



BUCKINGHAMSHIRE
NEW UNIVERSITY
EST. 1891

Secure Genomic Computing: Defence-in-Depth in a Hybrid HPC Cloud Environment

Authors:

Harihara Sudhan Shanmugam, Computational Research Technician

harihara.shanmugam@bnu.ac.uk

HPC Systems Engineering Team

Digital Technical Services

Buckinghamshire New University

Chamodi Korala Hewage, Research System Developer

chamodi.koralahewage@bnu.ac.uk

HPC Systems Engineering Team

Digital Technical Services

Buckinghamshire New University

Affiliation:

Pathogen Genomics Unit, Buckinghamshire New University.

Technical Report Contact:

Harihara Sudhan Shanmugam, Computational Research Technician

harihara.shanmugam@bnu.ac.uk

HPC Systems Engineering Team

Digital Technical Services

Buckinghamshire New University.

Document Type:

Research and Development Systems Technical Report

Date:

March 2025

High Wycombe Campus

Queen Alexandra Road
High Wycombe
Buckinghamshire HP11 2JZ

Aylesbury Campus

Walton Street
Aylesbury
Buckinghamshire HP21 7QG


Uxbridge Campus


106 Oxford Road
Uxbridge
Middlesex UB8 1NA

Telephone: 01494 522 141

International: +44 1494 605 259

Email: advice@bnu.ac.uk

 @BuckinghamshireNewUniversity

 @BuckinghamshireNewUniversity

 @BNUni_

 @_BNUni

 @_BNUni

[BNU.AC.UK](https://www.bnu.ac.uk)



Table of Contents

Abstract.....	4
Technical Implementation.....	4
Hardware Architecture	4
Authentication Framework.....	5
Data Transfer Security	5
Container Security Implementation	6
GPU Security Controls.....	6
BioBERT Model Security.....	7
Secure Pathogen Genomic Processing	8
Cross-Environment Authentication Security	8
Secure Variant Data Transfer.....	9
GPU Security for Deep Variant Processing	11
Containerized BioBERT Security.....	12
Secured BioBERT Model for Variant Annotation	13
Performance-Security Integration in Genomic Pipeline	14
Technical Challenges and Solutions	15
Cross-Environment Authentication	15
GPU Memory Protection.....	15
Secure Data Transfer Pipeline.....	16
Discussion.....	17
Security-Performance Balance.....	17
Genomic-Specific Security Considerations	18
Hybrid Architecture Security.....	18
Comparative Advantage	19
Future Work	19
Advanced Cryptographic Technologies	19
Confidential Computing	20
AI Security for Genomics.....	20
Enhanced Regulatory Compliance	21
Privacy-Enhancing Technologies.....	21
Conclusion	21
References.....	22



Abstract

This technical report presents the secure design and deployment of a hybrid High-Performance Computing (HPC) and cloud infrastructure for genomic data processing at Pathogen Genomics Unit (PenGU) in Buckinghamshire New University. The architecture seamlessly integrates on-premises Slurm-managed clusters with Microsoft Azure services, incorporating advanced security measures such as AES-256-GCM encryption, zero-trust authentication frameworks, and hardened containers using secure computing profiles.

Engineered with a defence-in-depth strategy, the solution achieves robust protection with minimal performance trade-offs—maintaining a 50% reduction in processing time and an 86% reduction in queue wait times compared to the baseline system. Comprehensive security testing demonstrated 99.8% effectiveness across the STRIDE threat model (Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, and Elevation of Privilege), validating the architecture's resilience in handling sensitive genomic workloads. This implementation sets a benchmark for secure, high-throughput genomic computing in hybrid HPC-cloud environments.

Technical Implementation

This section outlines the specific technologies and methodologies used to secure sensitive genomic data processing across the hybrid computational environment. The implementation addresses the unique security challenges of genomic data while maintaining the performance advantages of the distributed architecture.

Hardware Architecture

The secure hardware infrastructure builds upon the foundation described in the original report, with specialized security enhancements for both on-premises and cloud components.

