Securing Genomic Computations for Research and Clinical Decision Support

W. Knox Carey*, Nevenka Dimitrova†, Bart Grantham*,
Vartika Agrawal†, Jarl Nilsson*, Ray Krasinski†

* Genecloud, 920 Stewart Drive, Sunnyvale, CA 94085
† Philips Research North America, 345 Scarborough Road, Briarcliff Manor, NY 10510

Abstract

The use of genomic information holds great promise in understanding, diagnosing, and treating human disease. This promise, however, is tempered by an equally significant threat to individual privacy. Recent work has shown that even anonymized genomic data is vulnerable to reidentification attacks, demanding new techniques for protecting data in research and clinical settings. This paper describes Genecloud, a secure framework for storing and analyzing genomic data. Genecloud balances data access and privacy considerations by applying persistent governance to sensitive information and to the analytics that operate upon it, managing the interaction between the two. As a practical application of the framework, we describe several models for integration with PAPAyA, a full-featured clinical decision support platform created by Philips to bring next generation sequencing and analysis into the clinic.

1 Introduction

Data fuels scientific discovery, and any technology that restricts access to data has the potential to impede progress. This risk is particularly acute in healthcare systems, where information that sits idle is information that is not being used to treat patients or develop new cures. Making data broadly accessible to research is therefore imperative; the cost of not doing so is too great.

On the other hand, healthcare data is sensitive and its disclosure can be devastating. Genomic information in particular can reveal an uncomfortable amount of detail about health status and disease risk — not only for a patient, but for all of a patient's relatives as well. Recent research has shown that genomic information, even in an anonymized form, is susceptible to attacks that may compromise privacy [HOM, GYM]. Unfortunately, protecting privacy rights of patients and research subjects has often been treated as a secondary concern [LUN]. Researchers and clinicians who work with genomic data are faced with two apparently irreconcilable goals: to increase the amount of genomic data available for researchers while simultaneously preventing reidentification and other attacks on individual privacy.

This paper introduces the Genecloud™ framework, a system designed to mitigate privacy concerns in genomic applications. To illustrate the use of the Genecloud platform in practice, we consider methods for providing stronger security and privacy guarantees for the PAPAyA clinical decision support system from Philips Research North America [JAN].

2 The Genecloud Secure Genomics Framework

Genecloud is a framework for trusted cloud services that store and analyze genetic sequences and other medical information. It is specifically designed to address issues of genetic privacy by allowing researchers

and clinicians to interact with data through computer programs — trusted analytics — that can be managed in different ways under policies determined by the various stakeholders in the data.

Under most current access models, researchers that require access to sensitive data are screened carefully, but then given unfettered access to the information under the assumption that they can be trusted to respect the privacy interests of the subjects. This approach has two fundamental problems: (a) trust is highly contextual, and (b) trust is intransitive. Placing trust in one principal in a given context should not automatically apply to other principals whom he trusts in turn, nor to the same principal in a different context.

The most significant problem with the prevailing approach is that it expects *users* to enforce complex policies that govern the use of data. Once information has been revealed, the burden is on the recipient to act with discretion, a requirement that is often in conflict with the recipient's desire to accomplish his scientific task by sharing that information. However, researchers and clinicians increasingly interact with private data through the intermediary of a computer program, and computer programs can be governed to manage the risk of privacy violations.

2.1 Distributed Security Building Blocks

In the near future, research in genomics will be conducted across distributed data centers by international teams of researchers operating on massive, pooled data sets [CAL]. As promising as they are scientifically, these new models of collaboration raise novel legal, ethical, and privacy issues. In clinical settings, the massive storage and computing requirements for genetic data will favor cloud-based or hybrid solutions, where the same security and distribution issues arise. Supporting a trusted, decentralized, interconnected storage and computing network will require architectural support for features that are not commonly used in clinical and research projects today.

Trust management A trust management system allows relevant authorities to make verifiable assertions about principals and objects in a ecosystem. For example, various authorities may make assertions about the physical security of a data center, adherence to clinical laboratory guidelines, the identity of a human principal, the integrity or regulatory compliance of a software module, etc. A trust management system generates a set of cryptographic credentials that principals use to assert these properties to other actors in the system in an interoperable manner.

