Buchmann *et al. EURASIP Journal on Information Security* (2019) 2019:16
https://doi.org/10.1186/s13635-019-0099-x

EURASIP Journal on Information Security

RESEARCH Open Access

Long-term integrity protection of genomic data

Johannes Buchmann, Matthias Geihs†, Kay Hamacher, Stefan Katzenbeisser and Sebastian Stammler*†

Abstract

Genomic data is crucial in the understanding of many diseases and for the guidance of medical treatments. Pharmacogenomics and cancer genomics are just two areas in precision medicine of rapidly growing utilization. At the same time, whole-genome sequencing costs are plummeting below $ 1000, meaning that a rapid growth in full-genome data storage requirements is foreseeable. While privacy protection of genomic data is receiving growing attention, integrity protection of this long-lived and highly sensitive data much less so.

We consider a scenario inspired by future pharmacogenomics, in which a patient's genome data is stored over a long time period while random parts of it are periodically accessed by authorized parties such as doctors and clinicians. A protection scheme is described that preserves integrity of the genomic data in that scenario over a time horizon of 100 years. During such a long time period, cryptographic schemes will potentially break and therefore our scheme allows to update the integrity protection. Furthermore, integrity of parts of the genomic data can be verified without compromising the privacy of the remaining data. Finally, a performance evaluation and cost projection shows that privacy-preserving long-term integrity protection of genomic data is resource demanding, but in reach of current and future hardware technology and has negligible costs of storage.

Keywords: Long-term security, Integrity, Authenticity, Genomic data, Genomic privacy, Genomic security, Commitments, Timestamps, Renewable cryptography

1 Introduction

Full genome sequencing is becoming a standard medical procedure in the near future, not only in the assessment of many diseases but also in the research or consumer services setting. For example, in its recent annual report [1], the UK's chief medical officer called for a revolution of gene testing and wants whole-genome sequencing to become a standard procedure for National Health Service patients—not only for cancer treatment but also rare diseases testing, targeting of drugs etc.

With decreasing sequencing costs, periodic and tissue specific sequencing will be the next step forward. Thus, storage requirements are ever increasing and long-term data protection schemes become more complex. While genomic *privacy* is attracting much attention recently [2–4], the assurance of genomic data *integrity* has almost not been discussed yet. Genomic data not only requires hundreds of gigabytes of storage but also needs to be secured against loss and tampering for at least a human life span.

This paper is concerned with the integrity protection of genomic data for decades after data generation. As cryptographic primitives such as hash algorithms and signatures may become insecure in the future this undertaking is challenging.

1.1 Motivation

Endeavors like the 100,000 Genomes Project [5] in the UK show that one important scenario to consider is the outsourcing of genomic data storage to a trusted third party. The key challenge is to guarantee that none of the outsourced data gets ever modified, either by an outside attacker or even an insider, over a hundred years. In the future, doctors might get authorized access to parts of a patient's genome, stored in a national database, to support personalized medicine decisions. A renowned example from pharmacogenomics is the dosage determination for

*Correspondence: sebastian.stammler@crisp-da.de
†Matthias Geihs and Sebastian Stammler contributed equally to this work.
Technische Universität Darmstadt, Department of Computer Science, Hochschulstraße 10, 64289 Darmstadt, Germany

drug Warfarin based on just a few single-nucleotide polymorphisms (SNPs) [6–8]: for certain variants of CYP2C9, only a fifth of the normal dose is recommended. This prime example shows why even the change, or suppression, of a few entries in a database of genomic variants can have disastrous consequences on treatment decisions with implications for liability and legal procedures.

On the technical side, cryptographic primitives like symmetric encryption schemes, digital signature schemes, or hash functions are deemed to break over time. For example, in 1997 the widely used symmetric encryption scheme DES was broken by brute force for the first time[1] and can nowadays be broken for a small fee on `crack.sh`. Also in 1997, the results of Shor [9] showed that the RSA signature scheme is insecure against quantum computers. In 2004, Wang et al. [10] for the first time found collisions for the three then popular hash functions `MD5`, `HAVAL-128`, and `RIPEMD`. Thus, long-term security needs to take future breaches of cryptographic primitives into account.

1.2 Contribution

In this paper, we propose a solution that allows to store genetic data in a database, while guaranteeing integrity and authenticity over long time periods. Data may be stored in plain-text, encrypted, or secretly shared form. We examine a scenario in which a full set of raw sequencer reads, alignments, and genomic variant data files are generated and stored in a certified database (see Sections 2 and 3).

We propose a long-term protection scheme (Section 4) that uses unconditionally hiding commitments, Merkle hash trees, and digital signatures for protecting the integrity of the data while preserving confidentiality. The scheme allows querying and proving of integrity and authenticity of specific positions in the genome while leaving the remaining data undisclosed. No information can be inferred about adjacent positions. The scheme supports updating the integrity protection in case one of the used cryptographic schemes (i.e., commitments, hashes, or signatures) is expected to become insecure in the near future. The integrity update procedure uses timestamping while it is guaranteed that no information is leaked to the involved timestamp servers.

We also evaluate the performance of our scheme (Section 5), in a scenario with periodic updates of the timestamps, commitments and hashes. Our performance evaluation shows that long-term integrity protection of a human genome of size $3 \cdot 10^9$ is feasible on current hardware. Furthermore, verification of the integrity of a small subset of genomic data is fast.

[1] Achieved by the *DESCHALL Project*, the winners of the first $ 10,000 *DES Challenge* by *RSA Security*.

1.3 Related work

Various timestamping-based long-term integrity protection schemes for various use cases have been proposed in the literature [11, 12]. However, these schemes leak information to the involved timestamp services and therefore do not preserve long-term confidentiality of the protected data. Braun et al. [13] use unconditionally hiding commitments to combine long-term integrity with long-term confidentiality protection. However, they only consider the protection of a single large data item while genomic databases consist of a large number of relatively small data items. Computation and storage costs of their scheme scale unfavorably for such databases, because each data item needs to be protected by a separate signature-timestamp pair, which is costly to generate and store. We resolve this issue by using Merkle Hash Trees [14] which enable us to protect a whole dataset with just a single signature-timestamp pair.

As an alternative to computationally secure signature schemes, proposals for unconditionally secure signature schemes which do not rely on computational assumptions [15] exist as well. However, these schemes function inherently differently from their computationally secure counterparts and require a number of other strong assumptions, e.g., that data verifiers are known and active at scheme initialization. They are thus not applicable to the scenarios discussed here.

In the field of genomic data security, the recent work by Bradley et al. [16] explores several methods for the integrity protection of genomic data. Merkle hash trees are also studied to deliver integrity protection of single positional mutations while keeping the remaining positions confidential. Instead of commitments, they use a similar approach by salting the leaf values before hashing. The authors argue that, without salting, up to 32 neighboring base nucleotide leafs could be revealed by learning the hashes along the path to the MHT root. However, the paper does not consider the long-term aspect of data storage, with cryptographic primitives becoming insecure over time. Achieving long-term security is the main focus of this work.

The same can be said about recent works on blockchain-based integrity protection [17, 18]. While decentralized blockchain technology is a novel and promising approach to data integrity and time-stamping, it faces the same long-term security issues like any other scheme that does not include regular updates of hash functions. Hence, these works do not solve the problem of long-term protection. Recently, Bansarkhani et al. [19] explored long-term integrity of blockchains. When the time comes to replace a hash function, the authors propose to hash the whole blockchain and store this hash in a new block, resulting in extended data integrity. However, this approach is not applicable to the random-access queries that we will

introduce, where we only want to proof integrity of parts of the genomic data.

2 Genomic data

For completeness, we give a short overview of all relevant genomic file formats even thought our actual scheme will only be applied to variant data (VCF files).

2.1 File types

The initial data produced by genome sequencers goes through several steps of processing to reach different levels of representation and abstraction. In our scenario, we are interested in storing genomic variations, which have high utility in personalized medicine. They allow random access to specific positions and, at the same time, protection of adjacent genomic positions.

Sequencers produce short raw reads, that, in a first step, are aligned to form a contiguous genome. Those aligned genomes can then be compared to a reference genome to deliver a more interpreted view, highlighting the genomic variation.

2.1.1 Raw reads

Typically, sequencing machines produce output in the FASTQ format, consisting of billions of small unaligned so-called reads (of nucleotides, making up the full DNA) together with a quality score for each nucleotide. FASTQ files are usually stored in compressed form [20]. Depending on coverage and read length, they are typically of size between 10 GB and 70 GB.