On-Premises: 28 Dell PowerEdge R740 compute nodes with Mellanox InfiniBand High Data Rate (HDR) interconnect

Cloud Integration: Azure ExpressRoute (10 Gbps) with dedicated connection.



Security Hardware: Thales Luna Network Hardware Security Module (HSM) 7 FIPS 140-2 Level 3, Trusted Platform Module (TPM) 2.0 attestation.

Storage: 120TB Lustre parallel filesystem with Self-Encrypting Drives (SEDs) with Opal 2.0 compliance.

Authentication Framework

The authentication system ensures secure identity verification across hybrid environments, addressing the challenges of cross-domain authentication while maintaining seamless operation (Zhang et al., 2024).

Implementation: OAuth 2.0 with Proof Key for Code Exchange (PKCE), JSON Web Tokens (JWT) tokens (RSA-256 signing)

Token Specifications: 1-hour maximum lifetime, audience-restricted claims

Federation: Azure Active Directory (AD) integration with Microsoft Authentication Library (MSAL.js) for centralized identity

MFA: Time-based One-Time Password (OTP) with Conditional Access policies

Results: Authentication failures reduced from 2.3% to 0.02%

Data Transfer Security

The data transfer security implementation protects genomic data confidentiality and integrity during movement between on-premises and cloud environments, addressing both current and future cryptographic threats.

Encryption: Transport Layer Security (TLS) 1.3 with Advanced Encryption Standard (AES-256-GCM) and Elliptic Curve Diffie-Hellman Ephemeral (ECDHE) key exchange.

Advanced Cryptography: Elliptic curve cryptography using P-384 with perfect forward secrecy.

Integrity Verification: Hash-based Message Authentication Code with Secure Hash Algorithm 256 (HMAC-SHA256) with Merkle tree verification.



Network Security: Private endpoints with service endpoint policies.

Performance: 4.2% overhead compared to non-secured transfers.

Container Security Implementation

The containerization strategy implements defence-in-depth security controls for both Singularity (on-premises) and Azure Container Registry environments, significantly reducing the attack surface as recommended by Wagner et al. (2022).

Base Technology: Singularity 3.10.0 (on-premises), Azure Container Registry (cloud).

Hardening Measures:

- Secure computing (seccomp) profiles blocking 358 unnecessary system calls.
- Capability restrictions (cap_drop: ALL; cap_add: NET_BIND_SERVICE, SYS_PTRACE).
- Read-only filesystems with explicit volume mounts.
- User namespace isolation with dynamic User Identifier (UID) mapping.

Supply Chain Security:

- Image signing with Elliptic Curve Digital Signature Algorithm (ECDSA-P256) (Cosign)
- Vulnerability scanning with Trivy (pre-deployment)
- Centre for Internet Security (CIS) benchmark compliance (95% minimum score)

Results: Container attack surface reduced by 67%, vulnerabilities reduced from 24 per image to 3.1

GPU Security Controls

The GPU security implementation addresses the unique memory protection challenges for sensitive genomic data processing, particularly for Deep Variant (Poplin et al., 2018) and BioBERT operations, building on research by Ramirez et al. (2023).



Memory Protection:

- Compute Unified Device Architecture (CUDA) context isolation with dedicated memory regions
- Secure memory wiping between jobs
- Side-channel protection through timing normalization

Access Control: Fine-grained permissions based on data classification.

Monitoring: Graphics Processing Unit (GPU)-specific anomaly detection with Data Centre GPU Manager (DCGM) integration.

Performance: 2.8% overhead for secured GPU operations.

BioBERT Model Security

The BioBERT (Bidirectional Encoder Representations from Transformers for Biomedical Text Mining) implementation secures both the language model and its Application Programming Interface (API) endpoints, protecting the model from extraction attacks while maintaining the benefits of AI-driven variant annotation, addressing concerns raised by Wang et al. (2023) regarding security in genomic language models.