Policy management Different institutions have different policies regarding data access and sharing, use of computing resources, etc. To ensure interoperability and consistent policy enforcement, these polices must be encoded and exchanged. For example, if a study participant has consented to the use of their data by a specific participating institution or research study, but not to broader uses, a policy that expresses those conditions should be persistently associated with that data, and enforced across the network. Polices should be dynamic, so that they may be added or withdrawn. Sources of policy are diverse, and include patients, researchers, their funders and institutions, pharmaceutical companies, and governments.

Auditing Where trust and policy management systems prevent unwanted data accesses before they happen, the auditing subsystem allows accesses to be examined forensically. Auditing is used to inform data owners about usage, to meter the use of bioinformatics tools, and to provide crucial support for liability analysis in the presence of strict privacy-protection laws such as HIPAA/HITECH in the US and the European Data Protection Directive.

Encryption and Key Management Once data has been released in the clear, it is virtually impossible to enforce data management policies or even audit the use of sensitive information. A common technique for ensuring that data access is governed and auditable is to protect the data through strong encryption, and

then — crucially — to govern the encryption keys. Access to keys should be treated as equivalent to data access. While many systems incorporate encryption, few provide the required level of key management, tying it into trust management, policy, and auditing capabilities.

Secure Software If programs that access sensitive data are mobile, users that rely upon the results of those programs will need assurances that a) the software that they specified is indeed the software that produced the results they have received, b) other software in the remote system did not modify input data in a way that may change or compromise the results of the program, c) no keying material or secrets contained in the program were leaked. Software security is also important to the institutions that are executing the software: a) verifying that a software module comes from a trusted source, b) ensuring that the software module does not reveal sensitive data that it should not, e.g. by uploading it to a third-party site, c) limiting access of programs from certain sources to both computational resources and data.

2.2 Three Execution Models

Genomics has become an essential driver of new insights in biology and medicine, and research increasingly relies on extensive computational statistical analysis of massive data sets collected from individuals, each of whom has privacy rights. This section describes the security and privacy properties of three different models of computation that may be applied in bioinformatics, starting from the current status quo and building to the Genecloud security model.

The evolution described below reflects specific choices regarding threat models. Specifically, it is not a goal of this work to provide guarantees that a cloud service provider cannot intercept sensitive information. As they become more practical, techniques such as fully homomorphic encryption and secure multi-party computation hold promise in addressing the security of cloud computing. In the meantime, legal structures provide strong incentives for providers to design processes that limit access to private data. The threat that motivates the discussion below is that of revealing more information than strictly necessary in performing a computation. Any non-trivial computation on secure data must reveal *some* information; the goal is to limit the disclosure.

The Status Quo In the most common current case, sensitive data is supplied directly to analysis programs after verifying that the principal who will run the program is authorized. This approach, by far the most common, has several serious drawbacks from a security point of view, most notably:

- It assumes that trust is transitive, when in fact, the principal originally authorized may provide sensitive data to others whom he trusts in turn;
- There is no ability to audit usage of the data at a fine-grained level;
- All of the data for a large study needs to be centralized;
- Conservatively, one must assume that all sensitive information has been revealed;
- The policies governing the use of the data cannot change dynamically and are not technically enforced; the recipients must be trusted to enforce the policies themselves.

Genomic APIs An increasingly popular technique for storing genomic data addresses some of the security issues with direct access. This approach is based on defining interfaces (APIs) to trusted data stores. Genomic APIs have two significant advantages:

- If the principals that wish to access data are required to authenticate themselves, it is possible to discriminate amongst different principals and apply potentially different policies to their accesses;

- The system can provide fine-grained access; principals ask only for the data required for a particular task. The information disclosed — or potentially disclosed [NYH] — can be metered and audited. This level of auditing also allows potential sources of the leaks to be identified.

On the other hand, this approach means that sensitive data is ultimately returned into an untrusted environment. To see where this might pose a problem, consider a simple example: given a list of genome identifiers, determine the number of genomes that have a particular variant. Suppose that the API simply returns variants by identifier. Under these circumstances, the untrusted code will learn the individual variants for each of the genomes in the list, which were presumably chosen based on phenotypical characteristics. Those associations may be compromised, weakening security.

This specific problem might be solved by using a more sophisticated API, capable of returning statistics about the data directly, without revealing individual variants. This type of API still reveals information about the subjects in the cohort [SAN], but the information revealed may be quantified and is much more difficult to exploit. This approach effectively moves the information-revealing computation from untrusted code into a secure, trusted environment.

