique that reduces the storage requirements by eliminating unrelated redundant data [103]. Additionally, deduplication has two advantages when compared to compression algorithms: it may leverage the inter-file similarities, while most compression algorithms consider only intra-file data or use a single generic contiguous reference; and it usually achieves a better read performance than compression. However, traditional identity-based deduplication (e.g., chunk-based [202]) fails to provide a satisfactory reduction in the storage of genomes because FASTQ entries contain unique identifiers.

Similarity-based deduplication is an interesting alternative since there are several entries with very similar structure or content. Solutions for similarity-based deduplication commonly cluster similar entries into buckets and use identity-based deduplication within them [202], or they focus mostly on the delta-encoding problem [91] and employ inefficient global indexes [256]. We have proposed a solution that balances space savings and read performance by integrating efficient similarity-based deduplication based on Locality-Sensitive Hashing (LSH) [143] and specialised delta-encoding based on the Hamming distance for genome sequencing data [72]. This solution finds, separately for the DNA and QS lines of each FASTQ entry, the most similar base chunk in a deduplication index and replaces the original lines by a pointer to the best candidate and the transformations to recover the original sequence from it.

Preliminary analysis of our work indicates it achieves 81.7% of the reduction ratio of the best specialised tool (i.e., SPRING) and compresses 50% more than the fastest generic competitor used in practice (i.e., GZIP) using a human reference genome as the deduplication index for the DNA lines and 2^{20} synthetic candidates for the QS lines. Additionally, it restores data 9.96× faster than SPRING and 4.4× faster than GZIP. In summary, our solution compresses the selected genomes, on average, 4.92× (i.e., $r = 0.2032$).

6.1.4 Storage

Data from the DETECTION phase is divided into two subsets: a privacy-sensitive portion of human genomes and a non-sensitive one. These two portions are deduplicated and delta-encoded in the REDUCTION phase and are handled differently in the present phase. The privacy-sensitive portion requires stronger security premises, while the other portion can use affordable security techniques. From the moment they are delivered by the previous phases, the STORAGE phase applies commonly used dependability and security techniques to store data properly in public clouds.

Cloud computing is an economical alternative to expensive private infrastructures. We consider a system architecture composed of a single public cloud to store the non-sensitive portion of human genomes and a cloud-of-clouds to store the privacy-sensitive portion.

6.1.4.1 Single-Cloud Storage

This scenario is the simplest one, where we apply *standard encryption* on data from the REDUCTION phase and store it in a *single public cloud*. This encryption guarantees that only authorised users have access to data, and these users need to know the decryption key. The rationale for this decision is the fact that this data is the less (or non-) sensitive portion of human genomes, and thus the security and dependability provided by a single public cloud is acceptable.

There is an inherent execution cost in this scenario. The cost for reading data will be equals to the cost of transferring the encrypted, compressed file from the cloud, decrypting it, decompressing it, requesting the original sequences and quality scores from the deduplication index, and applying the delta-encoded transformations to recover the original data.

6.1.4.2 Multi-Cloud Storage

In this storage configuration, we also initiate by applying *standard encryption* on data. Then, data is *split in blocks* that will be sent to different clouds later [28]. It guarantees that no cloud has the whole genome in its infrastructure, which increases the privacy-protection if data stored in a subset of clouds is compromised. We opt to apply *secret sharing* [161] on the encryption key to distribute it together with the data blocks, which makes the system independent of key managers. Storage optimal *erasure codes* [207] are also employed to allow recovering the data in case of failures without the need for replicating all data blocks, which reduces the storage cost compared to full replication. Finally, data is sent to a quorum of clouds from the *cloud-of-clouds*, where each cloud stores different data blocks in a secure setting and provides increased availability.

There is an inherent execution cost in applying all these techniques over data. The cost for reading data will be equals to the cost of transferring the data from a subset of clouds plus: recovering the original blocks from the erasure codes and secret sharing methods, decrypting the data, and decoding the original data based on the entries used in the deduplication system and the delta-encoded transformations. The needed subset of clouds must result in clients receiving at least the minimum number τ of correct blocks to decode a data item.

We employ CHARON [183] as our backend since it is a complete storage solution that provides the two mentioned configurations (single public clouds and a cloud-of-clouds) and also allows the storage of data in private repositories. The original cloud-of-clouds configuration of CHARON assumes $n \geq 3f+1$, with $\tau \geq f+1$, which means that this storage configuration incurs in an optimal storage overhead of 50% ($f = 1$). As it will be explained in the next section, to support auditability, we consider a cloud-of-clouds configuration in our pipeline where $n \geq 5f+1$ and $\tau \geq 3f+1$ (i.e., resulting in a storage overhead of only 25% with $f = 1$).

6.1.5 Management

Several components (e.g., key distribution, performance monitoring, and billing) may fit in this generic phase. However, we are interested only in the access control and auditability ones because these are the main blocks responsible for guaranteeing that only authorised users can and have effectively accessed the data.

6.1.5.1 Access Control

Access control permits certain users to obtain and modify specific data items according to their roles. This mechanism may also have distinct access rules for the different portions of human genomes (i.e., the privacy-sensitive and the non-sensitive portions). Additionally, cryptographic solutions from the STORAGE phase complement access control mechanisms since an attacker that circumvents the access control does not obtain the data in clear. Finally, the cloud-of-clouds in CHARON provides a joint access control combining cloud providers, where its access control is satisfied even if up to f providers have been compromised [183].

6.1.5.2 Auditability

Auditability is the systematic ability to verify some property in an environment and is a deterrent measure that complements preventive ones, e.g., security, dependability, and privacy protection. In this work, we are interested in auditing exactly which users have effectively read each human genome stored in the system, which already separates them from the whole group of users that are authorised to read it but have never done so. Auditors need to access only metadata, such as filenames, access logs, and access control rules (i.e., they do not need to access the whole data sets).

Usually, auditability systems must keep an indelible tamper-proof track of data accesses to detect, analyse, and sanction misuses. However, the guarantees from this registry directly depend on the configuration of the system. For instance, non-replicated storage systems must trust in the single cloud provider they use to register and provide evidences for every action users perform in the system.

However, storage systems that employ multiple cloud providers have the opportunity to avoid this trust requirement by using the logs from a subset of providers to create a fault-tolerant track of records. In Chapter 5, we have identified the formal requirements of such auditability for systems based on secure information dispersal schemes [69]. Basically, auditability requires $n \geq 5f+1$ (i.e., $\tau \geq 3f+1$) to provide a weak form of auditability in systems supporting fast reads [117] or a strong form of auditability in systems with slow reads (i.e., reads with more than one communication round). Signing read requests enables both forms of auditability to be supported with $\tau \geq 2f+1$, but it would increase the storage overhead and was not considered in the evaluation in this chapter.

Privacy-awareness (from Section 6.1.2) allows us to provide adequate auditability guarantees for the different portions of human genomes. The fact that only the non-sensitive portions of human genomes are stored on single-cloud storage reduces the impact of losing auditability information on these configurations. The storage of the privacy-sensitive portion of human genomes guarantees that every effective read is reported by the audit process [69].

6.2 Feasibility Discussion

Storage costs directly impact the feasibility of collecting large sets of whole human genomes. Furthermore, storage solutions must benchmark their cost-efficiency to not burden institutions and to make dependability affordable [50].

Haussler *et. al* [127] estimated the costs of creating a data warehouse to store (and process) one million human genomes (compressed to 180GB each). Their calculated capital expenditure (CAPEX) was $65M for the first year and $35M per subsequent year to maintain and update the infrastructure.

One million human genomes is an interesting example of the scale biobanks will face since they already manage similar numbers of physical samples [251]. Assuming that each human genome sizes $s = 300$GB (i.e., 30–45$\times$ of coverage), one million genomes result in 300PB of data. Storing all this data is expensive, where even the cost of using only cold storage from a single cheap cloud provider (e.g., Microsoft Azure—see Table 6.1) is $3.6M per year. Investing in more dependable solutions (e.g., secure information dispersal using multiple clouds) increases this annual cost to approximately $13M.

In this section, we evaluate the feasibility of the presented composite pipeline. We use the estimated annual cost (in $) to store a single human genome as the metric of interest since it can easily be adapted to deployments of any size. We start by delineating three basic configurations typically used in cloud-based storage and present their pros and cons.

The first configuration (`NON-REP`) stores all data only in the cheapest single cloud provider from Table 6.1 (i.e., Microsoft Azure). It is the baseline of this evaluation and represents a non-replicated scenario, where the cloud provider has to be trusted and is a single point of failure at the administrative domain level. The second configuration

Cloud Storage	**Standard**	**Infrequent**	**Cold**
Microsoft Azure [184]	0.0184	0.01	0.001
Alibaba Cloud [7]	0.0185	0.01	0.0036
Google Cloud [112]	0.02	0.01	0.004
Amazon AWS [13]	0.023	0.01	0.004
IBM Cloud [135]	0.022	0.012	0.006
Oracle Cloud [199]	0.0425	0.0255	0.0026

Table 6.1: Cloud storage pricing (in \$/GB/Month) as in June, 2019.

