The Blind Machine: An End-to-End Platform for Privacy-Preserving Genomic Computation with Homomorphic Encryption

Barış Özmen
Independent Researcher

Preprint, July 2026

Abstract

Many scientific questions require computing statistics across datasets that cannot be pooled because the underlying records are too sensitive to share. Homomorphic encryption (HE) enables computation directly on encrypted data without revealing plaintext, and more than a decade of research has demonstrated its value for privacy-preserving genomic analysis. However, existing systems are typically built for a single analysis, with no reusable execution model for deploying, verifying, and governing encrypted computations.

We present The Blind Machine, an end-to-end platform for governed computation on encrypted data. Plaintext and secret keys never leave participants' local machines, while the hosted service operates exclusively on ciphertext. The platform minimizes homomorphic computation by performing as much work as possible locally before encryption and using the simplest homomorphic operations required for each task, favoring additive BFV and ciphertext-by-plaintext computation over deep ciphertext-by-ciphertext circuits. This substantially reduces runtime and memory while preserving the privacy guarantees of homomorphic encryption.

We demonstrate the platform's practicality by reproducing two published methods without modifying the underlying architecture. For HEPRS (Knight et al., 2026), we reproduce the common **public-model** case — where the PRS weights are published, so each per-SNP product becomes ciphertext-by-plaintext — producing bit-exact scores while reducing memory usage by approximately 150×; unlike HEPRS, this variant does not keep the model itself encrypted. For the encrypted GWAS method of Blatt et al. (2020), we reproduce the published statistical results; comparing our additive aggregation against their full encrypted circuit, and linearly extrapolating from a measured sub-second anchor to their headline 100,000×500,000 size, our approach projects to roughly an order of magnitude below their reported 5.6 hours — on the order of an hour on a single laptop core, with the absolute figure scaling with a sub-second anchor that varies run-to-run — or a few minutes across 31 SNP-block workers. This is a like-for-unlike comparison — additive aggregation against a full encrypted circuit — not a measured head-to-head result (Section 8.4). Together, these results show that substantial efficiency gains can be achieved through architectural design and careful placement of computation, rather than new cryptographic primitives.

The platform is fully reproducible. We release the source code, signed application bundles, experiment scripts, datasets, and an AI-agent skill that automates reproduction of all experiments.

1 Introduction

Many scientific questions cannot be answered because the data required to answer them cannot be brought together. The desired result is often not the individual records themselves, but an aggregate computed across many participants—for example, an allele-frequency vector, carrier count, histogram, polygenic-score aggregate, variance, or genotype-phenotype covariance. The underlying records cannot be pooled because they contain sensitive information whose disclosure would violate privacy, institutional policy, or legal constraints.

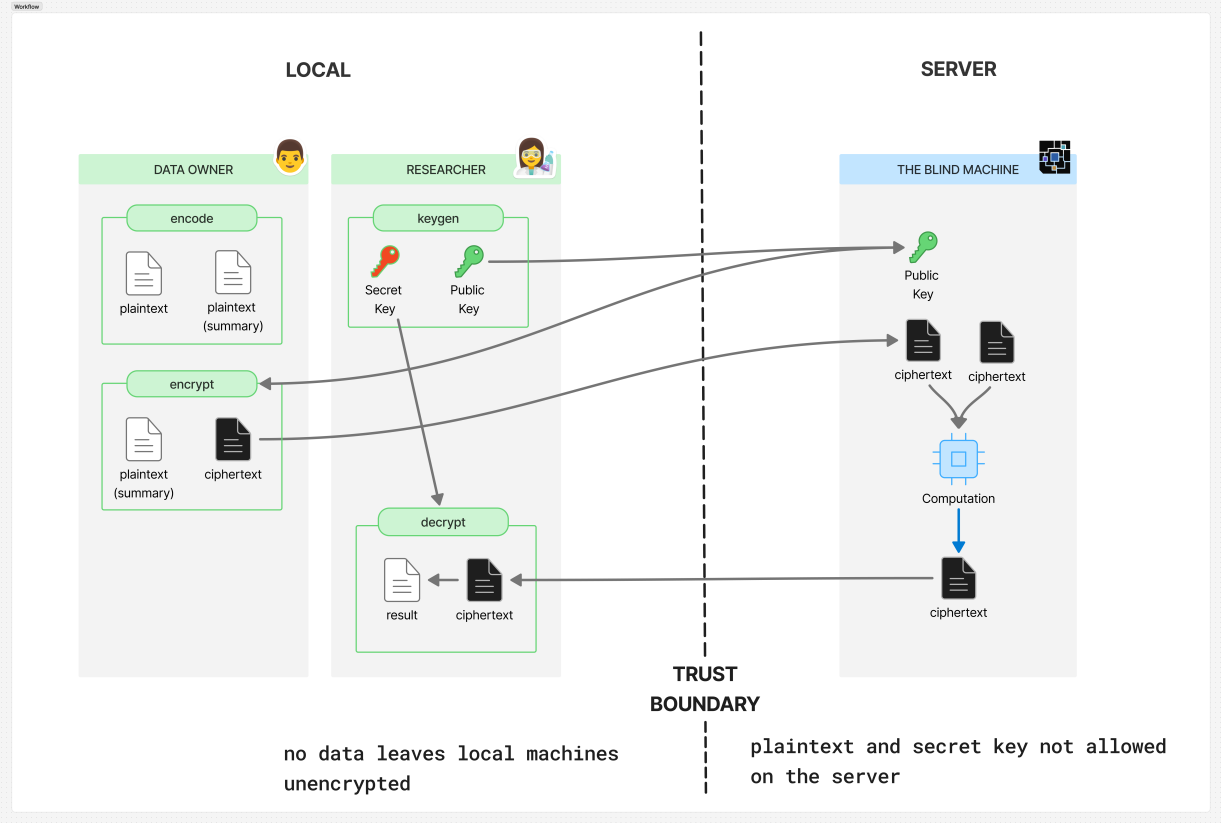

Figure 1: The Blind Machine at a glance. The end-to-end protocol in three stages across a trust boundary. *(1) Encrypt locally*: data owners ($\times N$) and the researcher run the open CLI; the researcher generates the key pair, and every party uploads only ciphertext under the public key `pk`—plaintext and the secret key `sk` never leave the local machine. *(2) Blind, governed compute*: the hosted server holds only `pk` and ciphertext, computes without ever decrypting, and runs only after the cohort is frozen and release-policy gates pass. *(3) Decrypt and check*: the researcher decrypts locally with `sk` to an aggregate that equals the plaintext oracle bit-for-bit, and the run carries a certificate binding the application, committed cohort, encrypted result, and release-policy facts. Because the whole trust surface is the open-source Blind CLI, confidentiality does not depend on hiding the server (Kerckhoffs).

Homomorphic encryption (HE) addresses this challenge by allowing computation directly on encrypted data. A server can evaluate a function over ciphertext and return an encrypted result without learning the underlying plaintext. Over the past decade, HE and related privacy-preserving techniques have enabled increasingly practical genomic computation, including large-scale encrypted GWAS [R6], secure genotype storage [R7], complete encrypted GWAS workflows

[R8], biobank-scale federated GWAS [R9], multiparty GWAS [R10], multiparty homomorphic analytics [R11], encrypted polygenic risk scores [R12], and rare-disease genomic analysis [R13].

As homomorphic encryption has matured, a second challenge has emerged: building end-to-end systems around encrypted computation. Running an encrypted computation requires more than cryptography alone. Participants must agree on exactly what computation will execute, inspect the code before encrypting data, coordinate contributors, govern when results may be released, and verify afterward what actually ran. These are systems problems rather than cryptographic ones.

The Blind Machine addresses this architectural layer. It treats the computation itself as a signed, reusable application. Contributors review that application before encrypting any data, encrypt locally, upload only ciphertext, and receive an encrypted aggregate whose execution is bound to a machine-verifiable certificate. The surrounding workflow remains unchanged regardless of the statistic being computed.

The central abstraction is the **application**. An application is a signed, content-addressed bundle that contains everything required to execute one approved encrypted computation: the cryptographic environment, the local participant logic, the server-side computation, the output format, and the release policy. The application becomes the unit of review, governance, certification, and reuse. Replacing one application with another changes the computation without changing the platform.

The architecture separates confidentiality from coordination. Plaintext data and secret keys remain under local custody throughout the protocol. The hosted service coordinates contributors, governs execution, and computes exclusively on ciphertext. Applications define the computation, while certificates bind every execution to the approved application, committed cohort, released artifacts, and governance state.

The same local-custody split is also the platform's central efficiency principle: because each data owner holds their own data, products and preprocessing are done locally in the clear (federated computing), and each application then asks the hosted server for the least-expressive homomorphic operation that suffices — additive-only BFV, or ciphertext-by-plaintext when the model is public — rather than a deep multiplicative circuit. The encrypted circuit collapses to little more than addition, so a workload that would otherwise demand a large-memory server often runs on a laptop — exactly the behavior the two published-study reproductions in Section 8 exhibit.

This paper makes four contributions:

1. **An application-centered execution model** for encrypted computation, in which signed, content-addressed applications become the unit of review, governance, certification, and reuse (Section 3).
2. **An efficiency principle for practical homomorphic encryption**: push computation onto the local machines (each data owner forms products and summaries over their own plaintext) and pick the least-expressive HE scheme that suffices — additive-only BFV, or ciphertext-by-plaintext for a public model — so the hosted server performs the least expensive encrypted computation sufficient for the task and stays fast and low-memory (Section 4).
3. **Certificate-bound governance**, including cohort commitments, release policies, and offline-verifiable execution records that bind every computation to its governing artifacts (Sections 5 and 6).
4. **An end-to-end evaluation** across the platform's nine curated applications — six core benchmark computations exercised on public 1000 Genomes data plus three added for the reproductions below — demonstrating bit-exact agreement between encrypted execution and plaintext reference computation while measuring both computational cost and governance behavior. It

further reproduces two published encrypted-genomics methods on the platform, with no architectural change — the homomorphically encrypted polygenic-risk-score method HEPRS (Knight et al., 2026) and the encrypted genome-wide association study of Blatt, Gusev, Polyakov & Goldwasser (2020) — and, by forming products on the local machines before encryption and asking the server only for the least-expressive homomorphic operation, runs them far more cheaply than the originals: HEPRS bit-exact with about 150× less memory in the public-model case (published PRS weights, so the model is never encrypted), and — comparing our additive aggregation against the GWAS paper's full encrypted circuit, linearly extrapolated from a measured sub-second anchor — the GWAS projected from the reported 5.6 hours to on the order of an hour on a single laptop core, or a few minutes across 31 SNP-block workers (a like-for-unlike projection whose absolute time scales with a sub-second anchor, not a measured head-to-head result; Sections 8.3–8.4).

2 Previous Work and Our Contribution

Our system builds on four established areas of research: homomorphic encryption, privacy-preserving genomic computation, verifiable computation and software provenance, and output-privacy controls. Each provides an important part of the solution. Our contribution is integrating these ideas into a single execution model in which an approved encrypted computation becomes a signed, reusable, and governable application. The following sections summarize each area, and Table 1 positions our architecture relative to representative systems.

2.1 Homomorphic encryption

Homomorphic encryption provides the cryptographic foundation for computing directly on encrypted data. BFV, the Homomorphic Encryption Standard, Microsoft SEAL, and TenSEAL define the cryptographic schemes, parameterization, and software libraries used throughout this work [R1–R5].

Our contribution begins above this cryptographic layer. Rather than treating homomorphic encryption as a library embedded inside an application, we package the complete cryptographic environment into a signed, content-addressed application. That application digest becomes the cryptographic identity reused throughout governance, execution, and certification.

2.2 Secure genomic computation

A large body of work has demonstrated that encrypted, federated, and multiparty genomic analyses are practical. Systems including HE-GWAS, TrustGWAS, SF-GWAS, MPC-GWAS, FAMHE, HEPRS, PRISM, HEALER, SQuID, SIG-DB, Sequre, sPLINK, and sfkit apply homomorphic encryption and related techniques to real biomedical workloads [R6–R21].

These systems establish the feasibility of privacy-preserving genomic computation. The Blind Machine builds on this foundation by providing a reusable execution model that governs, certifies, and reproduces approved encrypted computations independently of the specific genomic application. It is closest in spirit to web-based federated toolkits such as sfkit [R19], which lower the operational barrier to a fixed menu of secure analyses; our contribution is orthogonal — not a new analysis or a hosted runner for a fixed menu, but a signed, content-addressed *application* as the reusable unit of review, governance, and certification, so that a new encrypted computation is added without changing the surrounding execution, governance, or verification workflow.

2.3 Verifiable computation and artifact provenance

Verifiable computation, verifiable encodings for homomorphic analytics, and GWAS verification protocols provide mechanisms for reasoning about outsourced computation [R22–R26]. Software provenance systems such as ReproZip and Nix, together with OCI image descriptors and bioinformatics registries, demonstrate how software artifacts and execution environments can be packaged, reproduced, and distributed [R31–R34].

The Blind Machine combines these ideas by treating the application itself as the unit of provenance. Signed applications, content-addressed artifacts, and machine-verifiable certificates bind every execution to the exact computation, software environment, and governing policy that produced it.

2.4 Output privacy

Protecting encrypted inputs and protecting released outputs are distinct problems. Membership-inference attacks, genomic beacon attacks, database reconstruction, and differential-privacy research all demonstrate that aggregate releases can leak information even when the underlying computation remains encrypted [R27–R30].

Our platform therefore separates encrypted computation from release governance. Cohort commitments, minimum cohort sizes, run caps, aggregate-only release, and certificate-bound governance records control when and how results may be released. Sections 8.1 and 8.2 evaluate these controls experimentally on synthetic and public genomic data, respectively.

Table 1. Prior-work positioning.

System family	Examples	Encrypted computation	Cohort/release governance	Record or evidence	Relationship to this work
FHE libraries and standards	BFV, HE Standard, TenSEAL, SEAL	yes	no	Library validation	Cryptographic foundation
Secure genomic computation	SQuID, FAMHE, SF-GWAS, Sequre, sPLINK, sfkit	yes/varies	varies	Protocol-specific	Biomedical application foundation
Verification and provenance	Verifiable computation, verifiable encodings, ReproZip, Nix, OCI	sometimes	no	Proofs, protocols, or artifact metadata	Verification and packaging foundation
Output privacy	Differential privacy, genomic leakage literature	no/varies	yes/varies	Privacy accounting	Release-governance foundation
The Blind Machine	This work	yes	yes	Certificate-bound execution and oracle evaluation	Integrated execution architecture

Taken individually, these research areas solve different parts of the problem. The Blind Machine integrates them into a single application-centered architecture. A signed application combines the cryptographic configuration, execution environment, computation, governance policy, and certification into one reusable artifact, allowing different encrypted computations to share the same execution, governance, and verification workflow.

