

A MERGED HOMOMORPHIC CRYPTOGRAPHIC AND HASHING TECHNIQUE FOR SECURE AND INTEGRITY-PRESERVING CLOUD DATA

¹Pooja Gupta, ²Dr. Udai Shankar (Assistant Professor)

¹Research Scholar, ²Research Supervisor

¹⁻²Department of Computer Science, Sunrise University, Alwar, Rajasthan

Abstract

Cloud computing has transformed data storage and computation by allowing organizations to outsource infrastructure and processing to remote service providers. This convenience, however, creates a persistent security problem: sensitive information must remain confidential even while it is being processed, and its integrity must remain verifiable throughout storage, transmission, and computation. Conventional encryption protects data at rest and in transit but ordinarily requires decryption before computation, thereby creating a plaintext exposure point. Cryptographic hashing, on the other hand, provides a compact and highly sensitive fingerprint for detecting unauthorized modification but does not conceal the underlying information. This analytical study examines a merged security technique in which homomorphic encryption and cryptographic hashing are used as complementary mechanisms for secure cloud data management. The proposed conceptual framework encrypts data with a homomorphic cryptosystem before cloud outsourcing, computes and securely associates a cryptographic digest with the protected object or its defined canonical representation, permits authorized operations over ciphertext, and verifies integrity before accepting retrieved data or computed results. The analysis distinguishes confidentiality, computational privacy, integrity, authentication, and verifiability rather than treating them as interchangeable properties. It evaluates the merged approach against conventional encryption-only, hashing-only, and standalone homomorphic designs using security coverage, computational overhead, storage expansion, scalability, and cloud suitability as analytical dimensions. The study finds that the merged architecture can provide stronger end-to-end assurance because homomorphic encryption reduces plaintext exposure while hashing offers efficient tamper detection. Its principal limitations are ciphertext expansion, expensive homomorphic operations, key-management complexity, and the need to define exactly what object is hashed when ciphertexts may be randomized or transformed.

Keywords: Cloud computing; homomorphic encryption; cryptographic hashing; data confidentiality; data integrity; privacy-preserving computation; SHA-256; secure cloud storage.

1. Introduction

Cloud services now support databases, analytics, artificial intelligence pipelines, enterprise applications, backups, and collaborative computing. The security model is therefore no longer limited to protecting a file while it travels across a network. A cloud customer must consider what happens to information while stored by a third party, while copied between services, while processed by remote infrastructure, and while returned as a result. This expanded attack surface makes confidentiality and integrity two separate but simultaneous requirements. Confidentiality asks whether an unauthorized party can learn the content of data. Integrity asks whether unauthorized alteration can be detected. A cloud protection mechanism that addresses only one of these requirements leaves an important gap.

Traditional encryption is highly effective for confidentiality when keys are well protected, but ordinary encrypted data

must generally be decrypted before useful computation is performed. If the cloud server receives the decryption key or plaintext, the protection boundary moves inside infrastructure that the data owner may not fully control. Homomorphic encryption changes this model by supporting mathematical operations directly over ciphertext. The cloud can manipulate protected values and return an encrypted result that is decrypted only by an authorized key holder. Recent literature continues to identify this property as a major advantage for privacy-preserving cloud computation, while also emphasizing the substantial computation and storage overhead of practical HE deployments. Hash functions solve a different problem. A secure cryptographic hash maps an arbitrary-length input to a fixed-length digest and is designed to make preimage discovery and practical collision construction computationally infeasible. Even a small change in the input should produce a substantially different digest. This makes hashing useful for integrity verification, content identification, digital signatures, Merkle structures, and authenticated audit records. Yet a digest cannot replace encryption: hashing does not provide reversible confidentiality, and an unkeyed hash by itself does not prove who created a value. The analytical premise of this study is therefore that HE and hashing should be merged because they protect different dimensions of cloud trust.

The proposed merged technique is not presented as a claim that concatenating two cryptographic primitives automatically produces a secure protocol. Secure composition requires explicit choices concerning key generation, hash placement, metadata binding, freshness, access control, verification rules, and the threat model. In particular, randomized homomorphic encryption may generate different ciphertexts for the same plaintext, and legitimate cloud-side computation intentionally changes ciphertext. Consequently, a robust design must state whether the integrity digest covers the original plaintext, uploaded ciphertext, metadata, result ciphertext, or a combination of these objects. This study develops that distinction and uses it to construct a practical analytical framework.