The BioBERT represents a critical component of the genomic processing pipeline, enabling rapid and accurate annotation of genetic variants through advanced natural language processing techniques applied to biomedical literature. However, the deployment of such AI models in security-sensitive genomic environments introduces unique security challenges that require specialized protection mechanisms. Our security implementation addresses both the protection of the model itself and the securing of interfaces through which the model is accessed

Model Protection:

- Envelope encryption (Advanced Encryption Standard (AES-256) + Rivest–Shamir–Adleman (RSA-4096)) for model files
- Memory encryption for activation tensors during variant processing
- Integrity verification with signed manifest.



API Security:

- Input validation with schema enforcement for variant data
- Rate limiting with adaptive thresholds
- Output filtering to prevent sensitive genomic data leakage

Results: 99.7% effectiveness against Open Web Application Security Project (OWASP) attacks while maintaining 68% reduction in annotation time.

Secure Pathogen Genomic Processing

Our hybrid HPC-cloud architecture implements a comprehensive security framework to protect sensitive pathogen genomic data while preserving the performance benefits of our distributed computing environment. This section details the algorithmic approaches and implementation methods designed specifically for our Pathogen Genomics Unit (PenGU) workflows.

Cross-Environment Authentication Security

One of the most challenging aspects of our hybrid architecture was securing identity propagation between our on-premises HPC cluster and Azure cloud resources. Our initial deployment revealed significant security gaps that could have allowed unauthorized access during cloud bursting operations.

Implementation Steps for Genomic Pipeline Authentication:

1. Enhanced our Slurm job scheduler with Azure Active Directory integration using federated identity
2. Implemented genomic job-specific security contexts with role-based access mapped to dataset sensitivity
3. Developed token exchange service for secure identity propagation during cloud bursting operations
4. Applied cryptographic binding between job tickets and genomic dataset identifiers
5. Created automated audit logging for all cross-boundary authentication events



PenGU-Specific Authentication Algorithm:

```
function propagateIdentity(slurm_job_id,  
target_cloud_partition)  
    % Retrieve job credentials from secure store  
    job_context = getJobContext(slurm_job_id);  
    genomic_dataset_id = job_context.dataset_id;  
  
    % Generate dataset-specific claims  
    dataset_permissions =  
getDatasetPermissions(genomic_dataset_id);  
    restricted_claims =  
createRestrictedClaims(dataset_permissions);  
  
    % Create Azure-compatible token with limited scope  
    azure_token = createAzureToken(  
        restricted_claims,  
        lifetime=1800, % 30 minutes - our typical burst  
job duration  
        resource_target=target_cloud_partition  
    );  
  
    % Log the cross-boundary authentication  
    logSecurityEvent(  
        "CROSS_ENV_AUTH",  
        slurm_job_id,  
        genomic_dataset_id,  
        target_cloud_partition  
    );  
  
    return azure_token;  
end
```

This tailored approach reduced authentication failures during genomic pipeline execution from 2.3% to 0.02%, while ensuring that variant data access permissions were properly enforced across environment boundaries.

Secure Variant Data Transfer

Our pathogen genomics workflows frequently transfer sensitive variant data between our high-performance computing environments during cloud bursting operations. We implemented a specialized transfer protocol to ensure both performance and security.



Implementation Steps for Variant Data Protection:

1. Developed variant-specific compression before encryption to optimize transfer performance
2. Implemented AES-256-GCM encryption with environment-specific keys stored in Thales Luna HSM
3. Created integrity verification with specialized checksums for genomic data formats
4. Established dedicated ExpressRoute channel for variant data with traffic isolation
5. Implemented automated post-transfer verification specifically for genomic file formats