2.1.2 Aligned reads

Assembly of raw reads to a full genome is performed via an alignment of the short reads in FASTQ format to a reference genome (e.g., GRCh38 [21]). The alignment information is most commonly stored in SAM/BAM [22] or CRAM [23] files. By applying lossy compression to quality scores, CRAM achieves the smallest file sizes [24]. For example, the 1000 Genomes Project [5] distributes CRAM files with quality scores compressed into 8 bins. Depending on coverage, file sizes vary between 3 GB and 14 GB for full genome alignments [25] (excluding high-coverage alignments).

2.1.3 Variant calls

Variant calls[2] of aligned genomes are usually stored in the *variant call format* (VCF) [26], or its binary counterpart BCF. They represent a difference against the reference genome and are thus an abstract representation in comparison to the aforementioned alignment formats. Coverage and read length do not play a role anymore, as each line in a VCF file represents a called mutation at a unique position of the reference genome.

A human genome has approximately 4 to 5 million variations compared to a reference genome [27]. VCF files that store this information typically require a few hundred megabytes of storage. Usually, a single file per genome, or per chromosome, is produced. This translates to an average storage requirement of about 100 bytes per variation in VCF.

2.1.4 Efficient random access

Efficient random access for SAM, BAM, CRAM, and VCF files is realized by storing the data sorted by chromosome and position and then creating an index map, which stores for a chosen set of positions the corresponding location in the file.

2.2 Data access scenarios

The following scenarios describe different access patterns to genomic data for real-world applications. In particular, the first scenario motivates the solution developed in this work.

2.2.1 Personalized medicine and testing

A typical workflow in personalized medicine requires access to a few mutations in the genome during regular visits to a doctor or hospital. This random access to genomic variant data (e.g., stored in VCF) is roughly required at most once a month for older patients who routinely need to see a doctor. The same is true for ancestry and paternity tests, which primarily access tandem repeat variations.

2.2.2 Cancer

Cancer researchers need access to the full alignments (BAM/CRAM) of healthy and cancer tissue. That is, several full-genome datasets per patient are accessed.

2.2.3 Studies

Pan-genome studies like genome-wide association studies (GWAS) will probably access whole BAM/CRAM files to produce study-specific input files, for each study participant's genome.

3 Application scenario

We consider an application scenario for personalized medicine that involves a patient, a sequencing laboratory, a certified genome database and the patient's doctors and hospitals. The genome of the patient is stored in the certified database and the doctors regularly request parts of the patient's genome (e.g., to identify the best medication and dosage, or to detect possible genomic predispositions). The patient may also want to prove the authenticity of its genomic data towards a third party verifier (e.g., a judge in court in case of a law suit because of a wrong

[2] In the context of genomics, the verb *to call* is often used in the sense *to determine*. E.g. a variant call is a variant determined from the underlying data.

treatment). An overview of the application scenario is depicted in Fig. 1 and the details are described in the following subsections.

3.1 Data generation

When the genome of the patient is sequenced for the first time (e.g., at birth), the sequencing laboratory timestamps and signs the resulting FASTQ files. The laboratory then creates an alignment of those raw reads against some standardized current version of a human reference genome in the CRAM format. Additionally, variants are called and stored in a VCF file. Both the alignment and variants are timestamped and signed by the laboratory.

The data is then transferred to the genome database, who will also conduct future integrity proof updates, without any interaction with the laboratory. From this point on, the laboratory is not involved in any further protocol. The data may be stored in blocks of plain-text, encrypted with a symmetric block-cipher, or secretly shared, since our scheme works on any kind of data blocks. The block cipher would need to be seekable, e.g., AES in counter mode, so that blocks can be decrypted individually. A position in the human genome takes $\lceil \log(3 \cdot 10^9) \rceil = 32$ bits. A pseudorandom permutation could be applied to the 32-bit index of each block to hide the accessed positions. A detailed analysis of the different kinds of block storage are out of scope of this work and we focus on the long-term *integrity* of data blocks.

Note that we do not consider the scenario of re-sequencing a human's genome and the subsequent regeneration of the genomic data. This case is discussed in the outlook Section 6.3.

3.2 Data access

Consider a doctor who wants to identify the best medicine and dosage for their patient, or detect possible genomic predispositions that could influence future treatment. Such a procedure requires to query dozens (and in the future, possibly thousands) of variants from the most recently stored VCF file. A current real-world example is the medicine Warfarin, whose optimal dosage is highly dependent on a patient's genome (cf. motivation Section 1.1). More precisely, eight SNPs[3] were identified that significantly influence a person's dosage dependent response to the drug.

If the data blocks are stored in encrypted form, the patient or a designated doctor or hospital would need to manage the secret keys to assist the decryption of retrieved data blocks.

3.3 Protection goals and threat model

We demand that a solution for holistic genomic data protection achieves the following protection goals:

Integrity. The integrity of the genomic data as produced by the laboratory should be protected. That is, it should be infeasible for an adversarial entity to modify the data at rest or in transit without the modifications being detected at a subsequent data access.

Confidentiality. The confidentiality of genomic data that is not revealed should be protected. An authorized querier should only learn the requested genomic data. That is, a patient or database must be able to prove the integrity of parts of the genomic data without leaking information about the remaining parts of the data.

Authenticity. The database or patient should be able to prove authenticity of the genomic data to a third party verifier.

We allow the querier to be adversarial, i.e., they may try to infer any additional information beyond the authorized parts of the genomic data from their interaction with the database. An adversary within the certified database may have full read and write access to the, possibly encrypted, genomic data blocks. We furthermore consider two cases: if the database provider can be trusted to keep the data confidential, it may be stored in plain text. Otherwise, it should be encrypted or secretly shared. Note that after initial data generation and signing by the laboratory, only the database and requesters are involved in any protocol.

4 Protection scheme

To meet the above stated demands of long-term integrity and confidentiality protection, we have derived a protection scheme, which is described in this chapter.

4.1 Full-retrieval data

Unprocessed raw reads, e.g., stored in compressed FASTQ format, and resulting alignments, e.g., stored in CRAM format, are usually only accessed as a whole and a long-term protection scheme for that use case was proposed in [13]. The scheme presented here in Section 4.3 enhances the integrity protection scheme of [13], so that a large number of small data items can be protected together efficiently.

4.2 Random access data

As opposed to whole-data integrity proofs, our scheme provides random access integrity proofs of genomic variation data on the finest level possible—per position in the reference genome.

We view genomic variation data like VCF/BCF files as a table G, where for each genome position i, $G[i]$ denotes the corresponding variant data entry in G. If there is no mutation at position i, we set $G[i]$ to 0. Note that we do not need to actually store those 0s as the absence of a

[3] two in gene *CYP2C9*, one in gene *GGCX* and five in gene *VKORC1*

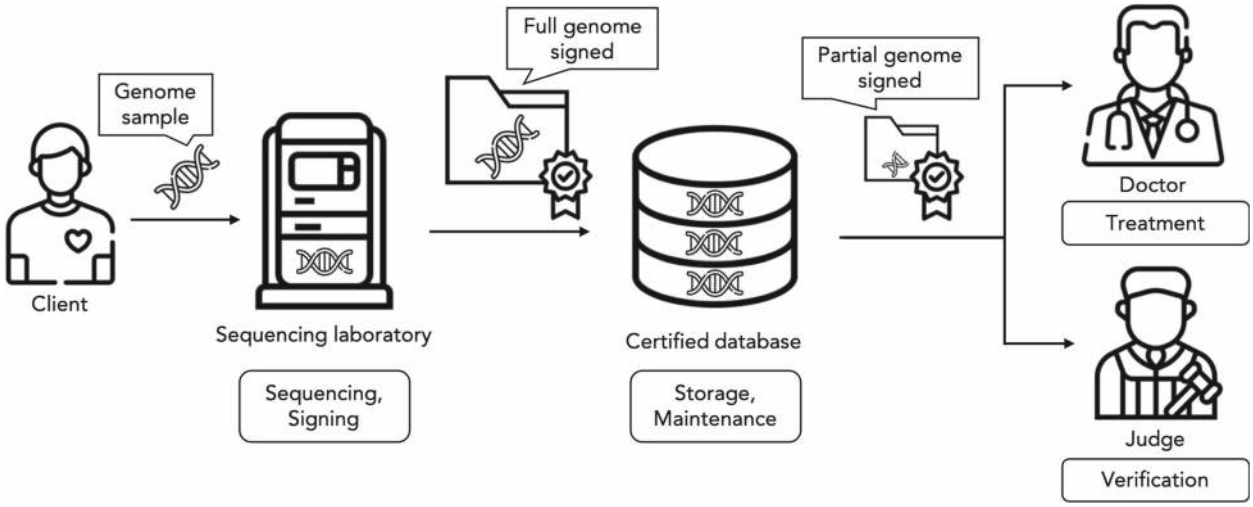

Fig. 1 Overview of the application scenario for our protection scheme

variation implicitly represents a 0. However, the scheme also needs to create commitments for the absence of variants so that absence can also be proven. Since a human genome has about $3 \cdot 10^9$ positions, this is the size of table G and the number of commitments that have to be created, independent of the underlying data format.