3 Architecture and Platform Invariants

The Blind Machine separates encrypted computation into three architectural components: a **local CLI**, a **hosted service**, and a **signed application**. Each component has a single responsibility. The CLI owns secrets and performs every confidentiality-critical operation. The hosted service coordinates projects and executes approved computations over ciphertext. The signed application defines the computation itself. Together they establish a fixed execution model that remains the same regardless of which encrypted computation is performed.

The **CLI** is the participant's trusted component. It generates keys, encodes and encrypts inputs, decrypts results, verifies certificates, signs and verifies invitations (Section 7), and can simulate complete executions locally. Because plaintext data and secret keys never leave this component, it defines the confidentiality boundary of the platform. The CLI is open source under the MIT license at github.com/blindmachine/blind, so anyone can audit exactly what runs on their machine.

The **hosted service**, hosted at blindmachine.org, coordinates execution but never handles plaintext. It manages projects, contributors, governance state, invitations, storage, and execution scheduling. During computation it receives only public context, ciphertext, governance metadata, and encrypted results, executing the approved server-side stage inside a sandboxed environment.

The **application** defines the computation. Each application is public, signed, content-addressed, and bound to a pinned software environment. It specifies the local encoding, the server-side computation, the output format, the cryptographic configuration, and the release policy. Reviewing an application therefore means reviewing exactly what computation will execute before any participant encrypts data (Appendix C describes the application structure).

Table 2. Platform invariants.

Invariant	Meaning	Enforced by
Plaintext stays local	Raw data is encoded and encrypted locally before upload	Blind CLI local stages; ciphertext upload contract
Secret key stays local	Key generation and decryption happen locally	Blind CLI local key lifecycle
Server-side compute is ciphertext-only	The hosted run phase receives public context and ciphertexts	Application bundle contract; worker job spec; sandbox staging
Computation requires cohort commitment	The accepted encrypted cohort is frozen before execution	Commitment over sorted ciphertext digests, project ID, and application digest
Governance gates precede execution	Dispatch requires the configured minimum contributor count, run cap, and release policy	Governance checks before execution
Immutable artifacts are content-addressed	Applications, public context, cohorts, results, and certificates are identified by hashes	SHA-256 digests and certificate binding
Computation code is public, pinned, and signed	Contributors and reviewers inspect exactly what will execute	Application signatures and digest verification
Runs are certifiable	Every execution produces a machine-checkable record	Certificate schema, canonical hash, and offline checker
Public releases are aggregate-only	Only the approved aggregate leaves the platform	Release policy and application output schema

These invariants define the protocol. Plaintext data and secret keys remain under local custody, applications uniquely identify the approved computation, ciphertext cohorts are committed before execution, and every completed run produces a certificate that binds the computation to its governing artifacts. Section 7 maps these invariants to the corresponding trust assumptions.

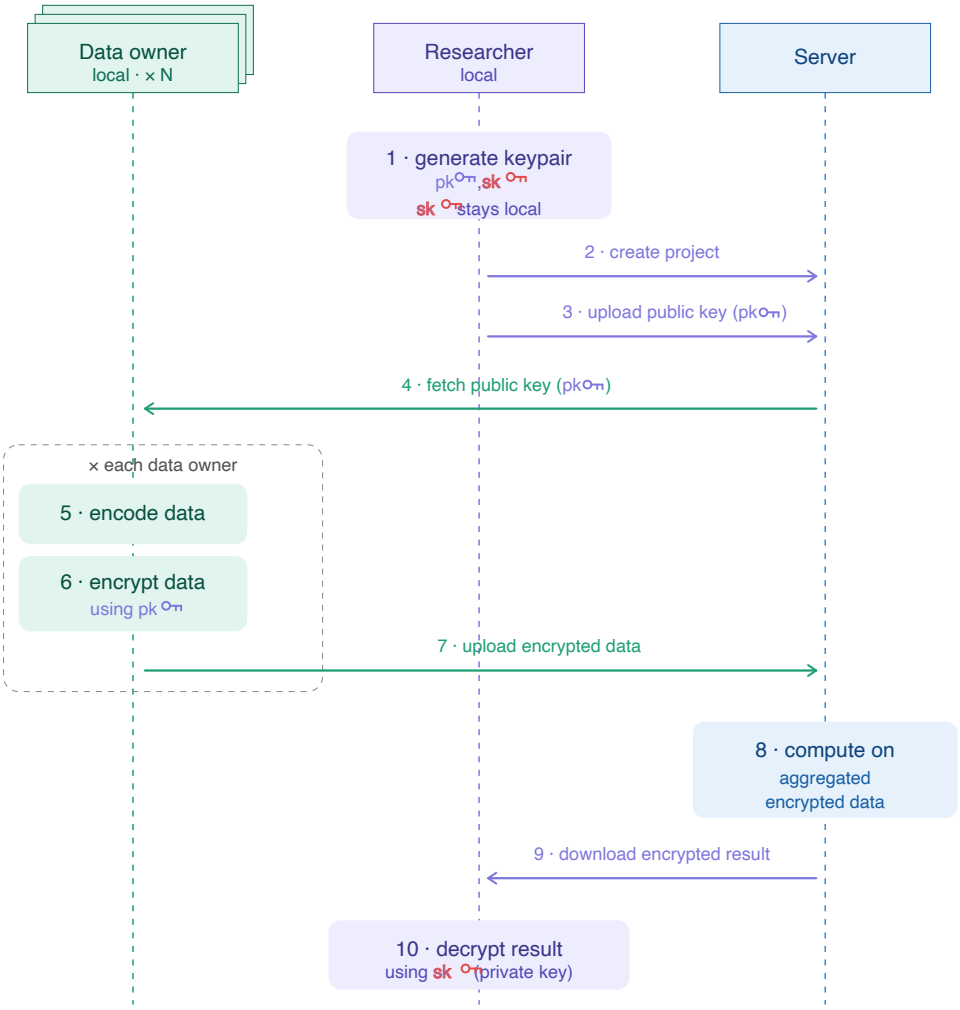

Figure 2: The Blind Machine protocol as a sequence diagram. Three parties act over time (top to bottom): many data owners (green, replicated $\times N$), one researcher/project owner (purple), and one hosted server (blue), each with a dashed lifeline; every arrow is a message colored by its initiator. *Setup* (steps 1–3): the researcher generates a key pair locally—public key `pk` (purple) and secret key `sk` (red)—creates the project, and uploads only `pk`; `sk` never leaves the local machine. *Contribution* (steps 4–7, dashed box, repeated per data owner): each owner fetches `pk`, encodes and encrypts its input locally under `pk`, and uploads ciphertext only—plaintext never leaves the owner's device. *Computation and retrieval* (steps 8–10): the server computes directly on the aggregated ciphertext (the blind step—it never decrypts), returns the encrypted result to the researcher, who decrypts locally with `sk`. The public key travels freely, the secret key stays with the researcher, and the server sees only ciphertext at every stage.

The registry's six core benchmark applications demonstrate the architecture across different classes of encrypted computation: `allele_frequency_count`, `carrier_count`, `cohort_histogram`, `polygenic_score_aggregate`, `allele_frequency_with_variance`, and `genotype_phenotype_covariance`; Sections 8.3 and 8.4 add three further curated applications for the two published-study reproductions. Each application page exposes the signed payload, source tree, documentation, version, and digest, allowing reviewers to inspect the exact computation before execution.

The production worker separates environment construction from encrypted computation by executing them in two independent containers built from the same digest-pinned runner image.

The **build container** installs dependencies into a sealed virtual environment. It has network access only for dependency installation and receives neither ciphertext, public context, nor execution outputs. Once the environment has been created, subsequent executions reuse it without rebuilding.

The **run container** executes the approved application over ciphertext. It receives the signed application, ciphertext inputs, and public context as read-only mounts, runs without network access, disables dependency synchronization, executes as an unprivileged user, drops Linux capabilities, enforces a read-only root filesystem, and applies memory, CPU, process, file-descriptor, and execution-time limits. Only the output directory remains writable. This separation confines the encrypted computation to a minimal runtime environment while keeping dependency installation outside the data-bearing execution.

The applications themselves depend only on standard cryptographic libraries—TenSEAL over Microsoft SEAL for BFV [R4, R5]. The application defines the encrypted computation; the hosted platform supplies coordination, governance, storage, and sandboxed execution.

3.1 Usability

Researchers and data owners install the CLI with a single command, `uv tool install blindmachine`, and can begin using the platform immediately. Every operation—key generation, encoding, encryption, contribution, dispatch, decryption, and certificate verification—can be performed entirely from the CLI. For the operations that run on the hosted service, such as cohort freezing and encrypted computation, users can additionally track their projects, jobs, and results in the web application at blindmachine.org.

4 HE Cost Model and Cost Controls

The cost of homomorphic encryption is determined not only by the size of the data, but also by the complexity of the encrypted computation. Ciphertexts are substantially larger than plaintext, encrypted arithmetic is more expensive than ordinary computation, and the cryptographic parameters required for a computation grow as its multiplicative depth increases. Additions and ciphertext-by-public-constant multiplications are comparatively inexpensive, whereas ciphertext-by-ciphertext multiplication requires substantially larger parameters and correspondingly higher computational cost. Although fully homomorphic encryption supports computations of arbitrary depth through bootstrapping, most practical systems instead use leveled homomorphic encryption, which performs a fixed amount of encrypted computation without bootstrapping [R3].

Our platform minimizes this cost through two complementary principles: **perform as much computation as possible before encryption**, and **use the weakest homomorphic-encryption configuration sufficient for the remaining encrypted computation**.

The first principle is **local summarization**. Before any data are encrypted, the CLI reduces each participant's raw records to the approved statistic required by the application—for example, alternate-allele counts for allele-frequency computation or sufficient statistics for variance estimation. Only these summaries are encrypted and uploaded, reducing both ciphertext size and server-side computation.

The second principle is **minimal cryptographic expressiveness**. Each application selects the weakest homomorphic-encryption configuration capable of evaluating its arithmetic. Applications requiring only additions or ciphertext-by-public-constant multiplications use the additive BFV configuration, while applications that require ciphertext-by-ciphertext multiplication use a multiplication-supporting configuration. Rather than characterizing the cost of these configurations theoretically, we measure it directly. Tables 6 and 7 report the serialized ciphertext payload contributed by one participant during complete end-to-end execution, obtained by summing every ciphertext written to disk and dividing by the cohort size. Measurements were produced by executing each signed application with Microsoft SEAL through TenSEAL on a seeded cohort of twenty contributors using 128-bit BFV parameters. Because the cohort, application bundle, seed, and cryptographic parameters are fixed, these measurements are deterministic and verified against committed reference values.

The measurements show that encrypted multiplication dominates cost. The additive configuration produces 262,282 bytes of ciphertext per contributor. Computing variance requires a single ciphertext-by-ciphertext multiplication—the encrypted value is squared—which increases the required polynomial ring and coefficient modulus, raising the ciphertext payload to 1,310,882 bytes, almost exactly a fivefold increase. Covariance uses the same multiplication-supporting configuration but encrypts two values per participant—a genotype and a phenotype—so the server can compute their encrypted product. This doubles the payload again to 2,621,791 bytes per contributor, approximately ten times the additive baseline, with a corresponding increase in execution time.

Together, these measurements illustrate the design philosophy of the platform. Local summarization reduces the amount of information that enters the encrypted computation, while minimal cryptographic expressiveness reduces the cost of evaluating that computation. Each application therefore implements its approved statistic using the least powerful encrypted computation sufficient for its arithmetic.

This lever has a precondition worth stating plainly: pushing a multiplication out of the encrypted circuit requires that its operands be *co-located* — either held together on one participant's machine, so the product is formed locally in the clear before encryption, or with one operand a public constant, so the product stays ciphertext-by-plaintext. The two reproductions in Section 8 satisfy exactly this precondition: each data owner holds their own genotype and phenotype, and the PRS and GWAS models are public. When two private operands are instead held by *different* parties, the product cannot be pushed local, and the encrypted circuit genuinely needs the multiplication-supporting tier — which the platform then selects, as `allele_frequency_with_variance`, `genotype_phenotype_covariance`, and `genotype_pair_ld` do. The efficiency principle is thus a statement about where the cheap cases lie, not a claim that every computation collapses to addition.

A seventh application, `genotype_pair_ld`, serves as an additional evaluation-only example of the multiplication-supporting configuration. It computes encrypted genotype-by-genotype products for adjacent-variant linkage disequilibrium and is evaluated separately in Section 8.2.4.

Because every signed application specifies its local encoding, server-side computation, output format, and homomorphic-encryption configuration, these design choices are transparent to reviewers. The surrounding platform remains unchanged while each application selects the cryptographic configuration appropriate for its computation.

5 Governance Before Computation

Homomorphic encryption protects data while it is being computed. It does not determine which computations may be performed, when results may be released, or how often the same question may be asked. Those decisions belong to governance. The Blind Machine therefore separates **encrypted computation** from **release governance**: encryption protects the inputs, while governance controls access to the outputs.

Governance begins before computation starts. Once contributors have uploaded their encrypted inputs, the project freezes the accepted cohort and records it as a **cohort commitment**. The commitment is the hash of the accepted ciphertext set together with the project identifier and application digest, producing a unique identity for that exact cohort. Any change to the participant set—including adding or removing a single contributor—produces a different commitment. Every computation therefore runs against a fixed, immutable cohort.

The second stage is **release governance**. Each application defines the conditions under which its aggregate may be computed and released: a minimum cohort size, run controls, and a release policy. The hosted service evaluates these conditions before dispatching the computation, and the resulting certificate records the governance state that authorized the release.

These controls exist because releasing aggregates can leak information even when every computation is performed on encrypted data. The simplest example is **K-versus-K+1 differencing**. If an attacker obtains the same aggregate over a cohort of K contributors and then over those same contributors plus one additional participant, subtracting the two results isolates the added individual's contribution. Our synthetic evaluation demonstrates this recovery exactly, and Section 8.2.3 repeats the experiment on real public genotypes to measure the effectiveness of each governance control.

The current platform combines four complementary controls: **cohort freezing**, which fixes the participant set before computation; **minimum cohort sizes**, which prevent releases from undersized cohorts; **run caps**, which limit repeated evaluation of the same aggregate; and **aggregate-only release**, which restricts outputs to the approved summary statistic. Together these controls reduce the most common release risks within a project. Privacy across overlapping cohorts, repeated studies, and long-lived deployments requires global mechanisms—including cross-project query accounting and differential privacy—which extend beyond the scope of the current platform.