Pathogen Variant Data Transfer Algorithm:

```
function secureVariantTransfer(variant_file,
destination_node)
    % Compress variant data with genomic-specific
    algorithm
    compressed_path = compressVariantFile(variant_file);

    % Generate genomic-format-aware checksum
    variant_checksum =
    calculateVariantChecksum(compressed_path);

    % Encrypt with pathogen data-specific key
    pathogen_key =
    getPathogenTypeKey(getPathogenType(variant_file));
    encrypted_path = encryptWithPathogenKey(
        compressed_path,
        pathogen_key
    );

    % Transfer through dedicated ExpressRoute channel
    result = transferThroughSecureChannel(
        encrypted_path,
        destination_node,
        "genomic-data"
    );

    % Format-specific integrity verification
    remote_integrity = verifyVariantFileIntegrity(
        destination_node,
        getFilename(encrypted_path),
        variant_checksum
    );

    return remote_integrity;
end
```



Our implementation achieved 99.998% integrity verification success rate for variant data transfers while maintaining the 12GB/s aggregate throughput necessary for our 120TB Lustre filesystem operations.

GPU Security for Deep Variant Processing

Our NVIDIA A100 GPU nodes (4 GPUs per node, 80GB VRAM each) posed unique security challenges when processing pathogen genetic sequences, particularly during the Deep Variant neural network operations.

Implementation Steps for GPU Pathogen Processing Security:

1. Developed pathogen-type memory isolation to prevent cross-contamination of variant data
2. Created specialized memory clearing routines for genomic data using randomized patterns
3. Implemented secure variant batching that prevents sample data from being co-located
4. Applied side-channel protection specifically for pathogen classification operations
5. Deployed anomaly detection focused on variant processing patterns

Deep Variant GPU Security Implementation:

```
function initializeSecureGPUContext(pathogen_type,
sample_id)
    % Allocate isolated memory region for specific
    pathogen type
    secure_context = cudaCreateSecureContext(
        pathogen_type_id=getPathogenTypeID(pathogen_type),
        sample_id=sample_id,
        isolation_level="high"
    );

    % Apply memory protection for genomic workloads
    cudaApplyGenomicProtection(secure_context);

    % Set up secure tensor operations for variant calling
    cudaConfigureVariantOperations(
        secure_context,
        operation_type="deep_variant",
        protection_level="classified"
    );
end;
```



These specialized GPU security controls eliminated memory remnants containing sensitive pathogen data (0% in 10,000 test cycles) with only 2.8% performance overhead for our Deep Variant operations.

Containerized BioBERT Security

Our BioBERT implementation for pathogen variant annotation required specialized security controls to protect both the model and the sensitive genomic data being processed.

Implementation Steps for Secure BioBERT Deployment:

1. Created variant-specific container profiles for different pathogen types
2. Implemented model partitioning to isolate critical components by security classification
3. Developed API input sanitization specifically for genomic variant formats
4. Applied inference monitoring specialized for unusual pathogen variant patterns
5. Implemented data loss prevention for classified pathogen information

BioBERT Container Security Configuration:

```
# PenGU BioBERT Container Security Profile
securityContext:
  # Read-only filesystem except for specific variant
  # directories
  readOnlyRootFilesystem: true
  # Non-root execution for variant processing
  runAsNonRoot: true
  runAsUser: 1000
  # Limited capabilities for variant processing only
  capabilities:
    drop:
      - ALL
    add:
      - NET_BIND_SERVICE # Required for variant API
  # Custom security profile for BioBERT variant processing
  seccompProfile:
    type: Localhost
    localhostProfile: profiles/biobert-variant
profile.json
# PenGU-specific security labels
labels:
  data-classification: "pathogen-genomic"
  variant-processing: "allowed"
  pathogen-types: "restricted"
```



This containerization approach enabled us to leverage BioBERT for variant annotation while maintaining strict security controls aligned with our pathogen data classification policies.

Secured BioBERT Model for Variant Annotation

Our BioBERT model for pathogen variant annotation represented a particularly sensitive component requiring specialized protection mechanisms.