For genome data G, generated and signed by a sequencing laboratory, the scheme generates an integrity proof P. The validity period of such a proof is limited in time because the cryptographic primitives used for its generation have a limited validity period. Therefore, the proof is updated regularly. Furthermore, we describe how a partial integrity proof for a subset $G' \subset G$ can be extracted from P, and how such a partial integrity proof is verified. Our scheme thus delivers random access to $G' \subset G$ with integrity proofs while keeping the remaining data $G \setminus G'$ private. We also present a security analysis of the proposed scheme. The scheme uses components of the schemes Lincos [13] and Mops [12]. More information on the used cryptographic primitives (i.e., timestamps, commitments, hashes, and signatures) can be found in the respective publications.

4.3 Scheme description

Our scheme for long-term integrity protection of genomic data provides the algorithms Protect, Update, PartialProof, and Verify. Algorithm Protect generates the initial integrity proof when genomic data is stored. Algorithm Update updates the integrity proof if a used cryptographic primitive (e.g., the hash function) is threatened to become insecure. Algorithm PartialProof generates a partial integrity proof for verification of a subset of the genomic data. Algorithm Verify allows a verifier to verify the integrity of a given genomic dataset using a given partial integrity proof.

4.3.1 *Initial protection*

The initial integrity proof P for sequenced genome data G is generated by the sequencing laboratory using algorithm Protect (Algorithm 1). The algorithm obtains as input genome data G, an information-theoretic hiding commitment algorithm Com [28], a hash algorithm Hash, a signing algorithm Sign, and a time-stamping algorithm TS. The algorithm first uses algorithm Com to generate commitments and decommitments to all entries in G. The commitments can be used as placeholders for the data items, which itself do not leak information, and the decommitments can be used to prove the connection between the commitment and the corresponding data item. Then, it uses the hash algorithm Hash to compute a Merkle hash tree (MHT) [14] for the generated commitment values. The root node of the generated tree is then signed using algorithm Sign and timestamped using the trusted timestamp authority TS [29]. Output of the initial protection algorithm is an integrity proof P which contains the commitments, the decommitments, the MHT, the signature, and the timestamp.

In our algorithm listings we denote by $\mathsf{MHT} : (\mathsf{Hash}, L) \rightarrow T$ an algorithm that on input a hash algorithm Hash and a set of leaf nodes L, outputs a MHT T. Furthermore, we denote the root of a MHT T by $T.r$.

4.3.2 *Protection update*

Timestamps, hash values, and commitments have a limited validity periods, which in turn limits the validity period of the corresponding integrity proof. The *overall validity* of an integrity proof is therefore prolonged regularly by the genome database by running Algorithm 2. The

Algorithm 1: Protect(G, Com, Hash, Sign, TS) $\rightarrow P$

```
/* commit to all entries of G                          */
for i ∈ G do
    (c_{1,i}, d_{1,i}) ← Com(G[i]);
C_1 ←[c_{1,i}]_{i∈G}; D_1 ←[d_{1,i}]_{i∈G};
/* create merkle hash tree for
   commitments, sign root of tree,
   and timestamp signature with tree
   root                                                */
T_1 ← MHT(Hash, C_1); σ ← Sign(T_1.r);
ts_1 ← TS([σ, T_1.r]);
op_1 ← init; P_1 ← (op_1, C_1, D_1, T_1, ts_1); P ←[σ, P_1];
return P;
```

input parameter $\mathsf{op} \in \{\mathtt{upCHT}, \mathtt{upHT}, \mathtt{upT}\}$ determines which primitives are updated; $\mathsf{op} = \mathtt{upCHT}$ updates commitments, hashes, and timestamps; $\mathsf{op} = \mathtt{upHT}$ updates only hashes and timestamps; and $\mathsf{op} = \mathtt{upT}$ updates only timestamps. For $\mathsf{op} = \mathtt{upCHT}$, first new information theoretically hiding commitments are generated. Then, a new MHT T is generated and finally the root of T is timestamped. Output of the update algorithm is an updated integrity proof P'.

In the algorithm listings, we denote by $\mathsf{AuthPath}(T, i) \rightarrow A$ an algorithm that on input MHT T and leaf index i, outputs the authentication path A from leaf node i to root node $T.r$.

4.3.3 Generate partial integrity proof

A data owner may want to create a partial integrity proof P' for a subset $G' \subset G$ such that P' does not reveal any information about $G \setminus G'$. This can be done using Algorithm 3. The algorithm extracts from P all information relevant for proving the integrity of G' and outputs them in form of a partial integrity proof P'. In particular, the partial integrity proof contains the commitments corresponding to the positions contained in G', the corresponding hash tree authentication paths, as well as the corresponding timestamps and the corresponding signature.

4.3.4 Verification

A verifier receives partial genome data G' and a corresponding partial integrity proof P'. Additionally, it uses a trusted verification algorithm Ver and reads the current time t_{n+1}. It then uses Algorithm 4 to verify the integrity of G'.

The trusted verification algorithm Ver is used for verifying the validity of timestamps, hashes, commitments, and signatures. It can be realized by leveraging trusted public key certificates that include verification parameters and validity periods. It must provide the following functionality. If $\mathsf{Ver}_{\mathsf{TS}}(m, \mathsf{ts}; t) = 1$, then ts is a valid timestamp

Algorithm 2:
Update(op, G, P, Com, Hash, Sign, TS) $\rightarrow P'$

```
P → (σ, P_1, ..., P_n);
∀i ∈[n]: P_i → (op_i, C_i, D_i, T_i, ts_i);
if op = upCHT then
    /* if update commitment, then
       create a new commitment to the
       corresponding entry and
       decommitments                                   */
    for i ∈ G do
        (c_{n+1,i}, d_{n+1,i}) ←
        Com([G[i], D_1[i], ..., D_n[i]]);
    C_{n+1} ←[c_{n+1,i}]_{i∈G}; D_{n+1} ←[d_{n+1,i}]_{i∈G};
else
    C_{n+1} ← ⊥; D_{n+1} ← ⊥;
if op ∈ {upCHT, upHT} then
    /* if update hash, then create a
       new hash tree to the
       corresponding commitments and
       authentication paths                            */
    CA(i) :=
    [[C_1[i], ..., C_{n+1}[i]], [AuthPath(T_1, i), ...,
    AuthPath(T_n, i)]];
    T_{n+1} ← MHT(Hash, [CA(i)]_{i∈G});
else
    T_{n+1} ← ⊥;
/* timestamp signature with tree
   roots and timestamps                                */
ts_{n+1} ← TS([σ, (T_1.r, ..., T_{n+1}.r), (ts_1, ..., ts_n)]);
op_{n+1} ← op;
P_{n+1} ← (op_{n+1}, C_{n+1}, D_{n+1}, T_{n+1}, ts_{n+1});
P' ←[σ, P_1, ..., P_{n+1}];
return P';
```

Algorithm 3: PartialProof(G', P) $\rightarrow P'$

```
P → (σ, P_1, ..., P_n);
∀i ∈[n]: P_i → (op_i, C_i, D_i, T_i, ts_i);
for i ∈ {1, ..., n} do
    /* extract the commitments,
       decommitments, and
       authentication paths
       corresponding to G' from P_i                    */
    if C_i ≠ ⊥ ∧ D_i ≠ ⊥ then
        C'_i ←[C_i[j]]_{j∈G'}; D'_i ←[D_i[j]]_{j∈G'};
    if T_i ≠ ⊥ then
        A'_i ←[AuthPath(T_i, j)]_{j∈G'};
    P'_i ← (op_i, C'_i, D'_i, A'_i, T_i.r, ts_i);
P' ←[σ, P'_1, ..., P'_n];
return P';
```