(FULL-REP) replicates all data into trusted cloud providers. It tolerates cloud outages (i.e., crash faults only) of a subset of providers (i.e., $n \geq f+1$). We consider tolerating one fault (i.e., $f = 1$) in this evaluation, which means that this configuration results in a storage overhead of 100% since $n = 2$. The third configuration (INFO-DISP) distributes data into multiple untrusted providers using secure information dispersal schemes [3, 28], which guarantee that no single cloud stores or has access to the entire data set. It tolerates a subset of malicious clouds (i.e., $n \geq 3f+1$) and results in a storage overhead of 50% (for $f = 1$) [28] compared to our baseline (i.e., NON-REP).

Our pipeline employs the steps described in Section 6.1 and stores each genome portion in an appropriate configuration. It results in a fourth configuration (PRI-AWR) that conservatively stores approximately 12% of each human genome (i.e., the privacy-sensitive portion) using a special case of secure information dispersal and the remaining 88% (i.e., the non-sensitive portion) in a non-replicated configuration. Our secure information dispersal uses more clouds than the configuration in INFO-DISP because it supports the auditability of who has effectively read data, as described in Section 6.1.5. This increase in the number of replicas results in a storage overhead of 25% (for $f = 1$) instead of the 50% from INFO-DISP. In the end, our pipeline has a storage overhead of only 3% since $0.12 \times 1.25 \times r \times s + 0.88 \times r \times s = 1.03 \times r \times s$.

Despite the configuration of choice, we assume all solutions encrypt data to protect confidentiality and use the n cheapest clouds on each different configuration. Additionally, this comparison considers that a human genome originally sizes $s = 300$GB (i.e., $r = 1$)

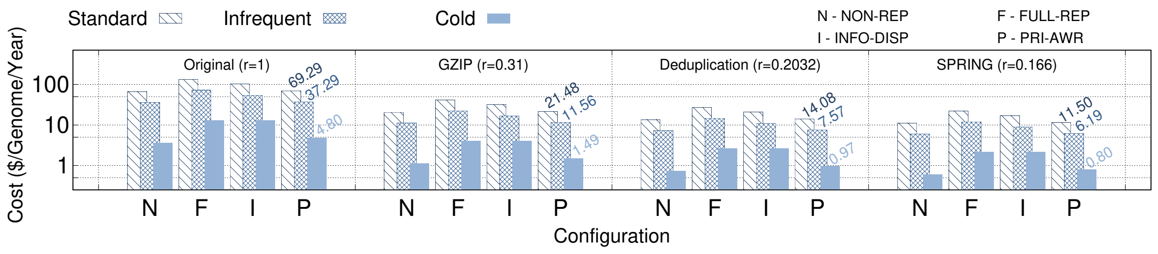


Figure 6.2: Estimated annual cost (in $) to store a human genome ($s = 300$GB) considering different configurations and reduction ratios (r).

and can be reduced by different algorithms with the following compression ratio r: GZIP reduces a genome to 93.45GB ($r = 0.31$), our similarity-based deduplication reduces it to 60.96GB ($r = 0.2032$), and SPRING reduces it to 49.8GB ($r = 0.166$).

Figure 6.2 presents the estimated annual cost (in $) to store a human genome in every configuration described in this section either uncompressed or compressed by one of the three mentioned reduction algorithms. Additionally, this figure considers scenarios using three storage service levels that are available in all evaluated cloud providers and differ in the expected frequency of data accesses: standard, infrequent, and cold storage (the less frequent, the cheaper—see Table 6.1).

While storing an uncompressed human genome can cost $66.24 per year in `NON-REP` using the cheapest standard cloud storage, it can drop to $0.59 storing this genome compressed by the SPRING in the cheapest cold storage provider. Considering dependable alternatives, fully replicating data always costs more than using secure information dispersal, which by its turn always costs more than using our privacy-aware pipeline. Storing an uncompressed genome using only our privacy-awareness (Section 6.1.2) and auditability phases (Section 6.1.5) can cost $69.29 per year in `NON-REP` standard storage, while $0.80 are enough to store it compressed with SPRING using cold storage. However, SPRING has a small restore throughput compared to our deduplication and the other competitors (see Section 6.1.3). Using all phases from our pipeline (i.e., privacy-awareness, auditability, and deduplicating instead of using SPRING) results in a storage cost of $14.08 using standard storage services and $0.97 using cold storage. It means that storing one million human genomes with our pipeline costs less than $1M per year. These results vouch for the utility of the mechanisms integrated into our composite pipeline to enable the efficient, dependable storage of human genomes in public cloud infrastructures.

6.3 Final Remarks

The complete storage pipeline presented in this thesis integrates the specific contributions of Chapters 3–5 and enables the efficient, dependable cloud-based storage of human genomes in public clouds since it provides reasonable privacy protection, security, and dependability guarantees at modest costs (less than $1/Genome/Year) using appropriate storage configurations.

Conclusion 7

7.1 Final Remarks

This thesis encompasses works that have proposed, implemented, and evaluated an end-to-end composite storage pipeline, which introduced privacy-awareness, cost-efficiency, and auditability into the data storage ecosystem for human genomes. By integrating the described mechanisms with appropriate storage configurations, one can obtain reasonable privacy protection, security, and dependability guarantees at modest costs (e.g., less than $1/Genome/Year). This pipeline enables the efficient, dependable cloud-based storage of human genomes since it provides enhanced dependability guarantees with adequate storage overhead (e.g., 3% compared to non-replicated systems). Moreover, the efficiency of the proposed pipeline is also attested by the fact it costs 48% less than fully-replicating data and 31% less than using secure information dispersal schemes exclusively.

More specifically, our first contribution was to propose a novel efficient solution to automatically detect privacy-sensitive DNA sequences from an input stream using, as a reference, a knowledge database of privacy-sensitive sequences. Data from this solution results into two sets: a small privacy-sensitive portion (less than 12%) and a large non-sensitive one (more than 88%). The assessment of the privacy-sensitivity detector demonstrated its feasibility to address the challenges imposed by some published attacks and to establish the basis for future developments in this field. We have shown that the privacy-sensitivity detector can easily be fitted inline with the NGS production cycle and fulfil the systematic detection promise, exhibiting adequate performance and scalability, by using Bloom filters.

However, we note again that besides the effectiveness of our solution, the proposed computational framework and architecture can be reused and evolved with new privacy-sensitive sequences being identified [71].

The second specific contribution from this thesis was the proposal of GenoDedup, the first method that integrates efficient similarity-based deduplication and specialised delta-encoding for genome sequencing data. Experimental results attested that GenoDedup (with 2^8 entries in the QS deduplication index) achieves 67.8% of the reduction gains of the best specialised tool in this metric (i.e., SPRING [49]) and restores data $1.62\times$ faster than SeqDB [133] (i.e., its fastest competitor). Additionally, GenoDedup restores data $9.96\times$ faster than SPRING and compresses files $2.05\times$ more than SeqDB. Finally, GenoDedup compresses 50% more than the fastest generic compressor used in practice (i.e., GZIP [88]) and restores data $4.4\times$ faster than it.

The third specific contribution initiated the study of auditable storage emulations, which provides the capability for an auditor to discover the previously executed reads in a register. This work defined auditable registers, their properties, and established tight bounds and impossibility results for auditable storage emulations in the presence of faulty storage objects. Our system model considered read-write registers that securely store data using information dispersal and support fast reads. In such a scenario, given a minimum number τ of data blocks required to recover a value from information dispersal schemes and a maximum number f of faulty storage objects (1) auditability is impossible with $\tau \leq 2f$; (2) when fast reads are supported, $\tau \geq 3f+1$ is required for implementing weak auditability, while strong auditability is impossible; (3) signing read requests overcomes the lower bound of weak auditability; and (4) totally ordering operations or using non-fast reads can provide such strong auditability.

7.2 Future Work

This thesis has explored several topics related to the storage of critical data, more specifically of human genomes. Opportunities arise from each thesis' contribution and every step in our storage pipeline.

Privacy-awareness. There is an opportunity to advance scientific knowledge and the systems' awareness of the privacy-sensitivity of human genomes. For instance, a natural next step is to delineate a comprehensive map of privacy-sensitivity in human genomes by organising privacy-sensitive information according to their location in the genome and correlating this distribution with biological reasoning. Another open contribution is the creation of an educational service that informs whether and why a small queried DNA sequence contains or not privacy-sensitive information and what is its privacy risk severity. Other colleagues, from which some are also co-authors of the first paper of this thesis [70], have been extending the proposed detection method to support FASTQ entries with longer DNA sequences [83] and to support additional privacy-sensitivity levels according to different risk classifications and allele frequencies [102]. There are opportunities to expand even further these risk levels by classifying information based on the correlation of genes and mutation with diseases by defining that sequences associated with some diseases (e.g., Alzheimer's) are more privacy-sensitive than when linked to others (e.g., allergic Rhinitis).