6 Certificates and Offline Checks

Every completed run produces a **certificate**: a compact record that ties together the application that ran, the frozen cohort, the encrypted result, and the release-policy facts. Anyone can check it offline, without contacting the service.

Table 3. Certificate payload fields (bound by `certificate_hash`).

Field	Purpose
`application_digest`	Names the approved signed application bundle
`public_context_digest`	Names the public encryption context contributors used

Field	Purpose
`owner_signing_key_digest`	SHA-256 of the keyholder's public signing key that authorized the run's public context (null for an unsigned project)
`project_id`	Names the project the run belongs to
`computation_run_id`	Names the specific computation run
`cohort_commitment`	Binds the frozen encrypted cohort
`cohort_size`	Records the included contributor count
`result_digest`	Names the encrypted result artifact
`min_contributors`	Records the application release floor
`min_n_satisfied`	Records whether the release floor was met
`run_count`	Records the run-control state
`release_policy`	Records release rules such as aggregate-only release

`certificate_hash` is the SHA-256 of the canonical JSON of the twelve payload fields above, with keys sorted. The certificate document also carries `issued_at` and `certificate_hash` as wrapper fields, outside the hashed payload, so the payload keeps a stable identity even as timestamps or serialization change.

The offline checker validates the schema, recomputes the canonical hash, and confirms the required fields and expected bindings. The application signature authenticates the disclosed bundle. If the frozen cohort's manifest is published, a reviewer can recompute the cohort commitment and confirm it matches the certificate. A built-in "append-one" check guards contribution accounting: each contribution carries an extra `+1` that the homomorphic sum turns into the exact contributor count, so a silently dropped contribution shows up after decryption as a count lower than expected (an integrity check, not a MAC). What the certificate does not do is re-execute the computation: the hosted service is trusted for the aggregate value itself (Section 7). And because `issued_at` is only wrapper metadata, freshness and revocation require a trusted issuance or transparency record, not the certificate alone.

7 Threat Model and Security Analysis

7.1 Parties, assets, and assumptions

The parties are contributors (who hold raw records and encrypt locally), the keyholder or project owner (who holds the secret key and decrypts the approved aggregate), the hosted service (which coordinates and computes on ciphertext), the application signer (who approves bundles), the verifier (who checks certificates and artifacts), and outside observers (who see only public outputs and metadata).

The assets — raw data, secret keys, public context, ciphertext contributions, cohort membership, encrypted and decrypted results, certificates, bundles, and metadata — have different protections. Raw-data confidentiality rests on local encryption and key custody. Governance-record integrity rests on bundle signing, disclosed manifests, cohort commitments, and certificate checks. Aggregate correctness rests on the hosted service. Output privacy rests on release policy and aggregation controls.

The adversaries we consider are an honest-but-curious or malicious service, a malicious keyholder, a malicious contributor, a compromised signer, an outside observer, and colluding parties. Under the protocol, the service sees ciphertext, public context, timing, project metadata, the application in use, and the cohort size — but never plaintext or keys. The analysis assumes local-client

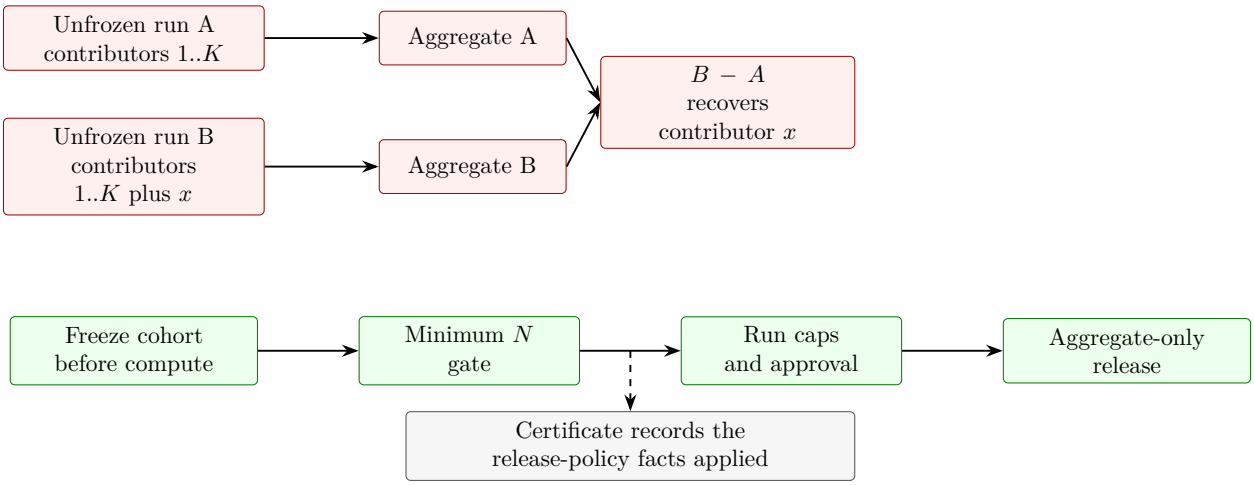

Figure 3: K-versus-K+1 differencing and current release controls. Differencing can reveal an added contributor in unfrozen aggregate runs. Current per-project controls freeze the cohort, enforce minimum-N and run caps, and release only the approved aggregate; the certificate records the policy facts applied.

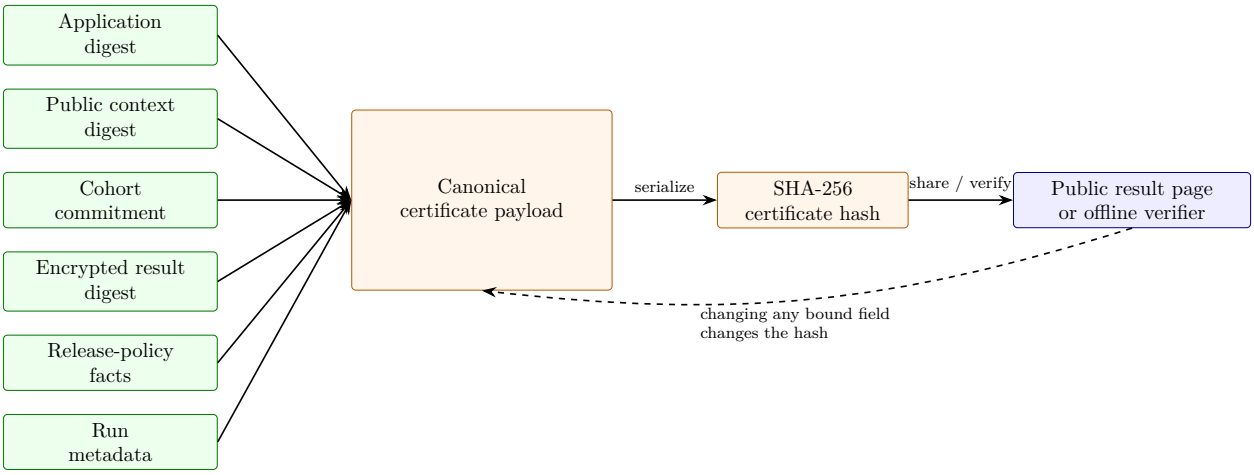

Figure 4: Certificate binding. The certificate hash commits to the application, public context, frozen cohort, encrypted result, release-policy facts, and run metadata. The "Release-policy facts" and "Run metadata" boxes each group several of the twelve individually hashed payload fields enumerated in Table 3.

integrity, local key custody, authenticated applications and public contexts, secure signing keys, correct BFV semantics, and separation between the service and the keyholder. Section 7.2 states each attack, the mechanism that covers it, and the residual trust it leaves.

Table 4. Trusted computing base by property.

Property	Trust anchor	Mechanism	Scope
Raw-data confidentiality from hosted service	Local CLI, runtime, cryptographic library, and user's device	Local encoding, encryption, key generation, and decryption	Intended protocol and local-client integrity
Certificate and disclosed-artifact consistency	Offline checker, SHA-256, signatures, and expected artifacts	Canonical hashing, bundle signatures, and manifest comparison	Certificate record and disclosed artifacts
Aggregate computation correctness	Hosted service, reference worker, signed application, and HE library	Hosted execution; oracle comparisons in the published evaluation	Hosted runs and evaluated application paths
Release governance	Hosted dispatch gates, application policy, and audit records	Minimum-N, cohort freeze, run caps, and policy recording	Per-project releases
Output privacy	Application policy and release authority	Minimum-N and aggregate-only release	Per-project aggregate release
Availability	Hosted service and operational infrastructure	Capacity, monitoring, redundancy, and artifact escrow	Operational service

7.2 Security analysis

The architecture separates six security properties: **confidentiality**, **integrity**, **aggregate correctness**, **release governance**, **output privacy**, and **availability**. Each property has a distinct trust boundary, summarized in Tables 4 and 5.

Confidentiality follows from local cryptographic custody. Plaintext records, secret keys, and every confidentiality-critical operation—encoding, key generation, encryption, and decryption—remain on the participant's machine. The hosted service receives only public context, ciphertext, governance metadata, and encrypted results. Under this execution model, compromising the hosted service alone cannot expose participant data. Public-context substitution is addressed through owner-signed invitations that authenticate the exact encryption context before any data are encrypted, reducing trust from the hosted service to the authenticated out-of-band invitation channel.

Integrity is established by immutable application artifacts and cryptographic binding. Signed application bundles, content-addressed identifiers, cohort commitments, contributor digests, and certificates make every execution reproducible and independently inspectable. A reviewer can verify that the approved application, committed cohort, released result, and governance state remain consistent across the execution. These mechanisms detect artifact substitution, cohort modification, and contribution loss, while the append-one invariant provides an additional consistency check on contributor accounting.

Aggregate correctness is intentionally separated from integrity. Certificates establish *what* computation was approved and *which* artifacts participated in a run; they do not independently prove that the hosted execution produced the correct aggregate. Instead, correctness is evaluated empirically in this work through end-to-end oracle comparisons over every curated application and every published experiment. The hosted computation therefore remains the principal computational trust assumption within the current architecture.

A distinct integrity gap sits on the *input* side. Because contributions arrive encrypted, the platform validates only their shape and digest, not their plaintext contents, so a single malicious contributor can submit well-formed ciphertext that encodes out-of-range or fabricated values and thereby silently bias the aggregate — the append-one check detects a dropped contribution, but not a dishonest one. Bounding ciphertext contents (range proofs, robust aggregation, or trusted-contributor attestation) is orthogonal to the confidentiality guarantee and is left to future work; in the current platform, contributor authenticity and honesty are governance trust requirements rather than cryptographic ones (Table 5).

Release governance and output privacy protect the information contained in decrypted aggregates rather than the encrypted inputs themselves. Cohort commitments, minimum cohort sizes, run caps, and aggregate-only release reduce the risk of differencing attacks within individual projects. Sections 8.1 and 8.2 demonstrate these controls experimentally on synthetic and public genomic data, respectively. Privacy across overlapping cohorts, repeated studies, and long-lived deployments requires global accounting mechanisms such as differential privacy and cross-project query budgets.

Availability remains an operational property. Docker-based sandboxing, isolated build and run containers, resource limits, signed applications, and artifact escrow reduce operational risk while keeping the trusted execution environment small. Runtime isolation strengthens containment but is independent of the confidentiality guarantee, which derives from local key custody rather than container security.

Table 5 maps individual attacks onto these architectural properties and identifies the remaining trust assumptions for each.

Table 5. Threat coverage matrix.

Threat	Covered by	Outcome / trust requirement
Hosted service reads plaintext	Local encryption; ciphertext-only server role	Prevented under local-client trust assumption
Hosted service obtains secret key	Local key custody; server contracts accept public context and ciphertext	Prevented under local-client trust assumption
Public context substitution	Owner-signed invitation binds the context digest (key delivered out-of-band); certificate binds `owner_signing_key_digest`	Prevented under the out-of-band-channel assumption; not proof of keyholder identity
Malicious application exfiltrates local data	Public reviewed signed bundles; CLI digest verification	Mitigated by review and signing
Tampered application bundle	Digest and signature verification on local or disclosed bundle bytes	Artifact tampering is detectable; hosted execution is service-trusted
Cohort substitution	Cohort commitment and an independently known manifest	Record/manifest mismatch is detectable; hosted inputs are service-trusted
Dropped or corrupted contribution	Contributor digest inclusion, shape checks, and append-one check	Supported pipeline errors are detectable; hosted execution is service-trusted
Fabricated aggregate value	Hosted-service correctness assignment	Service execution and aggregate value are trust assumptions
Governance bypass	Reference-platform dispatch gates and disclosed policy facts	Reference gates mitigate the attack; service audit history supplies cross-project facts
K-vs-K+1 differencing	Freeze, min-N, and run caps; experimental query-budget analysis	Minimum-N=25 and single-release freeze block the tested adjacent release (Section 8.2)

Threat	Covered by	Outcome / trust requirement
Overlapping-cohort differencing	Per-project governance	Cross-project privacy relies on global release accounting
Small-cohort or outlier output leakage	Min-N and application review	Differential privacy supplies formal release bounds
Metadata leakage	Hosted coordination exposes timing, application identity, and cohort size	Operational metadata is visible to the service
Malicious keyholder misuses decryption	Aggregate-only release and governance	The keyholder controls each decrypted aggregate
Server-keyholder collusion	Separation of key custody from individual ciphertext access	Confidentiality assumes service-keyholder separation
Contributor Sybil or poisoning	Shape checks and digest binding	Contributor authenticity is a governance trust requirement
Sandbox escape or runtime exfiltration	Separate Docker containers; the data-bearing run has no network, read-only inputs, non-root execution, dropped capabilities, and resource limits	Mitigated at the container boundary; the host kernel, Docker daemon, and job orchestrator remain trusted
Denial of service	Observable missing or delayed results; operational redundancy	Availability belongs to the hosted-service trust boundary
Replay or stale artifact use	Certificate payload identity	Currentness relies on a trusted issuance or transparency record

8 Experiments

We evaluate the platform along two tracks, organized as ten reproducible experiments (E1–E10; the commands are in Appendix B). The synthetic correctness suite of Section 8.1 is the exactness taxonomy (E1, Tables 6–7), the security-level matrix (E2, Table 8), the feasibility sweep (E3), and the differencing recovery (E4); the public-genome studies of Section 8.2 are E5–E8; and the two published-study reproductions are E9 (Section 8.3) and E10 (Section 8.4). The first track establishes correctness: we run every curated application on seeded synthetic cohorts (Section 8.1) and on public human genotypes (Section 8.2), in each case through the same loop — encode and encrypt locally, run the application's server-side BFV stage, decrypt locally, and compare against a plaintext oracle (the same statistic computed directly in the clear, which gives the known-correct answer). Across all six core curated applications on synthetic data and all four public-genome studies, every decrypted result matched its oracle exactly. The second track puts the platform's central bet to the test — that pushing work onto the local machine and onto public plaintext lets the least-expressive HE scheme do the job — by reproducing two published FHE genomics results: a privacy-preserving polygenic-risk-score method (Section 8.3, E9) and an encrypted genome-wide association study (Section 8.4, E10). In both, recasting the computation for our multiparty, public-model setting collapses the encrypted circuit to the cheap additive tier and reproduces each prior result to within its published precision — bit-exact against our own plaintext oracle, and matching the published methods to near-perfect agreement (the covariate-adjusted GWAS is concordant rather than bit-exact) — at a fraction of its memory and cost.