2. Research Problem and Motivation

Cloud users frequently depend on service providers for both storage and computation. A malicious outsider, compromised account, insider, faulty service, or transmission error may expose or alter outsourced data. Encryption-only systems reduce disclosure but do not necessarily give the owner a lightweight independent method to recognize unauthorized modification. Hash-only systems detect change but leave readable information exposed. Conventional authenticated encryption can efficiently protect stored objects, but it does not by itself enable general computation over ciphertext. This creates a three-part requirement: preserve confidentiality, retain useful computation, and verify integrity.

The research problem addressed here is how to combine homomorphic cryptography and secure hashing so that cloud data can remain encrypted during selected computations while unauthorized changes to stored or returned objects can be detected. The problem is analytically important because HE is often discussed as if it also automatically guarantees every form of integrity. In reality, confidentiality of encrypted computation and integrity of a stored object are distinct security goals. The merged model therefore introduces an explicit integrity layer rather than assuming that encryption alone is sufficient.

The motivation is strongest in domains where the cloud is useful precisely because it can compute on large datasets, yet plaintext disclosure is undesirable. Examples include aggregated financial statistics, privacy-preserving medical analytics, encrypted sensor processing, outsourced scoring, and confidential enterprise metrics. The proposed model

is intended as a general architecture rather than a domain-specific implementation.

3. Objectives of the Study

1. To analyze the security limitations of encryption-only and hashing-only approaches for outsourced cloud data.
2. To examine the role of homomorphic encryption in maintaining confidentiality during cloud-side computation.
3. To examine the role of cryptographic hashing in detecting unauthorized data modification.
4. To develop a merged conceptual framework that combines encrypted computation with explicit integrity verification.
5. To compare the merged framework with conventional and standalone security approaches on security coverage, efficiency, scalability, and implementation complexity.
6. To identify practical limitations, attack considerations, and deployment requirements for real-world cloud environments.

4. Conceptual Background

4.1 Cloud Data Security Requirements

A secure cloud design should distinguish confidentiality, integrity, availability, authentication, authorization, and accountability. Confidentiality prevents unauthorized disclosure; integrity provides evidence that protected information has not been modified outside authorized rules; authentication establishes an identity or origin; authorization controls permitted operations; availability concerns timely access; and accountability supports traceable actions. The merged technique in this study concentrates on confidentiality and integrity but must interoperate with identity, access control, logging, and key-management mechanisms.

4.2 Homomorphic Encryption

Homomorphic encryption permits one or more operations on ciphertext such that decrypting the resulting ciphertext yields a value corresponding to an operation on the underlying plaintext. If Enc denotes encryption and f is a supported function, the desired relationship can be expressed conceptually as $Dec(Eval(f, Enc(m))) = f(m)$. Partially homomorphic encryption supports a restricted operation such as repeated addition or multiplication. Somewhat or leveled homomorphic encryption supports bounded circuits, while fully homomorphic encryption aims to evaluate arbitrary circuits subject to practical parameter and performance constraints. Paillier is a well-known additive homomorphic system, whereas modern lattice-based FHE schemes are designed for broader encrypted computation.

4.3 Cryptographic Hashing

A cryptographic hash function H maps input x to a fixed-length digest $h = H(x)$. For integrity use, desirable properties include preimage resistance, second-preimage resistance, and collision resistance. SHA-256 produces a 256-bit digest and remains widely used for integrity-oriented applications. When a stored object is later retrieved, its hash can be recomputed and compared with a trusted reference digest. Equality indicates that the compared byte representation is unchanged; inequality indicates modification, corruption, or representation mismatch.

4.4 Why Merge the Two Techniques?

Homomorphic encryption and hashing are complementary. HE answers the question, “Can the cloud compute without learning the protected values?” Hashing answers the question, “Can a party detect that a protected representation has changed?” Merging them provides layered assurance. The cloud stores and processes ciphertext rather than plaintext,

while a digest-based verification path detects unauthorized changes to protected objects and associated metadata. The combination is therefore stronger conceptually than treating either primitive as a complete cloud-security solution.

5. Proposed Merged Homomorphic Cryptographic and Hashing Framework

5.1 System Entities

The conceptual system contains four principal entities: a data owner, an authorized data user, a cloud service provider, and an optional independent integrity/audit component. The data owner generates or controls cryptographic keys, prepares data, computes trusted integrity values, and defines permitted operations. The cloud stores ciphertext and performs authorized homomorphic evaluation. The authorized user obtains only the keys and permissions required by policy. The integrity component may be client-side, a trusted verification service, or an authenticated append-only record that prevents silent replacement of the reference digest.