Reduction of sequencing data. The proposed similarity-based deduplication highly depends on the size and quality of the deduplication index, as presented in Section 4.3.2. There is an opportunity in exploring other feature selection and clustering algorithms to evaluate more genomes to model better deduplication indexes faster. One can create multiple indexes considering the different sequencing profiles—e.g., one index for sequencing resulting mostly in high quality scores and another for lower values. Other engineering optimisations are possible, namely: improving the LSH scalability and distribution into several nodes (however the implemented LSH scales to more entries than one can model in practical time), experimenting different LSH configurations to improve the search for the best candidates, and testing other distance metrics and their encodings. The theoretical aspects of deduplication are also an open problem, where one can explore the required amount of memory in deduplicating registers or even impossibility results depending on the system model and operations available.

Auditability. The system model considered in this thesis is strongly grounded on practical systems deployed in multi-cloud environments (e.g., [3, 28, 183, 212]). More specifically, this thesis used CHARON [183] as the storage back-end because it provided all the configurations our pipeline needed. As future work, it would be interesting to study how our results will change if considering different models, such as server-based (where base objects can run protocol-specific code and communicate with each other) and synchronous systems. Having audit operations that wait for all n base objects can be an interesting alternative to be explored in the future as well. The results can also be made more generic considering any system with three or more available quorum-based operations with distinct intersections. Exploring blockchains to audit these datasets is another alternative, but with the inconvenience of requiring additional system components to support the blockchain.

Blockchain opens another venue for innovation, such as creating a market place for human genomic data where donors keep full control over their data and determine exactly when and who may have access to which portions of their genomes. There are several ongoing discussions on the ELSI (ethical, social, and legal) implications of a marketplace for genomic data [131], but it has not deterred some solutions from being proposed [9, 186].

Bibliography

[1] 1000 Genomes Project. *Current directory tree*. Available at http://ftp.1000genomes.ebi.ac.uk/vol1/ftp/current.tree. Accessed on Apr. 19, 2020. 2020.

[2] I. Abraham, G. V. Chockler, I. Keidar, and D. Malkhi. “Byzantine disk paxos: optimal resilience with Byzantine shared memory”. In: *Proc. of the 23rd annual ACM symposium on Principles of Distributed Computing (PODC)*. 2004, pp. 226–235.

[3] H. Abu-Libdeh, L. Princehouse, and H. Weatherspoon. “RACS: A Case for Cloud Storage Diversity”. In: *Proc. of the 1st ACM Symposium on Cloud Computing (SoCC)*. 2010, pp. 229–240.

[4] M. Adler. *pigz - Parallel gzip*. Available at https://zlib.net/pigz/. Accessed on Apr. 19, 2020. 2020.

[5] M. K. Aguilera, B. Englert, and E. Gafni. “On using network attached disks as shared memory”. In: *Proc. of the 22nd Annual ACM Symposium on Principles of Distributed Computing (PODC)*. 2003, pp. 315–324.

[6] S. Al Yami and C.-H. Huang. “LFastqC: A lossless non-reference-based FASTQ compressor”. In: *PlOS ONE* 14.11 (2019).

[7] Alibaba, Inc. *Alibaba Storage Pricing*. Available at https://www.alibabacloud.com/product/oss/pricing. Accessed on Apr. 19, 2020. 2020.

[8] A. L. Allen. “What Must We Hide: The Ethics of Privacy and the Ethos of Disclosure”. In: *St. Thomas Law Review* 25 (2012), pp. 1–18.

[9] AllSeq, Inc. *AllSeq - The Sequencing Marketplace*. Available at `https://allseq.com/`. Accessed on Apr. 19, 2020. 2020.

[10] F. Alves, V. V. Cogo, and A. Bessani. “Indexação sob Demanda para a Compressão Referencial de Ficheiros de ADN”. In: *Proc. of the 6th Simposio de Informatica (INFORUM)*. 2014.

[11] F. Alves, V. V. Cogo, S. Wandelt, U. Leser, and A. Bessani. “On-Demand Indexing for Referential Compression of DNA Sequences”. In: *PLOS ONE* 10.7 (2015), e0132460.

[12] Amazon Web Services, Inc. *Amazon Glacier Pricing*. Available at `https://aws.amazon.com/glacier/pricing/`. Accessed on Apr. 19, 2020. 2020.

[13] Amazon Web Services, Inc. *Amazon S3 Pricing*. Available at `https://aws.amazon.com/s3/pricing/`. Accessed on Apr. 19, 2020. 2020.

[14] Amazon Web Services, Inc. *AWS CloudTrail*. Available at `https://aws.amazon.com/cloudtrail/`. Accessed on Apr. 19, 2020. 2020.

[15] E. Androulaki, C. Cachin, D. Dobre, and M. Vukolić. “Erasure-coded Byzantine storage with separate metadata”. In: *Proc. of the 18th International Conference on Principles of Distributed Systems (OPODIS)*. 2014, pp. 76–90.

[16] H. Attiya, A. Bar-Noy, and D. Dolev. “Sharing memory robustly in message-passing systems”. In: *Journal of the ACM (JACM)* 42.1 (1995), pp. 124–142.

[17] T. K. Attwood, S. R. Pettifer, and D. Thorne. *Bioinformatics challenges at the interface of biology and computer science: Mind the gap*. John Wiley & Sons, 2016.

[18] S. Axelsson, U. Lindqvist, U. Gustafson, and E. Jonsson. “Approach to UNIX security logging”. In: *Doktorsavhandlingar vid Chalmers Tekniska Hogskola* 1530 (1999), pp. 137–158.

[19] E. Ayday, J. L. Raisaro, U. Hengartner, and et al. “Privacy-preserving processing of raw genomic data”. In: *Data Privacy Management and Autonomous Spontaneous Security*. Springer, 2014, pp. 133–147.

[20] E. Ayday, E. De Cristofaro, J.-P. Hubaux, and G. Tsudik. “Whole genome sequencing: Revolutionary medicine or privacy nightmare?” In: *IEEE Computer* 48.2 (2015), pp. 58–66.

[21] M. M. A. Aziz, M. N. Sadat, D. Alhadidi, et al. “Privacy-preserving techniques of genomic data—a survey”. In: *Briefings in bioinformatics* 20.3 (2019), pp. 887–895.

[22] S. Bakhtiari, R. Safavi-Naini, and J. Pieprzyk. *Cryptographic Hash Functions: A Survey*. Tech. rep. University of Wollongong, 1995.

[23] N. S. Bakr and A. A. Sharawi. “DNA lossless compression algorithms: review”. In: *American Journal of Bioinformatics Research* 3.3 (2013), pp. 72–81.

[24] C. Basescu *et al.* “Robust data sharing with key-value stores”. In: *Proc. of the 42nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. 2012.

[25] BBMRI Consortium. *BBMRI: Biobanking and Biomolecular Resources Research Infrastructure*. Originally available at http://bbmri.eu/. Accessed on Jul. 24, 2014. Permanently available at https://web.archive.org/web/20140724151631/http://bbmri.eu/. 2014.

[26] BBMRI-ERIC Consortium. *BBMRI-ERIC: Biobanking and BioMolecular resources Research Infrastructure (European Research Infrastructure Consortium)*. Available at http://bbmri-eric.eu/. Accessed on Apr. 19, 2020. 2020.

[27] A. Berger, I. Keidar, and A. Spiegelman. “Integrated Bounds for Disintegrated Storage”. In: *Proc. of the 32nd International Symposium on Distributed Computing (DISC)*. 2018, 11:1–11:18.

[28] A. Bessani, M. Correia, B. Quaresma, F. Andre, and P. Sousa. “DepSky: Dependable and Secure Storage in Cloud-of-Clouds”. In: *ACM Transactions on Storage (TOS)* 9.4 (2013).

[29] A. Bessani, V. V. Cogo, P. Verissimo, et al. *State of the Art and Preliminary Architecture*. BiobankCloud Project, Deliverable D4.1, 75 pages. 2013.

[30] A. Bessani, S. Wandelt, M. Bux, et al. *Reference-based Compression Algorithms*. BiobankCloud Project, Deliverable D2.2, 94 pages. 2014.

[31] A. Bessani, R. Mendes, V. V. Cogo, et al. *The Overbank Cloud Architecture, Protocols and Middleware*. BiobankCloud Project, Deliverable D4.2, 85 pages. 2014.

[32] A. Bessani, J. Brandt, M. Bux, et al. "BiobankCloud: a Platform for the Secure Storage, Sharing, and Processing of Large Biomedical Data Sets". In: *Proc. of the 1st International Workshop on Data Management and Analytics for Medicine and Healthcare (DMAH 2015)*. 2015.

[33] A. Bessani, R. Mendes, T. Oliveira, and V. V. Cogo. *Overbank Implementation and Evaluation*. BiobankCloud Project, Deliverable D4.3, 40 pages. 2015.

