Homomorphic Encryption for Machine Learning in Medicine and Bioinformatics

ALEXANDER WOOD and KAYVAN NAJARIAN, University of Michigan
DELARAM KAHROBAEI, University of York, United Kingdom

Machine learning and statistical techniques are powerful tools for analyzing large amounts of medical and genomic data. On the other hand, ethical concerns and privacy regulations prevent free sharing of this data. Encryption techniques such as fully homomorphic encryption (FHE) enable evaluation over encrypted data. Using FHE, machine learning models such as deep learning, decision trees, and Naive Bayes have been implemented for privacy-preserving applications using medical data. These applications include classifying encrypted data and training models on encrypted data. FHE has also been shown to enable secure genomic algorithms, such as paternity and ancestry testing and privacy-preserving applications of genome-wide association studies.

This survey provides an overview of fully homomorphic encryption and its applications in medicine and bioinformatics. The high-level concepts behind FHE and its history are introduced, and details on current open-source implementations are provided. The state of fully homomorphic encryption for privacy-preserving techniques in machine learning and bioinformatics is reviewed, along with descriptions of how these methods can be implemented in the encrypted domain.

CCS Concepts: • **Security and privacy → Cryptography**; **Usability in security and privacy**; • **Social and professional topics → Medical information policy**;

Additional Key Words and Phrases: Fully homomorphic encryption, homomorphic encryption, neural networks, machine learning, bioinformatics, data privacy and security, secure outsourcing computation

ACM Reference format:
Alexander Wood, Kayvan Najarian, and Delaram Kahrobaei. 2020. Homomorphic Encryption for Machine Learning in Medicine and Bioinformatics. *ACM Comput. Surv.* 53, 4, Article 70 (August 2020), 35 pages.
https://doi.org/10.1145/3394658

1 INTRODUCTION

Increasing access to big data continues to transform medicine. The cost of genome sequencing has reduced dramatically, leading to the proliferation of commercial personalized genetic analysis and enabling large-scale genome sequencing for research, and hospitals collect far more data on patients than clinicians are able to examine. Machine learning techniques harness the power of this data by learning rules from datasets too large or complex for any human to analyze.

Delaram Kahrobaei is supported in part by the Office of Naval Research grant N000141512164.

Authors' addresses: A. Wood and K. Najarian, University of Michigan, Ann Arbor, MI, 48109, USA; emails: {anwood, kayvan}@med.umich.edu; D. Kahrobaei, University of York, Heslington, York YO10 5DD, United Kingdom; email: delaram.kahrobaei@york.ac.uk.

0360-0300/2020/08-ART70 $15.00
https://doi.org/10.1145/3394658

Transformational applications in clinical medicine include prognosis models trained over massive, many-featured datasets, interpretation of medical imaging, and generation of differential diagnoses [42, 161].

Machine learning on genomic and medical data requires careful consideration of privacy concerns. A breach in the privacy of this data can have negative consequences for an individual, leading to the passage of laws such as the Health Insurance Portability and Accountability Act (HIPAA) in the United States [162] and the General Data Protection Regulation in the European Union [91].

Commercial genomic analysis opens the door to both privacy and civil liberties concerns. The United States federal government and all 50 states allow the creation of DNA databases and admit DNA evidence in courts [196]. This data can affect not just the individual, but also relatives with whom an individual shares DNA [42]. Remarkably, the notorious “Golden State Killer,” accused of a series of decades-old murders in California, was arrested in 2018 after law enforcement used a commercial genealogy website to locate relatives of the suspect using crime scene DNA. These methods raise concerns over the impact such powerful tools could have on the general population [130].

Privacy-preserving techniques for computation and machine learning can reduce the risk to patients who choose to share their data for personalized medical analysis. Encryption enables patients to secure and prevent third parties from analyzing their data. While traditional encryption does not generally allow for computation over encrypted data points, fully homomorphic encryption (FHE), described as the “holy grail” of cryptography, enables arbitrary computation over encrypted data [95, 199]. This opens the door to privacy-preserving applications of computational techniques such as machine learning and statistical analysis to genomic and medical data and outsourcing of computation.

This work provides a broad overview of FHE, its history, current techniques and open-source implementations, and the state-of-the-art in applications of FHE for machine learning and statistical analysis in the medical field. In contrast, previous surveys provide overviews of FHE schemes for mathematicians [97, 190, 199] or engineers [148]. One 2015 survey focuses on genomic applications [15, 158], and another on the state of fully homomorphic encryption research [11], though many new techniques and applications have been presented since its publication . A recent survey provides an overview of FHE but is not focused on applications [1]. Other recent surveys discusses applications of secure outsourced computation more generally [187, 203], while this survey focuses instead on specifics of HE for applications in the medical field. Additional surveys focus on HE for signal processing [3, 136], hardware implementation [154], theory [12, 111], and lattice cryptography [166].

This survey is targeted toward all who are interested in privacy-preserving techniques in the medical field using fully homomorphic encryption. This includes clinicians, computer scientists, engineers, developers, graduate students, and any with some background in encryption, genomics, or machine learning. It seeks to introduce readers to the high-level concepts behind fully homomorphic encryption and familiarize readers with terms, definitions, and schemes necessary to begin applying FHE. A detailed summary of currently available open-source implementations of FHE is given. Applications of FHE for privacy-preserving machine learning in medicine and bioinformatics are reviewed, and methods for performing these privacy-preserving techniques are summarized. This survey gives a comprehensive overview of the applications of FHE in for machine learning in medicine. In particular, details are provided on how to implement privacy-preserving logistic regression, Naive Bayes, decision tree, neural networks, and unsupervised learning using FHE. In addition, privacy-preserving methods for DNA/RNA sequence comparisons, Hamming

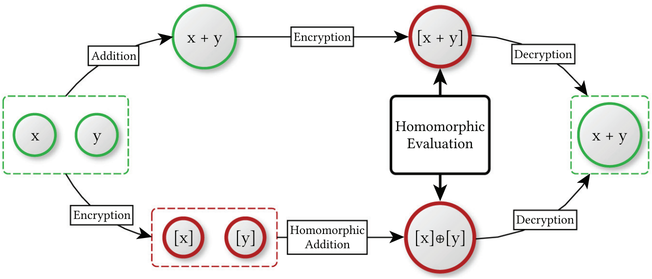

Fig. 1. Under an additive homomorphism, encryption followed by homomorphic addition is equal to addition followed by encryption.

and edit distance, genetic testing, and statistical tests for performing genome-wide association studies are reviewed.

1.0.1 Organization. Section 2 discusses the history of homomorphic encryption, defines the related terminology, and discusses schemes and techniques in the field. Section 3 provides an overview of open-source FHE libraries and implementation tools. Section 4 surveys the current status of homomorphic encryption for privacy-preserving machine learning in medicine. Section 6 discusses applications of FHE in privacy-preserving bioinformatics algorithms.

2 HOMOMORPHIC ENCRYPTION

Cryptography was originally developed as a technique for secure communication between multiple parties, where one party encrypts a message and sends it to another party, who then decrypts it [199]. The concept of computing over encrypted data was first introduced as a "privacy transformation" by Rivest, Adleman, and Dertouzos in 1978 and developed into the study of homomorphic encryption today [179]. Broadly, homomorphic encryption enables computation over encrypted data. A scheme called additively (or multiplicatively) homomorphic if

$$[x] \oplus [y] = [x + y] \quad \text{and} \quad [x] \otimes [y] = [x \cdot y]$$

for addition and multiplication, respectively, where $[m]$ denotes the encryption of some plaintext m. The symbols $\oplus$ and $\otimes$, respectively, denote the homomorphic addition and multiplication operations in the ciphertext space. In other words, if an encryption scheme is additively homomorphic, then encryption followed by homomorphic addition is equal to addition followed by encryption, a concept illustrated in Figure 1.

A partially homomorphic encryption (PHE) scheme can perform a single operation, such as addition or multiplication, over encrypted data an arbitrary number of times. ElGamal [85] and RSA [180] are multiplicatively homomorphic, although versions of RSA with improved security tend to lose this capability [199]. Goldwasser and Micali's quadratic reciprocity scheme [107, 108] and subsequent improvements [19, 20, 156, 163] led to Paillier's additively homomorphic encryption scheme [164], one of the most well-known HE schemes [87]. Other PHE schemes include adaptations of Paillier [73, 74, 90] and some lattice-based schemes [4, 104, 174].

While PHE allows for arbitrary computation of a single operation on encrypted data, somewhat homomorphic encryption (SHE) allows for bounded computation of a set of operations. For instance, the somewhat homomorphic Boneh-Goh-Nissim (BGN) encryption scheme can perform unlimited homomorphic addition and a single homomorphic multiplication operation [26].

FHE enables arbitrary addition *and* multiplication over encrypted data. Because arbitrary functions can be described as Boolean circuits, an encryption scheme that can compute addition and multiplication can theoretically evaluate any function [166]. Schemes that can perform homomorphic operations over a circuit of some pre-defined depth are called leveled homomorphic encryption (LHE) schemes. In practice, considerations such as execution time, ciphertext noise accumulation, key sizes, ciphertext sizes, and circuit depth limit the class of functions that can be practically computed using FHE/LHE.

The first fully homomorphic scheme was published by Craig Gentry in his 2009 thesis [93, 94]. The introduction of Gentry's revolutionary FHE scheme quickly led to a series of improvements that laid the foundation for a wealth of future work. This development has been categorized in multiple ways. Fully homomorphic encryption schemes have been sorted according to their theoretical underpinnings.

Peikert categorizes FHE schemes into three generations of research [166]. First-generation FHE schemes are based on Gentry's initial breakthrough, with security based on ad-hoc average-case assumptions about ideal lattices or the *approximate greatest common divisor (GCD)* problem, while second-generation FHE schemes are based on *learning with errors (LWE)* and *ring learning with errors (RLWE)*. He describes third-generation FHE as marked by practical improvements and further refinement of security assumptions. Additional works follow Peikert's blueprint [111]. Acar et al. [1] and Martins et al. [148] describe four categories of FHE schemes: those based on ideal lattice assumptions, the approximate GCD problem, (R)LWE, and N^{th} degree–truncated polynomial ring unit (NTRU)-like assumptions [1].

FHE research has continued rapidly past Peikert's third generation of schemes. The current generation of FHE research is marked by a focus on open-source implementations, real-world applications, and standardization of methodology and security requirements. The Homomorphic Encryption Standardization Consortium[1] has described contemporary FHE as being based in three models of homomorphic computation: Boolean circuits, modular (exact) arithmetic, and approximate number arithmetic [62]. This section provides a high-level overview of the develop of fully homomorphic encryption.

2.1 Gentry's Breakthrough

Gentry's breakthrough scheme is built upon an abstract algebra construct called a lattice [166]. He begins by constructing a somewhat homomorphic encryption scheme that is able to homomorphically compute low-degree polynomials. Each homomorphic evaluation adds noise to a ciphertext, and the eventual accumulation of too much noise results in an undecryptable ciphertext.

Gentry handles noise by introducing *bootstrapping*, which reduces the rate of noise accumulation in ciphertexts. A bootstrappable SHE scheme is able to handle its own decryption function. A noisy ciphertext is double-encrypted, then "homomorphically" decrypted using an encryption of the private key. To illustrate, Gentry compares bootstrapping to a locked glovebox, where the glovebox represents encryption, the lock represents the decryption key, and manipulation of items in the box using the gloves represents homomorphic evaluation [95]. Noise accumulation is symbolized by the gloves developing defects from use. When the gloves become defective the original box and its key are placed inside of a second locked glovebox, where it can be unlocked. Object manipulation continues inside of the second glovebox. Implementing the fully homomorphic properties of Gentry's original scheme is impractical largely due to the computational cost of bootstrapping.

Gentry's bootstrapping method blazed the trail for the first wave of rapid innovation in FHE [157]. An FHE scheme introduced by Smart and Vercauteren has smaller key and ciphertext sizes, and a SHE version of the scheme was implemented [191]. However, Cramer et al.'s 2016 key-recovery attack compromises this system's security assumptions [67]. Additional variants

[1] https://homomorphicencryption.org/.

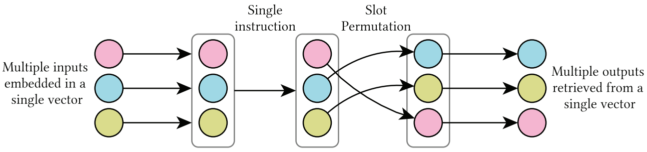
Single instruction
Slot Permutation
Multiple inputs embedded in a single vector
Multiple outputs retrieved from a single vector

Fig. 2. From left to right, multiple values are embedded within a single vector. A single operation simultaneously manipulates each value. Slot permutation re-orders the values within the vector.

follow Gentry's blueprint [96, 98, 193]. Gentry and Halevi provided the first full implementation of Gentry's FHE scheme [99]. In this implementation, a single bootstrapping operation takes up to 30 minutes to run. An implementation on a GPU yielded faster run times but is still impractical for real-world deployment [201].

In another work, Smart and Vercauteren describe an SHE scheme that supports Single-Instruction Multiple-Data (SIMD) operations, enabling parallelization in FHE via ciphertext packing [192]. SIMD instructions parallelize computation by encoding multiple plaintext values within a single ciphertext vector. Ciphertext packing can be used for batch computation in many FHE schemes. Additional usages permute the plaintext slots inside of a packed ciphertext, enabling, for instance, evaluation speedups [102]. These concepts are illustrated in Figure 2. From left to right, multiple values embedded within a vector can be processed via a single instruction, and their slots may be permuted.

2.2 New Theoretical Underpinnings

After Gentry's introduction of an FHE scheme rooted in ideal lattice-based assumptions, a new wave of innovation yielded FHE schemes with different theoretical underpinnings [166]. In one category, security is based on the *LWE* [174] and *RLWE* [145] assumptions. In the second category, security is based on the approximate-GCD problem [123]. Another category of schemes are based on NTRU-like assumptions [121].

2.2.1 R/LWE: BGV Encryption. A series of papers Brakerski and Vaikuntanathan published in 2011 present schemes based on LWE and ring-LWE that follow the SHE-bootstrap-FHE model [35, 36]. Their initial SHE scheme, based on LWE, is turned into a FHE scheme using their modulus reduction and dimension reduction techniques. These reduce parameter sizes of the ciphertext, making decryption independent from the number of levels the scheme can evaluate. Modulus reduction changes the modulus used during encryption to a smaller value while dimension reduction reduces the dimension of the ciphertext.

Joint work with Gentry followed that resulted in the Brakerski-Gentry-Vaikuntanathan (BGV) scheme [34, 38]. BGV removes the need for expensive bootstrapping procedures using techniques such as modulus reduction in order to mitigate noise growth during homomorphic evaluation. Bootstrapping is left as an optional optimization procedure. Without bootstrapping, modulus reduction yields an LHE scheme that can handle noise up to a pre-specified circuit depth. Further work on BGV encryption incorporates the batching technique of Smart and Vercauteren in BGV [100, 102] and simpler bootstrapping techniques [101]. Gentry, Halevi, and Smart implement BGV-based LHE to evaluate an AES circuit, the first homomorphic evaluation of a complex circuit [103].