5.2 Processing Workflow

Stage	Analytical Function
1. Data preparation	The owner canonicalizes or serializes the data so that hashing is deterministic and representation ambiguity is minimized.
2. Hash generation	A SHA-256 digest is computed over the selected protected object and security-relevant metadata, or an HMAC/signature is used when origin authentication is required.
3. Homomorphic encryption	The prepared plaintext is encrypted with an HE public key using parameters appropriate to the intended computation.
4. Secure outsourcing	Ciphertext, permitted metadata, and integrity-related identifiers are uploaded; the trusted reference digest is protected separately or authenticated.
5. Encrypted computation	The cloud evaluates only approved functions over ciphertext and returns a result ciphertext.
6. Verification	The client verifies the integrity of the retrieved stored object and relevant metadata. Result integrity may additionally require a verifiable-computation mechanism, authenticated transcript, or application-specific proof rather than a simple hash alone.
7. Decryption and acceptance	Only after required checks succeed is the ciphertext result decrypted and accepted by the authorized party.

5.3 Architecture of the Merged Technique

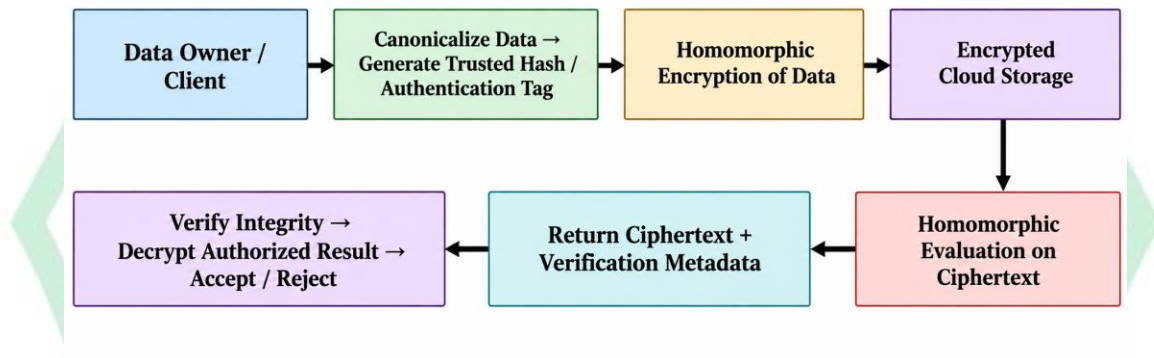


Figure 1. Conceptual workflow of the merged homomorphic encryption and hashing technique.

5.4 Cryptographic Representation

Let m denote a plaintext data object, pk the HE public key, sk the secret key, and H a collision-resistant hash function. The owner computes ciphertext $c = \text{Enc}(pk, m)$. For storage-integrity checking, the owner may compute $h_c = H(c \parallel \text{meta})$, where meta contains immutable security-relevant metadata such as object identifier, version, algorithm identifier, and policy context. The trusted value h_c must be retained outside the attacker's modification domain or authenticated. When c' is retrieved, integrity succeeds only when $H(c' \parallel \text{meta}')$ equals the trusted h_c and metadata validation succeeds.

For homomorphic computation, the cloud evaluates $c_r = \text{Eval}(pk, f, c)$. Because c_r is intentionally different from c , the original upload hash cannot be used to prove that c_r is the correct evaluation result. A new hash of c_r can protect that result against later alteration, but it does not prove that the cloud evaluated f correctly. This is an important analytical boundary. Correctness of outsourced computation may require a homomorphic signature, zero-knowledge proof, trusted execution evidence, SNARK-style proof, or another verifiable-computation mechanism. The merged technique therefore uses hashing primarily for object integrity and tamper evidence, not as a substitute for a proof of computational correctness.

5.5 Hash Placement Strategies

Strategy	Protected Object	Strength	Main Limitation
Plaintext hash before encryption	Canonical plaintext	Verifies recovered content	Digest may leak equality information if exposed; must be protected.
Ciphertext hash	Uploaded ciphertext + metadata	Efficient storage tamper detection	Randomized re-encryption changes digest.
Dual hash	Plaintext and ciphertext in separate trust domains	End-to-end and storage-level checks	More metadata and management.
HMAC over ciphertext	Ciphertext + metadata	Integrity plus keyed authentication	Requires secret MAC-key management.