[34] A. N. Bessani, R. Mendes, T. Oliveira, et al. "SCFS: A Shared Cloud-backed File System." In: *Proc. of the USENIX Annual Technical Conference (ATC)*. 2014, pp. 169–180.

[35] V. Bhola, A. S. Bopardikar, R. Narayanan, K. Lee, and T. Ahn. "No-reference compression of genomic data stored in FASTQ format". In: *Proc. of the IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*. 2011, pp. 147–150.

[36] B. H. Bloom. "Space/time trade-offs in hash coding with allowable errors". In: *Communications of the ACM* 13.7 (1970), pp. 422–426.

[37] W. J. Bolosky, S. Corbin, D. Goebel, and J. R. Douceur. "Single instance storage in Windows 2000". In: *Proc. of the USENIX Windows Systems Symposium*. 2000, pp. 13–24.

[38] J. K. Bonfield and M. V. Mahoney. "Compression of FASTQ and SAM format sequencing data". In: *PlOS ONE* 8.3 (2013), e59190.

[39] E. T. Borer, E. W. Seabloom, M. B. Jones, and M. Schildhauer. “Some simple guidelines for effective data management”. In: *Bulletin of the Ecological Society of America* 90.2 (2009), pp. 205–214.

[40] S. E. Brenner. “Be prepared for the big genome leak”. In: *Nature* 498.7453 (2013), pp. 139–139.

[41] A. Z. Broder. “On the resemblance and containment of documents”. In: *Proc. of the IEEE Compression and Complexity of Sequences*. 1997, pp. 21–29.

[42] J.-M. C. Brook et al. *Top Threats to Cloud Computing: Egregious Eleven*. Tech. rep. Cloud Security Alliance (CSA), 2019.

[43] J. Brown, M. Ahamad, M. Ahmed, et al. “Redactable and auditable data access for bioinformatics research”. In: *Proc. of AMIA Joint Summits on Translational Science* (2013), pp. 18–22.

[44] K. V. Brown. *A $100 Genome Within Reach, Illumina CEO Asks If World Is Ready*. Available at `https://www.bloomberg.com/news/articles/2019-02-27/a-100-genome-within-reach-illumina-ceo-asks-if-world-is-ready`. Accessed on Apr. 19, 2020. 2019.

[45] J. M. Butler. “Genetics and genomics of core short tandem repeat loci used in human identity testing”. In: *Journal of Forensic Sciences* 51.2 (2006), pp. 253–265.

[46] S. Byma et al. “Persona: a high-performance bioinformatics framework”. In: *Proc. of the USENIX Annual Technical Conference (ATC)*. 2017, pp. 153–165.

[47] V. R. Cadambe, Z. Wang, and N. Lynch. “Information-theoretic lower bounds on the storage cost of shared memory emulation”. In: *Proc. of the 35th ACM Symposium on Principles of Distributed Computing (PODC)*. 2016, pp. 305–313.

[48] R. Cattell. “Scalable SQL and NoSQL Data Stores”. In: *ACM SIGMOD Record* 39.4 (2011), pp. 12–27.

[49] S. Chandak, K. Tatwawadi, I. Ochoa, M. Hernaez, and T. Weissman. “SPRING: a next-generation compressor for FASTQ data”. In: *Bioinformatics* 35.15 (2018), pp. 2674–2676.

[50] R. Charette. “Why software fails”. In: *IEEE Spectrum* 42.9 (2005), pp. 42–49.

[51] G. Chockler and A. Spiegelman. “Space complexity of fault-tolerant register emulations”. In: *Proc. of the 36th Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC)*. 2017, pp. 83–92.

[52] G. Chockler, R. Guerraoui, and I. Keidar. “Amnesic distributed storage”. In: *Proc. of the 21st International Symposium on Distributed Computing (DISC)*. 2007, pp. 139–151.

[53] Chronicle Software. *Chronicle-Map*. Available at `https://github.com/OpenHFT/Chronicle-Map`. Accessed on Apr. 19, 2020. 2020.

[54] J. Y. Chung, C. Joe-Wong, S. Ha, J. W.-K. Hong, and M. Chiang. “CYRUS: Towards Client-Defined Cloud Storage”. In: *Proc. of the 10th ACM European Systems Conference (EuroSys)*. 2015.

[55] D. Cirillo and A. Valencia. “Big data analytics for personalized medicine”. In: *Current opinion in biotechnology* 58 (2019), pp. 161–167.

[56] M. e. Clamp. “Distinguishing protein-coding and noncoding genes in the human genome”. In: *Proceedings of the National Academy of Sciences (PNAS) of the United States of America* 104.49 (2007), pp. 19428–19433.

[57] L. Clarke et al. “The 1000 Genomes Project: data management and community access”. In: *Nature methods* 9.5 (2012), pp. 459–462.

[58] Cloud Harmony. *Service Status*. Available at `https://cloudharmony.com/status-1year-of-storage-group-by-regions-and-provider`. Accessed on Apr. 19, 2020. 2020.

[59] Cloud Security Alliance. *CloudAudit: Automated Audit, Assertion, Assessment, and Assurance*. Originally available at `http://cloudaudit.org`. Accessed on Mar. 22, 2015. Permanently available at `http://web.archive.org/web/20150322234732/http://cloudaudit.org/CloudAudit/Home.html`. 2020.

[60] G. Cochrane, I. Karsch-Mizrachi, and Y. Nakamura. “The international nucleotide sequence database collaboration”. In: *Nucleic Acids Research* 39.suppl 1 (2011), pp. D15–D18.

[61] P. Cock et al. “The Sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants”. In: *Nucleic Acids Research* 38.6 (2010), pp. 1767–1771.

[62] V. V. Cogo. *DNA Privacy Detector*. Available at: `https://github.com/vvcogo/dna-privacy-detector`. Accessed on Apr. 19, 2020. 2020.

[63] V. V. Cogo and A. Bessani. “A Privacy-Assuring Disclosure Filter for Genetic Information”. In: *the Ph.D. Students’ Session of the 32nd IEEE Symposium on Reliable Distributed Systems (SRDS 2013)*. 2013.

[64] V. V. Cogo and A. Bessani. “BiobankCloud Platform as a Service for Biobanking”. In: *the 2nd Annual Next Generation Sequencing Data Congress (Poster)*. 2014.

[65] V. V. Cogo and A. Bessani. *Efficient Storage of Whole Human Genomes*. Poster in the 11th ACM European Systems Conference (EuroSys). 2016.

[66] V. V. Cogo and A. Bessani. “From Data Islands to Sharing Data in the Cloud: the Evolution of Data Integration in Biological Data Repositories”. In: *Communications and Innovations Gazette (ComInG)* 1.1 (2016), pp. 1–11.

[67] V. V. Cogo and A. Bessani. *FS-BioBench: A File System Benchmark from Bioinformatics Workflows*. Technical report integrated into [183]. 2016.

[68] V. V. Cogo and A. Bessani. “Enabling the Efficient, Dependable Cloud-based Storage of Human Genomes”. In: *Proc. of the 1st Workshop on Distributed and Reliable Storage Systems (DRSS’19)*. 2019, pp. 1–6.

[69] V. V. Cogo and A. Bessani. “Auditable Register Emulations”. In: *arXiv:1905.08637* (2020), pp. 1–16.

[70] V. V. Cogo, A. Bessani, F. M. Couto, and P. Verissimo. “A high-throughput method to detect privacy-sensitive human genomic data”. In: *Proc. of the 14th ACM Workshop on Privacy in the Electronic Society (WPES)*. 2015, pp. 101–110.

[71] V. V. Cogo, A. Bessani, F. M. Couto, et al. "How can photo sharing inspire sharing genomes?" In: *Proc. of the 11th International Conference on Practical Applications of Computational Biology & Bioinformatics (PACBB)*. Springer. 2017, pp. 74–82.

[72] V. V. Cogo, J. Paulo, and A. Bessani. "GenoDedup: Similarity-Based Deduplication and Delta-Encoding for Genome Sequencing Data". In: *IEEE Transactions on Computers (TC)* Early Access (2020), pp. 1–12.

[73] V. V. Cogo. *GenoDedup*. Available at `https://github.com/vvcogo/GenoDedup`. Accessed on Apr. 19, 2020. 2020.

[74] COMMIT. *COMMIT*. Originally available at `http://www.amolf.nl/research/bims/research-activities/e-biobanking/`. Accessed on Feb. 03, 2015. Permanently available at `http://web.archive.org/web/20150203044617/http://www.amolf.nl/research/bims/research-activities/e-biobanking/`. 2020.

[75] A. Cornish-Bowden. "Nomenclature for incompletely specified bases in nucleic acid sequences: recommendations 1984". In: *Nucleic acids research* 13.9 (1985), p. 3021.

[76] A. J. Cox, M. J. Bauer, T. Jakobi, and G. Rosone. "Large-scale compression of genomic sequence databases with the Burrows–Wheeler transform". In: *Bioinformatics* 28.11 (2012), pp. 1415–1419.