Genecloud Execution Model The genomic API model described above addresses many of the deficiencies of the direct access model, especially when computations that may reveal information as a side effect are performed behind the API, out of reach of untrusted client-side code. Yet it is difficult to design an API that obscures all such computations, and thus it is inevitable that untrusted code will have access to some intermediate products, revealing more information than is strictly necessary. This problem can be solved by creating a general-purpose computational capability within a trusted boundary.

Moving computations into a trusted environment allows for much more precise control over the information disclosed, as only the information revealed by the final result is visible to untrusted code. Intermediate results remain within the trusted boundary. However, allowing arbitrary code to execute within a trusted environment changes the threat model — malicious or incorrect code might compromise sensitive data, revealing it in unanticipated ways.

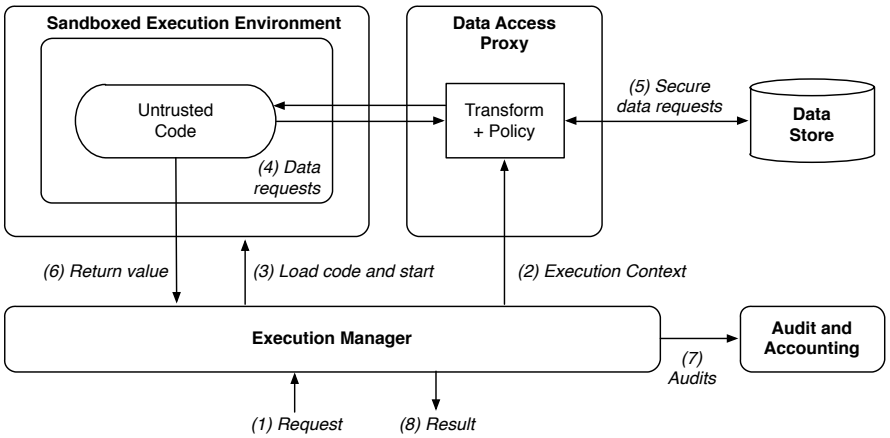

Figure 1: The Genecloud Execution Model: sandboxing untrusted code.

Genecloud relies on several techniques to mitigate these threats:

- Foreign code may be required to be signed by competent authorities, and the signature checked as a condition for code loading or execution;

```
import json, sys, urllib

def query_geneserver(gid, gene, exon)
  url = 'http://geneserver.genecloud.com/genome/%s/variant/gene/%s/exon/%s' % (gid, gene, exon)
  http_handle = urllib.urlopen(url)
  return http_handle.read()

if __name__ == "__main__":
  genome_id = sys.argv[1]
  results = []
  for (gene, exons) in ["PIK3CA", [1, 3, 8, 19]], ["PTEN", [3, 6, 7]]:
    for exon in exons:
      result_raw = query_geneserver(genome_id, gene, exon)
      result = json.loads(result_raw)
      results += result['variants']
  print json.dumps({'variants': results})
```

Figure 2: A program to check for variants in particular exons of the PIK3CA and PTEN genes.

- Code signing provides enhanced auditing and reproducibility; it is possible to know *precisely* which code accessed information;
- Foreign code executes in a sandboxed environment that prevents it from accessing arbitrary storage or network locations and allows the system to limit exposure to sensitive data;
- Foreign code may be required to access data through a fine-grained API, improving auditability and minimizing the amount of personal information that might be compromised by a given computation.

2.2.1 Performing a Computation

Writing Programs Programs written for the Genecloud use REST APIs for data access. Each server hosting sensitive data defines its own semantically appropriate APIs. For example, a server that exposes variant data presents an interface that allows a user to fetch the data via HTTP:

```
http://geneserver.genecloud.com/genome/5685c028bf7811e3a21a12470ec1d3b5/variant/rsid/rs1933437
```

Genecloud is language agnostic. For example, the program shown in Figure 2, written in the Python language, checks for variants in a given set of genes and exons. Programs handle user I/O using the standard `stdin`, `stdout`, and `stderr`.

Loading Programs Programs can be developed for the Genecloud in an untrusted environment, as if developing for a genomic API as shown in §2.2. Programs may be tested by developers using public data, over unsecured HTTP to ensure that they work properly. Once a program is tested, it is uploaded to the Genecloud to be run in a governed environment. This step requires that the developer be authenticated, and allows the developer to set policies and conditions governing the use of the program.