Implementation Steps for Variant Annotation Security:

1. Implemented model encryption with pathogen-type specific keys
2. Developed secure loading routine that validates model integrity before variant processing
3. Created variant-specific input validation aligned with expected pathogen patterns
4. Applied rate limiting based on normal variant annotation patterns
5. Implemented output filtering to prevent leakage of classified pathogen information

Secure Variant Annotation Algorithm:

```
function secureVariantAnnotation(variant_data,  
pathogen_type)  
    % Load secured BioBERT model with pathogen-specific  
    key  
    model_path = getPathogenModelPath(pathogen_type);  
    key_id = getPathogenKeyID(pathogen_type);  
  
    % Verify model integrity before loading  
    if ~verifyPathogenModelIntegrity(model_path,  
pathogen_type)  
        logSecurityAlert("MODEL_INTEGRITY_FAILURE",  
pathogen_type);  
        error("Security violation: model integrity check  
failed");  
    end  
  
    % Load model into secure environment  
    model = loadModelSecurely(  
        model_path,  
        key_id,  
        classification_level="pathogen"  
    );  
  
    % Validate variant format before processing  
    if ~isValidVariantFormat(variant_data, pathogen_type)  
        logSecurityAlert("INVALID_VARIANT_FORMAT",  
pathogen_type);
```



```
        error("Security violation: invalid variant  
format");  
    end  
  
    % Perform secure annotation within isolated  
environment  
    annotations = performSecureAnnotation(model,  
variant_data);  
  
    % Filter annotations to prevent data leakage  
    filtered_annotations = filterSensitivePathogenInfo(  
        annotations,  
        pathogen_type  
    );  
  
    return filtered_annotations;  
end
```

These specialized BioBERT security measures achieved 99.7% effectiveness against potential attacks while maintaining the 65.6% improvement in annotation time (from 3.2 hours to 1.1 hours per batch) critical for our pathogen surveillance workflows.

Performance-Security Integration in Genomic Pipeline

Our security implementation demonstrates that genomic data protection can coexist with high-performance computing when tailored to the specific needs of pathogen workflows. By focusing security controls on the most sensitive components of our pipeline, we achieved comprehensive protection with minimal performance impact.

The specialized security controls for our BioBERT model and GPU-accelerated variant processing were particularly critical, as these components handle the most sensitive pathogen data. Our security design enables the rapid genomic processing necessary for time-sensitive pathogen surveillance while ensuring data protection throughout the workflow.

- **Risk-based security allocation:** We mapped security controls to data sensitivity classifications, applying the strongest protections to variant data and pathogen classification results.
- **Optimized cryptographic operations:** For variant data transfers between environments, we implemented robust pipeline-aware optimization techniques that overlap cryptographic operations with I/O wait times.



Technical Challenges and Solutions

Throughout the implementation, several significant technical challenges required innovative solutions to maintain both security and performance.

Cross-Environment Authentication

One of the foremost challenges involved securing cross-environment authentication across on-premises HPC clusters and cloud-based Azure services. The initial deployment revealed gaps that posed risks of credential theft and token reuse—two major vulnerabilities in hybrid identity architectures. The system experienced inconsistent authentication handoffs between local and cloud identity providers, primarily due to incompatible token lifetimes, unverified session contexts, and lack of dynamic claim validation. These discrepancies created opportunities for session hijacking and unauthorized access under certain load conditions.

Challenge: Initial authentication failure rate of 2.3% with potential for token reuse attacks.

Solution:

Implemented JWT token exchange service with claims transformation

Code:

```
def exchange_token(source_token, target_audience):
    claims = validate_token(source_token)
    restricted_claims = restrict_permissions(claims)
    return generate_token(restricted_claims,
                        lifetime=3600,
                        audience=target_audience)
```

Result: Authentication failures reduced to 0.02%, token compromise risk reduced by 97%.

GPU Memory Protection

GPU memory presented unique security challenges for genomic data processing, particularly for Deep Variant operations that handle sensitive variant information.



The architecture of modern Graphics Processing Units (GPUs) creates specific security concerns when processing sensitive genomic data, as GPUs typically lack the memory protection mechanisms found in CPU environments. Our security analysis identified several critical vulnerabilities in the default GPU memory handling that could potentially expose sensitive genomic variants.

Challenge: Data remnants detected in GPU memory in 8.2% of cases after job completion.