[77] L. C. Crosswell and J. M. Thornton. "ELIXIR: a distributed infrastructure for European biological data". In: *Trend in Biotechnology* 30.5 (2012), pp. 241–242.

[78] J. W. Dale, M. Von Schantz, and N. Plant. *From genes to genomes: concepts and applications of DNA technology*. John Wiley & Sons, 2012.

[79] Dat Project. *rabin—Node native addon module for Rabin fingerprinting data streams*. Available at `https://github.com/datproject/rabin`. Accessed on Apr. 19, 2020. 2020.

[80] J. G. Day and G. N. Stacey. "Biobanking". In: *Molecular Biotechnology* 40.2 (2008), pp. 202–213.

[81] H. Debar, M. Dacier, and A. Wespi. “Towards a taxonomy of intrusion-detection systems”. In: *Computer Networks* 31.8 (1999), pp. 805–822.

[82] T. Debatty. *Implementation of various string similarity and distance algorithms*. Available at `https://github.com/tdebatty/java-string-similarity`. Accessed on Apr. 19, 2020. 2020.

[83] J. Decouchant, M. Fernandes, M. Voelp, F. M. Couto, and P. Esteves-Verissimo. “Accurate filtering of privacy-sensitive information in raw genomic data”. In: *Journal of biomedical informatics* 82 (2018), pp. 1–12.

[84] M. A. C. Dekker. *Critical Cloud Computing: A CIIP perspective on cloud computing services (v1.0)*. Tech. rep. European Network and Information Security Agency (ENISA), 2012.

[85] S. Deorowicz and S. Grabowski. “Compression of DNA sequence reads in FASTQ format”. In: *Bioinformatics* 27.6 (2011), pp. 860–862.

[86] S. Deorowicz and S. Grabowski. “Robust relative compression of genomes with random access”. In: *Bioinformatics* 27.21 (2011), pp. 2979–2986.

[87] S. Deorowicz and S. Grabowski. “Data compression for sequencing data”. In: *Algorithms for Molecular Biology* 8.1 (2013), p. 1.

[88] P. Deutsch. *GZIP file format specification version 4.3*. RFC 1952. RFC Editor, 1996, pp. 1–11.

[89] H. B. F. Dixon et al. “Nomenclature and Symbolism for Amino Acids and Peptides”. In: *Pure and Applied Chemistry* 56.5 (1984), pp. 595–624.

[90] DNAnexus. *DNAnexus*. Available at `https://dnanexus.com/`. Accessed on Apr. 19, 2020. 2020.

[91] F. Douglis and A. Iyengar. “Application-specific Delta-encoding via Resemblance Detection”. In: *Proc. of the USENIX Annual Technical Conference (ATC)*. 2003, pp. 113–126.

[92] C. Dwork. “Differential privacy”. In: *Automata, languages and programming*. Springer, 2006, pp. 1–12.

[93] V. J. Dzau, G. S. Ginsburg, K. Van Nuys, D. Agus, and D. Goldman. "Aligning incentives to fulfill the promise of Personalized Medicine." In: *Lancet (London, England)* 385.9982 (2015), p. 2118.

[94] A. El Allali and M. Arshad. "MZPAQ: a FASTQ data compression tool". In: *Source code for biology and medicine* 14.1 (2019), p. 3.

[95] ELIXIR. *ELIXIR*. Available at `http://www.elixir-europe.org/`. Accessed on Apr. 19, 2020. 2020.

[96] EMBL-EBI. *1000 Genomes Project: A Deep Catalog of Human Genetic Variation*. Available at `http://www.1000genomes.org/`. Accessed on Apr. 19, 2020. 2020.

[97] Y. Erlich and A. Narayanan. "Routes for breaching and protecting genetic privacy". In: *Nature Reviews Genetics* 15.6 (2014), pp. 409–421.

[98] K. Eshghi and H. K. Tang. *A framework for analyzing and improving content-based chunking algorithms*. Tech. rep. 2005. 2005.

[99] B. Ewing, L. Hillier, M. C. Wendl, and P. Green. "Base-calling of automated sequencer traces using Phred. I. Accuracy assessment". In: *Genome Research* 8.3 (1998), pp. 175–185.

[100] H. Fan and J.-Y. Chu. "A brief review of short tandem repeat mutation". In: *Genomics, Proteomics & Bioinformatics* 5.1 (2007), pp. 7–14.

[101] FARGEN. *FARGEN - Faroe Genome Project*. Available at `https://www.fargen.fo/en/`. Accessed on Apr. 19, 2020. 2020.

[102] M. Fernandes, J. Decouchant, M. Volp, F. M. Couto, and P. Verissimo. "DNA-SeAl: Sensitivity Levels to Optimize the Performance of Privacy-Preserving DNA Alignment". In: *IEEE Journal of Biomedical and Health Informatics* Early Access (2019), pp. 1–8.

[103] L. Freeman, R. Bolt, and T. Sas. *Evaluation criteria for data de-dupe*. INFOSTOR. 2007.

[104] D. Frey, A.-M. Kermarrec, and K. Kloudas. “Probabilistic deduplication for cluster-based storage systems”. In: *Proc. of the ACM Symposium on Cloud Computing (SoCC)*. 2012, p. 17.

[105] M. H.-Y. Fritz, R. Leinonen, G. Cochrane, and E. Birney. “Efficient storage of high throughput DNA sequencing data using reference-based compression”. In: *Genome research* 21.5 (2011), pp. 734–740.

[106] Galaxy Project. *The Galaxy Project – Online bioinformatics analysis for everyone*. Available at http://galaxyproject.org/. Accessed on Apr. 19, 2020. 2020.

[107] Y. Gelfand, A. Rodriguez, and G. Benson. “TRDB—the tandem repeats database”. In: *Nucleic Acids Research* 35.suppl 1 (2007), pp. D80–D87.

[108] Genealogy by Genetics, Ltd. *Ysearch: The number one Y-DNA public database*. Available at: http://www.ysearch.org/. Accessed on Apr. 19, 2020. 2020.

[109] C. Gentry et al. “Fully homomorphic encryption using ideal lattices.” In: *Proc. of the 41st ACM Symposium on Theory of Computing (STOC)*. Vol. 9. 2009, pp. 169–178.

[110] R. Gerhards. *The syslog protocol*. RFC 5424. 2009.

[111] L. Goldsmith, L. Jackson, A. O’Connor, and H. Skirton. “Direct-to-consumer genomic testing: systematic review of the literature on user perspectives”. In: *European Journal of Human Genetics* 20.8 (2012), pp. 811–816.

[112] Google, Inc. *Google Storage Pricing*. Available at https://cloud.google.com/storage/pricing. Accessed on Apr. 19, 2020. 2020.

[113] I. Grebnov. *BSC: High performance data compression library*. Available at http://libbsc.com/. Accessed on Apr. 19, 2020. 2020.

[114] D. Greenbaum, A. Sboner, X. J. Mu, and M. Gerstein. “Genomics and privacy: Implications of the new reality of closed data for the field”. In: *PLOS Computational Biology* 7.12 (2011), e1002278.

[115] D. Grishin, J. L. Raisaro, J. R. Troncoso-Pastoriza, et al. “Citizen-Centered, Auditable, and Privacy-Preserving Population Genomics”. In: *bioRxiv* (2019), pp. 1–15.

[116] S. Grumbach and F. Tahi. “A new challenge for compression algorithms: genetic sequences”. In: *Information Processing & Management* 30.6 (1994), pp. 875–886.

[117] R. Guerraoui and M. Vukolić. “How Fast Can a Very Robust Read Be?” In: *Proc. of the 25th Annual ACM Symposium on Principles of Distributed Computing (PODC)*. 2006, pp. 248–257.

[118] M. Gymrek, A. L. McGuire, D. Golan, and et al. “Identifying personal genomes by surname inference”. In: *Science* 339.6117 (2013), pp. 321–324.

[119] F. Hach, I. Numanagić, C. Alkan, and S. C. Sahinalp. “SCALCE: boosting sequence compression algorithms using locally consistent encoding”. In: *Bioinformatics* 28.23 (2012), pp. 3051–3057.

[120] J. Hadfield. *NGS Mapped*. Available at `http://enseqlopedia.com/ngs-mapped/`. Accessed on Apr. 19, 2020. 2020.

[121] V. Hadzilacos and S. Toueg. *A modular approach to fault-tolerant broadcasts and related problems*. Tech. rep. Cornell University, 1994.

[122] A. Haeberlen. “A case for the accountable cloud”. In: *ACM SIGOPS Operating Systems Review* 44.2 (2010), pp. 52–57.

[123] A. Haeberlen, P. Kouznetsov, and P. Druschel. “PeerReview: Practical Accountability for Distributed Systems”. In: *Proc. of 21st ACM SIGOPS Symposium on Operating Systems Principles (SOSP)*. 2007.