Algorithm 4: Verify$(\mathsf{Ver}, G', P'; t_{n+1}) \rightarrow b$

$P' \rightarrow (\sigma, P'_1, \ldots, P'_n)$;
$\forall i \in [n]\colon P'_i \rightarrow (\mathsf{op}_i, C'_i, D'_i, A'_i, T_i.r, \mathsf{ts}_i)$, $t_i \leftarrow \mathsf{ts}_i.t$;
$b \leftarrow 1$;
for $i \in \{n, \ldots, 2\}$ **do**
  if $op \in \{upCHT, upHT, upT\}$ **then**
    /* check that timestamp ts_i is valid at time $t_{\mathsf{NxTs}}(i)$ for the signature, the previous tree roots, and the previous timestamps */
    $\mathsf{R} := (T_1.r, \ldots, T_i.r)$; $\mathsf{T} := (\mathsf{ts}_1, \ldots, \mathsf{ts}_{i-1})$;
    $b \leftarrow b \wedge \mathsf{Ver}_{\mathsf{TS}}([\sigma, \mathsf{R}, \mathsf{T}], \mathsf{ts}_i; t_{\mathsf{NxTs}}(i))$;
  for $j \in G'$ **do**
    if $op \in \{upCHT, upHT\}$ **then**
      /* check that $A'_i[j]$ is a valid authentication path at time $t_{\mathsf{NxHa}}(i)$ from root $T_i.r$ to the previous commitments and authentication paths */
      $\mathsf{CA}'(i,j) :=$
      $[[C'_1[j], \ldots, C'_i[j]], [A'_1[j], \ldots, A'_{i-1}[j])]$;
      $b \leftarrow$
      $b \wedge \mathsf{Ver}_{\mathsf{MHT}}(\mathsf{CA}'(i,j), A'_i[j], T_i.r; t_{\mathsf{NxHa}}(i))$;
    if $op = upCHT$ **then**
      /* check that $D'_i[j]$ is a valid decommitment at time $t_{\mathsf{NxCo}}(i)$ from commitment $C'_i[j]$ to the corresponding entry and previous decommitments */
      $\mathsf{GD} := [G'[j], D'_1[j], \ldots, D'_{i-1}[j]]$;
      $b \leftarrow b \wedge \mathsf{Ver}_{\mathsf{Com}}(\mathsf{GD}, C'_i[j], D'_i[j]; t_{\mathsf{NxCo}}(i))$;
/* check that the first timestamp is valid for the initial signature and the first tree root */
$b \leftarrow b \wedge \mathsf{Ver}_{\mathsf{TS}}([\sigma, T_1.r], \mathsf{ts}_1; t_{\mathsf{NxTs}}(1))$;
/* check that the signature is valid for the first tree root */
$b \leftarrow b \wedge \mathsf{Ver}_{\mathsf{Sign}}(T_1.r, \sigma; t_1)$;
for $i \in G'$ **do**
  /* check that $A'_1[i]$ is a valid authentication path from the first tree root $T_1.r$ to the corresponding commitment $C'_1[i]$ */
  $b \leftarrow b \wedge \mathsf{Ver}_{\mathsf{MHT}}(C'_1[i], A'_1[i], T_1.r; t_{\mathsf{NxHa}}(1))$;
  /* check that $D'_1[i]$ is a valid decommitment from commitment $C'_1[i]$ to genome entry $G'[i]$ */
  $b \leftarrow b \wedge \mathsf{Ver}_{\mathsf{Com}}(G'[i], C'_1[i], D'_1[i]; t_{\mathsf{NxCo}}(1))$;
return b;

for m at time t, meaning that the cryptographic algorithms used for generating the timestamp are considered secure at time t. The time that the timestamp ts refers to is denoted by ts.t. Hence, $\mathsf{Ver}_{\mathsf{TS}}(m, \mathsf{ts}; t) = 1$ means that it is safe to believe at time t that data m existed at time ts.t. Similarly, $\mathsf{Ver}_{\mathsf{MHT}}(m, a, r; t) = 1$ means that at time t, a is a valid authentication path for m through a hash tree with root r. $\mathsf{Ver}_{\mathsf{Com}}(m, c, d; t) = 1$ means that at time t, d is a valid decommitment from commitment c to message m. $\mathsf{Ver}_{\mathsf{Sign}}(m, \sigma; t) = 1$ means that at time t, σ is a valid signature for message m. We refer to Section 5.2 for more details on how the validity periods of the cryptographic primitives are derived.

We use the following shorthand notations $t_{\mathsf{NxTs}}(i)$, $t_{\mathsf{NxHa}}(i)$, $t_{\mathsf{NxCo}}(i)$ to denote update times with respect to a given partial integrity proof $P' = [\sigma, P'_1, \ldots, P'_n]$. By $t_{\mathsf{NxTs}}(i)$ we denote the time of the next timestamp update after P_i, i.e., $t_{\mathsf{NxTs}}(i) = \min\{\mathsf{ts}_j.t : j > i\}$. Likewise, by $t_{\mathsf{NxHa}}(i)$ we denote the time of the next hash tree update after P_i, and by $t_{\mathsf{NxCo}}(i)$ we denote the time of the next commitment update after P_i.

The verification function Verify of the genome data protection scheme works as follows. It checks whether the integrity proof has been constructed correctly, and whether the cryptographic primitives have been updated before becoming invalid. We refer the reader to the next section (Section 4) for more details on the security of this scheme.

4.4 Security analysis

We now analyze the security of the proposed scheme and argue that it fulfills the requirements described in Section 3.3.

4.4.1 Confidentiality

We observe that a partial integrity proof P' for genome data $G' \subset G$ does not reveal any information about the remaining data $G \setminus G'$ by the following argument. Let $P' = (\sigma, P'_1, \ldots, P'_n)$ be a partial integrity proof for G', where $P'_i = (\mathsf{op}_i, C'_i, D'_i, A'_i, T_i.r, \mathsf{ts}_i)$. We observe that for every $i \in \{1, \ldots, n\}$, op_i, C'_i, and D'_i are independent of $G \setminus G'$ because of the information-theoretic hiding property of the commitments. Furthermore, A'_i contains authentication paths that only depend on information theoretically hiding commitments and thus does not reveal any information as long as the decommitment values are not revealed. Hence, also the tree root $T_i.r$, the timestamp ts_i, and the signature σ are independent of $G \setminus G'$.

4.4.2 Integrity

Next, we show that it is infeasible for an adversary, who cannot break any of the used cryptographic primitives within their validity period, to present a valid partial integrity proof P' for partial genome

data G' if G' has not been originally signed by the laboratory.

For our security analysis, we consider an adversary that can potentially become computationally more powerful over time and use methods developed in [30–32] for arguing about the knowledge of an adversary at an earlier point in time. For this, we require that the timestamp, commitment, and hash algorithms chosen by the user are *extractable*. Thereby, we are able to show that if an adversary presents a valid integrity proof, then the signed data together with the signature must have been known at a point when the corresponding signature scheme was considered valid. If the signature is valid for the data, then it follows that the data is authentic.

Here, we use the following notation to express the knowledge of the adversary. For any data m and time t, we write $m \in \mathcal{K}[\,t\,]$ to denote that the adversary knows m at time t. We remark that for any $t < t'$, $m \in \mathcal{K}[\,t\,]$ implies $m \in \mathcal{K}[\,t'\,]$.

Extractable timestamping [30, 32] guarantees that if at some time t, a timestamp ts and message m are known and ts is considered valid for m at time t, then m must have been known at time $\mathsf{ts}.t$, or in the notation introduced above:

$$(m, \mathsf{ts}) \in \mathcal{K}[t] \wedge \mathsf{Ver}_{\mathsf{TS}}(m, \mathsf{ts}; t) \implies m \in \mathcal{K}[\,\mathsf{ts}.t\,] \ . \quad (1)$$

Moreover, extractable commitments [31] guarantee that if a commitment value is known at time t, and a message m and a valid decommitment value are known at a later time $t' > t$, then the message m was already known at commitment time t, i.e.:

$$c \in \mathcal{K}[t] \wedge (m, d) \in \mathcal{K}[t'] \wedge \mathsf{Ver}_{\mathsf{Com}}(m, c, d; t') \implies m \in \mathcal{K}[t] \ . \quad (2)$$

Extractable hash trees [32] provide similar guarantees, i.e., for any hash tree root value r, time t, message m, hash tree authentication path a, and times t, t':

$$r \in \mathcal{K}[t] \wedge (m, a) \in \mathcal{K}[t'] \wedge \mathsf{Ver}_{\mathsf{MHT}}(m, a, r; t') \implies m \in \mathcal{K}[\,t\,] \ . \quad (3)$$

Furthermore, we know that if a signature σ and a message m are known at some time t, and σ is considered valid for m at time t, then by the existential unforgeability of the signatures it follows that m is authentically signed [30, 33]:

$$(m, \sigma) \in \mathcal{K}[\,t\,] \wedge \mathsf{Ver}_{\mathsf{Sign}}(m, \sigma; t) \implies m \text{ is authentic} \ . \quad (4)$$

Finally, it is known that signing the root of a Merkle tree preserves the integrity of the leafs. Furthermore, if the leafs are commitments, the authenticity of the committed messages is preserved. That is, for any hash tree root value r, signature σ, commitment c, hash tree authentication path a, message m, decommitment d, and times t, t', t'':

$$\begin{aligned} &(r, \sigma) \in \mathcal{K}[\,t\,] \wedge \mathsf{Ver}_{\mathsf{Sign}}(r, \sigma; t) \wedge \\ &(c, a) \in \mathcal{K}[\,t'\,] \wedge \mathsf{Ver}_{\mathsf{MHT}}\left(c, a, r; t'\right) \wedge \\ &(m, d) \in \mathcal{K}[\,t''\,] \wedge \mathsf{Ver}_{\mathsf{Com}}\left(m, c, d; t''\right) \\ &\qquad\qquad \implies m \text{ is authentic} \ . \end{aligned} \quad (5)$$

We now show that it is infeasible to produce a valid integrity proof for genome data that is not authentically signed. Assume an adversary outputs (G', P') at some point in time t_{n+1} and let Ver be a verification function trusted by the verifier. We show that if P' is a valid partial integrity proof for data G' (i.e., $\mathsf{Verify}(\mathsf{Ver}, G', P') = 1$), then the signature σ for G' is not a forgery.