2.2.2 R/LWE: BFV Encryption. Fan and Vercauteren's 2012 scheme follows the SHE-bootstrap-FHE framework [86]. It is a modification of a "scale-invariant" scheme introduced by Brakerski that does not require modulus switching [33]. Because of this, it is commonly referred to as the Brakerski-Fan-Vercauteren (BFV) or the Fan-Vercauteren (FV) scheme. Fan and Vercauteren port Brakerski's LWE scheme to the ring-LWE setting and implement SIMD for batch computing. They introduce two variants of this scheme. The first minimizes error that occurs due to noise accumulation, and the second method reduces the time and space used during computation using techniques similar to modulus-switching methods.

2.2.3 Approximate-GCD Based Schemes. In 2010, van Dijk et al. presented their DGHV scheme [78], with security based on the approximate GCD problem [123]. This scheme follows Gentry's SHE-bootstrap-FHE blueprint. A subsequent optimization reduces the public key size in DGHV [65]. Further improvements include an extension to batch FHE [50, 63] and more efficient key-reduction and bootstrapping techniques [64]. The bootstrap-free framework of the BGV scheme is extended to the DGHV scheme via a new modulus-switching technique [66].

2.2.4 NTRU-Like Assumptions. Stehle and Steinfield's 2011 work introduces a modification of an encryption scheme called NTRUEncrypt [121] that makes it as secure as worst-case problems over ideal lattices [194]. López-Alt et al. use the homomorphic properties of NTRUEncrypt to propose the LTV scheme, the first FHE scheme based on NTRU-like assumptions [143]. Rohloff and Cousins introduce another NTRU-like FHE cryptosystem with modified bootstrapping techniques and ciphertext representations [181]. Further customization of the LTV scheme is presented by Doröz et al [79], and YASHE, or "yet another somewhat homomorphic encryption scheme" [29], removed some non-standard assumptions from LTV. However, despite initial promise, Albrecht et al. present attacks on some of the "overstretched" security NTRU-like security assumptions used in LTV and YASHE [6].

2.3 Practical Improvements

From 2012 into 2013 marked the beginning to what Peikert terms the "third generation" of FHE, which introduced new internal methods used within the schemes [21, 166]. In this new wave, practical improvements simplify and raise the efficiency of FHE schemes [8, 36, 105] and the first open-source FHE implementations are released [116, 175].

2.3.1 R/LWE-Based Encryption. Gentry, Sahai, and Waters published an FHE scheme in 2013, known as the GSW scheme, that is based on LWE and does not require key-switching [105]. The GSW scheme is foundational to a number of following schemes and provides a Boolean circuit-based approach to FHE. This method reduces homomorphic multiplication in most cases to matrix multiplication and introduces the approximate eigenvector method of constructing an FHE scheme. Brakerski and Vaikuntanathan apply modulus and dimension reduction to GSW to obtain a leveled FHE scheme [37]. Alperin-Sheriff and Peikert introduce faster bootstrapping [8], which is then applied by Hiromasa et al. to a variant on GSW that encrypts matrices and supports homomorphic matrix addition and multiplication [120].

2.3.2 Approximate-GCD Based Encryption. Cheon and Stehlé introduce a decision variant on the approximate-GCD problem that is at least as hard as the LWE problem, as well as a (non-batched) variant of the DGHV scheme with security based on this new reduction [55]. Benarroch et al. present two more schemes [21]. One of their LHE schemes is non-batched with security based on the approximate-GCD decision problem. The other is a batched scheme with security based in part on the batched approximate-GCD decision problem.

2.4 Implementation and Standardization

The current generation of research is marked by a shift toward real-world, open-source implementation and standardization of FHE methods and security. Technical improvements yielded an assortment of open-source software libraries, which are discussed in Section 3.

Prompted in part by this rapid innovation, starting in 2017 a consortium of experts from the government, industry, and academia formed the Homomorphic Encryption Standardization Consortium[2] and began holding regular workshops to create a community standard for HE. These workshops led to the development of a *Homomorphic Encryption Standard* in 2018 [7]. This standard outlines the security assumptions used in major FHE applications, possible attacks, and recommends security parameters for implementations.

The Consortium has described contemporary FHE as being based in three models of homomorphic computation: Boolean circuits, modular (exact) arithmetic, and approximate number arithmetic [62]. The modular (exact) arithmetic approach follows from the groundwork laid in second-generation FHE by R/LWE-based schemes, notably BGV [34] and BFV [33, 86]. This leveled approach evaluates arithmetic circuits over values encrypted modulo an integer t, with optional bootstrapping. Recent developments include residue number system (RNS) variants of BFV, which provide speedups for modular arithmetic operations [16, 17, 112]. Improvements to the plaintext space in the BFV scheme allows for homomorphic evaluation of deep circuits with rational number inputs [137] and improved bootstrapping techniques [49].

The Boolean circuit-based approach toward FHE evaluates Boolean circuits over encrypted bits. The major schemes following this approach are known as FHEW and TFHE. Ducas and Micciancio's Fasted Homomorphic Encryption in the West (FHEW) scheme [82] introduces a ring variant of Alperin-Sheriff and Peikert's [8] bootstrapping for the GSW scheme [105] and a new homomorphic NAND gate. Bootstrapping runs in less than a second after evaluation of this NAND gate on a single bit. Another Boolean circuit-based scheme, Fast Fully Homomorphic Encryption Over the Torus (TFHE) [59], extends the GSW scheme [105] and encodes polynomials over the torus and evaluates over Boolean circuits [56]. Their techniques also apply to the FHEW cryptosystem. Recent developments include improved bootstrapping methods for TFHE [57, 58], expansion of the bootstrapping operation in FHEW to multiple bits [23, 27], multi-value bootstrapping for TFHE and FHEW [44], and refreshing of multiple ciphertexts at the cost of a single refresh procedure [149].

The approximate number arithmetic approach is led by the Cheon-Kim-Kim-Song (CKKS) scheme, also known as Homomorphic Encryption for Arithmetic of Approximate Numbers (HEAAN) [197]. This article refers to the scheme itself as CKKS and uses HEAAN to refer to the eponymous software. Evaluation of an approximate circuit over a ciphertext returns an approximation rather than an exact result. These approaches are suited for fast polynomial approximation and floating point computations. Fast polynomial approximation in CKKS can evaluate the multiplicative inverse, exponential, and logistic functions, as well as the discrete Fourier transform [197]. Additional works provide bootstrapping improvements [47, 51] and RNS variants [24, 52, 176].

Additional developments include a somewhat homomorphic variation on the GSW scheme presented by Genise et al. [92]. This scheme uses an inhomogeneous variant of the NTRU assumption (iNTRU) for security, a stronger assumption than LWE. This scheme evaluates encrypted, nondeterministic finite automata more efficiently than other existing schemes. Boura et al. present a unifying framework called Chimera that combines the best aspects of the BFV, CKKS, and TFHE schemes [31]. This framework enables switching between the plaintext spaces of the three schemes and describes a collection of bridges to connect the schemes.

3 IMPLEMENTING FULLY HOMOMORPHIC ENCRYPTION

The current generation of FHE research has led to efficient, publicly available FHE libraries and software. These libraries continue to develop with respect to cutting-edge FHE research and the recently drafted community standards for homomorphic encryption [117]. While FHE continues

[2] https://homomorphicencryption.org/.

Table 1. Fully Homomorphic Encryption Libraries and Software

Product	Creator	Language	License	Summary
SEAL [177]	Microsoft	C++	MIT	Widely-used FHE library that implements BFV for modular arithmetic and CKKS for approximate arithmetic.
HElib [116]	IBM	C++	Apache-2.0	Widely-used FHE library that implements BGV for modular arithmetic and CKKS for approximate arithmetic.
TFHE [59]	Gama et al.	C++	Apache-2.0	Implements an optimized ring variant of the GSW scheme.
HEAAN [115]	CryptoLab, Inc.	C++	CC-BY-NC-3.0	Implements the CKKS approximate number arithmetic scheme.
PALISADE [165]	New Jersey Institute of Technology	C++	BSD-2-Clause	Lattice cryptography library that supports multiple protocols for FHE, including BGV, BFV, and StSt.
$\Lambda \circ \lambda$ [69]	E. Crockett and C. Peikert	Haskell	GPL-3.0-only	Pronounced “LOL.” Implements a BGV-type FHE scheme.
Cingulata [45]	CEA LIST	C++	CECILL-1.0	Compiler and RTE for C++ FHE programs. Implements BFV and supports TFHE.
FV-NFLlib [89]	CryptoExperts	C++	GPL-3.0-only	Implements FV scheme. Built on the NFLLib lattice cryptography library. Last updated 2016.
Lattigo [138]	Laboratory for Data Security	Go	Apache 2.0	Implements BFV and HEAAN in Go.

to rapidly increase in efficiency, implementation presents a continuing challenge for non-experts. Implementations must be custom-built, requiring a high level of expertise both as a programmer and as a cryptographer. For example, conversions between plaintext and ciphertext computations are not trivial, noise must be carefully managed to maintain correctness upon decryption, and security parameters must be selected manually [71]. Some early compilers and tools have been introduced to make FHE more approachable and widely usable [45, 124, 167].

An discussion of the major implementations, compilers, and FHE tools is given below. Table 1 provides an overview of the implementations, including the name of the software, its creator, language, license, and a short summary of the implemented schemes.

Microsoft's Simple Encrypted Arithmetic Library (SEAL). The Simple Encrypted Arithmetic Library (SEAL)[3] is a popular homomorphic encryption library developed and maintained by researchers in the Cryptography Research Group at Microsoft Research since 2015. SEAL is self-contained and developed in C++, and includes a .NET wrapper [177]. SEAL has undergone changes in terms of the underlying encryption scheme since 2015. Early implementations of SEAL used a variant on the YASHE scheme [29]. Version 3.3 of SEAL [177] implements the BFV and CKKS schemes. SEAL 3.3 was released in June of 2019 and is available on Microsoft's website [177]. Intel AI has used SEAL's implementation of the CKKS scheme in the Intel HE transformer for nGraph (nGraph-HE)[4], a homomorphic backend to Intel's graph compiler for artificial neural networks [25].

[3] https://github.com/Microsoft/SEAL.
[4] https://github.com/IntelAI/he-transformer.

IBM's Homomorphic Encryption Library (HElib). Shai Halevi and Victor Shoup are creators of the Homomorphic Encryption Library (HElib)[5], another popular FHE library that implements versions of the BGV and CKKS [113]. HElib is implemented in C++ and requires the Library for Doing Number Theory (NTL) mathematical library [189]. HElib began with an implementation of the BGV scheme. Re-implementation led to significant speedups in March of 2018, and in January of 2019 HElib began to support an implementation of CKKS. HElib is available on Github [116].

Fast Fully Homomorphic Encryption over the Torus (TFHE). TFHE[6] is an open-source C/C++ software for FHE available on Github [59]. It is designed around an earlier library, TFHEW, which is no longer maintained [82, 83]. This implementation enables homomorphic evaluation of Boolean circuits on single-bit messages. FHEW requires the Fastest Fourier Transform in the World (FFTW) library [88]. TFHE implements a ring variant of GSW [105] with optimizations [56, 57, 82]. The CUDA-accelerated Fully Homomorphic Encryption Library (cuFHE)[7] implements TFHE on CUDA-enabled GPUs in C++ [72]. It includes an optional Python wrapper. Another library, NuCypher fully homomorphic encryption (NuFHE)[8], written in Python, implements TFHE using CUDA and OpenCL [160].

HEAAN. The Homomorphic Encryption for Approximate Numbers (HEAAN) library[9] implements the CKKS scheme [115]. HEAAN is written in C++ and uses the NTL library [189]. Since its release in 2016, HEAAN has seen speed improvements including the addition of bootstrapping [52]. HEAAN is available on Github [115].

PALISADE. PALISADE[10] is a C++ library created by the New Jersey institute of Technology (NJIT) that uses lattice cryptography primitives to support public-key encryption, several SHE and leveled SHE schemes, proxy re-encryption (PRE), and other multiparty capabilities [170]. PALISADE 1.5.0 was released in March 2019 and is available on GitLab. PALISADE implements versions of a variety of HE schemes: LTV [143], Stehle-Steinfeld (StSt) [194], BGV [34], and BFV [86], and PRE schemes [170]. The authors include LTV for historical purposes, as the scheme itself is no longer considered secure [170].

$\Lambda \circ \lambda$ *(LOL).* The $\Lambda \circ \lambda$: Functional Lattice Cryptography ("LOL") library[11] is a general lattice cryptography library that includes an implementation of a BGV-type FHE scheme [70]. $\Lambda \circ \lambda$ is implemented in Haskell and provides methods for high-level implementation of lattice cryptography primitives, including a BGV-type FHE scheme.

A Language and Compiler for Homomorphic Encryption Made easY (ALCHEMY) compiles plaintexts written in a modular domain-specific language with $\Lambda \circ \lambda$ implemented on the backend [71, 167]. The ALCHEMY compiler is able to simplify the implementation process for users by automatically generating many of the parameters.

Cingulata. Cingulata[12] (pronounced "tchingulata," formally Armadillo) is a compiler toolchain and runtime environment for C++ FHE programs [45]. It provides a high-level C++ interface for FHE applications over a "low-level" implementation of the scheme. Initially Cingulata supported

[5] https://github.com/homenc/HElib.
[6] https://github.com/tfhe/tfhe.
[7] https://github.com/vernamlab/cuFHE.
[8] https://github.com/nucypher/nufhe.
[9] https://github.com/snucrypto/HEAAN.
[10] https://palisade-crypto.org/software-library/.
[11] https://github.com/cpeikert/Lol.
[12] https://github.com/CEA-LIST/Cingulata.

the BFV scheme. In June 2019, support was added for TFHE in gate bootstrapping mode. Cingulata is available on Github [45].

(FV-)NFLlib. FV-NFLlib[13] implements the FV scheme in C++ [89]. FV-NFLlib is available on Github and was last updated in 2016.

Lattigo. Lattigo[14] is recently published lattice-based cryptographic library in Go [138]. It implements a number of schemes, including versions of BFV [16, 112] and HEAAN [52, 197].