[124] M. A. Hamburg and F. S. Collins. “The path to personalized medicine”. In: *New England Journal of Medicine* 2010.363 (2010), pp. 301–304.

[125] S. Han, H. Shen, T. Kim, et al. “MetaSync: File Synchronization Across Multiple Untrusted Storage Services”. In: *Proc. of the USENIX Annual Technical Conference (ATC)*. 2015.

[126] L. Harding. *The Snowden files: The inside story of the world's most wanted man*. Guardian Faber Publishing, 2014.

[127] D. Haussler et al. *A million cancer genome warehouse*. Tech. rep. University of Berkley, Dept. of Electrical Engineering and Computer Science, 2012.

[128] J. Hendricks, G. R. Ganger, and M. K. Reiter. "Low-overhead Byzantine Fault-tolerant Storage". In: *Proc. of 21st ACM SIGOPS Symposium on Operating Systems Principles (SOSP)*. 2007.

[129] M. P. Herlihy and J. M. Wing. "Linearizability: A correctness condition for concurrent objects". In: *ACM Transactions on Programming Languages and Systems (TOPLAS)* 12.3 (1990), pp. 463–492.

[130] J. N. Hirschhorn and M. J. Daly. "Genome-wide association studies for common diseases and complex traits". In: *Nature Reviews Genetics* 6.2 (2005), pp. 95–108.

[131] J. Hoang. *The Genome Marketplace*. Available at `http://www.immpressmagazine.com/the-genome-marketplace/`. Accessed on Apr. 19, 2020. 2018.

[132] N. Homer, S. Szelinger, M. Redman, and et al. "Resolving individuals contributing trace amounts of DNA to highly complex mixtures using high-density SNP genotyping microarrays". In: *PLOS Genetics* 4.8 (2008), e1000167.

[133] M. Howison. "High-throughput compression of FASTQ data with SeqDB". In: *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)* 10.1 (2013), pp. 213–218.

[134] D. A. Huffman. "A method for the construction of minimum-redundancy codes". In: *Proc. of the Institute of Radio Engineers (IRE)* 40.9 (1952), pp. 1098–1101.

[135] IBM. *IBM*. Available at `https://www.ibm.com/cloud-computing/bluemix/pricing-object-storage`. Accessed on Apr. 19, 2020. 2020.

[136] Illumina, Inc. *HiSeq 2000 Sequencing System*. Available at `https://www.illumina.com/documents/products/datasheets/datasheet_hiseq2000.pdf`. Accessed on Apr. 19, 2020. 2010.

[137] Illumina, Inc. *Quality Scores for Next-Generation Sequencing*. In: https://www.illumina.com/documents/products/technotes/technote_Q-Scores.pdf. Accessed on Apr. 19, 2020. 2011.

[138] Illumina, Inc. *Estimating Sequencing Coverage*. Available at https://jp.support.illumina.com/content/dam/illumina-marketing/documents/products/technotes/technote_coverage_calculation.pdf. Accessed on Apr. 19, 2020. 2014.

[139] Illumina, Inc. *Human Whole-Genome Sequencing with the HiSeq X Sequencing System*. In: https://www.illumina.com/content/dam/illumina-marketing/documents/products/appnotes/appnote-hiseq-x.pdf. Accessed on Apr. 19, 2020. 2014.

[140] Illumina, Inc. *NovaSeqTM 6000 System Quality Scores and RTA3 Software*. In: https://www.illumina.com/content/dam/illumina-marketing/documents/products/appnotes/novaseq-hiseq-q30-app-note-770-2017-010.pdf. Accessed on Apr. 19, 2020. 2017.

[141] Illumina, Inc. *Illumina BaseSpace*. Available at https://basespace.illumina.com/. Accessed on Apr. 19, 2020. 2020.

[142] Illumina, Inc. *Illumina introduces the HiSeq X Ten sequencing system*. Available at: https://www.businesswire.com/news/home/20140114006291/en/Illumina-Introduces-HiSeq-X%E2%84%A2-Ten-Sequencing-System. Accessed on Apr. 19, 2020. 2020.

[143] P. Indyk and R. Motwani. "Approximate nearest neighbors: towards removing the curse of dimensionality". In: *Proc. of the 30th ACM Symposium on Theory of Computing (STOC)*. 1998, pp. 604–613.

[144] International Human Genome Sequencing Consortium. "Finishing the euchromatic sequence of the human genome". In: *Nature* 431.7011 (2004), pp. 931–945.

[145] International Nucleotide Sequence Database Collaboration. *INSDC*. Available at http://www.insdc.org/. Accessed on Apr. 19, 2020. 2020.

[146] A. K. Jain. “Data clustering: 50 years beyond K-means”. In: *Pattern Recognition Letters* 31.8 (2010), pp. 651–666.

[147] R. K. Jakobsen. “Sequencing the genome of an entire population”. In: *ScienceNordic* (2012). Available at: http://sciencenordic.com/sequencing-genome-entire-population. Accessed on Apr. 19, 2020.

[148] M. Janitz. *Next-generation genome sequencing: towards personalized medicine*. John Wiley & Sons, 2011.

[149] P. Jayanti, T. D. Chandra, and S. Toueg. “Fault-tolerant Wait-free Shared Objects”. In: *J. ACM* 45.3 (1998), pp. 451–500.

[150] A. Johnson and V. Shmatikov. “Privacy-preserving data exploration in genome-wide association studies”. In: *Proc. of the 19th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*. 2013, pp. 1079–1087.

[151] D. C. Jones, W. L. Ruzzo, X. Peng, and M. G. Katze. “Compression of next-generation sequencing reads aided by highly efficient de novo assembly”. In: *Nucleic acids research* 40.22 (2012), e171–e171.

[152] A. Juels and B. S. Kaliski Jr. “PORs: Proofs of retrievability for large files”. In: *Proc. of the 14th ACM conference on Computer and communications security (CCS)*. 2007, pp. 584–597.

[153] K. Zhang *et al.* “Sedic: privacy-aware data intensive computing on hybrid clouds”. In: *Proc. of the 18th ACM conference on Computer and communications security (CCS)*.

[154] S. D. Kahn et al. “On the future of genomic data”. In: *Science(Washington)* 331.6018 (2011), pp. 728–729.

[155] D. J. Kaufman, J. Murphy-Bollinger, J. Scott, and et al. “Public opinion about the importance of privacy in biobank research”. In: *The American Journal of Human Genetics* 85.5 (2009), pp. 643–654.

[156] T. E. King and M. A. Jobling. “What’s in a name? Y chromosomes, surnames and the genetic genealogy revolution”. In: *Trends in Genetics* 25.8 (2009), pp. 351–360.

[157] R. K. Ko. “Data accountability in cloud systems”. In: *Security, Privacy and Trust in Cloud Systems*. Springer, 2014, pp. 211–238.

[158] R. K. Ko, B. S. Lee, and S. Pearson. “Towards achieving accountability, auditability and trust in cloud computing”. In: *Proc. of the 1st International Conference on Advances in Computing and Communications (ACC)*. 2011, pp. 432–444.

[159] M. Kolhar, M. M. Abu-Alhaj, and S. M. A. El-atty. “Cloud data auditing techniques with a focus on privacy and security”. In: *IEEE Security & Privacy* 15.1 (2017), pp. 42–51.

[160] C. Kozanitis, C. Saunders, S. Kruglyak, V. Bafna, and G. Varghese. “Compressing genomic sequence fragments using SlimGene”. In: *Journal of Computational Biology* 18.3 (2011), pp. 401–413.

[161] H. Krawczyk. “Secret sharing made short”. In: *Proc. of the 13th Annual International Cryptology Conference (CRYPTO)*. 1993, pp. 136–146.

[162] L. Lamport. “On interprocess communication”. In: *Distributed computing* 1.2 (1986), pp. 86–101.

[163] L. Lamport. “Lower Bounds for Asynchronous Consensus”. In: *Distrib. Comput.* 19.2 (2006), pp. 104–125.

[164] A. Lesk. *Introduction to bioinformatics*. Oxford University Press, 2013.

[165] H. Li and N. Homer. “A survey of sequence alignment algorithms for next-generation sequencing”. In: *Briefings in bioinformatics* 11.5 (2010), pp. 473–483.

[166] H. Li, B. Handsaker, A. Wysoker, et al. “The sequence alignment/map format and SAMtools”. In: *Bioinformatics* 25.16 (2009), pp. 2078–2079.

[167] P. Li and C. König. “b-Bit minwise hashing”. In: *Proc. of the 19th International Conference on World Wide Web (WWW)*. 2010, pp. 671–680.

[168] M. Lillibridge et al. “Sparse Indexing: Large Scale, Inline Deduplication Using Sampling and Locality”. In: *Proc. of the USENIX Conference on File and Storage Technologies (FAST)*. Vol. 9. 2009, pp. 111–123.

[169] Z. Lin, A. B. Owen, and R. B. Altman. “Genomic Research and Human Subject Privacy”. In: *Science* 305.5681 (2004), p. 183.