Signed digest	Digest + metadata signed by owner	Publicly verifiable origin and integrity	Signature/key lifecycle adds complexity.
---------------	--------------------------------------	---	---

6. Threat Model and Security Analysis

6.1 Threat Model

The baseline model assumes an honest-but-curious or potentially faulty cloud that may inspect stored values and metadata and may lose, replace, reorder, or modify stored objects. External attackers may intercept traffic or compromise cloud credentials. The cryptographic assumptions are that HE keys remain protected, the selected HE scheme is secure under its stated hardness assumptions and parameter set, and the selected hash remains collision resistant. Denial-of-service is not prevented by encryption or hashing and is therefore outside the principal protection scope.

6.2 Confidentiality Analysis

If data are encrypted locally and the secret key is not disclosed to the cloud, the provider sees ciphertext rather than plaintext. Supported computation can proceed in the encrypted domain, reducing the need for plaintext exposure. This is the defining security advantage of HE over conventional server-side computation. Confidentiality still depends on secure parameters, key storage, side-channel resistance, metadata minimization, and careful control of decryption interfaces. HE should therefore be considered a powerful cryptographic layer, not a complete access-control system.

6.3 Integrity Analysis

A trusted digest enables efficient detection of unauthorized changes to the exact representation covered by the hash. If a ciphertext bit, immutable metadata field, or stored block changes, recomputation produces a different digest except with negligible collision probability under the assumed hash security. The integrity guarantee fails if an attacker can replace both the object and unauthenticated reference digest. Accordingly, the reference must be retained by the client, protected by HMAC, signed, or anchored in another authenticated mechanism.

6.4 Replay, Substitution and Metadata Attacks

Hashing only the ciphertext is insufficient against every logical attack. An adversary might replay an older but valid ciphertext, substitute one valid object for another, or alter an unbound object identifier. Binding a version number, owner identifier, object identifier, algorithm suite, timestamp or monotonic counter, and policy context into authenticated metadata reduces these risks. Freshness must be explicitly enforced; collision resistance alone does not tell the verifier whether a valid object is the newest authorized version.

7. Comparative Analytical Evaluation

The following comparison is qualitative and architecture-based; it is not presented as a benchmark measured on a specific hardware platform. Actual performance depends heavily on the HE scheme, polynomial degree, modulus chain, security level, batching strategy, data size, network conditions, and implementation library.

Criterion	Hashing Only	Conventional Encryption	Standalone HE	Merged HE + Hashing
Confidentiality	No	Yes	Yes	Yes

Tamper detection	Yes, if digest trusted	Depends on authenticated mode	Not automatically complete	Explicit integrity layer
Compute on encrypted data	No	Generally no	Yes	Yes
Storage overhead	Very low	Low	High	High + small digest
Compute overhead	Very low	Low	High to very high	High to very high
Key-management burden	Low/none	Moderate	High	High
Cloud analytics suitability	Low for secrecy	Moderate	High for privacy-sensitive workloads	High where privacy + integrity both matter
Implementation complexity	Low	Low–moderate	High	High

The merged design does not remove the cost of HE. Hash computation is comparatively lightweight, so most incremental overhead arises from homomorphic encryption, ciphertext expansion, and encrypted evaluation. The integrity layer adds only a small fixed-size digest per object or block, although Merkle trees, signatures, or versioned metadata add further storage. This means optimization efforts should focus mainly on HE parameter selection, batching, operation depth, and avoiding unnecessary fully homomorphic functionality when a partially or leveled scheme is sufficient.

A 2025 edge-cloud simulation reported substantial reductions in encryption time and ciphertext size from optimized hybrid HE configurations relative to its standard HE baseline, illustrating that implementation architecture can materially change performance. A 2024 cloud-storage study likewise combined cryptographic mechanisms rather than relying on one primitive and reported measurable encryption and storage overhead. These results support the broader analytical position that secure cloud systems are often best treated as composed protocols with workload-specific trade-offs rather than single-algorithm solutions.

8. Performance and Scalability Considerations

8.1 Computational Cost

Hashing scales approximately linearly with the number of input bytes and is normally inexpensive compared with HE. HE cost depends on the supported arithmetic and cryptographic parameters. Additive Paillier operations may be practical for sums and selected statistics, while FHE circuits involving multiplication, comparisons, nonlinear functions, or repeated bootstrapping can be substantially more expensive. For this reason, the merged architecture should select the least powerful HE category that satisfies the workload rather than defaulting to FHE.