Solution:

Implemented custom CUDA memory handler:

Code:

```
cudaError_t secureAllocate(void** devPtr, size_t
size)
{
    cudaError_t result = cudaMalloc(devPtr, size);
    if (result == cudaSuccess) {
        // Initialize with random data
        secureRandomize<<<blocks, threads>>>(*devPtr,
size);
    }return result;
}
```

Memory clearing between jobs:

Code:

```
cudaError_t secureFree(void* devPtr, size_t size)
{
    // Zero memory before freeing
    secureWipe<<<blocks, threads>>>(devPtr, size);
    cudaDeviceSynchronize();
    return cudaFree(devPtr);
}
```

Result: GPU memory remnants eliminated (0% in 10,000 test cycles).

Secure Data Transfer Pipeline

Ensuring the confidentiality and integrity of sensitive genomic data during cloud bursting operations required a multi-layered approach to data protection.



Challenge: Ensuring integrity and confidentiality of genomic data with cloud bursting.

Solution:

Implemented hybrid transfer protocol with integrity verification:

Code:

```
def secure_transfer(file_path, destination):  
    # Generate checksum before transfer  
    checksum = calculate_sha256(file_path)  
  
    # Encrypt with strong elliptic curve crypto  
    encrypted_path = ec_encrypt(file_path,  
                                algo="P-384")  
  
    # Transfer with AzCopy  
    result = azcopy_transfer(encrypted_path,  
                             destination)  
  
    # Verify integrity after transfer  
    remote_checksum =  
    get_remote_checksum(destination)  
    return checksum == remote_checksum
```

Result: 99.998% integrity verification success rate, high-strength encryption for all transfers.

Discussion

The implementation of comprehensive security controls in the hybrid HPC-cloud architecture presents several important findings regarding the balance between security and performance in genomic computing. Our approach demonstrates that with proper architectural design and optimization, strong security controls can be implemented with minimal performance impact.

Security-Performance Balance

One of the most significant findings is the relatively small performance overhead (3-8%) compared to previous implementations. Wagner et al. (2022) reported performance penalties of 35-45% when implementing comparable security controls, primarily due to their reliance on full-disk encryption rather than selective data protection.



Our selective approach to encryption, applying different levels of protection based on data sensitivity classification, significantly reduced this overhead while maintaining strong security guarantees.

The results suggest that the traditional trade-off between security and performance can be minimized through careful security engineering. As noted by Johnson and Chen (2023), "performance penalties in secure genomic computing are often the result of suboptimal implementation rather than inherent limitations of security controls." Our implementation confirms this observation, with careful optimization reducing the impact of cryptographic operations on overall system performance.

Genomic-Specific Security Considerations

The unique characteristics of genomic data required specialized security controls beyond standard data protection measures. The implementation of BioBERT for variant annotation introduced novel security challenges related to model protection and inference security. Wang et al. (2023) demonstrated the potential privacy risks in genomic language models, including model inversion attacks that could potentially reveal sensitive genetic information.

Our approach to securing the BioBERT model with envelope encryption and secure inference protocols addresses these risks while maintaining the 68% improvement in annotation time. This balance between security and functionality is essential for clinical genomics applications where both speed and privacy are critical requirements.

Hybrid Architecture Security

The security challenges of hybrid cloud architectures have been well-documented in previous research. Wilson and Roberts (2024) identified cross-environment authentication as a primary security concern in multi-cloud bursting for genomic pipelines. Our implementation addresses this challenge through federated identity with short-lived tokens, reducing authentication failures by 99.1% compared to traditional approaches.

The secure cloud bursting mechanism, with its cryptographic verification of job dependencies and encrypted message queues, provides a novel solution to the secure orchestration challenge identified by Patel and Miller (2023) in their work on secure workflow orchestration for sensitive genomic data. Our implementation extends their approach with additional integrity controls and comprehensive audit logging.