In order to provide isolation and security, programs in the Genecloud are executed in virtualization containers. In the current implementation, virtualization is performed using Docker (`http://docker.io`), a virtualization system based on the Linux containers mechanism (lxc). When user-created programs are loaded into the Genecloud, the system automatically creates executable images and saves them to a repository for retrieval at execution time.

Trusted Execution In step (1) of Figure 1, a request to execute a given program is received by an *Execution Manager* component, which is responsible for managing the lifecycle of a computation running in the Genecloud. Although not shown in the figure, the request is assumed to have first passed through authentication and authorization stages ensuring that the principal requesting the execution of a given computation is allowed by policy to do so. In the Genecloud architecture, this request triggers the loading of third-party code and a further policy check that places conditions on the code itself, such as requiring a digital signature from a relevant authority.

In step (2), the Execution Manager creates an execution context for a particular execution of this program. The execution context allows the Genecloud system to associate sensitive information with the running instance without placing that information into the address space of the untrusted program, where it might be vulnerable. In the example of Figure 2, the program is passed an ephemeral genome ID as its first argument, and the execution context stores its mapping to a real identifier.

The program execution begins in step (3), with the Execution Manager starting the virtual machine instance and passing in the necessary parameters. As the program executes, it may request data from a data store over an HTTP API, as shown in step (4). Because the program is running inside a container, its access to network resources can be restricted to only trusted endpoints. Before these calls reach the data store, however, several additional things happen:

- The request is potentially transformed using information stored in the execution context. For example, in this step, the system might determine how the ephemeral identifiers given to a program as parameters map to actual identifiers in a data store. Information about the execution context may also be passed along to the data store as part of the request.
- The request, which was made over plain HTTP, is promoted to HTTPS, with certificates at both the client and server. This ensures that only certified systems can interact with trusted data stores, and keeps keying information out of the address space of the third-party program.
- Policies are applied to authorize the request. These policies offer more granular control over access to sensitive data because they are applied as a computation proceeds, rather than before it begins. In general, the set of requests a program will make cannot be determined in advance.
- The destination of the request may be rewritten. For example, if a request can be satisfied by a number of different servers, the proxy may direct the request to the most appropriate.
- The request is logged.

The results of the request — possibly after being transformed again with information in the execution context — are returned into the address space of the untrusted code. When the program terminates, its output is captured by the Execution Manager (step 6), audited (step 7), and returned back to the original requester (step 8). Audits are digitally signed so that they may be verified later. Because the Genecloud execution system depends on virtualization on known virtual machines, and because the Genecloud stores, signs, and tags all state information, it is possible to completely recreate a computation at a later time — and even in a different location — to validate these audits.

Computational Networks In the Genecloud model, potentially information-revealing computations are sandboxed, limiting access to intermediate products. More complex computations, however, may consist of multiple stages that can be further isolated in order to improve security and reduce information leakage.

For example, consider a simple computation that evaluates the genetic disease carrier compatibility of two subjects. It is possible to create a single program that a) looks up two subjects by a phenotypic identifier b) checks for the presence or absence of a variant and c) determines whether both subjects are carriers. In the worst case, this single program could leak personally-identifiable information. However, it is possible to

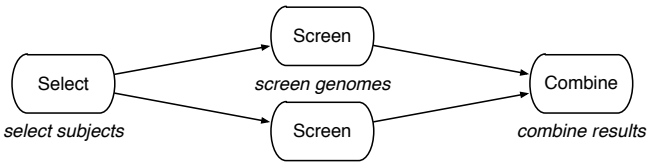

Figure 3: A simple computational network for carrier compatibility detection

transform this program into a network of three separate programs such that each, run in isolation, cannot make the connection between genotype and phenotype.

Figure 3 shows a computational network that separates the three different computations into isolated address spaces. The *Select* computation chooses two subjects, presumably based on phenotypical criteria. Each of these identifiers is mapped by the system into a genome identifier outside of the address space of either of the programs. Each genome is then assessed separately in a *Screen* process, which checks for the presence of a particular variant and passes a boolean value on to the *Combine* process, which performs a logical AND to determine carrier compatibility. Although the computation performed by this network is the same as that computed by a single program, the amount of personally-identifiable information that might possibly be released is greatly reduced.