3.1 Other Homomorphic Encryption Libraries

Pyfhel (PYthon For Homomorphic Encryption Libraries) implements homomorphic encryption techniques in Cython using SEAL, HElib, and PALISADE as backends [172]. PySyft is a Python library that implements a variety of privacy-preserving techniques, including homomorphic encryption [184]. Additional libraries implement partially homomorphic encryption and somewhat homomorphic encryption. The PythonPaillier library implements the partially homomorphic Paillier scheme in Python 3 [75]. The CUDA Homomorphic Encryption Library (cuHE) provides a GPU-accelerated implementation of the somewhat homomorphic DHS scheme [79]. The *Awesome Homomorphic Encryption List*[15] is an online resource that is periodically updated with new and developing FHE libraries.

4 PRIVACY-PRESERVING ARTIFICIAL INTELLIGENCE IN MEDICINE

Big data continues to transform medicine by improving prognostic models, aiding clinicians in interpretation of clinical data, and improving diagnostic accuracy [161]. However, health data is highly sensitive and sharing or obtaining medical data is challenging due to privacy regulations such as HIPAA [109]. Cloud computing has great potential in the medical field in applications such as personal health monitoring tools, but strict privacy regulations have slowed adoption of such technologies [134]. As fully homomorphic encryption becomes increasingly practical and standardized, its applications for artificial intelligence in the medical domain have grown.

Privacy-preserving machine learning broadly describes a variety of frameworks for training and classification of sensitive data. These methods can take place between multiple parties that can include the data owner(s), model owner(s), and cloud server(s). Computation can be delegated to the cloud server, which performs some computation over a client's encrypted data. The (encrypted) result is returned to the client, who then decrypts to privately view her result. Privacy-preserving machine learning has applications in personalized medicine [150, 151], DNA sequence analysis [15], and diagnostic tools [134]. Possible models include:

(1) *Private outsourced computation*: A medical institution uses homomorphic encryption to privately outsource computation-intensive tasks to a cloud server.
(2) *Private prediction*: A client classifies her data using a model owner's trained model without direct access to the trained model and without giving the model owner any partial information about her data. An intermediary cloud server may be used for outsourcing storage and computation. An example of this model is seen in Figure 3. This model assumes no collusion between the client and the server.
(3) *Private training*: A cloud server trains a classification model on clients' encrypted data.

[13] https://github.com/CryptoExperts/FV-NFLlib.
[14] https://github.com/ldsec/lattigo.
[15] https://github.com/jonaschn/awesome-he.

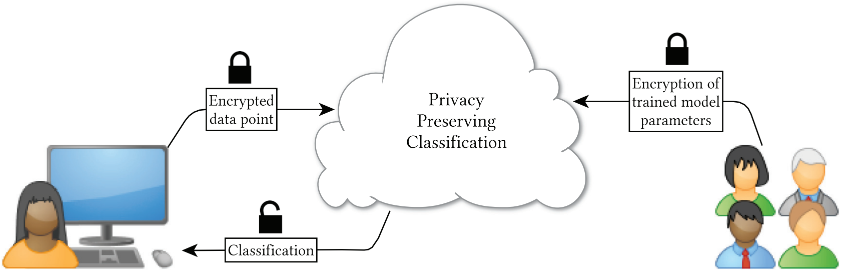
Encrypted data point
Privacy Preserving Classification
Encryption of trained model parameters
Classification

Fig. 3. A privacy-preserving classification model. A user encrypts her private medical data and uploads it to some cloud computing service. Researchers encrypt their trained model parameters under the same public key and upload to the cloud, which performs the required computation for privacy-preserving classification. This result is then returned to the user, who decrypts with her private key.

Private prediction is a major application for FHE techniques [30]. This scenario has applications in health care as well. Clinicians, for instance, could use outside, private diagnostic prediction tools to classify their patients based on their private medical records.

Additional methods develop models of information without specific knowledge of individual data records using different methods [2]. Secure multiparty computation (MPC) techniques such as collaborative learning, private set intersection, Yao's garbled circuits [204], and secret-sharing can be used for this task. Some models focus on the problem of multiple parties jointly training a classifier that is revealed to parties, without sharing individual data points [39, 141, 198]. This model can be useful in the medical field when multiple hospitals wish to jointly train a model over sensitive patient data that must remain confidential. Differential privacy, on the other hand, enables responses to queries from a database with protection from re-identification [84]. Models and classifiers can be constructed from these query responses with a bounded amount of data leakage regarding each data point [30].

This section focuses on applications of homomorphic encryption for privacy-preserving machine learning techniques. The reviewed methods are logistic regression, Naive Bayes, decision tree evaluation, support vector machine, neural networks, and unsupervised clustering. An overview of each method is provided along with techniques for homomorphic implementation.

4.1 Logistic Regression

Logistic regression is a popular method for statistical learning that models two or more outcomes using a logistic function. Logistic regression is widely used in the medical community to predict binary outcomes such as whether a patient needs treatment or not, or whether a condition is present or not [114]. It has been used in applications such as assessing treatment of diabetic patients [22].

Logistic regression models uses linear functions to model the posterior probabilities of the classes. Given a set of points $\{1, x_1, x_2, \ldots, x_n\}$, where $x_i \in \mathbb{R}^d$, and binary classes $\{-1, 1\}$, the model takes the form

$$\Pr(Y = -1|X = x) = \frac{1}{1 + e^{-\beta^T x}},$$

where β is a weight vector and $\Pr(Y = 1|X = x) = 1 - \Pr(Y = -1|X = x)$. Training a logistic regression model involves learning the parameter set β.

Training a Logistic Regression Model. The parameters in a logistic regression model can be learned using maximum likelihood estimation. Let $p_{y_i}(x_i; \beta) = \Pr(Y = y_i|X = x)$. The log-likelihood for N

observations with binary class labels is given by

$$\ell(\beta) = \sum_{i=1}^{N} \log p_{y_i}(x_i;\beta) = \sum_{i=1}^{N} \left(y_i \beta^T x_i - \log\left(1 + e^{y_i \beta^T x_i}\right)\right),$$

where $y_i \in \{-1, 1\}$. The log-likelihood can be maximized in multiple ways. Newton's method approximates the zeros of the gradient of ℓ with iterative updates to

$$\beta_{k+1} = \beta_k - \frac{\nabla_\beta \ell(\beta)}{\nabla_\beta^2 \ell(\beta)}.$$

Gradient descent minimizes a cost function C by moving in the direction of the gradient, the steepest descent. It iteratively calculates $\beta_{k+1} = \beta_k - \gamma_k \nabla C(\beta)$, where γ_k is the learning rate at iteration k. In the case of logistic regression the cost function is given by the negative log likelihood. Calculating the gradient $-\nabla_\beta \ell(\beta)$ of the negative log likelihood yields the iterative formula

$$\beta_{k+1} = \beta_k + \gamma_k \sum_{i=1}^{N} (1 - \sigma(\beta^T x)) y_i x_i, \tag{1}$$

where σ is the sigmoid function, $\sigma(x) = 1/(1 + \exp(x))$.

Training on Encrypted Data. In 2017, the Center for Integrating Data for Analysis, Anonymization and Sharing (iDASH) held a competition at its annual iDASH Privacy and Security Workshop calling for methods to train a logistic regression model over encrypted data using HE.[16] The winning team used HEAAN to train the model [132].

Training logistic regression models on encrypted data requires HE-friendly modifications to the optimization function. The training methodology should require the least number of iterations for convergence possible due to the expense of homomorphic computation. Kim et al. use Nesterov's accelerated gradient [159], which converges faster than standard gradient descent, to learn the parameter set in their winning algorithm [132]. Nesterov's gradient descent adds a parameter ν that looks further ahead each iteration in order to avoid overshooting minima. It is given by

$$\begin{aligned} \beta_{k+1} &= \nu_k - \gamma_k \nabla \ell(\beta) \\ \nu_{k+1} &= (1 - \alpha_k)\beta_{k+1} + \alpha_k \beta_k, \end{aligned}$$

where $\alpha_k \in (0, 1)$ is a smoothing parameter. The authors approximate the sigmoid σ over the domain $[-8, 8]$ with polynomials of degree 3, 5, and 7. They also introduce an improved method to encrypt a matrix within an HEAAN ciphertext. In this way, the entire dataset $\{x_1, \ldots, x_N\}$, where $x_i = (x_{i1}, \ldots, x_{id})$, can be encoded within a single matrix $\mathbf{x} = (x_{ij})_{1 \le i \le n, 1 \le j \le d}$. The weights β_k are encoded in a matrix n times, where each row represents a copy of β_k. The server can then compute Equation (1) using a polynomial approximation of sigmoid.

Other approaches to learning logistic regression parameters over encrypted data implement a variety of techniques. Bonte and Vercauteren use Newton's method combined with a Hessian approximation in FV-NFLib [28]. They use a Taylor series approximation in place of the sigmoid function. Crawford et al. approximate the log-likelihood with a Taylor expansion and evaluate a closed-form formula approximation of the logistic regression parameters using HElib [68]. Blatt et al. approximate the sigmoid function with Chebyshev polynomials for an implementation of logistic regression in PALISADE along with a custom variant of the CKKS scheme [24]. Carpov et al. perform logistic regression over data encrypted using TFHE and HEAAN [43]. Additional methods use HElib [200] and gradient descent with a minimax approximation for the sigmoid in

[16] iDASH Security & Privacy Workshop 2017. Retrieved August 21, 2019 from http://www.humangenomeprivacy.org.

SEAL [48]. An earlier method presented by Aono et al. in 2016 trained a logistic regression model using additive HE [10]. However, due to the limitation of encrypted computation to addition, the client is required to perform initial and intermediary plaintext computations.

4.2 Naive Bayes

When diagnosing a patient, clinicians often search for conditionally independent attributes that may be indicative of a disease [129]. Naive Bayes is a well-known classifier that assumes independence between data features. The result is a transparent and understandable classification method that frequently outperforms more sophisticated methods [114]. In medical applications, Naive Bayes has outperformed more sophisticated diagnosis systems, including localization of a primary tumor, prediction of recurrence of breast cancer, and rheumatological diagnosis [135], as well as in applications predicting heart disease [129].

Given a set of classes Y and a sample X, Bayes Theorem states that

$$\Pr(Y = Y_i|X) = \frac{\Pr(X|Y = Y_i)\Pr(Y = Y_i)}{\Pr(X)}.$$

In this formula, $\Pr(Y = Y_i|X)$ represents the posterior probability of a class given an attribute and $\Pr(X|Y = Y_i)$ is the likelihood of an attribute given a class. The classification of a vector X is given simply by taking the maximum posterior probability over all classes and applying Bayes' Theorem:

$$\hat{Y} = \underset{Y_i \in Y}{\operatorname{argmax}} \left[\Pr(Y = Y_i|X)\right] = \underset{Y_i \in Y}{\operatorname{argmax}} \left[\Pr(Y = Y_i) \prod_{j=1}^{d} \Pr(X = X_j|Y = Y_i)\right].$$

The denominator term $\Pr(X)$ can be omitted in the equation above because X is fixed.

4.2.1 Private Naive Bayes Classification. Bost et al. present a privacy-preserving method for Naive Bayes classification [30]. In their model, a client learns the classification of her data point X without learning any partial information about the classification model or giving away any information about her input. The model's learned parameters are uploaded to a cloud server in encrypted form.

Assume each vector X has d features, each of which can take on a finite number of possible values. Continuous data can be quantized to satisfy this assumption. The authors formulate the trained Naive Bayes model as two tables. The table $P = (P_i)$, where $P_i = \Pr(Y = Y_i)$, provides the probability of class Y_i. The table T is a matrix where the (i, j)th entry is the likelihood of an attribute given a class, $T_{i,j} = \Pr(X_j|Y = Y_i)$. A new data point can be classified by looking up the probability for each attribute given each class and computing the argmax of the resulting values.

Bost et al. use two partially homomorphic encryption schemes. Algorithm 1 describes a simplified version of this protocol that uses a single FHE scheme. First, the server encrypts P and T. The client then downloads these tables from the server and homomorphically computes $[p_i]$, the classifier score, for each class i. The client and server then perform a private argmax protocol and the client learns the index $\hat{i}$ of the largest value for p_i.

In order to compute the argmax, the client and server perform a linear sequence of comparisons to determine the largest value. The client randomizes the order that she sends the posterior probability for each class to the server. The server compares this value to the maximum observed value using an encrypted comparison protocol and returns the new maximum value to the client, masked so that she cannot learn a partial ordering on the values. A number of encrypted comparison protocols have been published [30, 195], and additional methods are discussed in Section 6.

Bost et al. implement Naive Bayes classification using the logarithm of the probability distributions, where $p_i = \log(\Pr(G = G_i|X))$, in order to compute the posterior probabilities using an

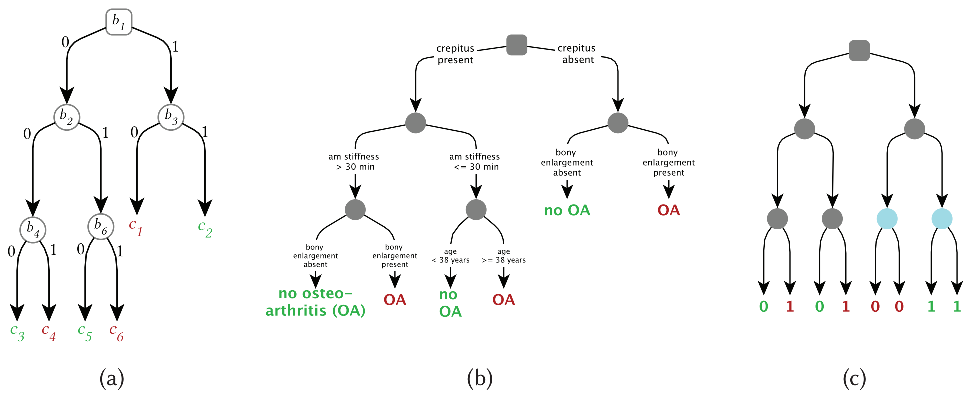

Fig. 4. (a) A binary decision tree with decision nodes b_i and leaf nodes c_i. (b) A decision tree for classification of osteoarthritis of the knee [9]. (c) A completion of the binary tree in (a) and (b).

ALGORITHM 1: Private Naive Bayes Classification

```
Client input: Data point $x \in \mathbb{Z}^d$, FHE public key PK.
Server input: Tables $P$ and $T$, FHE secret key SK.
Client output: The index of the class with the highest classifier score.

The server encrypts the tables $P$ and $T$.
The client homomorphically evaluates
    $[p_i] = [P_i] \prod_{j=1}^{d} [T_{i,j}(X_j)]$
  for $i = 1, \ldots, c$.
The client uses the server to privately compute $\hat{i} = \text{argmax}_i p_i$.
The client outputs $\hat{i}$.
```

additively homomorphic encryption scheme. The scheme can be implemented with or without the logarithm using a fully homomorphic encryption scheme. Sun et al. implement a fully homomorphic version of this scheme [195].