8.1 Synthetic-data correctness

We first demonstrate exact agreement between decrypted BFV execution and a plaintext reference across six integer computations on seeded synthetic inputs, spanning both HE tiers.

BFV is the right scheme here because every computation is exact integer arithmetic: coordinate-wise counts and sums, one-hot histograms, fixed-point public-weighted aggregates, and bounded-depth products. BFV, along with its parameter-selection and security literature, is standard [R1–R3]; we implement it with TenSEAL over Microsoft SEAL [R4, R5]. A run passes only if the decrypted result and the oracle agree with zero error.

The concrete parameters are fixed and public. The additive tier uses a BFV ring of degree $N = 8192$ with plaintext modulus $t = 1{,}032{,}193$ (a 20-bit batching prime), which packs the full contribution vector into a single ciphertext and keeps every count and public-weighted sum exact. The multiplication-supporting tier raises the ring to $N = 16{,}384$ with $t = 786{,}433$ ($\equiv 1 \bmod 2N$, required for batching at that degree) and a depth-1 coefficient-modulus chain [60, 40, 40, 60], sufficient for the single ciphertext-by-ciphertext product that variance and covariance require. In every case the security level (128/192/256-bit) is selected only by the coefficient-modulus chain at fixed ring degree, validated against the Homomorphic Encryption Standard's parameter tables [R3]; because a higher security level shrinks the coefficient modulus at a fixed ring degree, stronger security yields *smaller* ciphertext — exactly the trend Table 8 records. The GWAS reproduction (Section 8.4) keeps $t = 1{,}032{,}193$ for the chi-square sufficient statistics and widens it to $t = 274{,}877{,}562{,}881$ for the fixed-point covariate cross-terms, so the integer sums stay exact for cohorts up to about 500,000.

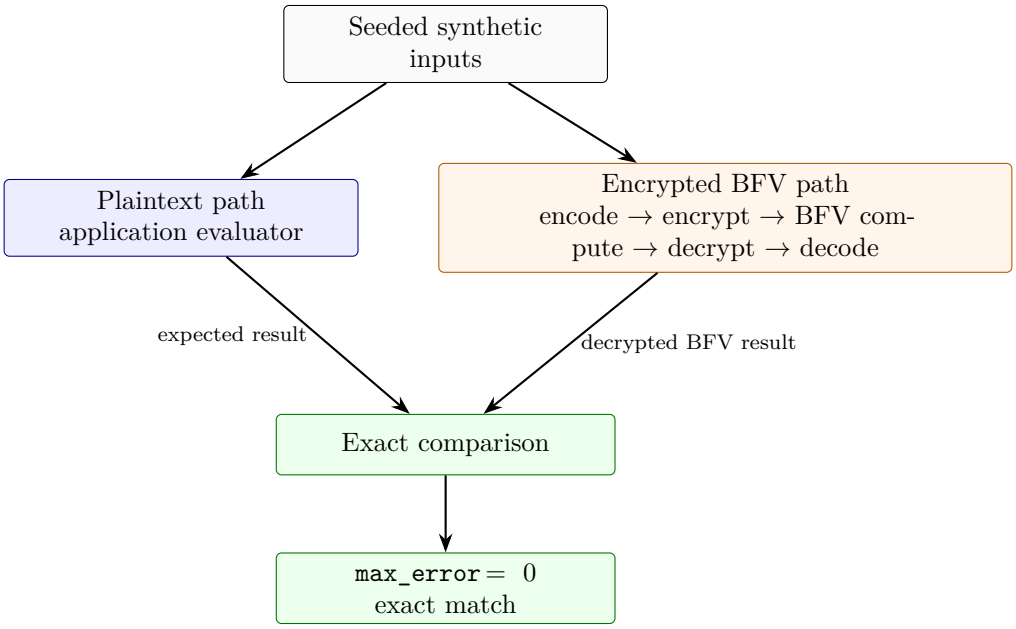

Figure 5: BFV correctness experiment. The verifier evaluates the same seeded inputs through plaintext simulation and encrypted BFV execution, then compares decoded outputs exactly.

All 61 machine-independent checks pass. They cover bit-exactness, the two-tier taxonomy, the ordering of the payload premium, the differencing-attack recovery, and the equality of every deterministic column against its committed reference value.

Table 6. BFV exactness taxonomy over six applications.

Application	Tier	Crypto	N	Length	Security	Ciphertext bytes/contribution	Compute ms	Max error	Exact
allele_frequency_count	additive-BFV-exact	bfv-add	20	10	128	262,282	204	0	yes
carrier_count	additive-BFV-exact	bfv-add	20	10	128	262,282	219	0	yes
cohort_histogram	additive-BFV-exact	bfv-add	20	10	128	262,282	223	0	yes
polygenic_score_aggregate	additive-BFV-exact	bfv-add	20	10	128	262,282	264	0	yes
allele_frequency_with_variance	mult-supporting-BFV-exact	bfv-mul	20	10	128	1,310,882	733	0	yes
genotype_phenotype_covariance	mult-supporting-BFV-exact	bfv-mul	20	10	128	2,621,791	1,317	0	yes

The table makes the two tiers concrete. Four applications are exact in the additive tier — including `polygenic_score_aggregate`, whose only multiplications are by public weights and therefore stay ciphertext-by-plaintext. The other two need the multiplication-supporting tier because the server forms a product under encryption: variance squares an encrypted value, and covariance multiplies an encrypted genotype by an encrypted phenotype. The four additive applications share a single ciphertext size and differ only in compute time. Ciphertext byte counts are deterministic and asserted; the compute-ms column is a single representative development-machine run and varies across runs and machines. Tables 6–8 cover the six core curated applications; Section 8.2.4 reports the evaluation-only, non-curated `genotype_pair_ld` path separately.

Table 7. Payload premium at the 128-bit artifact setting.

Arm	Ciphertext bytes/contribution	Premium vs additive
additive (`allele_frequency_count`)	262,282	1.0x
multiplicative (`allele_frequency_with_variance`)	1,310,882	5.0x
multiplicative (`genotype_phenotype_covariance`)	2,621,791	10.0x

This premium is deterministic under the current artifacts, and it is the byte-size evidence behind Section 4's cost argument: multiplication-supporting BFV earns its cost only when the server must form the product itself. When a multiplication can instead be done locally or against a public constant, the additive tier is the cheaper choice — the lever the two reproductions in Sections 8.3 and 8.4 pull.

Table 8. Security-level matrix.

Application	Crypto	128-bit	192-bit	256-bit	Exact across levels
allele_frequency_count	bfv-add	262,282	235,701	220,323	yes
carrier_count	bfv-add	262,282	235,734	220,314	yes

Application	Crypto	128-bit	192-bit	256-bit	Exact across levels
`cohort_histogram`	bfv-add	262,282	235,722	220,309	yes
`polygenic_score_aggregate`	bfv-add	262,282	235,718	220,320	yes
`allele_frequency_with_variance`	bfv-mul	1,310,882	786,582	657,704	yes
`genotype_phenotype_covariance`	bfv-mul	2,621,791	1,573,191	1,315,750	yes

Ciphertext size is set by two BFV parameters. With the first held fixed — the ring degree, which controls how many values pack into one ciphertext — a higher security level forces the second, the coefficient modulus (the numeric headroom for arithmetic), to shrink. Stronger security therefore yields smaller ciphertext here, from the 128-bit down to the 256-bit column, and exactness holds across the whole matrix. The 128-bit sizes are byte-deterministic; the 192-bit and 256-bit columns are representative single runs that vary by a few tens of bytes across TenSEAL builds. Runtime, CPU, RAM, and cost are representative development-machine measurements, pending a pinned-environment benchmark.

A feasibility sweep runs `allele_frequency_count` at N in {20, 100} crossed with length in {10, 100}; all four cells are bit-exact. Appendix B gives the commands, the invariant checker, and the public evidence package for regenerating these machine-independent results.

8.2 Public human genome studies

These four studies test application-stage exactness on real human genotypes, under real TenSEAL BFV with the open reference worker.

All four draw from the IGSR/1000 Genomes Project Phase 3 integrated callset (release `20130502`, GRCh37, chromosome 22) — public, openly consented, redistributable reference genotypes [R35, R36]. Each study queries a bounded chromosome-22 interval with `bcftools`. E5–E7 use curated signed applications; E8 uses the disclosed, evaluation-only `genotype_pair_ld` application, deliberately run at N=25 against its manifest release floor of N=30.

The goal is pipeline validation, not population genetics: the F_ST-like statistic (a rough measure of how differently a variant's frequency splits across populations, defined in Section 8.2.2) and the LD `r2` values are simplified small-panel descriptive statistics. Each study commits only its aggregate outputs; VCF slices, sample lists, and genotype vectors remain in ignored local `work/` directories, referenced by committed SHA-256. The evidence packages are the public `/verify/paper/...` pages, whose `not_published` certificate status records local execution. Appendix B lists the scripts that regenerate every table and figure in this section.

Table 9. Real-data studies on public human genomes.

Study	Region (chr22)	Samples	Variants / pairs	Application(s)	Headline result	Evidence page
E5 Allele-frequency panel	16.05-17.00 Mb	10	12 SNPs	`allele_frequency_count`; `allele_frequency_with_variance`	exact first/second moments; mean absolute AF delta vs IGSR global 0.0609	(local)
E6 Cross-population differentiation	16.05-17.00 Mb	50	24 SNPs	`allele_frequency_count`; `allele_frequency_with_variance`	max F_ST-like 0.1363; 624 small-cell rows suppressed; exact moments	`/verify/paper/public-genomics-e6-af-fst`
E7 Release-policy differencing	16.05-17.25 Mb	25 vs 24	40 SNPs	`allele_frequency_count` + release-policy harness	min-N alone leaves recovery 1.000; single-release freeze / simulated query budget yield 0.000 / 0.125	`/verify/paper/public-genomics-e7-beacon-policy`
E8 LD window (evaluation only)	16.05-16.90 Mb	25	11 pairs	`genotype_pair_ld` (non-curated)	exact product moments; max `r2` 1.0000	`/verify/paper/public-genomics-e8-ld-window`

8.2.1 Allele-frequency panel (E5)

The first study is the real-data analogue of the additive tier. Ten public samples (two per super-population, in panel order) over twelve complete-call biallelic SNPs in `22:16050000-17000000` are encrypted and aggregated by `allele_frequency_count` (first moment) and `allele_frequency_with_variance` (first and second moments). For all twelve variants, the decrypted counts and both moments matched the oracle exactly.

Table 10. E5 per-variant allele frequency and genotype distribution (10 public samples).

#	Coordinate	Alt count	Panel AF	IGSR global AF	Abs Δ	Heterozygote rate	Hom-alt rate
1	`22:16051249:T:C`	4	0.2000	0.1124	0.0876	0.40	0.00
2	`22:16052080:G:A`	3	0.1500	0.1412	0.0088	0.30	0.00
3	`22:16052962:C:T`	3	0.1500	0.0938	0.0562	0.30	0.00
4	`22:16052986:C:A`	1	0.0500	0.0741	0.0241	0.10	0.00
5	`22:16053444:A:T`	1	0.0500	0.0719	0.0219	0.10	0.00
6	`22:16053659:A:C`	15	0.7500	0.8576	0.1076	0.50	0.50

#	Coordinate	Alt count	Panel AF	IGSR global AF	Abs Δ	Heterozygote rate	Hom-alt rate
7	22:16053791:C:A	5	0.2500	0.1659	0.0841	0.30	0.10
8	22:16053862:C:T	4	0.2000	0.1146	0.0854	0.40	0.00
9	22:16053863:G:A	3	0.1500	0.1404	0.0096	0.30	0.00
10	22:16054454:C:T	4	0.2000	0.1158	0.0842	0.40	0.00
11	22:16054740:A:G	7	0.3500	0.4956	0.1456	0.30	0.20
12	22:16055070:G:A	3	0.1500	0.1348	0.0152	0.30	0.00

The panel frequencies track the IGSR global frequencies with the sampling noise expected from ten samples: the mean absolute deviation is 0.0609, and the largest deviations (variants 6 and 11) fall exactly where a ten-sample panel over- or under-represents a common variant. The genotype distribution confirms that the encrypted second moment carries variant-specific signal: the mean heterozygote rate is 0.3083 and the mean recovered dosage variance is 0.2542. Variant 6 (`22:16053659:A:C`) is the informative case — five heterozygotes and five homozygous-alternate samples give it a 0.75 panel AF and a 0.25 dosage variance — and its encrypted first and second moments match exactly.

8.2.2 Cross-population differentiation (E6)

The second study looks for population-differentiation signal in encrypted aggregates. Fifty public samples (ten per super-population: AFR, AMR, EAS, EUR, SAS) over twenty-four SNPs — chosen for a source super-population AF range of at least 0.15 and at least 200 bp of spacing — were aggregated by the same additive and multiplication-supporting applications, exact against the oracle. After decryption, a per-variant F_ST-like statistic was computed as the equal-weight heterozygosity contrast `(H_T - mean_s H_S) / H_T` across the five groups, where `H_T` is the heterozygosity of the pooled panel and `H_S` the heterozygosity within a group. This equal-weight contrast is a workflow signal, not a population estimator such as Weir-Cockerham F_ST [R37].

Table 11. E6 top cross-population differentiation variants (50 public samples, 5 super-populations).