Genecloud provides a mechanism for specifying computational networks such as the one shown in Figure 3. The system takes care of the necessary transformations between modules, lifecycle management, storage and transport of intermediate products, and returning the final result.

3 Securing Clinical Decision Support: PAPAyA

In this section, we explain how the security models described above are used to enhance security and privacy for PAPAyA, the platform for clinical decision support created by Philips Research.

PAPAyA is a software platform designed around management, analysis, and delivery of clinical sequencing data in an oncology setting. PAPAyA can be deployed wholly within the hospital IT infrastructure or in a cloud-based environment, allowing clinical users to access the platform through a browser. The interface is designed to allow oncologists and pathologists to explore and interact with clinically relevant information, presented through a sequence of modules that address specific clinical questions. The utility of this approach has been demonstrated in early studies such as [ROY]. The screen capture in Figure 4 shows several of these clinical modules operating on simulated patient data. The module results are based on both specimen-specific information and patient population data.

PAPAyA provides a platform for executing two different types of computations: modules and pipelines. Modules may be thought of as "in-silico assays" that address specific clinical questions such as calling actionable mutations or identifying targetable genomic aberrations like deletions or gene fusions. Modules are implemented as plug-ins to the system whose source code, input and output parameters, versioning, and dependencies are managed by the PAPAyA platform. The results of module computations — which may depend upon the results of other modules, clinical knowledge bases, and the like — are stored in a database and presented by the web interface on demand. While modules answer specific clinical questions, pipelines typically perform more computationally intensive processing that generates intermediate products that ultimately serve as input to clinical modules.

For example, the module that calls actionable mutations answers clinical questions about suitability for trials, predisposition to certain diseases and conditions, and the probability that a patient may respond to a

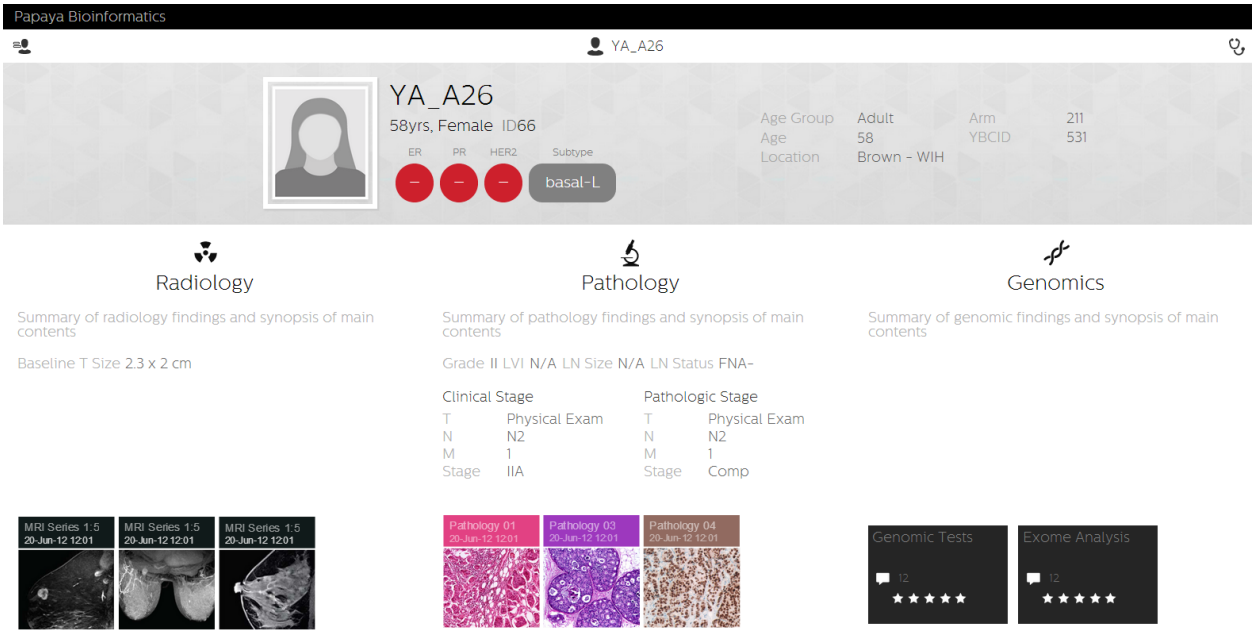

Figure 4: The PAPAyA Web Interface

specific targeted therapy. This module compares germline and tumor DNA sequences in order to identify potentially actionable somatic mutations in the tumor and annotates them for clinical relevance.