4.3 Decision Trees

Binary decision trees are a classification method with a representation as a simple diagram consisting of internal decision nodes and terminal leaf nodes. Decision nodes apply a sequence of binary splits to the data and a final class is assigned based on the value of the terminal leaf node. In Figure 4(b), these classes are "osteoarthritis" and "no osteoarthritis." While this example is for qualitative data with two classes, the concept also applies to multi-class data with quantitative features. A binary tree model is straightforward to draw regardless of data dimension. In fact, this method is favored among scientists in the medical community as it is easy to visualize and it "mimics the way a doctor thinks" by "stratify[ing] the population into strata of high and low outcome, on the basis of patient characteristics" [114]. Decision trees are used for a variety of medical applications, including early diagnosis of myocardial infarction, probability of admittance in the emergency room based on chest pain, and computer-aided decision-making in mental healthcare and wound care [169].

During training, a greedy algorithm chooses binary partitions that minimize the classification error in each region at each step. For the first split, splitting variable j and split point s are selected to minimize the node impurity in each of the two resulting sub-regions, $R(j,s) = \{X : X_j \leq s\}$ and $R'(j,s) = \{X : X_j > s$. The combined node impurity in these regions is minimized by testing all potential values for the splitting point s [114]. The node impurity in region R is measured by looking at the proportion of class ℓ vectors in the region. Possible measures for node impurity include classification error, $1 - p_{k\ell}$; Gini impurity, $\sum_{\ell=1}^{c}(1 - p_{k\ell})$; or entropy, $-\sum_{\ell=1}^{c} p_{k\ell} \log p_{k\ell}$.

This process iterates until a stop condition is met. Determining when to stop requires finding a balance between over-fitting with a large tree and under-fitting with a tree too small to adequately describe the data. Methods include restricting tree depth, restricting number of leaf nodes, and restricting the leaf nodes to a minimum node size. This can be followed by pruning to shrink the tree. Gini impurity and entropy are often used as measures of node impurity during the tree growing stage while the classification error is often used during the pruning phase [114].

4.3.1 Private Decision Tree Classification. A trained decision tree T can be described by decision nodes $b = (b_i)_{1 \leq i \leq N}$ and terminal nodes $c = (c_i)_{1 \leq i \leq M}$, as in Figure 4(a). Evaluation at each decision node yields a representation of that node as a Boolean value, 0 or 1. A new data point X is classified within T by applying the condition b_1 of the decision tree to X, then following the branches sequentially and applying decision conditions until reaching to a leaf node. X is assigned the class value contained in this leaf node.

Privately classifying data requires hiding both the tree structure from the client and hiding the client's data from the model owner. An intermediary server should learn nothing about either party's information. One approach to hiding tree structure converts a decision tree into its polynomial form [30, 195]. Another approach to hiding tree structure converts a decision tree into a complete binary tree. In either approach, every internal node in the tree must be evaluated, regardless of each binary outcome, in order to mask tree structure. Tree structure can present a computational bottleneck for private decision tree classification using FHE when tree structure is too large, depending on the speed of the underlying scheme.

Polynomial Tree Representation. A decision tree T for binary classification with decision nodes $b = (b_i)_{1 \leq i \leq N}$ and terminal nodes $c = (c_i)_{1 \leq i \leq M}$ can be represented as a polynomial P that outputs the predicted class of an input data point. Bost et al. [30] describe the a recursive procedure, $\mathcal{F}$, for constructing P given a non-empty tree T:

1: If T is one leaf node with class assignment c_i, then $\mathcal{F}(T) = c_i$.
2: If T has a decision node, boolean b, with left and right sub-trees T_0 and T_1, return $\mathcal{F}(T) = b \cdot \mathcal{F}(T_1) + (1 - b) \cdot \mathcal{F}(T_0)$.

For example, the tree in Figure 4(a) has polynomial representation

$$\begin{aligned} P(b,c) = {} & b_0 \cdot [b_3 \cdot (b_5 \cdot c_6 + (b_5 - 1) \cdot c_5) + (b_3 - 1) \cdot (b_4 \cdot c_4 + (b_4 - 1) \cdot c_3)] \\ & + (b_0 - 1) \cdot (b_2 \cdot c_2 + (b_4 - 1) \cdot c_1). \end{aligned}$$

Protocols for private decision tree classification using the polynomial form of the tree follow the broad steps based on the work of Bost et al. [30] and outlined in Algorithm 2 below.

In this setting, privacy-preserving means that the client does not directly learn the evaluation of her data point on each node or the tree structure, and the model owner does not learn anything about the client's data point. First, the client and server perform a series of private comparisons to determine the binary output on each internal decision node. Evaluation of each decision node is modeled as comparison of an input x to a threshold w, yielding bit $b = [x > w]$. Private comparison

```
ALGORITHM 2: Private Tree Evaluation
    Client input: Data point $X$, FHE secret key SK.
    Server input: The polynomial form $P$ of a decision tree, FHE public key PK.
    Client output: The index of the leaf node containing the classification of $X$.

    The client and model owner perform comparisons so that the server learns $[b_i]$, the (encrypted)
      binary output of each internal node.
    The server uses $[b_i]$ to homomorphically evaluate the polynomial $P$.
    The client receives and decrypts the output, $[P(X)]$, to receive their final classification.
```

in this setting requires that the client learn no information about the comparison result, and the server only learns the comparison result encrypted under the client's public key.

As with Naive Bayes, a number of private comparison methods using homomorphic encryption are available. Bost et al. implement two partially homomorphic encryption schemes to perform comparisons [30]. Sun et al. [195] implement private decision tree with comparison protocols using a BGV-type FHE scheme. Khedr et al. [131] also implement private decision tree using FHE. They formulate the decision on each node as a word matching problem and perform comparison using their presented encrypted word matching protocol.

Complete Binary Trees. Another method for hiding tree structure uses complete binary trees. This model introduces "dummy" nodes in order to impose a uniform depth upon tree structure [202]. The dummy nodes give a random binary output, where each response leads to the same outcome, as in Figure 4(c).

Wu et al. use tree randomization to hide the structure of a complete binary tree [202]. Tree randomization constructs a randomized tree T' from a given tree T with the same classification output as the original tree, where T is a complete binary tree with b decision nodes, via the following process:

1: Initialize $T' = T$.
2: Randomly choose bits $s = \{s_1, \ldots, s_b\} \in \{0,1\}^b$.
3: For each i, if $s_i = 1$ negate the decision function on the node t'_i of the tree T'. Swap the subtrees originating at the left and right child nodes.
4: Re-order the node indexes and output T'.

Step 3 of the above procedure randomly leaves or swaps the left and right sub-trees at each node, ensuring that evaluation of the tree will have a random path after each tree randomization procedure.

During the private complete tree evaluation protocol presented by Wu et al., a client learns the depth of the tree and no other information about its structure. An overview of this protocol is given below in Algorithm 3.

Lines 1 through 4 allow the client to evaluate each decision node in T. The client is only able to see randomized results of these comparisons, while the server only has access to the encrypted results. The server then reverses the initial permutation and homomorphically computes σ_i, the correct result on each node, in Line 5. The server then randomizes the tree and sends the client the decision node, which results in the randomized tree.

The client then uses the randomized decision path to determine the index of the leaf node containing the desired output. She retrieves the result associated with this index from the randomized tree using a cryptographic protocol called oblivious transfer [61, 173]. Oblivious transfer enables the client to learn the value contained in a exactly one leaf node in the randomized tree without revealing the node index to the server.

ALGORITHM 3: Private Evaluation of Complete Binary Trees

```
Client input: Data point $X$, FHE secret key SK.
Server input: A complete binary tree $T$ with depth $d$, FHE public key PK.
Client output: The index of the leaf node containing the classification of $X$.
  The client sends $[x_i]$ to the server for $i = 1, \dots, d$.
  The server generates random bits $b_i$ for $i = 1, \dots, d$ and negates the decision function at node $i$
    whenever $b_i = 1$.
  The server and client perform comparison for each node. The server provides the results to the client
    in random order according to some permutation $\pi$.
  The client sends $[b'_{\pi(i)}]$ to the server for all $i$, where $b'_{\pi(i)}$ is the Boolean result of each comparison.
  The server reverses $\pi$ on the clients encrypted bits and computes $\sigma_i = [b_i \oplus b'_i]$, correct result on
    each node.
  The server chooses random bits $s_i$ for $i = 1, \dots, d$ to construct the permuted tree $T'$ and sends the
    client $\sigma'_{i_k} = \sigma_{\tau(i)} \oplus s_{\tau(i)}$, where $\tau$ is the permutation on node indexes of $T$ arising from the
    randomization procedure.
  The client decrypts and computes the index $I$ of the leaf node containing the response in $T'$.
  The client and server perform an Oblivious Transfer protocol, which allows the client to learn the
    result on leaf node $I$ in $T'$ without revealing the node index to the server.
```

4.4 Neural Networks and Deep Learning

A neural network consists of layers containing nodes that compute functions over values fed from a previous layer. Deep learning uses deep neural networks, which are neural networks with multiple hidden layers between input and output. While conventional machine learning algorithms rely on feature selection prior to training, deep learning can discover data-driven and complex features from the raw data [60, 150]. The goal is to derive features as linear combinations of the input values, which are then combined in a nonlinear model of the target [114].

Increasing data availability in health care has led to a growing number of applications for deep learning in the medical field. Implementations have been successful for processing clinical imaging, prediction and classification based on aggregated electronic health records, genomics, and analyzing data from wearable devices and smart phone apps [150, 151]. Convolutional neural networks have proven valuable for applications in medical imaging such as UNet [182].

This section first provides an overview of common layers used in neural networks. This is followed by a summary of existing methods for privacy-preserving neural network classification and homomorphic methods for evaluating non-linear layers.

4.4.1 *Layers in Neural Networks.*

Neural networks contain layers such as dense (or fully connected) layers, convolutional layers, pooling layers, and activation layers. The functions used in some of these layers are not polynomial functions, making the direct application of homomorphic encryption unfeasible. The approximation of non-linear layers by polynomial functions and the bits of precision on inputs and network weights must be chosen to mitigate loss of network accuracy during homomorphic evaluation. Various techniques have been employed for approximating non-linear layers for homomorphic evaluation.

Convolutional layers. Convolutional layers perform feature extraction via discrete convolution. Discrete convolution of vectors f and g at index m is given by $f * g = \sum_{i=-\infty}^{\infty} f(i)g(m-i)$ where i runs over all valid indexes for f and g. This feeds a vector of input values f to the next layer in the network by computing its weighted sums with the vector g. If f and g are matrices, as in image processing applications, then

$$f * g = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} f(i,j)g(m-i, n-j).$$

Convolutional layers are common in image classification networks, where f is a sub-region of the image centered at index m and g is a kernel matrix [114]. After training, the weights of the kernel g can be encrypted. As a polynomial function, discrete convolution can be computed using FHE [31, 80].

Activation layers. Activation layers and introduce non-linearity to the neural network model. Convolution layers are generally followed by activation layers. Common activation functions include the *ReLU* (REctified Linear Unit) function, the sigmoid activation function, and the hyperbolic tangent function. The ReLU function is given by $f(x) = \max(0, x)$, and the sigmoid is given by

$$\phi(z) = \frac{1}{1 + \exp(-z)}.$$

Homomorphic-friendly variants of these functions are discussed in Section 4.4.2.

Pooling layers. Pooling layers reduce the dimensionality of the input layer. In image classification, pooling often follows convolution, as in UNet, a popular framework for medical image classification [182]. Max pooling layers yield the maximum value of some input components, while mean pooling layers yield the mean value of some input components [80]. For instance, a 2 × 2 max pool will yield the maximum value in non-overlapping regions in a 2 × 2 matrix.

Max pooling is not a polynomial function. Mean pooling, which calculates the mean value of a collection of input components, is a FHE-friendly function. Experiments have shown that mean pooling is more stable in the presence of perturbations than max pooling [31]. Therefore, mean pooling is advantageous for FHE applications in two ways: ease of computation and its relative stability in the presence of ciphertext noise accumulation.

Fully connected layer. Fully connected layers appear at the end of the network, combining all data from the preceding layer. Each node in a fully connected layer is connected to all nodes in the previous layer. The value at each node in the layer is computed as a weighted sum of all values in the previous layer. The kth node of layer ℓ, y_k^ℓ, is given by

$$y_k^\ell = b_k + \sum_i y_i^{\ell-1} w_{i,k},$$

where $y_i^{\ell-1}$ are the node values from the previous layer, $w_{i,k}$ are the weights, and b_k is a bias term. Because this is a polynomial function, it can be evaluated using FHE [31].

4.4.2 *Private Deep Learning.*

Fully homomorphic encryption is used in a collection of works that carry out private classification of encrypted data over a neural network trained in the clear. A summary of these works is provided in Table 2. In these models, an encrypted data point is classified using either unencrypted or unencrypted trained model parameters. When the parameters are unencrypted, a client encrypts a data point under their own private key and sends the encrypted value directly to the model owner. Because the model owner applies their (unencrypted) model to this encrypted data point, the client never needs direct access to the model. Encryption of both model parameters and input data would apply to the scenario of outsourcing all computation to an external service.

Dowlin et al.'s methodology, CryptoNets, carries out private classification over neural networks with unencrypted trained model parameters [106]. CryptoNets replaces ReLU with the square

Table 2. Frameworks for Private (Deep) Neural Network Evaluation Using FHE

Name	Software	Non-linear layer(s)	Pooling layer
Chabanne et al. [46]	HElib	ReLU approximated with low degree polynomials.	Mean pooling
Chimera [31]	—	ReLU approximated with low-degree trigonometric polynomials	Mean pooling
CryptoDL [119]	HElib	Low-degree polynomial approximations of sigmoid and tanh. ReLU replaced with the antiderivative of an approximation of its gradient.	Mean pooling
CryptoNets [106]	SEAL	ReLU replaced with the square activation function.	Sum pooling
E2DM [127]	HEAAN	Square activation function	Replaced by a fully connected layer.
FHE-DiNN [32]	TFHE	Sigmoid replaced by sign activation function.	—
Gazelle [128]	PALISADE	Encryption switching to evaluate non-linear functions using garbled circuits	Encryption switching to evaluate using garbled circuits

If an open-source FHE implementation was used, it is listed in the Software column. The non-linear layer column outlines the activation function used for fully homomorphic implementation, and the pooling layer column describes the FHE-friendly pooling operation.

activation function, $f(x) = x^2$. This function is FHE-friendly due to its multiplicative depth of just one. However, the authors point out that the square activation function can yield unstable behavior during backpropagation. Networks deeper than 10 layers may have difficulty training with this activation function. CryptoNets replaces max pooling with sum pooling—a scalar multiple of the mean. Both training and testing are performed with these modified functions. The sigmoid function is applied in a final activation layer during training and is replaced with argmax during classification.