8.2 Ciphertext Expansion

Homomorphic ciphertexts can be much larger than plaintext values. Expansion affects storage cost, memory pressure, transmission time, and cache behavior. Packing or batching multiple plaintext slots into one ciphertext can improve amortized efficiency for vectorized workloads. The hash layer adds comparatively little size—32 bytes for a raw SHA-256 digest—although secure metadata, signatures, proofs, and version history can increase total overhead.

8.3 Parallelism and Cloud Suitability

Cloud platforms offer parallel compute resources that can partly offset HE latency. Independent ciphertext operations can often be distributed across cores, machines, or accelerators. However, cloud scaling does not eliminate algorithmic cost and may increase financial expense. Practical deployments should therefore measure throughput per cost unit, not only wall-clock latency. Workloads with repeated aggregation over large encrypted vectors are generally more suitable than highly interactive applications requiring millisecond-level responses.

9. Application Scenarios

9.1 Privacy-Preserving Financial Analytics

A financial organization can encrypt transaction attributes before cloud outsourcing and permit selected aggregate computations such as sums or risk statistics without exposing raw values. Hash-based verification protects stored ciphertext objects and versioned metadata from silent alteration.

9.2 Healthcare and Biomedical Data

Hospitals and research groups can outsource encrypted numerical datasets for approved statistical analysis while limiting plaintext disclosure. Integrity verification helps identify corrupted or substituted records before decrypted results are accepted. Access policy, consent, and regulatory requirements remain separate governance obligations.

9.3 IoT and Smart Infrastructure

Sensor readings can be encrypted at gateways and aggregated in the cloud. Hashes or authenticated tree structures can support integrity checking of batches or blocks. Resource-constrained devices may favor partial HE or edge-assisted preprocessing because full HE can exceed device capabilities.

9.4 Confidential Enterprise Analytics

Organizations can perform selected cloud-based analysis over protected business metrics, forecasts, or operational data. A merged design is particularly useful when the organization wants both privacy from the infrastructure operator and independent evidence that stored encrypted objects remain unchanged.

10. Limitations and Implementation Challenges

Challenge	Analytical Implication
High HE overhead	Homomorphic operations remain slower and ciphertexts larger than ordinary computation, especially for deep circuits.
Scheme–workload mismatch	An HE scheme optimized for integer addition may be unsuitable for approximate real-number analytics or complex branching.
Integrity scope ambiguity	The system must specify whether the digest covers plaintext, ciphertext, metadata, blocks, or result objects.
Computation verification gap	A hash detects alteration but does not by itself prove that the cloud executed the requested function correctly.

Key compromise	If secret decryption keys are stolen, hashing cannot restore confidentiality.
Metadata leakage	File size, access patterns, timing, and query structure can leak information even when content is encrypted.
Freshness and rollback	A valid old object can pass a simple hash comparison unless versions or counters are authenticated.
Cryptographic agility	Algorithms and parameter sets may need migration as standards, implementations, and attack capabilities evolve.

11. Discussion

The central analytical result is that homomorphic encryption and cryptographic hashing should be treated as complementary rather than competing techniques. HE is valuable because it extends confidentiality into the computation stage, a point at which conventional encryption frequently requires plaintext. Hashing is valuable because it provides an efficient and compact mechanism for detecting unauthorized change to a precisely defined object. Their merger therefore addresses two different security questions within one cloud workflow.

The most important design decision is not simply which hash function or HE scheme to choose, but how the primitives are composed. A digest that is stored beside the protected object without authentication can be replaced. A plaintext digest exposed to an adversary may reveal equality or enable dictionary testing for low-entropy data. A ciphertext digest cannot validate a legitimately transformed homomorphic result against the original upload. These observations show why secure protocol design requires a threat model and authenticated metadata rather than algorithm names alone.

The merged approach is most compelling when cloud computation is necessary and the data owner does not wish to expose plaintext to the computing service. For storage-only use cases, authenticated encryption may be simpler and faster. For analytics requiring only additive aggregation, partial HE may offer a better cost-security balance than FHE. For adversarial computation where the server may intentionally return a false result, a verifiable-computation layer should be integrated. Thus, the framework should be adapted to the actual security objective instead of maximizing cryptographic complexity.