Let $P' = (\sigma, P'_1, \ldots, P'_n)$, where $P'_i = (\mathsf{op}_i, C'_i, D'_i, A'_i, T_i.r, \mathsf{ts}_i)$. Define $P''_i = (\sigma, P'_1, \ldots, P'_i)$ and $t_i = \mathsf{ts}_i.t$. In the following, we show recursively for $i \in [\,n, \ldots, 1]$, that given $\mathsf{Verify}(\mathsf{Ver}, G', P') = 1$, statement $\mathsf{St}(i) = \langle (G', P''_i) \in \mathcal{K}[\,t_{i+1}\,] \rangle$ holds.

We observe that $\mathsf{St}(n)$ is trivially true because the adversary presents valid (G', P') at t_{n+1} by assumption. Next, we show that assuming $\mathsf{St}(i)$ holds, then also $\mathsf{St}(i-1)$ holds. Given $\mathsf{St}(i)$, we observe that by $\mathsf{Ver}_{\mathsf{TS}}([\,\sigma, (T_1.r, \ldots, T_i.r), (\mathsf{ts}_1, \ldots, \mathsf{ts}_{i-1})\,], \mathsf{ts}_i; t_{\mathsf{NxTs}}(i)) = 1$ and (1), we have $[\,\sigma, (T_1.r, \ldots, T_i.r), (\mathsf{ts}_1, \ldots, \mathsf{ts}_{i-1})\,] \in \mathcal{K}[\,t_i\,]$. Furthermore, by

$$\mathsf{Ver}_{\mathsf{MHT}}(\mathsf{CA}'(i,j), A'_i[j]\,, T_i.r; t_{\mathsf{NxHa}}(i)) = 1$$

and (3), we have $\mathsf{CA}'(i,j) \in \mathcal{K}[t_i]$ for every $j \in G'$. Finally, by

$$\mathsf{Ver}_{\mathsf{Com}}([\,G'[j]\,, D'_1[j]\,, \ldots, D'_{i-1}[j]\,]\,, C'_i[j]\,, D'_i[j]\,; t_{\mathsf{NxCo}}(i)) = 1$$

and (2) we have $[\,G'[j]\,, D'_1[j]\,, \ldots, D'_{i-1}[j]\,] \in \mathcal{K}[\,t_i\,]$ for every $j \in G'$. Combined, we obtain $(G', P''_{i-1}) \in \mathcal{K}[\,t_i\,]$, which means that $\mathsf{St}(i-1)$ holds.

We observe that $\mathsf{St}(1)$,$\mathsf{Ver}_{\mathsf{TS}}([\,\sigma, T_1.r\,], \mathsf{ts}_1; t_{\mathsf{NxTs}}(1)) = 1$, and (1) implies that $[\,\sigma, T_1.r] \in \mathcal{K}[\,t_1\,]$. Furthermore, by $\mathsf{Ver}_{\mathsf{Sign}}(T_1.r, \sigma; t_1) = 1$ and (4), we obtain that σ is genuine for $T_1.r$. Finally, we observe that for every $i \in G$, $\mathsf{Ver}_{\mathsf{MHT}}(C'_1[\,i\,]\,, A'_1[\,i\,]\,, T_1.r; t_{\mathsf{NxHa}}(1)) = 1$, $\mathsf{Ver}_{\mathsf{Com}}(G[\,i\,]\,, C'_1[\,i\,]\,, D'_1[\,i\,]\,; t_{\mathsf{NxCo}}(1)) = 1$, and we obtain by (5) that σ is a genuine signature for G'.

5 Performance evaluation

In order to illustrate the applicability of our scheme to today's challenges in bioinformatics and medicinal informatics, in the following, we evaluate the performance of the scheme described in Section 4.3 in this chapter.

5.1 Protection scenario

We focus on the following situation: a human genome is sequenced and protected for a human lifespan of 100 years. The scenario starts with sequencing the genomic data G in 2019 and creating an integrity proof P. Here, we are only interested in the protection of a single-genome dataset, that is, we do not consider additional genomic data generated due to resequencing.

We assume that the lifetime of signature-based timestamps is based on the lifetime of the corresponding public key certificate, which is typically 2 years. For our commitments and hash functions, we assume a longer validity period of 10 years, as they are not dependent on secret parameters which may leak over time. The integrity protection update schedule is summarized in Table 1.

5.2 Instantiation of cryptographic primitives

For our analysis, we instantiatiate the cryptographic algorithms of our protection scheme as follows. As hash functions, we use the ones from the SHA-2 hash function family [34], which are extractable if modeled as a random oracle [35]. As timestamp schemes, we employ signature-based timestamps [29] based on the XMSS signature scheme [36], which is a hash-based signature scheme conjectured secure against quantum computers. As commitment schemes, we use the construction proposed by Halevi and Micali [37], which uses a hash function and is extractable if the hash function is extractable [35]. When generating Merkle hash trees, we use an optimization where we take commitments to the data directly as the leafs of the hash trees in order to save one hash tree level. Cryptographic parameters are chosen based on the recommendations by Lenstra and Verheul [38, 39]. The chosen parameters are summarized in Table 2.

5.3 Evaluation results

We show the storage space consumed by an integrity proof P corresponding to genome data G containing $3 \cdot 10^9$ entries, which is roughly the number of nucleotides of a human genome. We also show the storage space required by a partial integrity proof P' corresponding to partial genome data G' containing $1,100$ or 10^5 entries. As the Warfarin example shows, current personalized medicine applications would only be concerned with a few dozen entries. To take future medical scientific advances into accounts, we choose to evaluate partial proofs of size up to 10^5. We also measure the time it takes to generate the initial integrity proof, to update an integrity proof, and to verify a partial integrity proof.

We remark that we measure the space consumed in terms of the size of the commitments, timestamps, and hashes to be stored. Likewise, we measure the time consumed for generating and updating an integrity proof in terms of the computation time required to generate the commitments, timestamps, and hashes. For the verification time, we sum up the time required for verification of the individual cryptographic elements. The time and sizes required for hashing, signing, and committing to a message of size 128 B are shown in Table 3. This is an upper bound on the average storage requirement for a variation in VCF, cf. 2.1.3. For XMSS, the height parameter is chosen as 10. The timings were taken on a computer with a 2.9 GHz Intel Core i5 CPU and 8 GB RAM running Java.

Table 1 Schedule for updating the integrity proof

Update method	Update time
Initial protection	Once in the beginning (i.e., in 2019)
Update Ts	Every 2 years (i.e., 2021, 2023, …)
Update ComHashTs	Every 10 years (i.e., 2029, 2039, …)

Table 2 Parameter selection based on Lenstra and Verheul [38, 39]

Validity	Hashes	Signatures	Commitments
2066	SHA-224	XMSS-256	HM-224
2090	SHA-256	XMSS-256	HM-256
2186	SHA-384	XMSS-512	HM-384

5.3.1 *Size of integrity proof*

Figure 2 shows the storage space over time required for storing the full integrity proof. The size of the initial integrity proof in year 2019 is 391 GB. The size only increases minimally when updating the timestamps. When updating the commitment, hashes, and timestamps together, the size grows significantly. After the first such update, the size of the integrity proof is 782 GB. After 100 years, the size of the integrity proof is 5309 GB. Comparing this to the size of an average 600 MB VCF file shows that after 100 years, the integrity proof is roughly 10,000 times larger than the actual variant data.