Use of the square activation function for classification of encrypted data causes CryptoNets to have low accuracy for networks when there is a high number of non-linear layers. Chabanne et al. extended this approach to greater depth networks using batch normalization with low-degree polynomial approximations for the ReLU function [46]. The authors implemented their framework using BGV encryption [34] with SV ciphertext packing [192] using HElib [116].

Chabanne et al. [46] combine a low-degree polynomial approximation of ReLU with batch normalization. Batch normalization adds a normalization layer to the network before each activation layer in order to control the distribution of the data. After normalization, the activation function chosen to replace ReLU only needs to be accurate on a fixed interval. The authors replace ReLU with 2-, 4-, and 6-degree polynomials that approximate well on the interval $[-3, 3]$.

The CryptoDL framework [118, 119], implemented using HElib [116], approximates both the tanh and sigmoid functions with Chebyshev polynomials [119]. Rather than directly approximating ReLU, the authors approximate its derivative, and calculate its antiderivative for use as an activation function. Bootstrapping within the scheme is replaced with additional communication steps between client and server.

Another framework, GAZELLE, uses a combination of additive homomorphic encryption and garbled circuit techniques to implement neural network classification [128]. Garbled circuits compute non-linear functions such as MaxPool and ReLU with linear-sized circuits. Experiments implement the BFV [33, 86] scheme using PALISADE [165] for homomorphic encryption, along with Yao's garbled circuits [204].

The Chimera framework unifies the plaintext spaces of B/GV, HEAAN, and TFHE [31]. It includes a method to pack complex exponential values in HEAAN ciphertexts, enabling evaluation of trigonometric polynomials up to a predefined depth. The authors approximate ReLU by low-degree trigonometric polynomials by decomposing the graph as a Fourier series.

FHE-DiNN addresses the issue of scale invariance, where noise accumulation in activation layers limits network depth, by applying bootstrapping for noise reduction to each neuron's output [32]. Their network is a discretized neural network with $\{-1, 1\}$ as the input space and integer weights, and they implement the sign activation function. FHE-DiNN uses the TFHE library [59] to implement FHE for deep learning [32].

The encrypted data and encrypted model (E2DM) framework performs classification of encrypted data using an *encrypted* neural network model [110]. Implementation uses the HEAAN library and the square activation function [127]. E2DM replaces ReLU with the square activation function and replaces a pooling layer with a fully connected layer that learns parameters to compute an array of weighted sums.

Additional methods use additive homomorphic encryption in conjunction with additional cryptographic techniques to perform privacy-preserving classification on neural networks. The MiniONN framework of Liu et al. combines additive homomorphic encryption, secret sharing, and oblivious transfer for private classification [142]. This method does not require any changes to the model during training but does have a higher computation cost for clients. Ma et al. combine additive homomorphic encryption with secret sharing in their non-interactive model [146].

4.4.3 Privacy-preserving Model Training. Training over encrypted data presents its own challenges. Training a neural network for a complex classification problem generally requires analyzing network accuracy to introduce modifications during training. However, training over encrypted data requires the network to be effective on the first try [46]. In addition, training requires a heavy computational load and the evaluation of many homomorphic operations when performed over homomorphically encrypted data.

Zhang et al. describe a backpropagation method for model training over encrypted data [206]. Between each backpropagation iteration on the cloud, the client must download, decrypt, and re-encrypt the parameters. This avoids the problem of the high computational cost associated with high multiplicative depth from homomorphic operations, but at the cost of high communication complexity. Li et al. [140] present a method for collaborative training of a deep learning model using FHE combined with additional cryptographic techniques. This method is associated with high computation and communication costs. Phong et al. [168] expand upon a method of Shokri and Shmatikov [188]. The authors combine asynchronous stochastic gradient descent with additively homomorphic encryption, allowing multiple participants to jointly train a model without sharing their datasets. Additional methods for privacy-preserving model training use multiparty computation techniques [152, 153, 178, 183].

5 CLUSTERING

Clustering is a method of unsupervised machine learning that groups data points into clusters in order to discover patterns and structure in the data. A data owner may wish to perform privacy-preserving clustering, outsourcing the computational burden of clustering to an external cloud server without giving that server direct access to the data. In the medical setting, clustering has been applied to neurological disease datasets, such as Alzheimer's disease, to find patterns that clinicians would otherwise have difficulty detecting [5].

The k-means clustering algorithm is a popular clustering method that calculates the center of data clusters, called centroids, by iteratively assigning points to the cluster with the nearest mean, then updating the clusters to minimize within-cluster variance [114]. Sakellariou and Gounaris [186] perform k-means clustering of encrypted data using Brakerski's FHE scheme [33], but require decryption by a trusted server during analysis. Jäschke and Armknecht present a modification of k-means that can be evaluated without intermediate decryption, but the estimated runtime for a two-dimensional dataset with 400 points is over 25 days on a CPU [126].

Cheon et al. [53] perform clustering over encrypted data using the mean shift algorithm and HEAAN. Mean-shift clustering separates data into clusters by finding local maxima of a density function, or kernel, via gradient descent. The most common kernels used in mean shift clustering, such as the Gaussian kernel, require non-polynomial operations. The authors implement a HE-friendly kernel K given by

$$K(x, y) = c \cdot (1 - \|x - y\|^2)^{2^\Gamma + 1}$$

for constant parameters c and Γ, and address the computational load of mean-shift clustering by performing each iteration on a sample of the data, rather than the entire dataset.

6 FULLY HOMOMORPHIC ENCRYPTION FOR BIOINFORMATICS

Genome sequencing is now practical on a large scale, and the cost of data storage continues to decrease. As a result, genome sequencing is now commercially available for personalized genetic analysis.[17] Genomic data has been used for applications in multiple domains including healthcare, biomedical research, disease risk tests, and forensics [158]. Because we are uniquely identified by our genetic code, a privacy breach compromising this data could have an unforeseen negative impact upon a person's life.

The human genome is constructed of chains nucleic acid bases, which exist in pairs wound in the now well-known double helix structure. For DNA sequences, the bases cytosine, guanine, adenine, and thymine yield the alphabet $\Sigma = \{a, c, g, t\}$, with complements $\overline{a} = t$ and $\overline{g} = c$. DNA sequences are represented as strings of the letters from this alphabet. For example,

$$tttgcggtgctggttgccgtatttacttggct$$

is the DNA strand fragment from *vibrio cholarae* []. Strings in an alphabet Σ can be written over a ω-bit alphabet, where $\omega = \lceil \log |\Sigma| \rceil$ [54]. In the case of DNA, $\omega = 2$, and each base has a 2-bit representation with the mapping $a \rightarrow 00$, $g \rightarrow 01$, $c \rightarrow 10$, and $t \rightarrow 11$ [133].

Homomorphic encryption has been used for a variety of applications in bioinformatics. A large portion of this work is driven by the annual iDASH workshop's homomorphic encryption competition tasks. Lauter et al. provided some of the first applications of FHE for computation over encrypted genomic data [139]. They provided algorithms for homomorphic computation of basic genomic algorithms such as the Pearson goodness-of-fit test, the D' and r^2-measures of linkage disequilibrium, the Estimation Maximization (EM) algorithm for haplotyping, and the Cochran-Armitage test for trend.

6.1 Sequence Comparisons

Comparisons of DNA and RNA sequences can be used in bioinformatics applications including sequence alignment, gene finding, and motif discovery [133]. Algorithms for motif discovery seek to identify common motifs, or patterns, among biosequences such as DNA or RNA. Identifying motifs enables researchers to carry out such tasks as classifying protein sequences and identifying important regions such as splice sites, promoters, or binding sites. Sequence alignment seeks to align strings of DNA or RNA. Global alignment assumes that the entirety of two sequences are similar, while local alignment seeks to align local portions of sequences [155].

[17] www.ancestry.com; www.23andme.com/.

Sequence comparison methods include edit distance and Hamming distance. One of the tasks for the 2015 iDASH competition called for HE-based protocols that compute edit and Hamming distance between encrypted datasets.

6.1.1 Hamming Distance. Hamming distance compares two equal-length strings by counting the number of positions at which they differ. It has applications in bioinformatics such as the Planted (ℓ, d)-Motif Problem, which seeks to identify the motifs M in a nucleotide sequence of length ℓ that are at most Hamming distance d from M [155].

Jarrous and Pinkas perform privacy-preserving Hamming distance computation using additive HE and oblivious transfer [125]. Yasuda et al. compute Hamming distance between binary vectors using a BV-like scheme [205]. Kim and Lauter provide a method for privacy-preserving computation of Hamming distance between two participants given Variation Call Format (VCF) files, a text file format used to store genetic variations [133]. The VCF for each participant summarizes their variants with respect to the reference genome with values such as insertion, deletion, or substitution.

Let $\mathcal{L}$ denote a list indexed by participants x and y, where x_i denotes the record in the VCF file for patient x at location i. The authors formulate Hamming distance as follows.

```
1  $h \leftarrow 0$
2  for $i \leftarrow 1$ to $|\mathcal{L}|$
3      if the structural variant given by $x_i$ or $y_i$ is an insertion or deletion
4          $h_i \leftarrow 0$
5      else if $x_i \wedge y_i = \emptyset$
6          $h_i \leftarrow 1$
7      else if $x_i$ and $y_i$ reference the same bases but have different alternate alleles
8          $h_i \leftarrow 1$
9      else
10         $h_i \leftarrow 0$
11 $h = h + h_i$
12 return $h$
```

This algorithm skips all insertions and deletions, and counts the remaining record locations where exactly one of the patients has a structural variant (line 5), or where both participants have a structural variant but they are *different* structural variants (line 7).

Each participant encodes their data according to the server-provided reference list $\mathcal{L}$. For each position i, they calculate $e_i, f_i \in \{0, 1\}$, where $e_i = 1$ if $\text{pos}_i \in \mathcal{L}$ and $f_i = 1$ if $\text{sv}_i \in \{\text{INS}, \text{DEL}\}$. The value e_i defines whether the genotype at locus i is missing, and f_i specificies whether the variant is an insertion/deletion or not. The values s_i encode the single nucleotide variants. The authors then formulate Hamming distance via evaluation of the circuit

$$(\mathrm{E}(s_i, s_i') \wedge (e_i \oplus e_i' \oplus 1) \oplus 1)) \wedge f_i \wedge f_i',$$

where $\oplus$ is binary addition (XOR gate) and $\wedge$ is binary multiplication (AND gate). The function $\mathrm{E}(s_i, s_i')$ equals 1 if and only if $s_i = s_i'$. The authors homomorphically evaluate these circuits over data encrypted in a BGV scheme.

6.1.2 Edit Distance. The edit distance between two strings is given by the minimum number of substitutions, deletions, and insertions that it would take to convert one string to the other. Edit distance is used in algorithms such as motif discovery.

A 2003 protocol for private edit distance computation by Atallah et al. uses additive HE plus oblivious transfer [13], resulting in a high communication complexity. Zhang et al. use a path-finding algorithm combined with integer comparison to evaluate private edit distance [207].

Kim et al. use a greedy algorithm to compute an upper bound for an approximate edit distance [133].

Cheon et al. compute edit distance by representing it as an arithmetic circuit, which they then can evaluate using bit-wise homomorphic encryption [54]. For speed, the number of homomorphic operations required to evaluate the circuit should be kept as low as possible. Furthermore, the depth of the final circuit is needed to parametrize the SHE scheme. This circuit is built from equality circuits, comparison circuits, and addition circuits. The equality circuit compares two unsigned μ-bit values in their binary representations $x = x_\mu \ldots x_1$ and $y = y_\mu \ldots y_1$:

$$\mathtt{EQU}(x, y) = \wedge_{i=1}^{\mu}(1 \oplus x_i \oplus y_i).$$

This circuit is implemented with μ homomorphic addition operations and $\mu - 1$ homomorphic multiplication operations. The comparison circuit can be written $\mathtt{COM}(x, y) = c_1 \oplus c_2 \oplus \cdots \oplus c_\mu$, where

$$c_i = (x_i \oplus 1) \wedge y_i \wedge \left(\wedge_{j=i+1}^{\mu} \left(x_j \oplus 1 \oplus y_j\right)\right).$$

This circuit can be evaluated with $2\mu - 2$ homomorphic additions and $2\mu - 3 + \frac{(\mu-1)\log(\mu-1)}{2}$ homomorphic multiplications. Lastly, the addition circuit $\mathtt{ADD}(x, y)$ is defined by $(s_1, \ldots, s_\mu)$, where the kth value s_k of the sum is denoted $\mathtt{ADD}(x, y)[k]$ and satisfies

$$s_i = \begin{cases} x_1 \oplus y_1 & \text{if } i = 1, \\ x_i \oplus y_i \oplus e_{i-1} & \text{else,} \end{cases} \quad \text{and} \quad e_i = \begin{cases} x_i \wedge y_i & \text{if } i = 1, \\ (x_i \wedge y_i) \oplus ((x_i \oplus y_i) \wedge e_{i-1}) & \text{else.} \end{cases}$$

Implementing this circuit requires $3m - 3$ homomorphic addition operations and $2m - 3$ homomorphic multiplication operations.

Given an alphabet Σ and μ-bit words α and β, the authors compute encrypted edit distance based on the Wagner-Fischer algorithm [208]. Let $d(i, j)$ denote the edit distance between the first i characters of α and the first j characters of β. The Wagner-Fischer algorithm proceeds as follows.

```
1 for $i \leftarrow 0$ to $n$
2     $d(i, 0) \leftarrow 0$
3 for $j \leftarrow 0$ to $m$
4     $d(0, j) \leftarrow 0$
5 for $i \leftarrow 1$ to $n$
6     for $j \leftarrow 1$ to $m$
7         $t(i, j) \leftarrow (\alpha_i = \beta_i)$
8         $d(i, j) \leftarrow \min\{d(i-1, j-1) + t(i, j), d(i, j-1) + 1, d(i-1, j) + 1\}$
9 return $d(n, m)$
```

In line 6, $(\alpha_i = \beta_i)$ denotes binary comparison. This comparison can be calculated by the equality circuit as $t(i, j) = \mathtt{EQU}(\alpha_i, \beta_j) \oplus 1$, and $d(i, j) + 1$ can be calculated by the addition circuit as $\mathtt{ADD}(d(i, j), 1)$. Homomorphic evaluation computes the encrypted distance $[d(n, m)]$ with circuit depth $O((n + m) \log(\log(n + m)))$ and $O(nm \log(n + m))$ homomorphic computations.

6.2 Genetic Testing

Genetic testing is used for a variety of applications in areas such as personalized medicine, which uses each patient's genetic makeup to personalize treatments and diagnoses. An individual's genetic markers and mutations can be used to test for a genetic predisposition to cancer and other late-onset diseases, drug response, and genetic compatibility testing for the risk of passing a recessive disease to offspring, among other applications [81]. Paternity tests can establish whether one individual is the father of another, while ancestry and genetic genealogical testing enables individuals to learn about where their ancestors might have come from [76].