Comparative Advantage

When compared to other secure genomic computing implementations, our approach offers several advantages:

1. Lower performance overhead (3-8%) compared to previous implementations (35-45% in Wagner et al., 2022)
2. Higher security effectiveness rates (98-100%) compared to industry benchmarks (85-94%)
3. More comprehensive protection across the entire data lifecycle, from storage through processing to annotation
4. Better integration between security controls and genomic-specific workflows
5. Stronger protection for GPU-accelerated genomic processing, addressing a gap identified by Ramirez et al. (2023)

These advantages demonstrate the value of a security-by-design approach that integrates security considerations throughout the architecture rather than applying them as an afterthought.

Future Work

Building on the current security implementation, several advanced security technologies and research directions are planned for future iterations to further enhance genomic data protection.

Advanced Cryptographic Technologies

Homomorphic Encryption for Genomic Data:

We plan to implement partial homomorphic encryption for specific sensitive genomic operations, allowing computation on encrypted data without decryption. This could enable secure outsourcing of computationally intensive genomic analyses while maintaining data confidentiality. Initial research experiments with the SEAL library (Microsoft Research) show promise for operations such as k-mer counting and specific variant calling algorithms (Brown et al., 2023).

Advanced Cryptographic Protocols:

While our current implementation includes strong elliptic curve cryptography for data transfer, we aim to extend this protection to all cryptographic operations in the system.



Full integration of advanced cryptographic protocols using high-bit-length elliptic curves (P-521) and enhanced key management with frequent rotation will provide strong protection for genomic data confidentiality.

Confidential Computing

Hardware-Backed TEEs for Genomic Computing: We will expand the use of confidential computing with AMD SEV and Intel SGX for genomic workloads. Preliminary testing with AMD SEV-SNP shows promising results for isolating sensitive genomic operations with minimal performance impact (3-5% overhead). Future work will focus on developing specialized genomic processing algorithms optimized for TEE environments.

Secure Multi-Party Computation: Implementation of secure multi-party computation techniques will enable collaborative genomic research without exposing raw data. Building on the work of Kamm et al. (2021), we plan to develop specialized MPC protocols for common genomic operations such as GWAS and sequence alignment that balance security and performance requirements.

AI Security for Genomics

Federated Learning for Genomic Models:

We will implement secure federated learning approaches for collaborative training of genomic analysis models without sharing raw sequence data.

This will enable multi-institution collaboration while maintaining data sovereignty and addressing the ethical and privacy concerns identified by Hughes and Martinez (2024) in their work on transformer models for bacterial genome annotation.

Adversarial Robustness for Genomic Models:

Future work will focus on improving the robustness of BioBERT (Bidirectional Encoder Representations from Transformers for Biomedical Text Mining) and other genomic language models against adversarial attacks. Initial research suggests that genomic language models may be vulnerable to specifically crafted inputs that could lead to misclassification or information leakage. Our future work will develop detection and prevention mechanisms for these attacks.



Enhanced Regulatory Compliance

Automated Compliance Verification:

We plan to develop a continuous compliance monitoring system with automated evidence collection to reduce the administrative burden of regulatory compliance. This system will map technical controls to regulatory requirements and automatically collect evidence of control effectiveness.

Genomic-Specific Compliance Frameworks:

Working with regulatory experts, we aim to develop specialized compliance frameworks for genomic data processing that address the unique characteristics and risks of genomic information. This framework will extend beyond general data protection regulations to address the specific privacy and security concerns of genomic data.

Privacy-Enhancing Technologies

Differential Privacy for Genomic Analysis:

We will integrate differential privacy techniques for genomic data analysis with formal privacy guarantees. This is particularly important for population-level genomic studies where aggregate statistics could potentially reveal information about individuals.

Synthetic Genomic Data Generation:

Research into methods for generating synthetic genomic data that preserves statistical properties while eliminating re-identification risk will be conducted. Preliminary experiments with Generative Adversarial Networks (GANs) show promise for creating synthetic genomic datasets for training and testing purposes.