3.1 Subsystems for Data, Analytics, Workflow, and Policy

This section introduces PAPAyA subsystems for data, analytics, workflow, and policy management, highlighting some of the security and privacy issues pertaining to each. The system is implemented according to a Model-View-Controller (MVC) architecture such as the one shown on the left side of Figure 6. The implementation encompasses not only genomic data and interpretation, but also storage and management of modules and pipelines, workflow data, execution logs, and biological clinical knowledge bases.

1. **Data Management**

   (a) *Genomic data management* — This subsystem manages genomic data and analysis, analyzing genomic aberrations in individual genomes, exomes, and transcriptomes. The genomic data management system spans multiple locales, from sequencing centers to local and cloud computing environments to clinical settings. As a result, PAPAyA is built on a trust management framework that enables secure integration across local and remote environments.

   (b) *Knowledge management* — Includes knowledge bases that contain valuable clinical evidence and information on biological pathways, biomarkers, and therapies associated with these markers. The knowledge bases themselves are sensitive data that must be protected and versioned.

2. **Module and Pipeline Management** In-silico assays consist of multiple pipelines and modules, executed in a particular sequence, that implement a specific multivariate index assays based on genomic information managed in PAPAyA. The individual modules and pipelines are managed and versioned to ensure integrity and regulatory compliance. This subsystem provides several features that require trust management and integrity protection:

- A framework for pipeline management taking into account versions of pipelines that are specific for certain sequencing protocols and chemistry;
- A trusted IPR broker to record licensed IPR used in pipelines under various business models;
- Change management and logging auditing information on the updates of specific pipelines.

3. **Workflow Execution Management** In order to understand potential risks to privacy for genomic data in the system, it is necessary to control and audit data accesses. In executing in-silico assays, access should be restricted to only the chromosomal regions that are required for the assay. Control over access at this granularity requires:
   - A trusted execution framework supporting application-specific modules and workflows and managing their data access (*cf.* §3.3 below);
   - A framework for managing intermediate data structures generated during a workflow execution and handling the interaction between stored genomic data and the visualization modules.
4. **Policy management framework** In the context of PAPAyA, the policy management subsystem is concerned with issues such as the reporting of incidental findings, rules for consistent data management within an institution, and presentation of information in different jurisdictions[1]. When rules differ across jurisdictions, the policy management system must be able to harmonize them or raise an exception so that they may be resolved manually.

3.2 Security of Various Processing Stages

The data processing workflow in PAPAyA is divided into four phases, each defined by its own specific security and privacy considerations. Through these four phases, sensitive information progresses from high-volume undifferentiated data to actionable low-volume data that can be tied to specific phenotypic traits. The four processing phases for variant calling using exome data are shown in Figure 5 and described in more detail below.

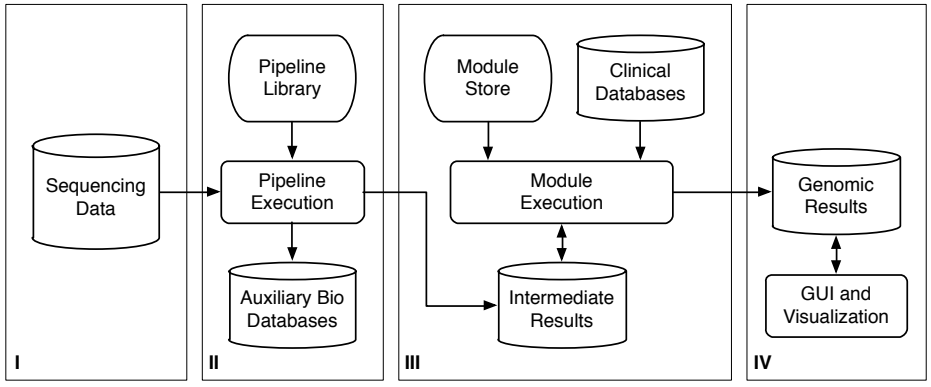

Figure 5: Exome Processing Stages in PAPAyA. (I) Sequencing and Ingestion (II) Pipelines (III) Modules (IV) Visualization and User Interaction. While stage (IV) happens in a clinical setting such as a hospital, stages (I)–(III) typically occur in remote computing environments such as a sequencing lab, a cloud computing environment, etc.