12. Conclusion and Future Scope

This study analyzed a merged homomorphic cryptographic and hashing technique for secure and integrity-preserving cloud data. The analysis demonstrates that homomorphic encryption can protect confidentiality during outsourced computation, while cryptographic hashing can provide efficient tamper detection for stored or transmitted representations. The two mechanisms solve different problems and can therefore strengthen one another when correctly composed. A robust implementation should encrypt data before outsourcing, bind security-relevant metadata to integrity checks, protect reference digests against replacement, distinguish storage integrity from computation correctness, and decrypt only after required verification succeeds.

The approach is not cost-free. Homomorphic encryption introduces significant computational and storage overhead, and secure deployment requires careful parameter selection and key management. Hashing also has limits: it does not

provide confidentiality, freshness, authorization, or proof that a remote computation was honestly executed. Future work should therefore examine benchmarked implementations using Paillier, BFV, BGV, or CKKS according to workload; Merkle-tree or HMAC-based block verification; zero-knowledge or succinct proofs for computation correctness; threshold key management; hardware acceleration; and adaptive selection between partial and fully homomorphic modes. Experimental validation should report encryption, evaluation, verification, decryption, ciphertext expansion, throughput, communication cost, and security parameters on explicitly identified hardware and datasets.

References

1. Ahmed, H. M. A., Zubair, S., & Tawfik, M. (2024). A privacy-preserving cloud storage framework with hybrid encryption, homomorphic keyword search, and blockchain-based integrity verification. *Scientific Reports*, 16, 16697. <https://doi.org/10.1038/s41598-026-48705-x>
2. Ayitey Junior, M., Appiahene, P., Appiah, O., et al. (2025). Cloud data privacy protection with homomorphic algorithm: A systematic literature review. *Journal of Cloud Computing*, 14, 84. <https://doi.org/10.1186/s13677-025-00774-5>
3. Deng, X. (2025). Homomorphic encryption-based data integrity verification and anti-tampering mechanism in cloud storage environment. *Preprints*. <https://doi.org/10.20944/preprints202512.0870.v1>
4. Gaitond, R., Biradar, G. S., & Terdal, S. (2025). Blockchain-integrated optimized cryptographic framework for securing cloud data. *Knowledge-Based Systems*, 324, 113830.
5. Parthasarathy, K., Panga, N. K. R., Bobba, J., Bolla, R. L., Ayyadurai, R., & Hemnath, R. (2025). Fortifying cloud security with advanced data encryption technique. *Journal of Science & Technology*, 10(3), 20–27. <https://doi.org/10.46243/jst.2025.v10.i03.pp20-27>
6. Comparative simulation of homomorphic encryption techniques in edge-cloud computing: Reducing computational and storage overhead. (2025). *Procedia Computer Science*, 269, 438–448. <https://doi.org/10.1016/j.procs.2025.08.296>
7. Fully homomorphic encryption-based optimal key encryption for privacy preservation in the cloud sector. (2025). *Journal of Information Security and Applications*, 91, 104048. <https://doi.org/10.1016/j.jisa.2025.104048>
8. Enhancing cloud data security using Paillier homomorphic encryption algorithm. (2025). 3rd International Conference on Integrated Circuits and Communication Systems (ICICACS). <https://doi.org/10.1109/ICICACS65178.2025.10968860>
9. Integrated protocol for verifiable confidential cloud computing and data integrity auditing. (2025). *Journal of King Saud University Computer and Information Sciences*.
10. Sharma, I., & Kak, S. (2016). Homomorphic encryption for security of cloud data. *Procedia Computer Science*, 79, 175–181. <https://doi.org/10.1016/j.procs.2016.03.023>
11. Gentry, C. (2009). Fully homomorphic encryption using ideal lattices. *Proceedings of the 41st Annual ACM Symposium on Theory of Computing*, 169–178.
12. Paillier, P. (1999). Public-key cryptosystems based on composite degree residuosity classes. *Advances in Cryptology—EUROCRYPT '99*, 223–238.

13. National Institute of Standards and Technology. (2015). Secure Hash Standard (SHS), FIPS PUB 180-4.
14. Boneh, D., Goh, E.-J., & Nissim, K. (2005). Evaluating 2-DNF formulas on ciphertexts. Theory of Cryptography Conference, 325–341.
15. Brakerski, Z., Gentry, C., & Vaikuntanathan, V. (2014). (Leveled) fully homomorphic encryption without bootstrapping. ACM Transactions on Computation Theory, 6(3), Article 13.