6.2.1 Paternity and Ancestry Testing. Ancestry testing can be performed on one individual, comparing her genome with a collection of sample genomes to infer information about the individual's genealogy. Another application is familial testing, such as paternity tests, which compare the genomes of two individuals to determine a genetic relationship. Privacy-preserving protocols for these tests seek to allow two parties to compare their genomes without sharing them.

Baldi et al. privately test for paternity using a combination of additive HE and private set intersection cardinality (PSI-CA) [18]. A PSI-CA protocol allows two parties to compute the cardinality of the intersection of their sets without revealing any other portion of their set to the other party. In their model, an individual and a testing service compare portions of their genomes without sharing any information about their contents. De Christofaro et al. expand on this work, implementing two privacy-preserving paternity testing algorithms on Android smartphones: one based on the work of Baldi et al. [18], and another based on private Hamming distance [76].

Bruekers et al. use short tandem repeats (STRs) to match DNA profiles for identity, paternity, and common ancestry testing [41]. STRs are portions of the DNA known to contain repetitive short sequences of nucleotides that are useful for identification. Two parties compare their encrypted STRs with only one party learning the result of the comparison. The identity testing protocol takes place between users A and B with respective STR profiles $\{(a_{i,1}, a_{i,2})\}$ and $\{(b_{i,1}, b_{i,2})\}$ for $1 \leq i \leq N$. Each pair $(x_{i,1}, x_{i,2})$ represents a STR pair from the 22 autosomal chromosomes, with one originating from the mother and one from the father. Identity testing is carried out by determining whether the sum

$$Z = \sum_{i=1}^{N} (a_{i,1} - b_{i,1}) + (a_{i,2} - b_{i,2})$$

is equal to zero. For private evaluation, A encrypts her values under her additive HE private key and sends $\langle \{[a_{i,1}], [a_{i,2}]\} \rangle$ to B. Then, B encrypts his values under A's public key and homomorphically evaluates $[Z]$ and sends $[rZ]$ to A for some random value r. If $[rZ]$ decrypts to zero, then A reports a match. Bruekers et al. use similar methods for paternity and common ancestry testing.

De Christofaro et al. perform privacy-preserving ancestry testing based off of the Jaccard similarity index between two genomes A and B, given by $J(A, B) = |A \cap B|/(|A| + |B|)$ [76]. This test allows two parties to receive a similarity score between their two genomes—for instance, an individual could be compared against a reference genomes held by a testing facility. Due to the large size of genomes, an approximation of this comparison can be sped up using MinHash techniques [40], which use hash functions to analyze similarity between two sets more quickly.

6.2.2 Personalized Medicine. In personalized medicine, an individual's genome is compared against genetic markers held by some testing facility. The results can be used by medical professionals to provide individualized treatment. Frequently, information about the test marker itself is proprietary—for instance, when the testing facility is a pharmaceutical company. Privacy-preserving personalized medicine seeks to hide the genetic markers from the individual being tested, while hiding information about the individual's genome and test results from the testing facility.

Ayday et al. use additive HE to for a privacy-preserving disease susceptibility test [14]. An individual sends their encrypted genome to a testing facility to test for some genetic marker(s). The individual does not learn any specific information about the markers being tested, and the testing facility only learns portions of the individual's genome that it has been authorized to access.

Baldi et al. present a protocol based on private set intersection (PSI) that enables two parties to privately compute the intersection of their sets [18]. This protocol allows an individual to test their genome for the presence of some genetic marker. The testing facility does have direct access to the individual's gene, but does learn the result of the test.

A protocol presented by De Chistofaro et al. allows a party to test a genome for an exact match against an external set of DNA markers without learning the DNA markers or granting any access to their genome data [77]. This protocol takes place between a client and testing laboratory. The client has a genome with nucleotides $t = \{t_1, \ldots, t_n\}$ and the secret key for an additively homomorphic encryption scheme. The testing laboratory has a substring $p = \{p_1, \ldots, p_m\}$ with starting position at index s in the genome. First, the client encrypts each nucleotide in the genome t using the additive HE scheme and sends the encryptions $\{[t_1], \ldots, [t_n]\}$ to the testing laboratory. The testing laboratory then performs the following computation.

```
$[b] \leftarrow [0]$
for $i \leftarrow 0$ to $m - 1$
    $r \leftarrow_{\$} \mathbb{Z}_q$
    $[b] \leftarrow [b] \oplus ([-p_i] \oplus [t_{s+i}])^r$
return $[b]$
```

The testing laboratory homomorphically adds each of the client's encrypted nucleotides to the inverse of its corresponding target substring character and raises this sum to a random exponent r to randomize the value. Whenever $t_{s+i} = p_i$, then $[-p_i] \oplus [t_{s+i}]$ is an encryption of zero. If these values match for all i, the final value $[b]$ will equal zero. Otherwise, the client will receive some random non-zero value due to the random exponentiation at each step.

6.3 Secure Genome-Wide Association Studies (GWAS)

Genome-wide association studies (GWAS) are used to identify the genes involved in a particular disease or trait. These observational studies mostly classify genetic alterations in single nucleotide variants (SNVs) and single nucleotide polymorphisms (SNPs) across groups of patients [28]. Subjects may be split into groups—say, those for whom a disease is present versus not present—and a number of statistical tests can be deployed to quantify the association between a SNP/SNV variant and a disease.

It is common to perform a GWAS on large sets of individuals, numbering in the tens of thousands. The large amount of individual genotype/phenotype data needed to perform large-scale GWAS presents major privacy concerns. De-identification strategies are not always sufficient, as patients have been re-identified from this "anonymized" data in the past [147]. Furthermore, Homer et al. demonstrate that an individual in a GWAS can be identified by analyzing the aggregated allele frequencies in a large number of SNPs [122].

6.3.1 Privacy-Preserving Genome-wide Association Studies. In 2015 and again in 2018, the iDASH competition called for development of secure outsourcing of GWAS via homomorphic encryption. Logistic regression (Section 4.1), a common approach in genome-wide association studies, is used in several solutions [24, 43]. Others use the χ^2 statistic [28, 133, 144, 185, 207], or statistical algorithms including the D' and r^2 measures of linkage disequilibrium [139, 185], the estimation-maximization (EM) algorithm for halotyping [139], and Fisher's exact test [171].

Blatt et al. provide the winning solution to the 2018 challenge using their own RNS-variant of the CKKS scheme [24]. Their experiments are performed using PALISADE, modified to implement their RNS-variant. Carpov et al. implement their solution for private GWAS via logistic regression in TFHE and HEAAN using the Chimera framework [43].

Additional approaches compute privacy-preserving versions of the χ^2 statistic for a GWAS [28, 133, 144, 207]. The χ^2 statistic is given by

$$\chi^2 = \sum_{i,j \in \{1,2\}} \frac{(O_{ij} - E_{ij})^2}{E_{ij}},$$

where O_{ij} and E_{ij} are the observed and expected values for the allele counts for allele j in either the case group ($i = 1$) or control group ($i = 2$).

Bonte et al. [28] compute χ^2 in the encrypted domain and only reveal whether or not this value is significant for the case, not the value itself, to protect against the aforementioned attack of Homer et al. For homomorphic evaluation, the authors rewrite the χ^2 statistic with a common denominator. The statistic is then computed homomorphically as the sum of the numerators, and the comparison threshold is scaled by the denominator. The authors use a masking technique for the final comparison. To compare two encrypted values $[x]$ and $[y]$, the server chooses a random positive r and random $r' \in [1, r)$ and homomorophically evaluates $[r(x - y) + r']$. The decryption of this value is positive if $x > y$; otherwise, $x < y$. The implementation of Bonte et al. uses SHE in FV-NFLlib [28].

Sadat et al.'s SAFETY system uses Paillier's HE scheme [185]. This method is only secure against Homer's attack [122] with a semi-honest querying party. An approach by Zhang et al. provides a secure division protocol that uses a lookup table [207]. However, the depth of the circuit required for this protocol increases rapidly for larger datasets.

7 CONCLUSIONS

Rapid research in the field of fully homomorphic encryption led the field quickly from its first presentation by Gentry to the efficient implementations available today. Implementations of FHE are available from a variety of open-source libraries today. Implementing these FHE libraries often requires an expert understanding of the theoretical foundations of the scheme(s). While these implementations remain difficult for non-experts to navigate, current and future works will provide more high-level, user-friendly technologies. These would enable researchers from a variety of backgrounds to utilize FHE for their own research.

The data-rich, privacy-concerned medical field is well poised to benefit from these innovations. Privacy-preserving classification models could allow a clinician to privately access a predictive model without having to share patients' medical data. FHE has been implemented for a variety of machine learning and statistical tasks, including private evaluation of deep networks, Naive Bayes classification, decision trees, and logistic regression. A variety of genomic applications such as genome testing, computation of Hamming and edit distance, and secure GWAS have also been carried out using FHE. As the speed and usability of FHE improves, new applications for FHE will continue to be discovered.

REFERENCES

[1] Abbas Acar, Hidayet Aksu, A. Selcuk Uluagac, and Mauro Conti. 2018. A survey on homomorphic encryption schemes: Theory and implementation. *ACM Comput. Surv.* 51, 4 (July 2018), 79:1–79:35.

[2] Rakesh Agrawal and Ramakrishnan Srikant. 2000. Privacy-preserving data mining. *SIGMOD Rec.* 29, 2 (May 2000), 439–450.

[3] C. Aguilar-Melchor, S. Fau, C. Fontaine, G. Gogniat, and R. Sirdey. 2013. Recent advances in homomorphic encryption: A possible future for signal processing in the encrypted domain. *IEEE Signal Process. Mag.* 30, 2 (March 2013), 108–117.

[4] Miklós Ajtai and Cynthia Dwork. 1997. A public-key cryptosystem with worst-case/average-case equivalence. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing (STOC'97)* (El Paso, Texas). ACM, New York, NY, USA, 284–293.

[5] Hany Alashwal, Mohamed El Halaby, Jacob J. Crouse, Areeg Abdalla, and Ahmed A. Moustafa. 2019. The application of unsupervised clustering methods to Alzheimer's disease. *Frontiers in Computational Neuroscience* 13, 31 (March 2019), 1–9.

[6] Martin Albrecht, Shi Bai, and Léo Ducas. 2016. A subfield lattice attack on overstretched NTRU assumptions. In *Proceedings, Part I, of the 36th Annual International Cryptology Conference on Advances in Cryptology (CRYPTO'16) Volume 9814*. Springer-Verlag, Berlin, 153–178.

[7] Martin Albrecht, Melissa Chase, Hao Chen, Jintai Ding, Shafi Goldwasser, Sergey Gorbunov, Shai Halevi, Jeffrey Hoffstein, Kim Laine, Kristin Lauter, Satya Lokam, Daniele Micciancio, Dustin Moody, Travis Morrison, Amit Sahai, and Vinod Vaikuntanathan. 2018. *Homomorphic Encryption Security Standard*. Technical Report. HomomorphicEncryption.org, Toronto, Canada.

[8] Jacob Alperin-Sheriff and Chris Peikert. 2014. Faster bootstrapping with polynomial error. In *Advances in Cryptology - CRYPTO 2014 (Lecture Notes in Computer Science)*. Springer, Berlin, 297–314.

[9] R. Altman, E. Asch, D. Bloch, G. Bole, D. Borenstein, K. Brandt, W. Christy, T. D. Cooke, R. Greenwald, M. Hochberg, D. Howell, D. Kaplan, W. Koopman, S. Longley III, H. Mankin, D. J. McShane, T. Medsger Jr., R. Meenan, W. Mikkelsen, R. Moskowitz, W. Murphy, B. Rothschild, M. Segal, L. Sokoloff, and F. Wolfe. 1986. Development of criteria for the classification and reporting of osteoarthritis: Classification of osteoarthritis of the knee. *Arthritis & Rheumatism* 29, 8 (Aug. 1986), 1039–1049.

[10] Yoshinori Aono, Takuya Hayashi, Le Trieu Phong, and Lihua Wang. 2016. Scalable and secure logistic regression via homomorphic encryption. In *Proceedings of the 6th ACM Conference on Data and Application Security and Privacy (CODASPY'16)* (New Orleans, Louisiana). ACM, New York, NY, 142–144.

[11] Frederik Armknecht, Colin Boyd, Christopher Carr, Kristian Gøsteen, Angela Jáschke, Christian A. Reuter, and Martin Strand. 2015. *A Guide to Fully Homomorphic Encryption*. Technical Report 1192.

[12] Frederik Armknecht, Stefan Katzenbeisser, and Andreas Peter. 2013. Group homomorphic encryption: Characterizations, impossibility results, and applications. *Designs, Codes and Cryptography* 67, 2 (May 2013), 209–232.

[13] Mikhail J. Atallah, Florian Kerschbaum, and Wenliang Du. 2003. Secure and private sequence comparisons. In *Proceedings of the ACM Workshop on Privacy in the Electronic Society (WPES'03)* (Washington, DC). ACM Press, 39.

[14] Erman Ayday, Jean Louis Raisaro, Jean-Pierre Hubaux, and Jacques Rougemont. 2013. Protecting and evaluating genomic privacy in medical tests and personalized medicine. In *Proceedings of the 12th ACM Workshop on Workshop on privacy in the Electronic Society (WPES'13)* (Berlin, Germany). Association for Computing Machinery, 95–106.

[15] Mohammad Momin Al Aziz, Mohammad Nazmus Sadat, Dima Alhadidi, Shuang Wang, Xiaoqian Jiang, Cheryl L Brown, and Noman Mohammed. 2017. Privacy-preserving techniques of genomic data—A survey. *Briefings Bioinf.* (Nov. 2017).

[16] Jean-Claude Bajard, Julien Eynard, M. Anwar Hasan, and Vincent Zucca. 2017. A full RNS variant of FV like somewhat homomorphic encryption schemes. In *Selected Areas in Cryptography – SAC 2016*, Roberto Avanzi and Howard Heys (Eds.). Springer International Publishing, Cham, 423–442.

[17] Jean-Claude Bajard, Julien Eynard, Paulo Martins, Leonel Sousa, and Vincent Zucca. 2019. An HPR variant of the FV scheme: Computationally Cheaper, Asymptotically Faster. *Cryptology ePrint Archive, Report 2019/500*. https://eprint.iacr.org/2019/500.

[18] Pierre Baldi, Roberta Baronio, Emiliano De Cristofaro, Paolo Gasti, and Gene Tsudik. 2011. Countering GATTACA: Efficient and secure testing of fully-sequenced human genomes. In *Proceedings of the 18th ACM Conference on Computer and Communications Security (CCS'11)* New York, NY. ACM, 691–702.

[19] Josh Benaloh. 1987. *Verifiable Secret-ballot Elections*. Ph.D. Dissertation. New Haven, CT. AAI8809191.

[20] Josh Benaloh. 1994. Dense probabilistic encryption. In *Proceedings of the Workshop on Selected Areas of Cryptography*. 120–128.