#	Coordinate	Panel AF	Max pairwise Δ	Min group	Max group	F_ST-like	IGSR F_ST-like
19	22:16067208:C:G	0.6500	0.4500	AMR 0.4500	EAS 0.9000	0.1363	0.0604
2	22:16052080:G:A	0.1600	0.3000	AFR 0.0500	EAS 0.3500	0.1071	0.0440
8	22:16055070:G:A	0.1500	0.3000	AFR 0.0500	EAS 0.3500	0.1020	0.0475

#	Coordinate	Panel AF	Max pairwise Δ	Min group	Max group	F_ST-like	IGSR F_ ST-like
4	22: 16053659: A:C	0.8000	0.3000	EUR 0.6500	AFR 0.9500	0.1000	0.0371
22	22: 16069141: C:G	0.6600	0.4000	AMR 0.5000	EAS 0.9000	0.0998	0.0640
17	22: 16063369: C:T	0.1100	0.2500	EUR 0.0000	SAS 0.2500	0.0960	0.0712
1	22: 16051249: T:C	0.1200	0.2500	AFR 0.0000	SAS 0.2500	0.0814	0.0766
20	22: 16067411: T:C	0.1300	0.2500	AFR 0.0000	EAS 0.2500	0.0760	0.0874

The panel mean F_ST-like value is 0.0611 and the maximum is 0.1363, at `22:16067208:C:G`, where the East-Asian frequency (0.90) is double the admixed-American one (0.45). The ranking of the most-differentiated variants agrees with the direction of the IGSR whole-cohort column, even though the small ten-per-group panels inflate the absolute values — the expected behavior of a differentiation statistic on small groups. The study also exercises small-cell governance: group-level count and frequency fields are withheld whenever a group has fewer than ten samples, suppressing 624 rows across the panel. The full twenty-four-variant panel is Table 16 (Appendix D).

8.2.3 Release-policy and the differencing attack on real genotypes (E7)

The third study runs the K-versus-K+1 differencing attack of Section 5 on real genotypes and measures the release controls that defeat it. A frozen public cohort of N=25 is compared against an adjacent N=24 cohort (one held-out sample) over 40 complete-call biallelic SNPs; the held-out sample carries a nonzero alternate dosage at 6 of the 40 positions. Both encrypted aggregates matched the cleartext counts exactly, and — as expected — subtracting the two exact `allele_frequency_count` outputs recovers the held-out sample's full dosage vector.

Table 12. E7 release-policy recovery on adjacent public cohorts (N=25 vs N=24, 40 SNPs).

Release policy	Adjacent releases available	Query budget	Exact-position recovery (of 40)	Nonzero-position recovery (of 6)
No policy (exact adjacent counts)	yes	40	1.000	1.000
Minimum-N = 20 only	yes	40	1.000	1.000
Minimum-N = 25 blocks adjacent base	no	0	0.000	0.000
Cohort freeze, single release	no	0	0.000	0.000
Query budget = 5	yes	5	0.125	0.167
Counts rounded to nearest 5	yes	40	0.850	0.000

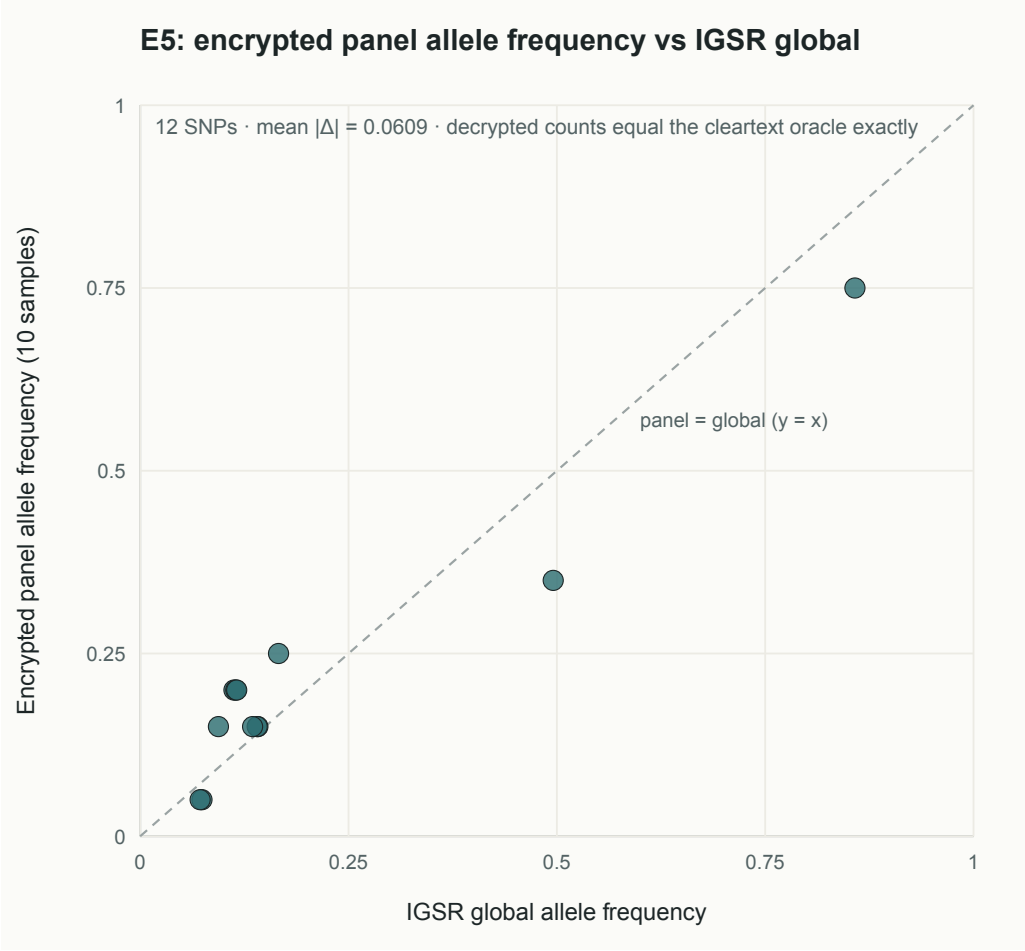

Figure 6: E5 allele-frequency concordance. Each point is one of the twelve E5 SNPs: the encrypted panel allele frequency (decrypted from BFV execution) against the IGSR global frequency, with a $y = x$ reference line. The panel tracks the global frequencies with the sampling scatter expected from ten samples (mean absolute deviation 0.0609), and every decrypted count equals the cleartext oracle exactly.

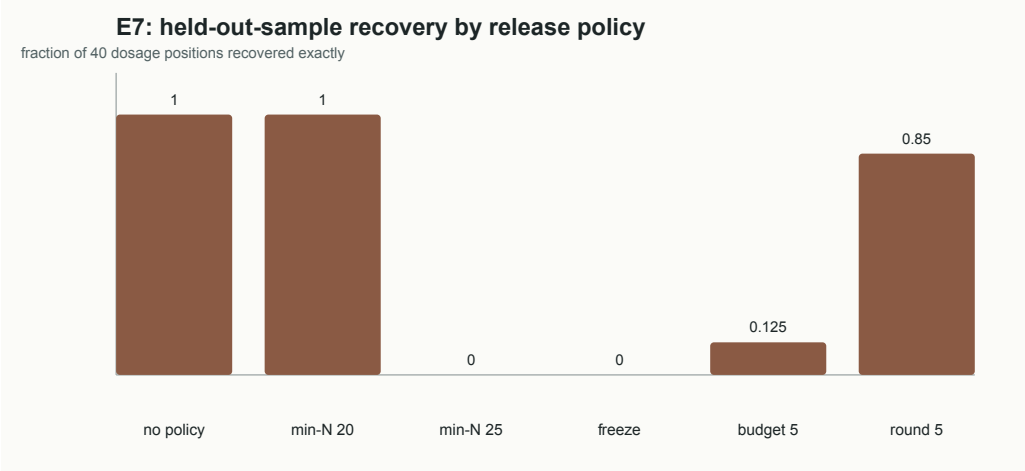

Figure 7: Held-out-sample recovery by release policy. Each bar is the fraction of the 40 dosage positions recovered exactly (the exact-position recovery rate) under one release policy on adjacent public cohorts (E7). A single release under cohort freeze and blocking the adjacent base with minimum-N=25 both reach 0.000; a query budget only rate-limits recovery.

Minimum-N=20 permits both adjacent releases and so leaves recovery at 1.000. Minimum-N=25 blocks the N=24 base, and a single-release cohort freeze permits only one aggregate; both drive recovery to 0.000. A simulated coordinate-release budget — a cap of B on how many of the 40 output positions may be read across all releases, the "query budget" column in Tables 12 and 13 — makes recovery equal to the accessible fraction, B of 40.

Table 13. E7 query-budget recovery curve.

Query budget	Positions compared	Exact positions recovered	Exact-position recovery rate
0	0	0	0.000
1	1	1	0.025
2	2	2	0.050
3	3	3	0.075
5	5	5	0.125
10	10	10	0.250
20	20	20	0.500
40	40	40	1.000

Rounding counts to the nearest five is a negative control: it destroys all six informative nonzero positions (recovery 0.000) while leaving 34 of 40 positions exact (0.850). The held-out public genotype trace stays under ignored `work/`, referenced by a committed SHA-256.

8.2.4 Linkage-disequilibrium window (E8, evaluation-only application)

The fourth study exercises the evaluation-only `genotype_pair_ld` application (the third multiplication-supporting example of Section 4), which releases the moments `sum_a, sum_b, sum_a2, sum_b2, sum_ab`. Over 25 public samples and 11 adjacent variant pairs in `22:16050000-16900000`, all five decrypted moment vectors matched the oracle exactly; covariance and `r2` were then computed from those moments after decryption.

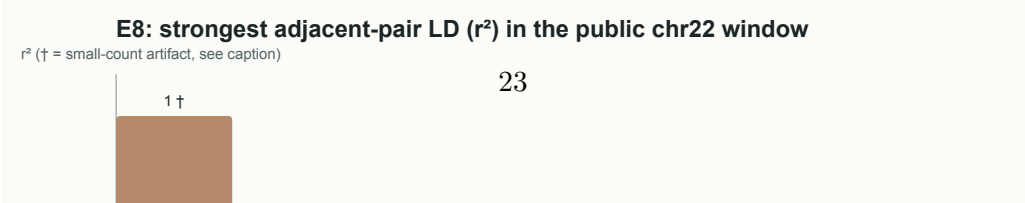

Table 14. E8 adjacent-pair LD on public genotypes (evaluation-only `genotype_pair_ld`, 25 samples).

Pair	Coordinate A	Coordinate B	Σa	Σb	Σab	Covariance	r2
1	22:16051249:T:C	22:16052080:G:A	9	5	0	-0.0720	0.1406
2	22:16052080:G:A	22:16052962:C:T	5	7	0	-0.0560	0.0972
3	22:16052962:C:T	22:16052986:C:A	7	3	1	0.0064	0.0019
4	22:16052986:C:A	22:16053444:A:T	3	3	3	0.1056	1.0000†
5	22:16053444:A:T	22:16053659:A:C	3	40	6	0.0480	0.0909
6	22:16053659:A:C	22:16053791:C:A	40	11	21	0.1360	0.1896
7	22:16053791:C:A	22:16053862:C:T	11	9	3	-0.0384	0.0157
8	22:16053862:C:T	22:16053863:G:A	9	5	0	-0.0720	0.1406
9	22:16053863:G:A	22:16054454:C:T	5	9	0	-0.0720	0.1406
10	22:16054454:C:T	22:16054740:A:G	9	21	3	-0.1824	0.2702
11	22:16054740:A:G	22:16055070:G:A	21	5	7	0.1120	0.1467

† Pair 4 reflects perfect co-occurrence of three alternate alleles in this 25-sample panel.

E8 validates the encrypted-product path with bit-exact moments. It runs at 25 samples while the `genotype_pair_ld` manifest keeps its production release floor of 30 contributors (Sections 5 and 7), so E8 is an evaluation artifact — deliberately outside the six curated registry applications and the frozen synthetic taxonomy of Section 8.1.

8.3 Replicating Knight et al. (2026): encrypted polygenic risk scores, bit-exact at ~150× less memory (E9)

To show that the platform's model reproduces prior FHE genomics work — and does so at far lower cost — we reproduce HEPRS, a recent method for privacy-preserving polygenic risk scores [R12]. HEPRS computes each individual's polygenic risk score `PRS_i = Σ_j w_j·g_ij` from a genotype dosage vector `g_i  {0,1,2}^L` and a GWAS weight vector `w`; it encrypts **both** the genotypes and the model, evaluating each per-SNP product as a ciphertext-by-ciphertext multiply under CKKS. Its headline result scores a 110,258-SNP schizophrenia model over 1,146 individuals in about 6 minutes using about 65 GB of memory on a single CPU.

We add one curated application, `polygenic_score_inference`, that computes the identical score for the common case where the PRS model is **public** — as published PGS-Catalog and GWAS

weight sets are. A public model changes the cryptography decisively: the per-SNP product becomes ciphertext-by-**plaintext**, so it needs no relinearization keys, never forms a ciphertext-by-ciphertext product, and never encrypts the model at all. The only homomorphism beyond addition is the intra-vector reduction Σ_j, performed once per individual as an encrypted rotate-sum (Section 4's additive tier plus Galois keys). Because each contributor is scored independently, the server streams them one at a time, so its memory never grows with the cohort. The score is exact in BFV — where HEPRS's CKKS is approximate — and, with the model public, the key-holding researcher learns each participant's score but never their genotype, while the compute server learns neither.

We reproduce HEPRS on its own public example (10,000 SNPs × 50 individuals, synthetic HAPGEN2 data redistributed under HEPRS's MIT licence) and on synthetic HAPGEN2-style cohorts at matched sizes, up to the 110,258-SNP × 1,146-individual schizophrenia scale. The real PsychENCODE genotypes behind HEPRS's schizophrenia result are controlled-access, so we do not use them. On the public example, our per-individual scores reproduce HEPRS's own plaintext predictions at Pearson r = 0.99999999 with an affine slope of exactly 1.000000: our encrypted score is precisely HEPRS's SNP-weighted sum, the only offset being the model's constant Ridge intercept (a public post-decrypt add), after which the residual is 3.6×10^-5 — HEPRS's own single-precision reference resolution. And at every scale, the decrypted score equals the plaintext oracle exactly.

Table 17. HEPRS reproduction: head-to-head at 128-bit security. HEPRS numbers are as reported (single CPU, Intel Xeon 6234, CKKS ring 2^{13}). Blind Machine numbers are measured on `polygenic_score_inference`, single-threaded, contributors streamed one at a time (12-core Apple-silicon, 34 GB); wall is encrypt + server-compute + decrypt across the whole cohort. The CPUs differ, so time is indicative; memory and exactness are architectural. Every row is drawn from the committed `results/head_to_head_benchmark.json` (peak RSS and per-scale wall times), which — like Table 18 — is a single unpinned development-machine run pending a pinned-environment benchmark (Section 8.1); the exactness and flat-memory columns are deterministic and asserted, the absolute times are not. Every Blind Machine result is bit-exact against the plaintext oracle.