[1]Individual states in the United States have widely divergent policies on matters of healthcare information management.

Sequencing and Ingestion In the first phase, a set of raw, unaligned reads is produced at a sequencing machine and transported (in FASTQ format) into the system for further processing. At this stage, it is not meaningful to make a policy distinction between different reads. Although the set of reads contains all of the sensitive information in the genome, this information is unorganized. Because the reads are not yet aligned, and in a random order, the reads are equivalent from a security point of view. The techniques appropriate at this phase will involve protection of the raw read data and secure transmission to a server for further processing.

Pipelines In this next stage, the raw reads are transformed into a set of annotated variants, first by alignment to a reference (producing BAM or CRAM files), then by variant calling and annotation. The annotated variants are stored in a database in PAPAyA, but represent the information that would typically be stored in a VCF file in a file-based system. This process organizes the information in a way that it would be more immediately useful to an attacker, but the data volume is still large. The programs that perform alignment require access to all of the reads on an equal basis, so there is no requirement to differentially protect various parts of the data. The system performing the alignment should protected by standard computer security techniques such as authentication of administrators, role-based access control, security audits, and the like. Pipelines may be executed without direct human interaction, in which case no authentication and authorization of end users is required. Third-party pipelines could be supported using the techniques described for modules, below.

Modules The ultimate output of the previous phase is an annotated set of variants, which are extremely private and may be subject to different policies depending upon the relative sensitivity of the variants. When running modules on this data, it is important to know who asked for a given module to be executed, for what purposes, and whether they are authorized to make the request. Ideally, the modules should run inside of a secure environment so that their access to sensitive data may be controlled more carefully, and their ability to perform illegitimate accesses (e.g. to post sensitive data in a third-party site) may be controlled. This phase is the most likely candidate for the techniques described in §2.2.

Visualization and User Interaction In the previous phase, computations are performed without necessarily revealing the results of these computations to users; the results of the computations may be stored in a database for later access. In this phase, on the other hand, sensitive information may be revealed to an end user, which requires that the user be authenticated, and any policies governing the user's access to the output of a given module be checked.

3.3 Module Integration Scenarios

This section describes integration scenarios for executing PAPAyA modules in the Genecloud framework.

Treat Modules as Trusted Code In some circumstances, the genomic API model described in §2.2 may provide sufficient protection. Because PAPAyA is built on an object-relational model (ORM), modules can be adapted in a natural way to interact with a secure API. An ORM allows developers using web services frameworks such as *Ruby on Rails* or *Django* to interact with automatically-generated model objects whose class corresponds to a database table, and whose object instances correspond to rows within that table. Many web frameworks allow developers to interact transparently with a model object stored in a remote server over a REST API in precisely the same manner as for a local object.

As an example of the Object/REST mapping, it is possible to translate a statement that would normally retrieve a database record like `patient.rsid(1933437)` into a URL (as shown in §2.2.1), where the genome ID is associated with the patient object in the local database and the path is constructed automatically by

the Object/REST mapping layer. This interaction is shown as interaction (A) in Figure 6. The request may also be coupled with an authentication mechanism that allows the code making the request to be identified and audited.

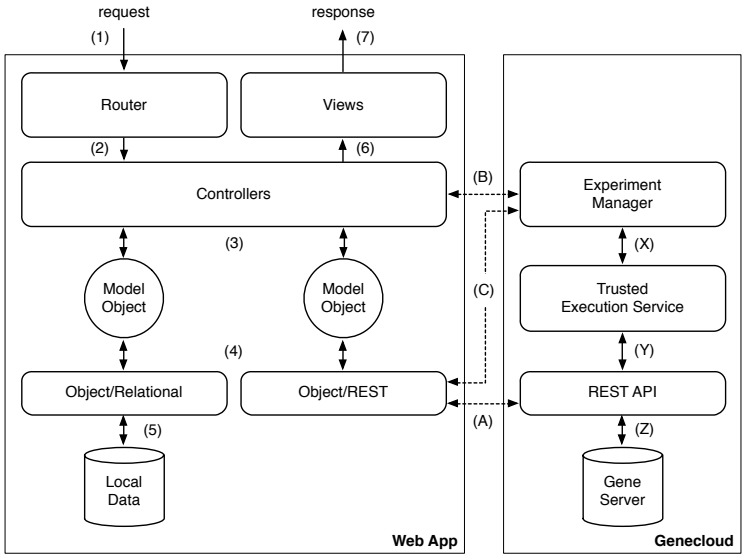

Figure 6: Integration Strategies. Numbered steps are exactly as in a standard MVC web application. (A), (B), and (C) illustrate various strategies for interacting with Genecloud. In (A), a model object corresponds to an entry accessed via a REST API through an Object/REST adapter. In (B) and (C), controller logic triggers execution of code in Genecloud either directly (B) or via a model object intermediary (C).