[21] Daniel Benarroch, Zvika Brakerski, and Tancrède Lepoint. 2017. FHE over the integers: Decomposed and batched in the post-quantum regime. In *Public-Key Cryptography - PKC 2017*, Serge Fehr (Ed.). Vol. 10175. Springer, Berlin, 271–301.

[22] R. Bender and U. Grouven. 1997. Ordinal logistic regression in medical research. *Journal of the Royal College of Physicians of London* 31, 5 (Oct. 1997), 546–551.

[23] Jean-François Biasse and Luis Ruiz. 2015. FHEW with efficient multibit bootstrapping. In *Progress in Cryptology – LATINCRYPT 2015 (Lecture Notes in Computer Science)*, Kristin Lauter and Francisco Rodríguez-Henríquez (Eds.). Springer International Publishing, 119–135.

[24] Marcelo Blatt, Alexander Gusev, Yuriy Polyakov, Kurt Rohloff, and Vinod Vaikuntanathan. 2019. Optimized Homomorphic Encryption Solution for Secure Genome-Wide Association Studies. *Cryptology ePrint Archive, Report 2019/223*. https://eprint.iacr.org//2019//223.

[25] Fabian Boemer, Yixing Lao, Rosario Cammarota, and Casimir Wierzynski. 2019. nGraph-HE: A graph compiler for deep learning on homomorphically encrypted data. In *Proceedings of the 16th ACM International Conference on Computing Frontiers (CF'19)* (Alghero, Italy). ACM, 3–13.

[26] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. 2005. Evaluating 2-DNF formulas on ciphertexts. In *Proceedings of the Second International Conference on Theory of Cryptography (TCC'05)* (Cambridge, MA). Springer-Verlag, Berlin, 325–341.

[27] Guillaume Bonnoron, Léo Ducas, and Max Fillinger. 2018. Large FHE gates from tensored homomorphic accumulator. In *Progress in Cryptology - AFRICACRYPT 2018 (Lecture Notes in Computer Science)*, Antoine Joux, Abderrahmane Nitaj, and Tajjeeddine Rachidi (Eds.). Springer International Publishing, 217–251.

[28] Charlotte Bonte, Eleftheria Makri, Amin Ardeshirdavani, Jaak Simm, Yves Moreau, and Frederik Vercauteren. 2018. Towards practical privacy-preserving genome-wide association study. *BMC Bioinf.* 19, 1 (Dec. 2018), 537.

[29] Joppe W. Bos, Kristin Lauter, Jake Loftus, and Michael Naehrig. 2013. *Improved Security for a Ring-Based Fully Homomorphic Encryption Scheme*. Springer, Berlin, 45–64.

[30] Raphael Bost, Raluca Ada Popa, Stephen Tu, and Shafi Goldwasser. 2015. Machine learning classification over encrypted data. *Symposium on Network and Distributed System Security* (NDSS).

[31] Christina Boura, Nicolas Gama, and Mariya Georgieva. 2018. Chimera: A unified framework for B/FV, TFHE and HEAAN fully homomorphic encryption and predictions for deep learning. *Cryptology ePrint Archive, Report 2018/758*. https://eprint.iacr.org//2018//758.

[32] Florian Bourse, Michele Minelli, Matthias Minihold, and Pascal Paillier. 2018. Fast homomorphic evaluation of deep discretized neural networks. In *Proceedings of the 38th Annual International Cryptology Conference, Santa Barbara, CA, August 19–23, 2018, Part III, Advances in Cryptology (CRYPTO'18)*. 483–512.

[33] Zvika Brakerski. 2012. Fully homomorphic encryption without modulus switching from classical GapSVP. In *Advances in Cryptology – CRYPTO 2012*, Reihaneh Safavi-Naini and Ran Canetti (Eds.). Springer, Berlin, 868–886.

[34] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. 2012. (Leveled) fully homomorphic encryption without bootstrapping. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference* (Cambridge, Massachusetts) *(ITCS'12)*. ACM, New York, NY, 309–325.

[35] Zvika Brakerski and Vinod Vaikuntanathan. 2011. Efficient fully homomorphic encryption from (standard) LWE. In *Proceedings of the 2011 IEEE 52nd Annual Symposium on Foundations of Computer Science (FOCS'11)*. IEEE Computer Society, Washington, DC, 97–106.

[36] Zvika Brakerski and Vinod Vaikuntanathan. 2011. Fully homomorphic encryption from ring-LWE and security for key dependent messages. In *Proceedings of the 31st Annual Conference on Advances in Cryptology (CRYPTO'11)*. Springer-Verlag, Berlin, 505–524.

[37] Zvika Brakerski and Vinod Vaikuntanathan. 2014. Lattice-based FHE as secure as PKE. In *Proceedings of the 5th Conference on Innovations in Theoretical Computer Science* (Princeton, New Jersey) *(ITCS'14)*. ACM, New York, NY, 1–12.

[38] Zvika Brakerski, Vinod Vaikuntanathan, and Craig Gentry. 2012. Fully homomorphic encryption without bootstrapping. In *In Innovations in Theoretical Computer Science*.

[39] Justin Brickell and Vitaly Shmatikov. 2009. Privacy-preserving classifier learning. In *Financial Cryptography and Data Security (Lecture Notes in Computer Science)*. Springer, Berlin, 128–147.

[40] Andrei Z. Broder. 1997. On the resemblance and containment of documents. In *SEQUENCES'97: Proceedings of the Compression and Complexity of Sequences 1997*. IEEE Computer Society, Washington, DC, 21.

[41] Fons Bruekers, Stefan Katzenbeisser, Klaus Kursawe, and Pim Tuyls. 2008. *Privacy-preserving Matching of DNA Profiles*. Technical Report.

[42] John M. Butler. 2015. The future of forensic DNA analysis. *Philosophical Transactions of the Royal Society B: Biological Sciences* 370, 1674 (Aug. 2015).

[43] Sergiu Carpov, Nicolas Gama, Mariya Georgieva, and Juan Ramon Troncoso-Pastoriza. 2019. Privacy-preserving semi-parallel logistic regression training with Fully Homomorphic Encryption. *Cryptology ePrint Archive, Report 2019/101*. https://eprint.iacr.org/2019/101.

[44] Sergiu Carpov, Malika Izabachène, and Victor Mollimard. 2019. New techniques for multi-value input homomorphic evaluation and applications. In *Topics in Cryptology—CT-RSA 2019*, Mitsuru Matsui (Ed.). Vol. 11405. Springer International Publishing, 106–126.

[45] CEA-LIST. 2018. *Cingulata*. Retrieved August 21, 2019 from https://github.com/CEA-LIST/Cingulata.

[46] Hervé Chabanne, Amaury de Wargny, Jonathan Milgram, Constance Morel, and Emmanuel Prouff. 2017. Privacy-Preserving Classification on Deep Neural Network. *Cryptology ePrint Archive, Report 2017/035*. https://eprint.iacr.org/2017/035.

[47] Hao Chen, Ilaria Chillotti, and Yongsoo Song. 2018. Improved Bootstrapping for Approximate Homomorphic Encryption. *Cryptology ePrint Archive, Report 2018/1043*. https://eprint.iacr.org//2018//1043.

[48] Hao Chen, Ran Gilad-Bachrach, Kyoohyung Han, Zhicong Huang, Amir Jalali, Kim Laine, and Kristin Lauter. 2018. Logistic regression over encrypted data from fully homomorphic encryption. *BMC Medical Genomics* 11, 4 (Dec. 2018), 81.

[49] Hao Chen and Kyoohyung Han. 2018. Homomorphic Lower Digits Removal and Improved FHE Bootstrapping. *Cryptology ePrint Archive, Report 2018/067*. https://eprint.iacr.org/2018/067.

[50] Jung Hee Cheon, Jean-Sébastien Coron, Jinsu Kim, Moon Sung Lee, Tancrède Lepoint, Mehdi Tibouchi, and Aaram Yun. 2013. Batch fully homomorphic encryption over the integers. In *Advances in Cryptology - EUROCRYPT 2013 (Lecture Notes in Computer Science)*, Thomas Johansson and Phong Q. Nguyen (Eds.). Springer Berlin, 315–335.

[51] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. 2018. Bootstrapping for Approximate Homomorphic Encryption. *Cryptology ePrint Archive, Report 2018/153*. https://eprint.iacr.org/2018/153.

[52] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. 2019. A full RNS variant of approximate homomorphic encryption. In *Selected Areas in Cryptography - SAC 2018 (Lecture Notes in Computer Science)*, Carlos Cid and Michael J. Jacobson Jr. (Eds.). Springer International Publishing, 347–368.

[53] Jung Hee Cheon, Duhyeong Kim, and Jai Hyun Park. 2019. Towards a Practical Cluster Analysis over Encrypted Data. *Cryptology ePrint Archive, Report 2019/465*. https://eprint.iacr.org/2019/465.

[54] Jung Hee Cheon, Miran Kim, and Kristin Lauter. 2015. *Homomorphic Computation of Edit Distance*. Springer, Berlin, 194–212.

[55] Jung Hee Cheon and Damien Stehlé. 2015. Fully homomophic encryption over the integers revisited. In *Advances in Cryptology – EUROCRYPT 2015*, Elisabeth Oswald and Marc Fischlin (Eds.). Vol. 9056. Springer Berlin, 513–536.

[56] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. 2016. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In *Proceedings of the 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam December 4–8, 2016, Part I, Advances in Cryptology (ASIACRYPT'16)*. 3–33.

[57] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. 2017. Faster packed homomorphic operations and efficient circuit bootstrapping for TFHE. In *Advances in Cryptology - ASIACRYPT 2017 (Lecture Notes in Computer Science)*, Tsuyoshi Takagi and Thomas Peyrin (Eds.). Springer International Publishing, 377–408.

[58] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. 2018. TFHE: Fast Fully Homomorphic Encryption over the Torus. Retrieved from http://eprint.iacr.org/2018/421.

[59] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. August 2016. TFHE: Fast Fully Homomorphic Encryption Library. Retrieved from https://tfhe.github.io/tfhe/.

[60] Ching Travers, Himmelstein Daniel S., Beaulieu-Jones Brett K., Kalinin Alexandr A., Do Brian T., Way Gregory P., Ferrero Enrico, Agapow Paul-Michael, Zietz Michael, Hoffman Michael M., Xie Wei, Rosen Gail L., Lengerich Benjamin J., Israeli Johnny, Lanchantin Jack, Woloszynek Stephen, Carpenter Anne E., Shrikumar Avanti, Xu Jinbo, Cofer Evan M., Lavender Christopher A., Turaga Srinivas C., Alexandari Amr M., Lu Zhiyong, Harris David J., DeCaprio Dave, Qi Yanjun, Kundaje Anshul, Peng Yifan, Wiley Laura K., Segler Marwin H. S., Boca Simina M., Swamidass S. Joshua, Huang Austin, Gitter Anthony, and Greene Casey S. 2018-04-30. Opportunities and obstacles for deep learning in biology and medicine. *J. R. Soc. Interface* 15, 141 (2018-04-30), 47.

[61] Tung Chou and Claudio Orlandi. 2015. The simplest protocol for oblivious transfer. In *Progress in Cryptology – LATINCRYPT 2015*, Kristin Lauter and Francisco Rodríguez-Henríquez (Eds.). Springer International Publishing, Cham, 40–58.

[62] Homomorphic Encryption Standardization Consortium. 2018. *Building Applications with Homomorphic Encryption*. Retrieved May 19, 2019 from http://homomorphicencryption.org/wp-content/uploads/2018/10/CCS-HE-Tutorial-Slides.pdf.

[63] Jean-Sébastien Coron, Tancrède Lepoint, and Mehdi Tibouchi. 2013. Batch Fully Homomorphic Encryption over the Integers. *Cryptology ePrint Archive, Report 2013/036*. https://eprint.iacr.org/2013/036.

[64] Jean-Sébastien Coron, Tancrède Lepoint, and Mehdi Tibouchi. 2014. Scale-invariant fully homomorphic encryption over the integers. In *Proceedings of the 17th International Conference on Public-Key Cryptography (PKC'14) Volume 8383*. Springer-Verlag New York, Inc., New York, NY, 311–328.

[65] Jean-Sébastien Coron, Avradip Mandal, David Naccache, and Mehdi Tibouchi. 2011. Fully homomorphic encryption over the integers with shorter public keys. In *Proceedings of the 31st Annual Conference on Advances in Cryptology* (Santa Barbara, CA) *(CRYPTO'11)*. Springer-Verlag, Berlin, 487–504.

[66] Jean-Sébastien Coron, David Naccache, and Mehdi Tibouchi. 2012. Public key compression and modulus switching for fully homomorphic encryption over the integers. In *Proceedings of the 31st Annual International Conference on Theory and Applications of Cryptographic Techniques* (Cambridge, UK) *(EUROCRYPT'12)*. Springer-Verlag, Berlin, 446–464.

[67] Ronald Cramer, Léo Ducas, Chris Peikert, and Oded Regev. 2016. Recovering short generators of principal ideals in cyclotomic rings. In *Advances in Cryptology - EUROCRYPT 2016 (Lecture Notes in Computer Science)*, Marc Fischlin and Jean-Sébastien Coron (Eds.). Springer, Berlin, 559–585.

[68] Jack L. H. Crawford, Craig Gentry, Shai Halevi, Daniel Platt, and Victor Shoup. 2018. Doing real work with FHE: The case of logistic regression. In *Proceedings of the 6th Workshop on Encrypted Computing and Applied Homomorphic Cryptography* (Toronto, Canada) *(WAHC'18)*. ACM, New York, NY, 1–12.

[69] Eric Crocket and Chris Peikert. 2017. *$\Lambda \circ \lambda$: Functional Lattice Cryptography*. Retrieved from https://hackage.haskell.org/package/lol-apps.

[70] Eric Crockett and Chris Peikert. 2015. $\Lambda \circ \lambda$: Functional Lattice Cryptography. *Cryptology ePrint Archive, Report 2015/1134*. https://eprint.iacr.org/2015/1134.

[71] Eric Crockett, Chris Peikert, and Chad Sharp. 2018. ALCHEMY: A language and compiler for homomorphic encryption made easY. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS'18*) (Toronto, Canada). ACM Press, 1020–1037.

[72] Vernam Lab. 2018. *CUDA-accelerated Fully Homomorphic Encryption (cuFHE)*. Vernam Lab. Retrieved April 13, 2020 from https:/github.com//vernamlab/cuFHE.

[73] Ivan Damgård and Mads Jurik. 2001. A generalisation, a simplification and some applications of paillier's probabilistic public-key system. In *Public Key Cryptography (Lecture Notes in Computer Science)*, Kwangjo Kim (Ed.). Springer Berlin, 119–136.