Configuration	HEPRS (reported)	The Blind Machine (measured)	Peak memory
1 individual × 110k SNP	4.9 s, 3.3 GB	0.13 s, exact	—
10,000 SNP × 50	5 s, 0.8 GB	2.2 s, exact	427 MB
130,000 SNP × 50	36 s, 9 GB	8.0 s, exact	427 MB
130,000 SNP × 200	1.5 min, 17 GB	35 s, exact	427 MB
130,000 SNP × 2,000	12.5 min, 125 GB	5.7 min, exact	427 MB
110,258 SNP × 1,146 (schizophrenia scale)	**6 min, 65 GB**	**2.5 min** (server 43 s), exact	**436 MB**

The memory result is the structural one. HEPRS holds the whole encrypted cohort and model in memory, so its footprint climbs into the tens or hundreds of gigabytes; the streaming public-model design keeps peak memory flat at about 430 MB regardless of cohort size — roughly 150× less at the schizophrenia scale, and about 290× less at 130k × 2,000. A workload that demands a large-memory server under HEPRS runs on a laptop here. End-to-end time is lower as well (about 2.5 vs 6 minutes at the schizophrenia scale, single-core), and the server's homomorphic compute alone is just 43 seconds; because that stage is embarrassingly parallel, it drops well under a minute across cores. The rest of the 2.5-minute wall is not hosted homomorphic work at all — it is per-contributor local encryption and the researcher's single local decrypt, both outside the blind server and both parallelizable across contributors. The only cost that grows with cohort size is thus a

linear, shardable stream of ciphertext-by-plaintext products that the server sums, which is exactly why peak memory stays flat and wall time scales gently. The lesson is Section 4's: pushing work onto the local machine and onto public plaintext — instead of encrypting everything — lets a much lighter, exact, additive-tier scheme do a job that otherwise reaches for the multiplication-supporting tier and a large-memory server.

This experiment also illustrates the platform's intended growth path. Reproducing HEPRS required no change to the architecture — only a new signed application, authored as three small pure-function files (Appendix C) and reproducible offline as experiment E9. The computation, its public model, its release policy, and its sealed environment all fold into the application's content-addressed digest, so an outside researcher can install it, inspect exactly what runs on their ciphertext, point it at their own published PRS model, and score their own encrypted cohort — or contribute an application of their own. We invite the community to do exactly that; the CLI, the application bundles, and the E9 reproduction harness are all in the public replication materials (Appendix B).

8.4 Replicating Blatt et al. (2020): an encrypted genome-wide association study at additive cost (E10)

The genome-wide association study is the workload that first established homomorphic encryption as practical at biobank scale. Blatt, Gusev, Polyakov, and Goldwasser [R6] ran an encrypted GWAS over a real cohort of more than 25,000 individuals with two protocols — a one-degree-of-freedom **allelic chi-square** test and a **logistic-regression approximation** — and reported that the chi-square scan extrapolates to a 100,000-individual, 500,000-SNP GWAS in about 5.6 hours on a single server node (or 11 minutes across 31 nodes), more than an order of magnitude faster than the multiparty-computation state of the art. In their single-key design, every individual's genotype and phenotype are encrypted to one key, and the case/control cross term — the count of minor alleles carried by cases at each SNP — is formed on the server with a ciphertext-by-ciphertext multiply under CKKS.

We reproduce their allelic chi-square GWAS with one curated application, `gwas_chi_square`, that computes the identical statistic under the platform's multiparty model. The design move is the one that made E9 cheap: **each data owner holds their own genotype and phenotype, so the cross term `g·y` is formed locally, in the clear, before anything is encrypted.** What then reaches the blind server is only per-SNP counts — `Σ_i g_ij` (minor alleles) and `Σ_i g_ij·y_i` (minor alleles in cases), plus the case count and cohort size — and the server does exactly one thing: add. The encrypted circuit collapses to the additive-only BFV tier of Section 4 (the flagship `allele_frequency_count` circuit): no ciphertext-by-ciphertext product, no relinearization keys, no Galois keys. The chi-square ratio, odds ratio, and p-value are computed on the researcher's machine after decryption, from exact integer sufficient statistics. The released quantities are the standard per-SNP association summaries that GWAS meta-analyses already share, over a cohort no smaller than the application's release floor. One disclosure-granularity difference from the single-key design is worth stating plainly: our analyst decrypts the per-SNP integer sufficient statistics (from which the 2×2 contingency table is recoverable), whereas Blatt et al. release only the per-SNP p-value — a finer versus a coarser release surface. Both keep every individual genotype and phenotype encrypted, and the sufficient statistics are the summaries GWAS consortia already exchange. The finer release does carry a residual risk the coarser one does not: a per-SNP 2×2 table is a membership-inference surface of the kind Homer et al. [R27] and the beacon literature [R28] exploit, and — as Section 8.2.3 shows on real genotypes — a minimum-N floor alone does *not* defeat differencing across adjacent releases. The control that does is the single-run cap (one release per frozen cohort), which the `gwas_chi_square` and `gwas_covariate_adjusted` manifests

set. Membership and differencing risk on this finer surface should therefore be read as governed by that single-run cap and the cohort freeze rather than by the release floor; coarser releases (per-SNP p-values only, or differentially private summaries) remain an application-level choice for deployments that need them.

Because the sufficient statistics are exact integers in `Z_t` (`t = 1032193 > 2N` keeps the sums exact for cohorts up to about 500,000), the reproduction is not merely concordant but **bit-identical** to the cleartext GWAS. Blatt et al. report that their encrypted chi-square matches the plaintext statistic at $R^2 = 1.00$; here the decrypted per-SNP counts equal the cleartext aggregate bit-for-bit, so the derived chi-square, p-values, and odds ratios agree exactly ($-\log10$ p concordance $R^2 = 1.000000$, verified at 128/192/256-bit security). We run it on the public demo cohort their prototype ships (`random_sample.csv`: 200 individuals, 16,384 SNPs, binary phenotype) and on seeded synthetic case/control cohorts of the same shape; their real data — a dbGaP-controlled age-related-macular-degeneration cohort — is access-gated, so, exactly as they did, we anchor on a measured cohort and extrapolate linearly in individuals and SNPs.

Table 18. Encrypted allelic-chi-square GWAS reproduction: head-to-head. Blatt et al. numbers are as reported — measured for $n \leq 25{,}000$, and their own linear extrapolations for n = 100,000 (two 14-core Xeon E5-2680 v4 sockets, 500 GB RAM, RNS-CKKS ring dimension 32768). Blind Machine numbers are for the **blind server's homomorphic aggregation** on `gwas_chi_square`, single-threaded on one modern laptop core (Apple silicon), 128-bit security. The anchor is the committed offline synthetic run of experiment E10, whose server aggregation of the 200×16,384 demo measures ≈ 0.15 s (and ≈ 0.23 s on Duality's public `random_sample.csv`, fetched separately); this sub-second anchor varies by roughly ±50% run-to-run on unpinned hardware. Every larger row is a linear $O(N \cdot M)$ projection from that anchor — the way Blatt et al. extrapolate their own runtime — so its absolute minutes should be read as order-of-magnitude, not a stopwatch figure, and a pinned-environment benchmark is future work (Section 8.1). All rows are anchored to the same committed synthetic measurement, so the table and the committed `results/` extrapolation block move together. Every Blind Machine result is bit-exact against the cleartext GWAS.

Configuration (individuals × SNPs)	Blatt et al. [R6] (reported)	The Blind Machine (server aggregation)
200 × 16,384 (public demo)	—	≈ 0.15 s committed (≈ 0.23 s on Duality's cohort), bit-exact
15,000 × 16,384	98 s	≈ 11 s
25,000 × 49,152 (full cohort)	8 min	≈ 1 min
100,000 × 16,384	11 min	≈ 1–2 min
100,000 × 500,000 (headline)	**5.6 h single node / 11 min on 31 nodes**	**≈ 40 min–1 h single core / ≈ 1–2 min on 31 SNP-block workers**
26,737 × 263,941 (their real AMD cohort)	—	≈ 5 min

The comparison must be read carefully: the claim is not that our cryptography is faster at the same task, but that in a multiparty setting where each contributor owns their own data, the association test's single multiplication can be done locally, leaving the blind server an additive-only job. We are therefore comparing our additive aggregation against their full encrypted chi-square circuit; the scientific output is identical — here bit-for-bit so. The hardware is asymmetric too: a single modern laptop core here against their two-socket, 28-core, 500 GB node. And the largest sizes on both sides are linear extrapolations from a measured anchor (their real cohort is a dbGaP-gated study, so we run their public demo cohort and extrapolate exactly as they do). The result is thus a work-split-and-scheme-tier finding, not a like-for-like cryptographic benchmark: the association

test's arithmetic simply does not need a deep encrypted circuit once each owner holds their own data. Per-owner encryption (about 20 ms per contributor, embarrassingly parallel across owners) plus local key generation and decoding (both well under a second) complete the loop, and peak memory stays around 0.4 GB because the server streams contributors and never holds the cohort. At demo scale the entire blind aggregation is a fraction of a second; at the paper's largest extrapolated size it is a few minutes across a handful of SNP-block workers — while returning the identical, bit-exact allelic chi-square. As with E9, reproducing this result required no architectural change, only a new signed application (three pure-function files, Appendix C), reproducible offline as experiment E10; a researcher can install it, read exactly what runs on their ciphertext, and point it at their own case/control cohort.

We reproduce the paper's other protocol — the covariate-adjusted **logistic-regression approximation** (LRA), a semi-parallel score test after Sikorska et al. — as a second signed application, `gwas_covariate_adjusted`. The same principle extends: each contributor holds their own covariates (sex, age, age^2) alongside their genotype and phenotype, so every product the semi-parallel method needs — the covariate Gram matrix `X'X`, the term `X'y`, and the per-SNP cross term `X'g` — is formed locally in the clear, while the small k×k covariate inverse and the per-SNP score test run in cleartext on the decrypted aggregate. The single-key prototype instead performs that k×k inverse homomorphically, under a deep circuit (their parameterization: RNS-CKKS, ring dimension 32768, ~16 multiplicative levels); here the encrypted circuit is once again additive-only. Because the covariates are continuous, they are fixed-point encoded (scale 1024), so this reproduction is highly concordant rather than bit-exact: on the demo cohort the per-SNP $-\log10(p)$ matches the cleartext covariate-adjusted regression at $R^2 = 0.99997$ (and at $R^2 = 0.999995$ on the committed offline synthetic run of E10) — the same order of approximation the paper's own LRA carries (it matches exact logistic regression at $R^2 = 1.00$). Blatt et al. report the LRA at 1.1 h for 15,000 individuals × 16,384 SNPs, extrapolating to 7.7 h at 100,000 × 16,384; the additive blind aggregation extrapolates to about 5 minutes single-core at that size (the covariate cross-series make each contribution about 3× the chi-square app's, so the demo aggregation is ~0.6 s, and 100,000 × 500,000 is a few hours single-core or minutes across SNP-block workers). Both protocols run together in experiment E10, on the public demo cohort and on seeded synthetic data. Their two published protocols are now two signed applications on the platform; we invite the community to add their own.

9 Conclusion and Future Directions

Homomorphic encryption now faces two practical engineering challenges. The first is computational: expanding the range of computations that can be performed efficiently over encrypted data. Today, many useful aggregate computations are already practical, while more complex workloads remain constrained by the cost of encrypted arithmetic. As homomorphic-encryption algorithms continue to improve, progressively richer classes of applications will become feasible. The second challenge is architectural: building end-to-end platforms that make encrypted computation practical to deploy, govern, reproduce, and verify. The Blind Machine addresses this second challenge by making the computation itself the unit of governance.

The central abstraction is the signed, content-addressed application. Rather than treating homomorphic encryption as a cryptographic library embedded inside a bespoke workflow, the platform packages the cryptographic environment, local and hosted computation, execution policy, and certification into a single reusable artifact. That artifact can be inspected before execution, reused

across studies, and verified afterward without changing the surrounding infrastructure. The platform therefore separates two concerns that are often intertwined: cryptography determines **how** a computation executes on encrypted data, while the application defines **what** computation is approved to execute.

The resulting architecture establishes a stable execution model for encrypted computation. Plaintext data and secret keys remain under local custody throughout the protocol, while the hosted service operates exclusively on ciphertext. Cohort commitments, release policies, sandboxed execution, signed applications, and offline-verifiable certificates bind each execution to an immutable description of the computation that produced it. Across the six core curated applications and four public-genome studies, the same architecture required no changes despite covering both additive and multiplication-supporting BFV workloads.

Local custody is also what keeps the platform fast. Because each data owner holds their own data, most computation is done locally in the clear, and each application asks the hosted server only for the least-expressive homomorphic operation that suffices — so a workload that would otherwise demand a deep encrypted circuit and a large-memory server often reduces to additive aggregation on a laptop. The two published-study reproductions of Section 8 make this concrete: the same design move that keeps confidentiality local also collapsed a ciphertext-by-ciphertext polygenic-score circuit and a full encrypted GWAS circuit into streamed, additive-tier aggregations that ran faster and in a fraction of the memory.

The architecture also defines clear boundaries. Aggregate correctness remains assigned to the hosted execution, certificates bind executions to approved artifacts rather than re-proving computation, governance currently operates within individual projects, and release privacy is enforced through project-level controls rather than global accounting. These boundaries are architectural choices that isolate future improvements without changing the surrounding execution model.

Several natural directions extend this platform.

Distributed trust. Threshold and multi-key homomorphic encryption [R8, R11, R13, R21] would distribute decryption authority across multiple participants or institutions while preserving the application model.

Stronger release privacy. Differential privacy, cross-project cohort-overlap accounting, and global query budgets [R29, R30] would extend the current governance model from individual projects to long-lived data ecosystems in which many studies reuse overlapping populations.

Independent execution verification. The current certificates establish provenance and reproducibility. Future systems may augment them with cryptographic proofs of correct execution, replicated execution across independent workers, or other mechanisms that reduce trust in the hosted computation itself while preserving the same application interface.

Transparency infrastructure. Append-only transparency logs for application digests, cohort commitments, certificate hashes, issuance events, and revocations would make the governance history itself independently auditable. Stronger contributor authentication would likewise strengthen project governance without changing the execution architecture.

Finally, although this paper focuses on genomics, the architecture is independent of genomics itself. Any domain in which participants retain sensitive data while releasing approved aggregates—including healthcare, finance, public statistics, and collaborative machine learning—shares the same execution pattern. As homomorphic encryption becomes faster and broader classes of computation become practical, the application-centered model presented here provides a reusable foundation for governed computation on encrypted data.

10 Declarations

Competing interests. The author develops The Blind Machine as a project intended for future commercialization and holds a financial interest in its success. The open-source components are released under the MIT license. No payments or services were received from any third party in connection with this work.

Funding. This work received no external funding.