[170] J.-E. Litton, J. Muilu, A. Bjorklund, A. Leinonen, and N. L. Pedersen. “Data modeling and data communication in GenomEUtwin”. In: *Twin Research* 6.5 (2003), pp. 383–390.

[171] J.-E. Litton, R. Martinez, J. Reichel, et al. *Disclosure Model*. Deliverable D1.4 of BiobankCloud Project. 2015.

[172] L. Liu, Y. Li, S. Li, et al. “Comparison of next-generation sequencing systems”. In: *Journal of Biomedicine and Biotechnology* 2012 (2012), pp. 1–11.

[173] Logical Clocks. *Hopsworks: The Platform for Data-Intensive AI with a Feature Store*. Available at `https://www.logicalclocks.com/`. Accessed on Apr. 19, 2020. 2020.

[174] *LongFastBloomFilter*. `http://code.google.com/p/java-longfastbloomfilter/`. Accessed on Apr. 19, 2020. 2020.

[175] M. P. Van der Loo. “The *stringdist* package for approximate string matching”. In: *The R* (2014), p. 2.

[176] M. Mahoney. *The LPAQ Compression Algorithm*. Available at `http://mattmahoney.net/dc/#lpaq`. Accessed on Apr. 19, 2020. 2007.

[177] M. Mahoney. *Data compression explained*. Available at `http://mattmahoney.net/dc/dce.html`. Accessed on Apr. 19, 2020. 2013.

[178] M. Mahoney. *The ZPAQ Compression Algorithm*. Available at `http://mattmahoney.net/dc/zpaq_compression.pdf`. Accessed on Apr. 19, 2020. 2015.

[179] D. Malkhi and M. Reiter. “Byzantine quorum systems”. In: *Distributed Computing* 11.4 (1998), pp. 203–213.

[180] E. R. Mardis. “Next-generation DNA sequencing methods”. In: *Annual Review of Genomics and Human Genetics* 9 (2008), pp. 387–402.

[181] V. Marx. “Biology: The big challenges of big data”. In: *Nature* 498.7453 (2013), pp. 255–260.

[182] G. Mayer. “Data management in systems biology I - Overview and bibliography”. In: *CoRR* abs/0908.0411 (2009).

[183] R. Mendes, T. Oliveira, V. V. Cogo, N. Neves, and A. Bessani. “CHARON: A Secure Cloud-of-Clouds System for Storing and Sharing Big Data”. In: *IEEE Transactions on Cloud Computing (TCC)* Early Access (2019), pp. 1–12.

[184] Microsoft Corporation. *Windows Azure Pricing*. Available at `https://azure.microsoft.com/pricing/details/storage/blobs/`. Accessed on Apr. 19, 2020. 2020.

[185] A. Moffat. “Implementing the PPM data compression scheme”. In: *IEEE Transactions on communications* 38.11 (1990), pp. 1917–1921.

[186] M. Molteni. *These DNA Startups Want to Put All of You on the Blockchain*. Available at `https://www.wired.com/story/these-dna-startups-want-to-put-all-of-you-on-the-blockchain/`. Accessed on Apr. 19, 2020. 2018.

[187] J. Muilu, L. Peltonen, and J.-E. Litton. “The federated database–a basis for biobank-based post-genome studies, integrating phenome and genome data from 600 000 twin pairs in Europe”. In: *European Journal of Human Genetics* 15.7 (2007), pp. 718–723.

[188] R. Myers et al. *Genome Reference Consortium*. Available at `http://genomereference.org/`. Accessed on Apr. 19, 2020. 2020.

[189] M. Naehrig, K. Lauter, and V. Vaikuntanathan. “Can homomorphic encryption be practical?” In: *Proc. of the 3rd ACM workshop on Cloud computing security workshop (CCSW)*. 2011, pp. 113–124.

[190] V. Navale and P. E. Bourne. “Cloud computing applications for biomedical science: A perspective”. In: *PLOS Computational Biology* 14.6 (2018).

[191] M. Naveed, E. Ayday, E. W. Clayton, et al. “Privacy in the genomic era”. In: *ACM Computing Surveys (CSUR)* 48.1 (2015), p. 6.

[192] S. Niazi, M. Ismail, S. Haridi, et al. “Hopsfs: Scaling hierarchical file system metadata using newsql databases”. In: *15th USENIX Conference on File and Storage Technologies (FAST 17)*. 2017, pp. 89–104.

[193] M. Nicolae. *Estimating Sequencing Coverage*. Available at `https://github.com/mariusmni/lfqc/issues/4`. Accessed on Apr. 19, 2020. 2020.

[194] M. Nicolae, S. Pathak, and S. Rajasekaran. “LFQC: a lossless compression algorithm for FASTQ files”. In: *Bioinformatics* 31.20 (2015), pp. 3276–3281.

[195] D. R. Nyholt, C.-E. Yu, and P. M. Visscher. “On Jim Watson’s APoE status: genetic information is hard to hide”. In: *European Journal of Human Genetics* 17 (2009), pp. 147–149.

[196] I. Ochoa, M. Hernaez, R. Goldfeder, T. Weissman, and E. Ashley. “Effect of lossy compression of quality scores on variant calling”. In: *Briefings in bioinformatics* 18.2 (2017), pp. 183–194.

[197] W. Ollier, T. Sprosen, and T. Peakman. “UK Biobank: from concept to reality”. In: *Pharmacogenomics* 6.6 (2005), pp. 639–646.

[198] OMIM. *OMIM*. Available at: `http://omim.org/`. Accessed on Apr. 19, 2020. 2020.

[199] Oracle, Inc. *Oracle Storage Pricing*. Available at `https://cloud.oracle.com/storage/pricing`. Accessed on Apr. 19, 2020. 2020.

[200] C. Orengo, D. T. Jones, and J. M. Thornton. *Bioinformatics: genes, proteins and computers*. Garland Science, 2003.

[201] E. Parliament. “Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation)”. In: *Official Journal of the European Union* L119 (2016), pp. 1–88.

[202] J. Paulo and J. Pereira. “A survey and classification of storage deduplication systems”. In: *ACM Computing Surveys (CSUR)* 47.1 (2014), p. 11.

[203] D. Pavlichin, T. Weissman, and G. Mably. “The quest to save genomics: Unless researchers solve the looming data compression problem, biomedical science could stagnate”. In: *IEEE Spectrum* 55.9 (2018), pp. 27–31.

[204] I. Pavlov. *LZMA*. Available at https://www.7-zip.org/. Accessed on Apr. 19, 2020. 2020.

[205] W. R. Pearson and D. J. Lipman. “Improved tools for biological sequence comparison”. In: *Proceedings of the National Academy of Sciences (PNAS) of the United States of America* 85.8 (1988), pp. 2444–2448.

[206] A. J. Pinho and D. Pratas. “MFCompress: a compression tool for FASTA and multi-FASTA data”. In: *Bioinformatics* 30.1 (2014), pp. 117–118.

[207] J. S. Plank. “Erasure codes for storage systems: A brief primer”. In: *The USENIX Magazine* 38.6 (2013), pp. 44–50.

[208] R. A. Popa, J. R. Lorch, D. Molnar, H. J. Wang, and L. Zhuang. “Enabling Security in Cloud Storage SLAs with CloudProof.” In: *Proc. of the USENIX Annual Technical Conference (ATC)*. Vol. 242. 2011.

[209] M. O. Rabin. “Efficient dispersal of information for security, load balancing, and fault tolerance”. In: *Journal of the ACM (JACM)* 36.2 (1989), pp. 335–348.

[210] M. O. Rabin et al. *Fingerprinting by random polynomials*. Center for Research in Computing Techn., Aiken Computation Laboratory, Univ., 1981.

[211] A. Regalado. *China’s BGI says it can sequence a genome for just $100*. Available at https://www.technologyreview.com/s/615289/china-bgi-100-dollar-genome/. Accessed on Apr. 19, 2020. 2020.

[212] J. K. Resch and J. S. Plank. “AONT-RS: Blending Security and Performance in Dispersed Storage Systems”. In: *Proc. of the 9th USENIX Conference on File and Storage Technologies (FAST)*. 2011, p. 14.

[213] Ł. Roguski and S. Deorowicz. “DSRC 2—Industry-oriented compression of FASTQ files”. In: *Bioinformatics* 30.15 (2014), pp. 2213–2215.

[214] L. Roguski, I. Ochoa, M. Hernaez, and S. Deorowicz. “FaStore: a space-saving solution for raw sequencing data”. In: *Bioinformatics* 34.16 (2018), pp. 2748–2756.

[215] C. M. Ruitberg, D. J. Reeder, and J. M. Butler. “STRBase: a short tandem repeat DNA database for the human identity testing community”. In: *Nucleic Acids Research* 29.1 (2001), pp. 320–322.

[216] M. Sabt, M. Achemlal, and A. Bouabdallah. “Trusted execution environment: what it is, and what it is not”. In: *2015 IEEE Trustcom/BigDataSE/ISPA*. Vol. 1. IEEE. 2015, pp. 57–64.