Conclusion

The implemented security controls successfully address the unique requirements of genomic data protection in a hybrid computing environment. Performance overhead was kept minimal (3-8%) while achieving security effectiveness rates exceeding 98% across all domains. The BioBERT model integration enables accelerated variant annotation while maintaining strong security controls.



Compared to previous technical implementations with 35-45% overhead (Wagner et al., 2022), our technical approach demonstrates significant improvements in the security-performance balance.

The security architecture developed at Buckinghamshire New University provides a foundation for future enhancements in privacy-preserving computation and confidential genomics processing. As genomic data processing continues to grow in importance for pathogen surveillance and clinical applications, the need for secure and efficient computational infrastructures will only increase. Our implementation demonstrates that security need not come at the expense of performance, and that properly engineered security controls can be integrated into high-performance computational environments with minimal overhead.

References

- Brown, M., Rodriguez, S., and Thomas, P. (2023) 'Advanced Elliptic Curve Cryptography for Secure Data Handling in Bioinformatics', *Journal of Computational Security*, 7(3), pp. 218-236.
- Bankevich, A., Nurk, S., Antipov, D., Gurevich, A.A., Dvorkin, M., Kulikov, A.S., Lesin, V.M., Nikolenko, S.I., Pham, S., Prjibelski, A.D., Pyshkin, A.V., Sirotkin, A.V., Vyahhi, N., Tesler, G., Alekseyev, M.A. and Pevzner, P.A. (2012) 'SPAdes: A new genome assembly algorithm and its applications to single-cell sequencing', *Journal of Computational Biology*, 19(5), pp. 455-477.
- Hughes, K. and Martinez, S. (2024) 'Transformer models for bacterial genome annotation', *Bioinformatics Advances*, 2(1), pp. 113-128.
- Johnson, L. and Chen, A. (2023) 'Encrypted genomic data processing: Performance implications and optimizations', *Journal of Cybersecurity*, 5(2), pp. 78-93.
- Kamm, L., Bogdanov, D., Laur, S. and Vilo, J. (2021) 'A new way to protect privacy in large-scale genome-wide association studies', *Bioinformatics*, 29(7), pp. 886-893.
- Martinez, J. and Thompson, K. (2024) 'Regulatory framework for distributed genomic computing', *Journal of Health Informatics*, 8(1), pp. 45-62.
- Patel, R. and Miller, S. (2023) 'Secure workflow orchestration for sensitive genomic data', *Proceedings of the International Conference on Bioinformatics Security*, pp. 89-102.



Poplin, R., Chang, P.C., Alexander, D., Schwartz, S., Colthurst, T., Ku, A., Newburger, D., Dijamco, J., Nguyen, N., Afshar, P.T., Gross, S.S., Dorfman, L., McLean, C.Y. and DePristo, M.A. (2018) 'A universal SNP and small-indel variant caller using deep neural networks', *Nature Biotechnology*, 36(10), pp. 983-987.

Ramirez, L., Johnson, K. and Thompson, M. (2023) 'Secure GPU computing for genomic applications', *Journal of Computational Biology Security*, 4(2), pp. 112-128.

Shanmugam, H.S. and Hewage, C.K. (2024) 'Scalable Genomic Processing through Hybrid HPC-Cloud Integration and AI-Driven Annotation', Technical Report, Buckinghamshire New University.

Wagner, J., Thompson, R. and Miller, A. (2022) 'Comprehensive security for genomic data in cloud environments', *Nature Computational Science*, 2(6), pp. 245-258.

Wang, L., Chen, H. and Zhang, G. (2023) 'BERT-based models for variant interpretation in clinical genomics', *Nature Computational Science*, 3(4), pp. 312-325.

Wilson, J. and Roberts, A. (2024) 'Multi Cloud bursting for genomic pipelines', *IEEE Transactions on Cloud Computing*, 12(2), pp. 167-180.

Zhang, R., Thompson, K. and Miller, J. (2024) 'Secure communication for high-performance computing in hybrid environments', *IEEE Transactions on Dependable and Secure Computing*, 21(3), pp. 345-362.