Table 3 Space and time required for storing, generating, and verifying, hashes (SHA), commitments (HM), and signatures (XMSS)

Primitive	Size	Generation	Verification
SHA-224	224 bit	1.33 µs	1.3 µs
SHA-256	256 bit	1.29 µs	1.29 µs
SHA-384	384 bit	1.43 µs	1.43 µs
HM-224	896 bit	5.68 µs	3.20 µs
HM-256	1024 bit	6.16 µs	3.11 µs
HM-384	1536 bit	6.08 µs	5.71 µs
XMSS-256	20000 bit	17.00 ms	2.43 ms
XMSS-512	72736 bit	46.37 ms	7.67 ms

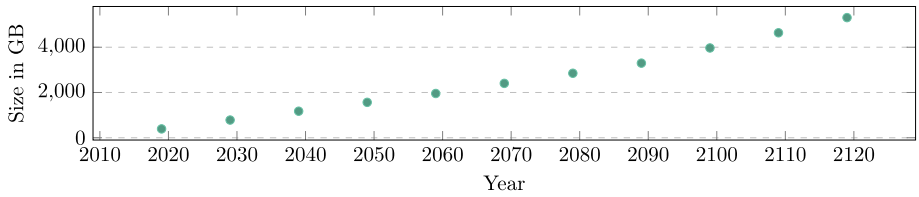

Fig. 2 Size of integrity proof for whole-genome data G

For $|G'| \in \{1, 100, 10^5\}$, Fig. 3 shows the size of a partial integrity proof P' for G' over time. As the number of elements covered by the partial integrity proof is considerably smaller, also its size is much smaller compared to the full integrity proof. For the largest partial proof parameter $|G'| = 10^5$, the size of P' ranges from 9.62 MB in 2019 to 130.67 MB in 2118, growing roughly linearly. For a fixed point in time and $|G'| \geq 100$, the size also grows roughly proportionally to $|G'|$.

5.3.2 Cost projection for integrity proof storage

Although it is impossible to predict long-term storage costs, we will nevertheless try to give a rough cost projection into the future. We examined two sources of historical hard disk prices and found that between 1980 and 2010 HDD storage costs per gigabyte roughly halved every 14 months [40], leading to a cost reduction by a factor of 10 roughly every 4 years. Then since 2009, this rapid decline in storage costs has slowed down, only showing a reduction in storage costs by a factor of 4–5 over the last 10 years [4]. However, new technologies like HAMR and MAMR [41] are on the horizon, which are expected to show HDDs of size 4 TB by 2025, according to Western Digital [42].

We calculated yearly expenses for the storage of a full integrity proof, considering three cost-per-storage projection scenarios: no change in storage costs and cost reductions by rates of $R = 2$ and 4 per 10 years. In view of past developments, we deem those rates conservative. We furthermore assumed that HDDs have to be replaced every 5 years and started with storage costs of $15 per TB[4].

The results can be seen in Fig. 4. The first year of storage costs 0.391 TB · $15/5 = $1.15. From then on, while the amount of data increases, thanks to the exponential decline in costs, the overall yearly costs decline sharply for $R = 2$ and 4. For $R = 1$, it is proportional to the amount of storage (Fig. 2). Even in the unrealistic case that storage costs do not drop over 100 years, the costs still only grow to $15.55 yearly in 2190. For $R = 2$, the costs decline to 22 cents in 2069 and 2 cents in 2119. For $R = 4$, the costs reach 1 cent in 2069 and after that are well below 1 cent. To be fair, in reality, this data would probably be stored redundantly to protect against data loss, so the actual costs would need to be multiplied by the amount of redundancy.

[4]On 28 July 2019, there were available a 4 TB HDD for $64 and 6 TB for $90 at the price comparison website newegg.com.

5.3.3 Computation time

The time required for the initial integrity proof generation in year 2019 is 5.85 h, for G with $|G| = 3 \cdot 10^9$. Figure 5 shows the time required for performing a commitment, timestamp, and hash update of the integrity proof. Computation time for each full update every 10 years is comparable to the computation time of the initial integrity proof. However, it should be considered that with more powerful computers in the future these update times can be expected to decrease significantly.

Figure 6 shows the time required for verifying a partial integrity proof P' corresponding to partial genome data G' with $|G'| \in \{1, 100, 10^5\}$. The computation time required for verification of P' of the largest partial size, generated in 2019, is 0.46 s. For P' generated in 2119 the verification time is 5.37 s.

5.4 Comparison with [13]

We briefly compare the performance of our scheme with performance of the integrity protection scheme of [13]. We observe that for protecting a dataset with [13], for each data item, a separate commitment, decommitment, signature, and timestamp need to be generated and stored. This results in an initial proof generation time of 28338 h (or 3.2 years) and a size of 14283 GB. In comparison, our scheme generates the initial proof in 5.9 h and the proof has a size of 391 GB.

6 Conclusion and future work

6.1 Conclusion

We have evaluated a scenario where the integrity of genomic data is protected over a time span of 100 years. We first described a scenario in which genomic data is generated and accessed for medical treatment and analyzed the protection requirements. Next, we proposed a long-term integrity protection scheme suitable for this scenario. Then, we analyzed the performance of the proposed scheme for the given scenario. We estimate that long-term integrity protection of a genome database

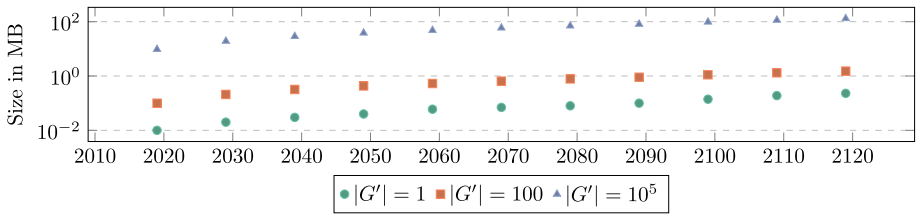

Fig. 3 Size of partial integrity proof for partial genome data G'

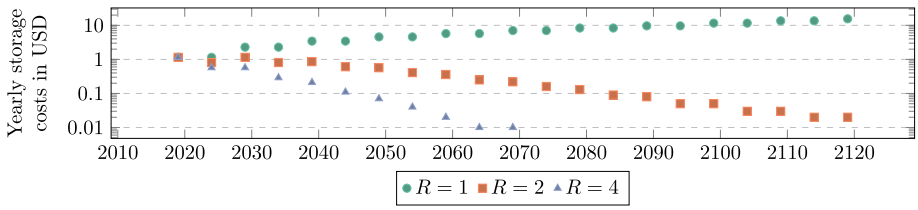

Fig. 4 Projected yearly storage costs of integrity proofs. The calculation starts with initially $ 15 per TB, then reduces the costs per TB by a rate of $R = 1$, 2 and 4 per 10 years. HDDs are assumed to be replaced every 5 years

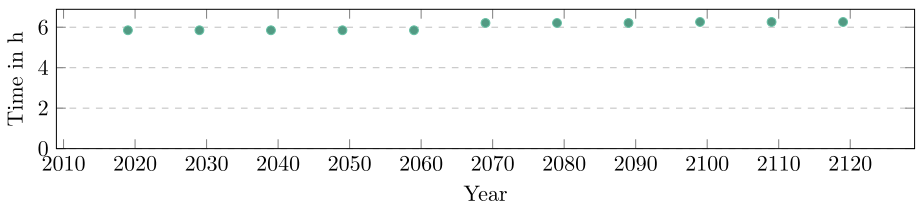

Fig. 5 Computation time for updating integrity proof for whole-genome data G

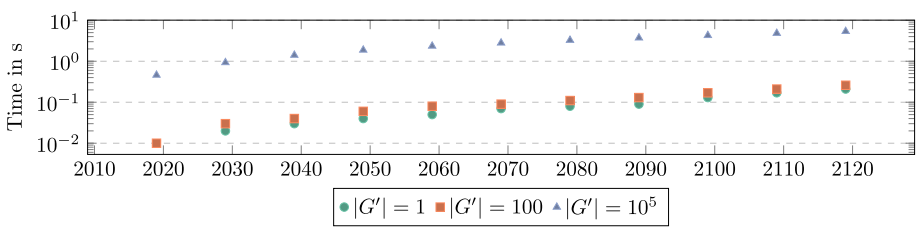

Fig. 6 Computation time for verifying partial integrity proof for partial genome data G'

with $3 \cdot 10^9$ independently verifiable entries for 100 years requires a storage space of approximately up to 5.3 TB in 2119. We estimated the yearly storage costs of the integrity proof to start at $\$1.15$ and, depending on the assumed reduction in general storage prices, reach $\$15.55$ in 2119 (no reduction) or fall to negligible levels for reduction rates of $R = 2$ or 4 per 10 years. We therefore deem this 10,000-fold increase in storage compared to the actual variation data as acceptable, considering the possible dangers of unprotected integrity and low actual yearly costs. It takes approximately 5.9 h to generate the initial integrity proof and up to 6.3 h to update it when used cryptographic primitives must be replaced. The size of a partial integrity proof for a genome subset of size 10^5, assumed to be a future-proof choice for personalized medicine, after 100 years is approximately 130 MB and the verification takes approximately 5 s. The computation times can be expected to decrease in the future when more powerful computers will be available.