Integration at the API level is appropriate when modules:

- Come from a trusted source that does not need to be authenticated during the transactions;
- Have been analyzed for undesired behavior;
- Do not require proof of integrity for access to the sensitive data;
- Are executed in a trusted environment that can ensure that the sensitive information being retrieved by the modules cannot be compromised by other components;
- Can pass intermediate products between one another safely.

Sandboxing Individual Modules In cases where one or more of the conditions above do not hold, some protection may be afforded by applying the sandboxing technique described in §2.2 at the individual module level. Approaches for implementing this type of module-level sandboxing include:

- Sandboxed modules may be invoked by controller code directly. If desired, the modules themselves can use the Object/REST mapping exactly as described above, with the adapter being injected as a dependency into the virtualization container. This approach may work well in cases where there are not many dependencies on other model objects, or where the necessary parameters can be passed to the sandboxed modules as parameters. See interaction (B) in Figure 6.
- By adding a level of indirection; rather than mapping instance variables and method accesses to REST calls that return those items, an object mapping can convert those requests into commands to execute the sandboxed models, passing in the necessary parameters. See interaction (C) in Figure 6.

Using this integration strategy, the system can safely handle third-party code, cryptographically verify code integrity, and so forth. However, it does not address the passing of intermediate products, which are still returned to the web application and must be trusted in that environment.

Sandboxing a Computational Network To mitigate risk to intermediate products, an entire computational network can be executed in the Genecloud framework. This approach, a superset of those described above, would involve specifying the computational network, loading all of the required modules, and then ordering an execution of the network within the trusted environment. This approach may also increase performance, as it avoids repeated roundtrips back to the web application made solely for the purpose of moving data from one processing stage to the other.

4 Conclusions

Taking patient rights seriously is imperative if the use of next generation sequencing data and analytics are to enter mainstream clinical practice. The standard techniques for addressing patient privacy, such as anonymization, have been shown to be vulnerable to reidentification attacks. As a result, the research and clinical communities have been moving in the direction of genomic access via APIs, enabling auditing, authentication, and some degree of policy management. In this paper, we have shown the limits of this model and described an alternative in which the interaction between secure data and the analytics that operate upon them is governed, reducing the potential threats to patient privacy. We have demonstrated that these techniques can be practically integrated with standard Model-View-Controller web applications in several different ways, depending on specific threat models.

References

[LUN] J. Lunshof et al. *From Genetic Privacy to Open Consent*. Nature Reviews Genetics 9.5 (2008): 406-411.

[HOM] N. Homer et al. *Resolving individuals contributing trace amounts of DNA to highly complex mixtures using high-density SNP genotyping microarrays.* PLoS Genetics, 4(8), 2008.

[GYM] M. Gymrek, McGuire, et al. *Identifying Personal Genomes by Surname Inference*. Science, vol. 339, no. 6117, pp. 321-324, 18 January 2013.

[JAN] A. Janevski et al. *PAPAyA: A Platform for Breast Cancer Biomarker Signature Discovery, Evaluation and Assessment*. BMC Bioinformatics, 10(Suppl 9):S7, 2009.

[CAL] E. Callaway. *Global Genomic Data-Sharing Effort Kicks Off*. Nature News, Mar 6, 2014. `http://www.nature.com/news/global-genomic-data-sharing-effort-kicks-off-1.14826`.

[SAN] S. Sankararaman et al. *Genomic privacy and limits of individual detection in a pool.* Nature Genetics, 41(9), 965-967, 2009.

[NYH] D. Nyholt et al. *On Jim Watson's APOE status: genetic information is hard to hide*. European Journal of Human Genetics, 17(2), February 2009.

[ROY] S. Roychowdhury et al. *Personalized Oncology Through Integrative High-Throughput Sequencing: A Pilot Study*. Science Translational Medicine, Vol. 3, Issue 111, p. 111–121, November 2011.