Ethics and data governance. This study uses no private or controlled-access human data. All real-genome experiments (E5–E10) draw exclusively on the 1000 Genomes Project Phase 3 integrated callset, an openly consented, publicly redistributable reference resource distributed by IGSR [R35, R36]; no controlled-access cohort was accessed, and the controlled-access datasets behind the reproduced studies (PsychENCODE for HEPRS, the dbGaP AMD cohort for the GWAS) were deliberately not used. The differencing and membership-inference demonstrations (Sections 5, 8.2.3, 8.4) are run on these public reference genotypes or on seeded synthetic cohorts and are included to motivate the platform's release-governance controls, not to re-identify any individual. No IRB approval was required. Deployments that ingest identifiable or controlled-access genomes must apply the appropriate institutional and legal data-governance regime in addition to the technical controls described here.

Data and code availability. The open-source `blind` CLI and the replication materials are openly available at https://github.com/blindmachine/blind and https://github.com/blindmachine/the-blind-machine-paper. The real-genome studies use publicly available data from the 1000 Genomes Project Phase 3 integrated callset (release 20130502, GRCh37, chromosome 22), distributed by the International Genome Sample Resource (IGSR, https://www.internationalgenome.org). An archival snapshot of this manuscript and the replication materials is deposited at Zenodo under the concept DOI https://doi.org/10.5281/zenodo.21421425, which always resolves to the latest version (the release archived here is v2026.07.21).

A Figures and Tables

ID	Title	Location	Source
Figure 1	The Blind Machine at a glance	Introduction	`figures/figure-1-overview.png`
Figure 2	The Blind Machine protocol	Section 3	`figures/figure-2-trust-zones.svg`
Figure 3	K-versus-K+1 differencing and current release controls	Section 5	`figures/figure-3-output-governance.tex`
Figure 4	Certificate binding diagram	Section 6	`figures/figure-4-certificate-binding.te`
Figure 5	BFV correctness experiment	Section 8.1	`figures/figure-5-bfv-correctness.tex`
Figure 6	E5 allele-frequency concordance	Section 8.2	`public_real_dna_summary_2026_07_09/results/figure_e5_af_concordance.svg`
Figure 7	Held-out-sample recovery by release policy	Section 8.2	`public_real_dna_summary_2026_07_09/results/figure_beacon_policy_recovery.svg`

ID	Title	Location	Source
Figure 8	Strongest adjacent-pair LD r2	Section 8.2	public_real_dna_summary_2026_07_09/results/figure_ld_top_r2.svg
Figure 9	Application authoring and execution loop	Appendix C	figures/figure-9-application-loop.tex
Table 1	Prior-work positioning	Section 2	Approved Stage 2 references
Table 2	Platform invariants	Section 3	Requirements, domain, application, and certificate docs
Table 3	Certificate payload fields	Section 6	ComputationCertificate canonical payload
Table 4	Trusted computing base by property	Section 7	Threat model and system docs
Table 5	Threat coverage matrix	Section 7	Threat model and the Section 7 per-attack catalog
Table 6	BFV exactness taxonomy	Section 8.1	results/expected/table_b_reference.json (committed invariant; results/table_b_exactness.csv is regenerated by run_all.sh, gitignored)
Table 7	Payload premium	Section 8.1	results/expected/table_b_reference.json (premium derived from the committed ciphertext sizes; results/table_c_premium.csv is regenerated by run_all.sh, gitignored)
Table 8	Security-level matrix	Section 8.1	results/expected/security_matrix_reference.json (committed invariant — only the 128-bit ciphertext-byte column and per-app exactness are asserted; the 192/256-bit sizes are non-deterministic across TenSEAL builds and vary by tens of bytes; results/security_matrix.csv is regenerated by run_all.sh full, gitignored)
Table 9	Real-data studies on public human genomes	Section 8.2	summarize_public_real_dna.py; IGSR/1000 Genomes Phase 3
Table 10	E5 per-variant allele frequency and genotype distribution	Section 8.2	real_human_dna_igsr_2026_07_09/results/allele_frequencies.csv, genotype_distribution.csv

ID	Title	Location	Source
Table 11	E6 top cross-population differentiation variants	Section 8.2	`public_af_fst_2026_07_09/results/fst_summary.csv`
Table 12	E7 release-policy recovery	Section 8.2	`beacon_release_policy_2026_07_09/results/policy_risk_summary.csv`
Table 13	E7 query-budget recovery curve	Section 8.2	`beacon_release_policy_2026_07_09/results/query_budget_curve.csv`
Table 14	E8 adjacent-pair LD	Section 8.2	`public_ld_window_2026_07_09/results/ld_pairs.csv`
Table 15	Application bundle surfaces	Appendix C	Application structure, manifest, worker, CLI, and test files
Table 16	E6 full cross-population differentiation panel	Appendix D	`public_af_fst_2026_07_09/results/fst_summary.csv`

B Reproduction and Certificate Checks

The published evaluation package contains the Blind CLI, the application code, the experiment scripts, the synthetic inputs, the bounded public-genome coordinates, the aggregate outputs, and the committed expected results. The commands below rebuild the synthetic and public-genome evaluations and check the machine-independent invariants of Section 8.

Getting the tools. The prerequisites are Python 3.11+ and `uv` (https://docs.astral.sh/uv/), the toolchain the bundles seal their environments with. The Blind CLI is open source at https://github.com/blindmachine/blind and is invoked as `blind` (for example, `uv run blind`); the six core curated applications are browsable at https://blindmachine.org/applications/, and the three published-study reproduction bundles (E9–E10) ship with the replication repository below. The full replication materials — the experiments harness, the nine signed application bundles (plus the unsigned evaluation-only `genotype_pair_ld` draft), and a vendored copy of the CLI — are in the companion repository https://github.com/blindmachine/the-blind-machine-paper; clone it and run the reproduction commands below from its root. On first run the harness seals each application environment once — downloading TenSEAL over Microsoft SEAL — and reuses it thereafter.

Reproduce the synthetic evaluation (Section 8.1) from seeded inputs once the environments are installed (the first setup may download dependencies; later runs can be offline). The fast profile writes Tables 6-7 (the exactness taxonomy and payload premium) plus the differencing check; the `full` profile additionally regenerates Table 8 (the security-level matrix) and the feasibility sweep:

```
cd experiments
bash run_all.sh              # fast: Tables 6-7 + differencing, assert invariants (~2-4 min)
bash run_all.sh full         # also Table 8 (security matrix) + feasibility sweep (~15-25 min)
python3 verify.py            # check machine-independent invariants only
```

Reproduce the real-genome studies (Section 8.2) from bounded public IGSR intervals and aggregate-only outputs:

```
cd experiments
bash e5_real_human_dna_igsr.sh            # Table 10
bash e6_public_af_fst_panel.sh            # Tables 11, 16
bash e7_beacon_release_policy.sh          # Tables 12, 13
bash e8_public_ld_window.sh               # Table 14
python3 summarize_public_real_dna.py      # rebuild Table 9 and Figures 6-8
```

Reproduce the HEPRS study (Section 8.3, E9) on `polygenic_score_inference`, offline, against the vendored HEPRS public example (synthetic HAPGEN2 data, MIT-licensed) — it asserts bit-exactness and the reproduction of HEPRS's predictions (Pearson r $>$ 0.9999):

```
cd experiments
bash e9_heprs_prs_reproduction.sh         # asserts exactness + reproduction; RESULT: PASS
```

Reproduce the GWAS study (Section 8.4, E10) on `gwas_chi_square` — reproducing the allelic chi-square GWAS of Blatt et al. (PNAS 2020) bit-exactly against a cleartext oracle — together with its covariate-adjusted companion `gwas_covariate_adjusted` (run as the second half of the same script, which asserts per-SNP $-\log10(p)$ concordance $R^2 \geq 0.999$ against a cleartext regression on the true covariates). Point it at Duality's own public CSV with `BLIND_GWAS_CSV=/path/to/random_sample.csv` to reproduce on their exact data:

```
cd experiments
bash e10_gwas_chi_square.sh               # asserts chi-square bit-exactness + covariate concordance; RESULT: PASS
```

Or reproduce everything with one command. `replicate_all.sh` is the canonical entrypoint: it drives the synthetic core (E1–E4), fetches the bounded public-genome slices and runs the real-DNA studies (E5–E8), then runs the published-study reproductions (E9–E10), printing a single PASS / SKIP / FAIL table. It exits non-zero only on a deterministic regression; missing prerequisites SKIP cleanly rather than fail:

```
cd experiments
bash replicate_all.sh                     # full E1-E10 (add `fast` to skip the E2/E3 sweeps)
```

Check a certificate offline. `blind certificates verify` validates a certificate's schema, canonical hash, and expected bindings; `blind applications install` verifies an application's digest and signature before installing it; and `blind explain` renders its computation and release rules for review.

Agent-driven replication and review. The reproduction workflow is also published as agent-executable surfaces. A concise plain-language summary of this work is hosted at https://blindmachine.org/papers/the-blind-machine (the full manuscript is this document and the archived preprint); the replication skill at https://blindmachine.org/skills/replicate is a self-contained recipe an agent can fetch and run to install the tools, repeat both the synthetic and real-genome experiments, and diff its outputs against our committed results; the review skill at https://blindmachine.org/skills/review guides an agent through an independent, evidence-checked review. Both are listed at https://blindmachine.org/skills and in `/llms.txt`.

`verify.py` asserts the machine-independent invariants and exits non-zero on any regression: bit-exactness (`max_error == 0`), the two-tier taxonomy, the payload-premium ordering, the differencing recovery, and equality of the deterministic columns against the committed references under

`results/expected/`. Wall-clock, RAM, and cost are reported as observed, hardware-dependent values.

The public synthetic-evidence package lists every evidence file with its SHA-256 and the package's canonical master hash; that page is generated from the live package, so it always reflects the current contents rather than a digest transcribed into this PDF. An outside replicator verifies the materials per file against the committed manifest — from the replication repo, `sha256sum -c MANIFEST.sha256`.

C Writing an Application

An application plugs a new encrypted aggregate into the surrounding governance loop. The author supplies only the computation-specific pieces: a manifest, the local role functions, one server compute function, a locked Python environment, documentation, and tests. The platform supplies everything else — the registry, content addressing, signing, the shimmed stage lifecycle, local CLI execution, the hosted sandbox, cohort freeze, certificates, and the offline checks.

The authoring contract is documented at https://blindmachine.org/application-structure, and the curated registry is browsable at https://blindmachine.org/applications/. Each page exposes the signed-payload boundary, README, version, hash, and source tree. Section 3 lists the six curated computations; Section 8.2.4 documents the evaluation-only `genotype_pair_ld` application.

The application digest covers `manifest.yml`, `server.py`, `local_project_owner.py`, `local_data_owner.py`, and `signed/env/`. Root-level `README.md`, `SECURITY.md`, benchmark notes, and `tests/` are public support artifacts outside the digest. The six numbered lifecycle scripts are kit-owned shims: they are materialized by the CLI or worker at run time and call the author's pure functions — they are never written by the author and never shipped in the bundle.

Table 15. Application bundle surfaces and how they plug into the loop.

Surface	Author writes	Platform uses it for	Plug-in boundary
Manifest	Application name, version, input/output shape, release policy, resources, role mapping, and display crypto hint	Registry listing, governance gates, resource limits, certificate fields, and review surface	Changing a signed manifest byte changes the application digest
Local project-owner role	`keygen`, `decrypt`, and `decode` pure functions	Local key generation, local result decryption, and final result interpretation	Secret-bearing operations run locally
Local data-owner role	`encode` and `encrypt` pure functions	Contributor-side validation, summarization, and ciphertext creation	Raw inputs are transformed locally before upload
Server role	`compute(inputs, public_context) -> bytes`	Author function run by the hosted worker	The function receives ciphertexts plus public context
Locked environment	`env/pyproject.toml`, `env/uv.lock`, and `.python-version`	Dependency-pinned build, environment sealing, `env_lock` digest, and benchmark provenance	Dependencies are application-owned rather than platform backends
Public tests	Vectors, expected aggregates, and local-loop equivalence tests	Review support, CI checks, and paper artifact validation	Public support evidence outside the application digest

Surface	Author writes	Platform uses it for	Plug-in boundary
Kit-owned shims	Platform-provided numbered lifecycle scripts	Stable CLI/worker lifecycle for keygen, encode, encrypt, compute, decrypt, and decode	Framework code maps file/argv stages onto pure functions
Signature and digest	The curator signs the canonical digest	Ingest, worker verification, certificate binding, and offline review	Registry and worker accept the curator-signed digest

This structure isolates a new encrypted aggregate to its manifest, local roles, server function, environment lock, and tests, while the platform keeps the surrounding governance loop. The review question becomes concrete: does this signed bundle encode the right inputs, compute the approved ciphertext function, release the intended aggregate, and pass its declared local-loop and encrypted-loop tests?

D Extended Real-Data Tables

This appendix holds the full per-variant listing behind the E6 cross-population study of Section 8.2.2. The E5 panel (Table 10) and E8 window (Table 14) already appear in full in Section 8.2; only the E6 panel is abridged there (to its eight most-differentiated variants, Table 11) and is given in full below. The Section 8.2 real-data figures and summary tables are regenerated by `python3 experiments/summarize_public_real_dna.py`.

Table 16. E6 full cross-population differentiation panel (50 public samples, 24 SNPs, 5 super-populations).