6.2 Confidentiality

In Section 3.1 we explain that our scheme works on any data that is stored in blocks, also in encrypted form. If the database is an untrusted cloud, it obviously makes sense to not store the data in plain text. To achieve *certain* long-term *confidentiality*, only information-theoretically secure methods such as secret sharing should be used. This stems from the simple fact that once data is obtained in encrypted form by an adversary, they only have to wait until the encryption is broken in the future. We leave it as future work to combine Oblivious RAM techniques [43] with our long-term integrity scheme to achieve better query pattern hiding.

6.3 Genome re-sequencing

Our scenario only considered a single production of genomic data, e.g., at birth. After that, only updated integrity proofs were generated. However, it is foreseeable that advanced sequencing technology will be used to re-sequence a human's genome periodically, e.g., every 10 years, once personalized medicine has gone mainstream. Additionally, it is already becoming standard procedure to sequence somatic cancer tissue of patients with certain types of cancers [44, 45]. More cancer types will follow to be subjected to genetic analysis. Furthermore, once cancer is detected, a re-sequencing of cancer tissue every few weeks seems plausible in the future, to observe the development of the cancer's genome.

Every (re)sequencing of either healthy or cancer tissue follows the alignment and variant calling procedures, so FASTQ, CRAM, and BCF files, or future enhanced versions thereof, are produced. How to provide long-term protection of this additional data, in combination with existing data, will be investigated in future work.

It could also become feasible to redo the alignment and variant calling step, once a new reference genome might be agreed upon on a (super)national health governance level.

An open question is whether alignments against obsolete reference genomes could be safely deleted, since they could still be reproduced from the raw reads. This, however, is solely determined by medical needs and legislative issues (liability and regulatory mechanisms).

6.4 Omics data

Other data apart from the genome itself, typically summarized under the term *omics*, like genome methylation pattern sequencing [46, 47] is receiving increasing attention in the area of precision medicine [48]. For these advanced but forseeable areas, an all-encompassing data integrity solution needs to combine integrity proofs of newly generated and updated data, taken at different time intervals. Such a full solution, however, is beyond the scope of the present study and will be pursued in the future.

Acknowledgements

Not applicable.

Authors' contributions

JB, KH, and SK played a major role in sparking the idea for this research and supervising it. KH and SS contributed with their background knowledge on genomic data formats and protection requirements. MG and SS together designed the protection scheme. MG evaluated the performance of the scheme. MG and SS are major contributors in writing the manuscript. KH and SK supported the revision of the manuscript. All authors read and approved the final manuscript.

Funding

The research reported in this paper has been supported by the German Federal Ministry of Education and Research (BMBF) [and by the Hessian Ministry of Science and the Arts] within CRISP (www.crisp-da.de), as well as by collaborations within the BMBF-funded HiGHmed consortium. This work has been co-funded by the DFG as part of project S6 within the CRC 1119 CROSSING.

Availability of data and materials

The timings presented in Section 5.3 were obtained by running implementations of the respective cryptographic algorithms. The source code for the timing measurements is available from the corresponding author on reasonable request.

Competing interests

The authors declare that they have no competing interests.

Received: 2 April 2019 Accepted: 30 September 2019
Published online: 29 October 2019

References

1. D. S. C. Davies, Chief Medical Officer annual report 2016: Generation Genome - GOV.UK. Technical Report 8, Department of Health (July 2017). https://www.gov.uk/government/publications/chief-medical-officer-annual-report-2016-generation-genome. Accessed 4 July 2017
2. M. Naveed, E. Ayday, E. W. Clayton, J. Fellay, C. A. Gunter, J.-P. Hubaux, B. A. Malin, X. Wang, Privacy in the Genomic Era. ACM Comput. Surv. **48**(1), 6–1644 (2015). https://doi.org/10.1145/2767007. Accessed 25 May 2016
3. M. Akgün, A. O. Bayrak, B. Ozer, M. Ş. Sağıroğlu, Privacy preserving processing of genomic data: a survey. J. Biomed. Inf. **56**, 103–111 (2015). https://doi.org/10.1016/j.jbi.2015.05.022. Accessed 28 July 2016