[217] M. Safran et al. “GeneCards Version 3: the human gene integrator”. In: *Database* 2010 (2010), baq020.

[218] D. Salomon and G. Motta. *Handbook of data compression*. Springer Science & Business Media, 2010.

[219] V. A. Schneider et al. “Evaluation of GRCh38 and de novo haploid genome assemblies demonstrates the enduring quality of the reference assembly”. In: *Genome Research* 27.5 (2017), pp. 849–864.

[220] O. Security. *The 21 Biggest Data Breaches Of The 21st Century*. Available at `https://optimumsecurity.ca/21-biggest-data-breach-of-21-century`. Accessed on Apr. 19, 2020. 2020.

[221] J. Seward. *bzip2 and libbzip2*. Available at `https://github.com/enthought/bzip2-1.0.6`. Accessed on Apr. 19, 2020. 2020.

[222] A. Shamir. “How to Share a Secret”. In: *Communications of the ACM* 22.11 (1979), pp. 612–613.

[223] J. Shendure and H. Ji. “Next-generation DNA sequencing”. In: *Nature biotechnology* 26.10 (2008), pp. 1135–1145.

[224] S. S. Shringarpure and C. D. Bustamante. “Privacy risks from genomic data-sharing beacons”. In: *The American Journal of Human Genetics* 97.5 (2015), pp. 631–646.

[225] D. Slamanig and C. Hanser. “On cloud storage and the cloud of clouds approach”. In: *Proc. of the IEEE International Conference for Internet Technology And Secured Transactions (ICITST)*. 2012, pp. 649–655.

[226] Sorenson Molecular Genealogy Foundation. *SMGF: Y-Chormosome database*. Originally available at `http://www.smgf.org/`. Accessed on Aug. 9, 2013. Permanently available at `http://web.archive.org/web/20130809023911/http://www.smgf.org/`. 2020.

[227] A. Spiegelman, Y. Cassuto, G. Chockler, and I. Keidar. “Space bounds for reliable storage: Fundamental limits of coding”. In: *Proc. of the 35th ACM Symposium on Principles of Distributed Computing (PODC)*. 2016, pp. 249–258.

[228] L. D. Stein et al. “The case for cloud computing in genome informatics”. In: *Genome Biology* 11.5 (2010), p. 207.

[229] M. Steinbach, G. Karypis, V. Kumar, et al. “A comparison of document clustering techniques”. In: *Proc. of the KDD Workshop on Text Mining*. Vol. 400. 1. 2000, pp. 525–526.

[230] Z. D. Stephens, S. Y. Lee, F. Faghri, et al. “Big data: astronomical or genomical?” In: *PLOS Biology* 13.7 (2015), e1002195.

[231] H. Takabi, J. B. Joshi, and G.-J. Ahn. “Security and privacy challenges in cloud computing environments”. In: *IEEE Security & Privacy* 8.6 (2010), pp. 24–31.

[232] G. Tan, L. Opitz, R. Schlapbach, and H. Rehrauer. “Long fragments achieve lower base quality in Illumina paired-end sequencing”. In: *Scientific Reports* 9:2856.1 (2019), pp. 1–7.

[233] H. Tang, F. Liu, G. Shen, Y. Jin, and C. Guo. “UniDrive: Synergize Multiple Consumer Cloud Storage Services”. In: *Proc. of the ACM/IFIP/USENIX Middleware*. 2015.

[234] W. Tembe, J. Lowey, and E. Suh. “G-SQZ: compact encoding of genomic sequence and quality data”. In: *Bioinformatics* 26.17 (2010), pp. 2192–2194.

[235] The 1000 Genomes Project Consortium. “An integrated map of genetic variation from 1,092 human genomes”. In: *Nature* 491 (2012), pp. 56–65.

[236] G. Tischler and S. Leonard. “biobambam: tools for read pair collation based algorithms on BAM files”. In: *Source Code for Biology and Medicine* 9.1 (2014), p. 13.

[237] UK Biobank. *UK Biobank*. Available at http://www.ukbiobank.ac.uk/. Accessed on Apr. 19, 2020. 2020.

[238] UniProt Consortium. “The universal protein resource (UniProt)”. In: *Nucleic Acids Research* 36.suppl 1 (2008), pp. D190–D195.

[239] U.S. National Library of Medicine. *Help Me Understand Genetics*. Available at https://ghr.nlm.nih.gov/primer. Accessed on Apr. 19, 2020. 2020.

[240] B. Vavala, N. Neves, and P. Steenkiste. “Secure tera-scale data crunching with a small TCB”. In: *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE. 2017, pp. 169–180.

[241] Vawlt Technologies. *Vawlt - Dynamic Storage Platform*. Available at https://vawlt.io/. Accessed on Apr. 19, 2020. 2020.

[242] E Vayena and U Gasser. “Between Openness and Privacy in Genomics.” In: *PLOS Medicine* 13.1 (2016), e1001937.

[243] J. C. Venter et al. “The sequence of the human genome”. In: *Science* 291.5507 (2001), pp. 1304–1351.

[244] P. E. Verissimo and A. Bessani. “E-biobanking: What Have You Done to My Cell Samples?” In: *IEEE Security&Privacy* 11.6 (2013), pp. 62–65.

[245] R. Wan and K. Asai. “Sorting next generation sequencing data improves compression effectiveness”. In: *Proc. of the IEEE International Conference on Bioinformatics and Biomedicine Workshops (BIBMW)*. 2010, pp. 567–572.

[246] R. Wan, V. N. Anh, and K. Asai. “Transformations for the compression of FASTQ quality scores of next-generation sequencing data”. In: *Bioinformatics* 28.5 (2012), pp. 628–635.

[247] S. Wandelt and U. Leser. “FRESCO: Referential compression of highly similar sequences”. In: *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)* 10.5 (2013), pp. 1275–1288.

[248] S. Wandelt, M. Bux, and U. Leser. “Trends in genome compression”. In: *Current Bioinformatics* 9.3 (2014), pp. 315–326.

[249] C. Wang, S. S. Chow, Q. Wang, K. Ren, and W. Lou. “Privacy-preserving public auditing for secure cloud storage”. In: *IEEE Transactions on Computers* 62.2 (2013), pp. 362–375.

[250] R. Wang, Y. F. Li, X. Wang, H. Tang, and X. Zhou. “Learning Your Identity and Disease from Research Papers: Information Leaks in Genome Wide Association Study”. In: *Proc. of the 16th ACM conference on Computer and communications security (CCS)*. 2009, pp. 534–544.

[251] R. W. G. Watson, E. W. Kay, and D. Smith. “Integrating biobanks: addressing the practical and ethical issues to deliver a valuable tool for cancer research”. In: *Nature Reviews Cancer* 10.9 (2010), pp. 646–651.

[252] H. Weatherspoon and J. D. Kubiatowicz. “Erasure coding vs. replication: A quantitative comparison”. In: *Peer-to-Peer Systems*. Springer, 2002, pp. 328–337.

[253] K. A. Wetterstrand. *DNA Sequencing Costs*. Available at `http://www.genome.gov/sequencingcostsdata`. Accessed on Apr. 19, 2020. 2020.

[254] D. A. Wheeler et al. “The complete genome of an individual by massively parallel DNA sequencing”. In: *Nature* 452.7189 (2008), pp. 872–876.

[255] C. Wilks, M. S. Cline, E. Weiler, et al. “The Cancer Genomics Hub (CGHub): overcoming cancer through the power of torrential data”. In: *Database* 2014 (2014), bau093.

[256] L. Xu, A. Pavlo, S. Sengupta, and G. R. Ganger. “Online Deduplication for Databases”. In: *Proc. of the ACM International Conference on Management of Data (SIGMOD)*. 2017, pp. 1355–1368.

[257] A. R. Yumerefendi and J. S. Chase. “The role of accountability in dependable distributed systems”. In: *Proc. of the 1st Workshop on Hot Topics in System Dependability (HotDep)*. Vol. 5. 2005, pp. 3–3.

[258] Y. Zhang, K. Patel, T. Endrawis, A. Bowers, and Y. Sun. “A FASTQ compressor based on integer-mapped k-mer indexing for biologist”. In: *Gene* 579.1 (2016), pp. 75–81.

[259] Y. Zhang et al. “Light-weight reference-based compression of FASTQ data”. In: *BMC Bioinformatics* 16.1 (2015), p. 188.

[260] J. Zhou, Z. Ji, Z. Zhu, and S. He. “Compression of next-generation sequencing quality scores using memetic algorithm”. In: *BMC Bioinformatics* 15.15 (2014), p. 1.

[261] B. Zhu, K. Li, and R. H. Patterson. “Avoiding the Disk Bottleneck in the Data Domain Deduplication File System”. In: *Proc. of the USENIX Conference on File and Storage Technologies (FAST)*. Vol. 8. 2008, pp. 1–14.