#	Coordinate	Panel AF	Max pairwise Δ	Min group	Max group	F_ST-like	IGSR F_ ST-like
1	`22:16051249:T:C`	0.1200	0.2500	AFR 0.0000	SAS 0.2500	0.0814	0.0766
2	`22:16052080:G:A`	0.1600	0.3000	AFR 0.0500	EAS 0.3500	0.1071	0.0440
3	`22:16052962:C:T`	0.1200	0.2000	AFR 0.0500	SAS 0.2500	0.0530	0.0750
4	`22:16053659:A:C`	0.8000	0.3000	EUR 0.6500	AFR 0.9500	0.1000	0.0371
5	`22:16053862:C:T`	0.1300	0.2000	AFR 0.0500	SAS 0.2500	0.0584	0.0787
6	`22:16054454:C:T`	0.1300	0.2000	AFR 0.0500	SAS 0.2500	0.0584	0.0764
7	`22:16054740:A:G`	0.4400	0.2000	SAS 0.3500	EAS 0.5500	0.0179	0.0318
8	`22:16055070:G:A`	0.1500	0.3000	AFR 0.0500	EAS 0.3500	0.1020	0.0475
9	`22:16055942:C:T`	0.7600	0.3000	EUR 0.6500	EAS 0.9500	0.0570	0.0368

#	Coordinate	Panel AF	Max pairwise Δ	Min group	Max group	F_ST-like	IGSR F_ ST-like
10	22: 16057417: C:T	0.1300	0.2000	AFR 0.0500	SAS 0.2500	0.0584	0.0797
11	22: 16058758: C:A	0.1400	0.1000	AFR 0.1000	SAS 0.2000	0.0116	0.0389
12	22: 16060513: T:C	0.5700	0.3000	AFR 0.4500	AMR 0.7500	0.0514	0.0284
13	22: 16060797: A:C	0.7600	0.3500	EUR 0.6000	EAS 0.9500	0.0735	0.0369
14	22: 16061016: T:C	0.1700	0.1500	AMR 0.0500	AFR 0.2000	0.0255	0.0294
15	22: 16061992: A:C	0.4600	0.3000	SAS 0.3000	AMR 0.6000	0.0459	0.0331
16	22: 16062988: C:T	0.1300	0.2000	AFR 0.0500	SAS 0.2500	0.0584	0.0722
17	22: 16063369: C:T	0.1100	0.2500	EUR 0.0000	SAS 0.2500	0.0960	0.0712
18	22: 16064992: G:A	0.1700	0.1500	AMR 0.0500	AFR 0.2000	0.0255	0.0337
19	22: 16067208: C:G	0.6500	0.4500	AMR 0.4500	EAS 0.9000	0.1363	0.0604
20	22: 16067411: T:C	0.1300	0.2500	AFR 0.0000	EAS 0.2500	0.0760	0.0874
21	22: 16067693: C:T	0.3200	0.1500	AMR 0.2500	EAS 0.4000	0.0165	0.0485
22	22: 16069141: C:G	0.6600	0.4000	AMR 0.5000	EAS 0.9000	0.0998	0.0640
23	22: 16069707: C:G	0.4100	0.2500	EUR 0.3000	EAS 0.5500	0.0389	0.0514
24	22: 16070003: C:T	0.1700	0.1500	AMR 0.1000	EAS 0.2500	0.0184	0.0292

Summary statistics (panel mean F_ST-like 0.0611, maximum 0.1363, mean absolute AF deviation 0.0311) and the 624 suppressed small-cell rows are discussed in Section 8.2.2; the eight most-differentiated variants appear in Table 11. The decrypted panel counts and second moments matched the cleartext oracle exactly for all twenty-four variants.

References

[R1] Z. Brakerski. "Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP." CRYPTO/IACR ePrint, 2012. https://eprint.iacr.org/2012/078

[R2] J. Fan and F. Vercauteren. "Somewhat Practical Fully Homomorphic Encryption." IACR ePrint, 2012. https://eprint.iacr.org/2012/144

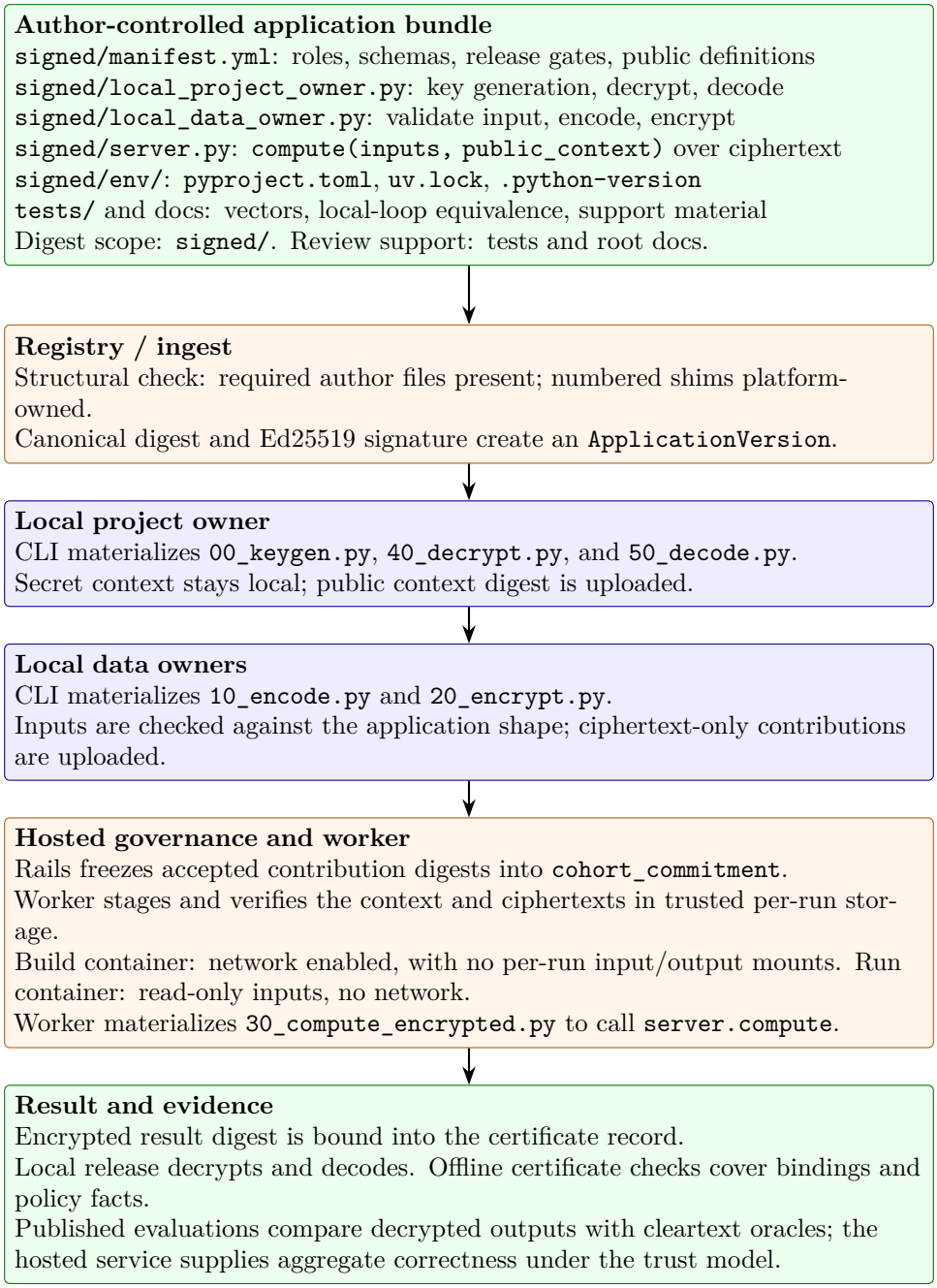

Figure 9: Application authoring and execution loop. Application authors provide the plug-in computation surfaces; the platform supplies registry checks, local and hosted shims, governance, sandboxed execution, certificate consistency checks, and evaluation evidence.

[R3] M. Albrecht et al. "Homomorphic Encryption Security Standard." HomomorphicEncryption.org/IACR ePrint, 2018/2019. https://homomorphicencryption.org/security-guidelines/

[R4] A. Benaissa, B. Retiat, B. Cebere, and A. E. Belfedhal. "TenSEAL: A Library for Encrypted Tensor Operations Using Homomorphic Encryption." arXiv, 2021. https://arxiv.org/abs/2104.03152

[R5] Microsoft Research. "Microsoft SEAL." https://github.com/microsoft/SEAL

[R6] M. Blatt, A. Gusev, Y. Polyakov, and S. Goldwasser. "Secure large-scale genome-wide association studies using homomorphic encryption." PNAS, 2020. https://www.pnas.org/doi/10.1073/pnas.1918257117

[R7] J. Blindenbach, J. Kang, S. Hong, C. Karam, T. Lehner, and G. Gursoy. "SQUiD: ultra-secure storage and analysis of genetic data for the advancement of precision medicine." Genome Biology, 2024. https://link.springer.com/article/10.1186/s13059-024-03447-9

[R8] M. Yang et al. "TrustGWAS: A full-process workflow for encrypted GWAS using multi-key homomorphic encryption and pseudorandom number perturbation." Cell Systems, 2022. https://www.sciencedirect.com/science/article/pii/S2405471222003143

[R9] H. Cho, D. Froelicher, J. Chen, M. Edupalli, A. Pyrgelis, J. R. Troncoso-Pastoriza, J.-P. Hubaux, and B. Berger. "Secure and federated genome-wide association studies for biobank-scale datasets." Nature Genetics, 2025. https://www.nature.com/articles/s41588-025-02109-1

[R10] K. Cho, D. J. Wu, and B. Berger. "Secure genome-wide association analysis using multiparty computation." Nature Biotechnology, 2018. https://doi.org/10.1038/nbt.4108

[R11] D. Froelicher et al. "Truly privacy-preserving federated analytics for precision medicine with multiparty homomorphic encryption." Nature Communications, 2021. https://www.nature.com/articles/s41467-021-25972-y

[R12] E. Knight, J. Li, M. Jensen, I. Yolou, C. Kockan, and M. Gerstein. "Homomorphic encryption enables privacy preserving polygenic risk scores." Cell Reports Methods, 6(1):101271, 2026. DOI 10.1016/j.crmeth.2025.101271. Software: https://github.com/gersteinlab/HEPRS (Zenodo doi:10.5281/zenodo.17486063). https://doi.org/10.1016/j.crmeth.2025.101271

[R13] G. Akkaya et al. "PRISM: privacy-preserving rare disease analysis using fully homomorphic encryption." Bioinformatics, 2025. https://academic.oup.com/bioinformatics/article/41/10/btaf468/8240325

[R14] S. Wang et al. "HEALER: homomorphic computation of ExAct Logistic rEgRession for secure rare disease variants analysis in GWAS." Bioinformatics, 2016. https://academic.oup.com/bioinformatics/article/32/2/211/1744166

[R15] M. Kim and K. Lauter. "Private genome analysis through homomorphic encryption." BMC Medical Informatics and Decision Making, 2015. https://link.springer.com/article/10.1186/1472-6947-15-S5-S3

[R16] A. J. Titus et al. "SIG-DB: Leveraging homomorphic encryption to securely interrogate privately held genomic databases." PLOS Computational Biology, 2018. https://doi.org/10.1371/journal.pcbi.1006454

[R17] H. Smajlovic, A. Shajii, B. Berger, H. Cho, and I. Numanagic. "Sequre: a high-performance framework for secure multiparty computation enables biomedical data sharing." Genome Biology, 2023. https://link.springer.com/article/10.1186/s13059-022-02841-5

[R18] A. Nasirigerdeh et al. "sPLINK: a hybrid federated tool as a robust alternative to meta-analysis in genome-wide association studies." Genome Biology, 2022. https://link.springer.com/article/10.1186/s13059-021-02562-1

[R19] S. Mendelsohn et al. "sfkit: a web-based toolkit for secure and federated genomic analysis." Nucleic Acids Research, 2023. https://academic.oup.com/nar/article/51/W1/W535/7184156

[R20] W. Zhou et al. "Secure bioinformatics: privacy-preserving federated analytics using homomorphic encryption." Bioinformatics, 2026. https://academic.oup.com/bioinformatics/article/42/5/btag081/8493202

[R21] M. Namazi, M. Farahpoor, E. Ayday, and F. Perez-Gonzalez. "Privacy-preserving framework for genomic computations via multi-key homomorphic encryption." Bioinformatics, 2025. https://academic.oup.com/bioinformatics/article/41/3/btae754/7994464

[R22] R. Gennaro, C. Gentry, and B. Parno. "Non-Interactive Verifiable Computing: Outsourcing Computation to Untrusted Workers." CRYPTO, 2010. https://link.springer.com/chapter/10.1007/978-3-642-14623-7_25

[R23] S. Chatel, C. Knabenhans, A. Pyrgelis, C. Troncoso, and J.-P. Hubaux. "Verifiable Encodings for Secure Homomorphic Analytics." arXiv, 2022. https://arxiv.org/abs/2207.14071

[R24] X. Wang, Y. Jiang, and J. Vaidya. "Efficient verification for outsourced genome-wide association studies." Journal of Biomedical Informatics, 2021. https://www.sciencedirect.com/science/article/pii/S1532046421000435

[R25] A. Halimi et al. "Privacy-Preserving and Efficient Verification of the Outcome in Genome-Wide Association Studies." Proceedings on Privacy Enhancing Technologies, 2022. https://pmc.ncbi.nlm.nih.gov/articles/PMC9536480/

[R26] Y. Jiang, T. Ji, and E. Ayday. "PROVGEN: A Privacy-Preserving Approach for Outcome Validation in Genomic Research." PoPETs, 2026. https://petsymposium.org/popets/2026/popets-2026-0064.php

[R27] N. Homer et al. "Resolving Individuals Contributing Trace Amounts of DNA to Highly Complex Mixtures Using High-Density SNP Genotyping Microarrays." PLOS Genetics, 2008. https://journals.plos.org/plosgenetics/article?id=10.1371/journal.pgen.1000167

[R28] S. S. Shringarpure and C. D. Bustamante. "Privacy Risks from Genomic Data-Sharing Beacons." American Journal of Human Genetics, 2015. https://www.sciencedirect.com/science/article/pii/S0002929715003742

[R29] I. Dinur and K. Nissim. "Revealing information while preserving privacy." PODS, 2003. https://doi.org/10.1145/773153.773173

[R30] J. P. Near, D. Darais, N. Lefkovitz, and G. S. Howarth. "Guidelines for Evaluating Differential Privacy Guarantees." NIST SP 800-226, 2025. https://csrc.nist.gov/pubs/sp/800/226/final

[R31] F. Chirigati, D. Shasha, and J. Freire. "Using Provenance to Support Computational Reproducibility." TaPP, 2013. https://www.usenix.org/conference/tapp13/technical-sessions/presentation/chirigati

[R32] E. Dolstra, M. de Jonge, and E. Visser. "Nix: A Safe and Policy-Free System for Software Deployment." LISA, 2004. https://nixos.org/research/

[R33] Open Container Initiative. "Image Descriptor Specification." https://github.com/opencontainers/image-spec/blob/main/descriptor.md

[R34] Bioconductor. "Open source software for bioinformatics." https://www.bioconductor.org/

[R35] 1000 Genomes Project Consortium (A. Auton et al.). "A global reference for human genetic variation." Nature, 526:68-74, 2015. https://doi.org/10.1038/nature15393

[R36] S. Fairley, E. Lowy-Gallego, E. Perry, and P. Flicek. "The International Genome Sample Resource (IGSR) collection of open human genomic variation resources." Nucleic Acids Research, 48:D941-D947, 2020. https://doi.org/10.1093/nar/gkz836

[R37] B. S. Weir and C. C. Cockerham. "Estimating F-Statistics for the Analysis of Population Structure." Evolution, 38(6):1358-1370, 1984. https://doi.org/10.2307/2408641