4. T. Dugan, X. Zou, in *2016 IEEE First International Conference on Connected Health: Applications, Systems and Engineering Technologies (CHASE)*. A Survey of Secure Multiparty Computation Protocols for Privacy Preserving Genetic Tests, (2016), pp. 173–182. https://doi.org/10.1109/CHASE.2016.71
5. M. Caulfield, J. Davies, M. Dennys, L. Elbahy, T. Fowler, S. Hill, T. Hubbard, L. Jostins, N. Maltby, J. Mahon-Pearson, G. McVean, K. Nevin-Ridley, M. Parker, V. Parry, A. Rendon, L. Riley, C. Turnbull, K. Woods, The 100,000 Genomes Project Protocol (2017). https://doi.org/10.6084/m9.figshare.4530893.v2. https://figshare.com/articles/GenomicEnglandProtocol_pdf/4530893
6. M. Wadelius, L. Y. Chen, K. Downes, J. Ghori, S. Hunt, N. Eriksson, O. Wallerman, H. Melhus, C. Wadelius, D. Bentley, P. Deloukas, Common VKORC1 and GGCX polymorphisms associated with warfarin dose. Pharmacogenomics J. **5**(4), 262–270 (2005). https://doi.org/10.1038/sj.tpj.6500313. Accessed 22 June 2017
7. T. I. W. P. Consortium, Estimation of the Warfarin Dose with Clinical and Pharmacogenetic Data. New Engl. J. Med. **360**(8), 753–764 (2009). https://doi.org/10.1056/NEJMoa0809329. Accessed 26 July 2017
8. J. A. Johnson, L. H. Cavallari, Warfarin pharmacogenetics. Trends Cardiovasc. Med. **25**(1), 33–41 (2015). https://doi.org/10.1016/j.tcm.2014.09.001. Accessed 26 July 2017
9. P. Shor, Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM J. Comput. **26**(5), 1484–1509 (1997). https://doi.org/10.1137/S0097539795293172. https://doi.org/10.1137/S0097539795293172
10. X. Wang, D. Feng, X. Lai, H. Yu, Collisions for hash functions MD4, MD5, HAVAL-128 and RIPEMD (2004). Cryptology ePrint Archive, Report 2004/199. https://eprint.iacr.org/2004/199
11. M. Vigil, J. Buchmann, D. Cabarcas, C. Weinert, A. Wiesmaier, Integrity, authenticity, non-repudiation, and proof of existence for long-term archiving: A survey. Comput. Secur. **50**, 16–32 (2015)
12. C. Weinert, D. Demirel, M. Vigil, M. Geihs, J. Buchmann, in *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security. ASIA CCS '17*. Mops: A modular protection scheme for long-term storage (ACM, New York, 2017), pp. 436–448
13. J. Braun, J. Buchmann, D. Demirel, M. Geihs, M. Fujiwara, S. Moriai, M. Sasaki, A. Waseda, in *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security. ASIA CCS '17*. Lincos: A storage system providing long-term integrity, authenticity, and confidentiality (ACM, New York, 2017), pp. 461–468
14. R. C. Merkle, in *Advances in Cryptology — CRYPTO' 89 Proceedings*, ed. by G. Brassard. A certified digital signature (Springer, New York, 1990), pp. 218–238
15. C. M. Swanson, D. R. Stinson, in *Information Theoretic Security*, ed. by S. Fehr. Unconditionally secure signature schemes revisited (Springer, Berlin, 2011), pp. 100–116
16. T. Bradley, X. Ding, G. Tsudik, Genomic Security (Lest We Forget). IEEE Secur. Priv. **15**(5), 38–46 (2017). https://doi.org/10.1109/MSP.2017.3681055. Accessed 9 May 2018
17. E. Gaetani, L. Aniello, R. Baldoni, F. Lombardi, A. Margheri, V. Sassone, in *Italian Conference on Cybersecurity (20/01/17)*. Blockchain-based database to ensure data integrity in cloud computing environments, (2017). http://ceur-ws.org/Vol-1816/paper-15.pdf. Accessed 20 July 2019
18. C. Esposito, A. D. Santis, G. Tortora, H. Chang, K. R. Choo, Blockchain: A Panacea for Healthcare Cloud-Based Data Security and Privacy? IEEE Cloud Comput. **5**(1), 31–37 (2018). https://doi.org/10.1109/MCC.2018.011791712
19. R. Bansarkhani, M. Geihs, J. Buchmann, PQChain: Strategic design decisions for distributed ledger technologies against future threats. IEEE Secur. Priv. **16**(04), 57–65 (2018). https://doi.org/10.1109/MSP.2018.3111246
20. J. K. Bonfield, M. V. Mahoney, Compression of FASTQ and SAM format sequencing data. PLoS ONE. **8**(3), 59190 (2013). https://doi.org/10.1371/journal.pone.0059190. Accessed 21 June 2017
21. The Genome Reference Consortium, The Genome Reference Consortium. http://genomereference.org/. Accessed 31 July 2017
22. H. Li, B. Handsaker, A. Wysoker, T. Fennell, J. Ruan, N. Homer, G. Marth, G. Abecasis, R. Durbin, The sequence alignment/map format and SAMtools. Bioinformatics. **25**(16), 2078–2079 (2009). https://doi.org/10.1093/bioinformatics/btp352. Accessed 20 Apr 2017
23. M. H.-Y. Fritz, R. Leinonen, G. Cochrane, E. Birney, Efficient storage of high throughput DNA sequencing data using reference-based compression. Genome Res. **21**(5), 734–740 (2011). https://doi.org/10.1101/gr.114819.110. Accessed 21 June 2017
24. S. Deorowicz, S. Grabowski, Data compression for sequencing data. Algoritm. Mol. Biol. **8**, 25 (2013). https://doi.org/10.1186/1748-7188-8-25. Accessed 15 June 2017
25. 1000 Genomes Project, IGSR: The International Genome Sample Resource. http://www.internationalgenome.org/ Accessed 31 July 2017
26. P. Danecek, A. Auton, G. Abecasis, C. A. Albers, E. Banks, M. A. DePristo, R. E. Handsaker, G. Lunter, G. T. Marth, S. T. Sherry, G. McVean, R. Durbin, The variant call format and VCFtools. Bioinformatics. **27**(15), 2156–2158 (2011). https://doi.org/10.1093/bioinformatics/btr330. Accessed 20 Apr 2017
27. The 1000 Genomes Project Consortium, A global reference for human genetic variation. Nature. **526**(7571), 68–74 (2015). https://doi.org/10.1038/nature15393. Accessed 31 July 2017-07-31
28. S. Halevi, S. Micali, in *Advances in Cryptology — CRYPTO '96*, ed. by N. Koblitz. Practical and provably-secure commitment schemes from collision-free hashing (Springer, Berlin, 1996), pp. 201–215
29. C. Adams, P. Cain, D. Pinkas, R. Zuccherato, RFC 3161: Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP) (2001). https://doi.org/10.17487/rfc3161
30. M. Geihs, D. Demirel, J. Buchmann, in *2016 14th Annual Conference on Privacy, Security and Trust (PST)*. A security analysis of techniques for long-term integrity protection, (2016). https://doi.org/10.1109/pst.2016.7906995
31. A. Buldas, M. Geihs, J. Buchmann, in *Information Security and Privacy: 22nd Australasian Conference, ACISP 2017, Auckland, New Zealand, July 3–5, 2017, Proceedings, Part I*, ed. by J. Pieprzyk, S. Suriadi. Long-term secure commitments via extractable-binding commitments (Springer, Cham, 2017), pp. 65–81
32. A. Buldas, M. Geihs, J. Buchmann, in *Provable Security*, ed. by T. Okamoto, Y. Yu, M. H. Au, and Y. Li. Long-term secure time-stamping using preimage-aware hash functions (Springer, Cham, 2017), pp. 251–260
33. S. Goldwasser, S. Micali, R. Rivest, A digital signature scheme secure against adaptive chosen-message attacks. SIAM J. Comput. **17**(2), 281–308 (1988). https://doi.org/10.1137/0217017. https://doi.org/10.1137/0217017
34. National Institute of Standards and Technology (NIST), FIPS PUB 180-4: Secure hash standard (SHS) (2015)
35. M. Geihs, Long-term protection of integrity and confidentiality ? security foundations and system constructions. PhD thesis, Technische Universität, Darmstadt (2018). http://tubiblio.ulb.tu-darmstadt.de/108203/
36. J. Buchmann, E. Dahmen, A. Hülsing, in *Post-Quantum Cryptography: 4th International Workshop, PQCrypto 2011, Taipei, Taiwan, November 29 – December 2, 2011. Proceedings*, ed. by B.-Y. Yang. Xmss - a practical forward secure signature scheme based on minimal security assumptions (Springer, Berlin, 2011), pp. 117–129
37. S. Halevi, S. Micali, in *Advances in Cryptology — CRYPTO '96: 16th Annual International Cryptology Conference Santa Barbara, California, USA August 18–22, 1996 Proceedings*, ed. by N. Koblitz. Practical and provably-secure commitment schemes from collision-free hashing (Springer, Berlin, 1996), pp. 201–215
38. A. K. Lenstra, E. R. Verheul, Selecting cryptographic key sizes. J. Cryptol. **14**(4), 255–293 (2001)
39. A. K. Lenstra, in *Bidgoli, Hossein. Handbook of Information Security, Information Warfare, Social, Legal, and International Issues and Security Foundations. Vol. 2*. Key lengths, (Wiley, 2006), pp. 617–635
40. M. Komorowski, A History of Storage Cost (2009). https://www.mkomo.com/cost-per-gigabyte. Accessed 28 July 2019
41. Y. Shiroishi, K. Fukuda, I. Tagawa, H. Iwasaki, S. Takenoiri, H. Tanaka, H. Mutoh, N. Yoshikawa, Future Options for HDD Storage. IEEE Trans. Magn. **45**(10), 3816–3822 (2009). https://doi.org/10.1109/TMAG.2009.2024879
42. T. S. Ganesh, Western Digital Stuns Storage Industry with MAMR Breakthrough for Next-Gen HDDs (2017). https://www.anandtech.com/show/11925/western-digital-stuns-storage-industry-with-mamr-breakthrough-for-nextgen-hdds. Accessed 28 July 2019
43. E. Stefanov, M. van Dijk, E. Shi, C. Fletcher, L. Ren, X. Yu, S. Devadas, in *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security. CCS '13*. Path ORAM: An Extremely Simple Oblivious RAM Protocol (ACM, New York, 2013), pp. 299–310. https://doi.org/10.1145/2508859.2516660
44. C. Tan, X. Du, KRAS mutation testing in metastatic colorectal cancer. World J. Gastroenterol. : WJG. **18**(37), 5171–5180 (2012). https://doi.org/10.3748/wjg.v18.i37.5171. Accessed 28 July 2017

45. S. Kummar, P. M. Williams, C.-J. Lih, E. C. Polley, A. P. Chen, L. V. Rubinstein, Y. Zhao, R. M. Simon, B. A. Conley, J. H. Doroshow, Application of molecular profiling in clinical trials for advanced metastatic cancers. JNCI: J. Natl. Cancer Inst. **107**(4) (2015). https://doi.org/10.1093/jnci/djv003. Accessed 28 July 2017
46. B. E. Bernstein, A. Meissner, E. S. Lander, The mammalian epigenome. Cell. **128**(4), 669–681 (2007). https://doi.org/10.1016/j.cell.2007.01.033
47. P. A. Jones, T. K. Archer, S. B. Baylin, S. Beck, S. Berger, B. E. Bernstein, J. D. Carpten, S. J. Clark, J. F. Costello, R. W. Doerge, M. Esteller, A. P. Feinberg, T. R. Gingeras, J. M. Greally, S. Henikoff, J. G. Herman, L. Jackson-Grusby, T. Jenuwein, R. L. Jirtle, Y.-J. Kim, P. W. Laird, B. Lim, R. Martienssen, K. Polyak, H. Stunnenberg, T. D. Tlsty, B. Tycko, T. Ushijima, J. Zhu, V. Pirrotta, C. D. Allis, S. C. Elgin, J. Rine, C. Wu, Moving AHEAD with an international human epigenome project. Nature. **454**(7205), 711–715 (2008). https://doi.org/10.1038/454711a. Accessed 1 Aug 2017
48. I. S. Chan, G. S. Ginsburg, Personalized Medicine: Progress and Promise. Ann. Rev. Genom. Hum. Genet. **12**(1), 217–244 (2011). https://doi.org/10.1146/annurev-genom-082410-101446. Accessed 1 Aug 2017

Publisher's Note