[74] Ivan Damgård, Mads Jurik, and Jesper Buus Nielsen. 2010-12. A generalization of Paillier's public-key system with applications to electronic voting. *International Journal of Information Security* 9, 6 (2010-12), 371–385.

[75] CSIRO Data61. 2013. Python Paillier Library. Retrieved April 13, 2020 from https://github.com/data61/python-paillier.

[76] Emiliano De Cristofaro, Sky Faber, Paolo Gasti, and Gene Tsudik. 2012. Genodroid: Are privacy-preserving genomic tests ready for prime time? In *Proceedings of the 2012 ACM Workshop on Privacy in the Electronic Society* (Raleigh, North Carolina) *(WPES'12)*. Association for Computing Machinery, 97–108.

[77] Emiliano De Cristofaro, Sky Faber, and Gene Tsudik. 2013. Secure genomic testing with size- and position-hiding private substring matching. In *Proceedings of the 12th ACM Workshop on Workshop on Privacy in the Electronic Society (WPES'13)* (Berlin, Germany). ACM Press, 107–118.

[78] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. 2010. Fully homomorphic encryption over the integers. In *Advances in Cryptology—EUROCRYPT 2010 (Lecture Notes in Computer Science)*. Springer, Berlin, 24–43.

[79] Yarkın Doröz, Yin Hu, and Berk Sunar. 2016-08. Homomorphic AES evaluation using the modified LTV scheme. *Designs, Codes and Cryptography* 80, 2 (2016-08), 333–358.

[80] Nathan Dowlin, Ran Gilad-Bachrach, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. 2015. Manual for Using Homomorphic Encryption for Bioinformatics. Retrieved from https://www.microsoft.com/en-us/research/publication/manual-for-using-homomorphic-encryption-for-bioinformatics/.

[81] Radoje Drmanac. 2011-02-09. The advent of personal genome sequencing. *Genetics in Medicine* 13 (2011-02-09), 188–190.

[82] Léo Ducas and Daniele Micciancio. 2015. FHEW: Bootstrapping homomorphic encryption in less than a second. In *Advances in Cryptology—EUROCRYPT 2015*, Elisabeth Oswald and Marc Fischlin (Eds.). Vol. 9056. Springer Berlin, 617–640.

[83] Leo Ducas and Daniele Micciancio. 2017. *FHEW A Fully Homomorphic Encryption Library*. https://github.com/lducas/FHEW.

[84] Cynthia Dwork. 2006. *Differential Privacy*. Springer, Berlin, 1–12.

[85] T. ElGamal. 1985. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory* 31, 4 (July 1985), 469–472.

[86] Jenfeng Fan and Frederik Vercauteren. 2012. Somewhat Practical Fully Homomorphic Encryption. *Cryptology ePrint Archive, Report 2012/144*. http://eprint.iacr.org/2012/144.

[87] Caroline Fontaine and Fabien Galand. 2007. A survey of homomorphic encryption for nonspecialists. *EURASIP Journal on Information Security*1 (Dec. 2007), 10.

[88] Matteo Frigo. 2004-04-01. A fast Fourier transform compiler. *ACM SIGPLAN Notices* 39, 4 (2004-04-01), 642.

[89] CryptoExperts. 2016. *FV-NFLlib*. CryptoExperts. Retrieved August 21, 2019 from https://github.com/CryptoExperts/FV-NFLlib.

[90] Steven D. Galbraith. 2002. Elliptic curve paillier schemes. *J. Cryptol.* 15, 2 (Jan. 2002), 129–138.

[91] 2016. Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC. *Office Journal of the European Union* L 119 (April 2016), 1–88.

[92] Nicholas Genise, Craig Gentry, Shai Halevi, Baiyu Li, and Daniele Micciancio. 2019. Homomorphic encryption for finite automata. *IACR Cryptology ePrint Archive* 2019 (2019), 176. https://eprint.iacr.org/2019/176.

[93] Craig Gentry. 2009. *A Fully Homomorphic Encryption Scheme*. Ph.D. Thesis. Stanford University, Stanford, CA, USA.

[94] Craig Gentry. 2009. Fully homomorphic encryption using ideal lattices. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing (STOC'09)*. ACM, 169–178.

[95] Craig Gentry. 2010. Computing arbitrary functions of encrypted data. *Commun. ACM* 53, 3 (March 2010), 97–105.

[96] Craig Gentry. 2010. Toward basing fully homomorphic encryption on worst-case hardness. In *Proceedings of the 30th Annual Conference on Advances in Cryptology* (Santa Barbara, CA) *(CRYPTO'10)*. Springer-Verlag, Berlin, 116–137.

[97] Craig Gentry. 2014. *Computing on the Edge of Chaos: Structure and Randomness in Encrypted Computation*. Technical Report 106. 609–632 pages.

[98] Craig Gentry and Shai Halevi. 2011. Fully Homomorphic Encryption without Squashing Using Depth-3 Arithmetic Circuits. *Cryptology ePrint Archive, Report 2011/279*. http://eprint.iacr.org/2011/279.

[99] Craig Gentry and Shai Halevi. 2011. Implementing gentry's fully-homomorphic encryption scheme. In *Proceedings of the 30th Annual International Conference on Theory and Applications of Cryptographic Techniques: Advances in Cryptology* (Tallinn, Estonia) *(EUROCRYPT'11)*. Springer-Verlag, Berlin, 129–148.

[100] Craig Gentry, Shai Halevi, Chris Peikert, and Nigel P. Smart. 2012. Ring switching in BGV-style homomorphic encryption. In *Security and Cryptography for Networks (Lecture Notes in Computer Science)*. Springer, 19–37.

[101] Craig Gentry, Shai Halevi, and Nigel P. Smart. 2012. *Better Bootstrapping in Fully Homomorphic Encryption*. Springer, Berlin, 1–16.

[102] Craig Gentry, Shai Halevi, and Nigel P. Smart. 2012. Fully homomorphic encryption with polylog overhead. In *Advances in Cryptology—EUROCRYPT 2012 (Lecture Notes in Computer Science)*. Springer, Berlin, 465–482.

[103] Craig Gentry, Shai Halevi, and Nigel P. Smart. 2012. Homomorphic evaluation of the AES circuit. In *Advances in Cryptology—CRYPTO 2012*, Reihaneh Safavi-Naini and Ran Canetti (Eds.). Springer Berlin, 850–867.

[104] Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. 2010. A simple BGN-type cryptosystem from LWE. In *Proceedings of the 29th Annual International Conference on Theory and Applications of Cryptographic Techniques* (French Riviera, France) *(EUROCRYPT'10)*. Springer-Verlag, Berlin, 506–522.

[105] Craig Gentry, Amit Sahai, and Brent Waters. 2013. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *CRYPTO*. Springer, 75–92.

[106] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. 2016. CryptoNets: Applying neural networks to encrypted data with high throughput and accuracy. In *PMLR*. 201–210.

[107] Shafi Goldwasser and Silvio Micali. 1982. Probabilistic encryption & how to play mental poker keeping secret all partial information. In *Proceedings of the 14th Annual ACM Symposium on Theory of Computing (STOC'82)*. ACM, New York, NY, 365–377.

[108] Shafi Goldwasser and Silvio Micali. 1984. Probabilistic encryption. *J. Comput. Syst. Sci.* 28, 2 (1984), 270–299.

[109] Alexey Gribov, Kelsey Horan, Jonathan Gryak, Kayvan Najarian, Vladimir Shpilrain, Reza Soroushmehr, and Delaram Kahrobaei. 2019. Medical diagnostics based on encrypted medical data. In *Bio-inspired Information and Communication Technologies* (Cham) *(Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering)*, Adriana Compagnoni, William Casey, Yang Cai, and Bud Mishra (Eds.). Springer International Publishing, 98–111.

[110] Alexey Gribov, Delaram Kahrobaei, and Vladimir Shpilrain. 2018. Practical private-key fully homomorphic encryption in rings. *Groups, Complexity, Cryptology* 10, 1 (2018), 17–27.

[111] Shai Halevi. 2017. Homomorphic encryption. In *Tutorials on the Foundations of Cryptography* (1st ed.), Yehuda Lindell (Ed.). Springer International Publishing, 219–276.

[112] Shai Halevi, Yuriy Polyakov, and Victor Shoup. 2019. An improved RNS variant of the BFV homomorphic encryption scheme. In *Topics in Cryptology—CT-RSA 2019 (Lecture Notes in Computer Science)*, Mitsuru Matsui (Ed.). Springer International Publishing, 83–105.

[113] Shai Halevi and Victor Shoup. 2018. Faster Homomorphic Linear Transformations in HElib. *Cryptology ePrint Archive, Report 2018/244*. https://eprint.iacr.org/2018/244.

[114] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. 2009. *The Elements of Statistical Learning* (2nd ed.). Springer New York Inc., New York, NY.

[115] CryptoLab Inc. 2018. *HEAAN Software Library*. CryptoLab Inc. Retrieved August 21, 2019 from https://github.com/snucrypto/HEAAN.

[116] IBM Corp. 2014. *HElib: An Implementation of Homomorphic Encryption*. IBM Corp. Retrieved August 21, 2019 from https://github.com/HomEnc/HElib.

[117] HES. 2019. *Homomorphic Encryption Standardization*. Retrieved April 13, 2020 from http://homomorphicencryption.org/.

[118] Ehsan Hesamifard, Hassan Takabi, and Mehdi Ghasemi. 2017. CryptoDL: Deep Neural Networks over Encrypted Data. (Nov 2017). arxiv:1711.05189

[119] Ehsan Hesamifard, Hassan Takabi, Mehdi Ghasemi, and Rebecca N. Wright. 2018-01-01. Privacy-preserving machine learning as a service. *Proceedings on Privacy Enhancing Technologies* 2018, 3 (2018-01-01).

[120] Ryo Hiromasa, Masayuki Abe, and Tatsuaki Okamoto. 2015. Packing messages and optimizing bootstrapping in GSW-FHE. In *Proceedings of the18th IACR International Conference on Practice and Theory in Public-Key Cryptography, Gaithersburg, MD, March 30 - April 1, 2015, (PKC'15)*. 699–715.

[121] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. 1998. NTRU: A ring-based public key cryptosystem. In *Lecture Notes in Computer Science*. Springer-Verlag, 267–288.

[122] Nils Homer, Szabolcs Szelinger, Margot Redman, David Duggan, Waibhav Tembe, Jill Muehling, John V. Pearson, Dietrich A. Stephan, Stanley F. Nelson, and David W. Craig. 2008-08-29. Resolving individuals contributing trace

amounts of DNA to highly complex mixtures using high-density SNP genotyping microarrays. *PLOS Genet.* 4, 8 (2008-08-29).

[123] Nick Howgrave-Graham. 2001. Approximate integer common divisors. In *Revised Papers from the International Conference on Cryptography and Lattices (CaLC'01).* Springer-Verlag, London, UK, 51–66.

[124] Galois Inc. 2018. *RAMPARTS: RApid Machine-learning Processing Applications and Reconfigurable Targeting of Security.* Galois Inc. Retrieved April 13, 2020 from https://galois.com/project/ramparts/.

[125] Ayman Jarrous and Benny Pinkas. 2009. Secure hamming distance based computation and its applications. In *RoboCup 2001: Robot Soccer World Cup V*, Andreas Birk, Silvia Coradeschi, and Satoshi Tadokoro (Eds.). Vol. 2377. Springer Berlin, 107–124.

[126] Angela Jäschke and Frederik Armknecht. 2019. Unsupervised machine learning on encrypted data. In *Selected Areas in Cryptography—SAC 2018*, Carlos Cid and Michael J. Jacobson Jr. (Eds.). Springer International Publishing, Cham, 453–478.

[127] Xiaoqian Jiang, Miran Kim, Kristin Lauter, and Yongsoo Song. 2018. Secure outsourced matrix computation and application to neural networks. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security* (Toronto, Canada) *(CCS'18).* ACM, New York, NY, 1209–1222.

[128] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. 2018. GAZELLE: A low latency framework for secure neural network inference. In *27th USENIX Security Symposium (USENIX Security'18).* USENIX Association, Baltimore, MD, 1651–1669.

[129] Soni Jyoti, Ansari Ujma, Sharma Dipesh, and Soni Sunita. 2011-03-31. Predictive data mining for medical diagnosis: An overview of heart disease prediction. *International Journal of Computer Applications* 17, 8 (2011-03-31), 43–48.

[130] Jocelyn Kaiser. 2018. We will find you: DNA search used to nab Golden State Killer can home in on about 60% of white Americans. *Science* (Oct. 2018). DOI: https://doi.org/10.1126/science.aav7021

[131] A. Khedr, G. Gulak, and V. Vaikuntanathan. 2016. SHIELD: Scalable homomorphic implementation of encrypted data-classifiers. *IEEE Trans. Comput.* 65, 9 (Sept. 2016), 2848–2858.

[132] Andrey Kim, Yongsoo Song, Miran Kim, Keewoo Lee, and Jung Hee Cheon. 2018-10-11. Logistic regression model training based on the approximate homomorphic encryption. *BMC Medical Genomics* 11, 4 (2018-10-11), 83.

[133] Miran Kim and Kristin Lauter. 2015. Private Genome Analysis through Homomorphic Encryption. *Cryptology ePrint Archive, Report 2015/965.* http://eprint.iacr.org/2015/965.

[134] Ovunc Kocabas and Tolga Soyata. 2020. Towards privacy-preserving medical cloud computing using homomorphic encryption. *Virtual and Mobile Healthcare: Breakthroughs in Research and Practice* (2020), 93–125.

[135] Igor Kononenko. 2001-08-01. Machine learning for medical diagnosis: History, state of the art and perspective. *Artif. Intell. Med.* 23, 1 (2001-08-01), 89–109.

[136] R. L. Lagendijk and M. Barni. 2013. Encrypted signal processing for privacy protection: Conveying the utility of homomorphic encryption and multiparty computation. *IEEE Signal Process. Mag.* 30, 1 (Jan. 2013), 82–105.

[137] Kim Laine, Hao Chen, Rachel Player, and Yuhou Xia. 2018. High-precision arithmetic in homomorphic encryption. In *Lecture Notes in Computer Science* (topics in cryptology ct-rsa 2018 ed.), Vol. 10808. Springer, Cham, 116–136.

[138] EPFL-LDS. 2019. *Lattigo 1.3.0.* EPFL-LDS. Retrieved April 13, 2020 from http://github.com/ldsec/lattigo.

[139] Kristin Lauter, Adriana López-Alt, and Michael Naehrig. 2015. *Private Computation on Encrypted Genomic Data.* Springer International Publishing, Cham, 3–27.

[140] Ping Li, Jin Li, Zhengan Huang, Tong Li, Chong-Zhi Gao, Siu-Ming Yiu, and Kai Chen. 2017. Multi-key privacy-preserving deep learning in cloud computing. *Future Generation Computer Systems* 74 (Sept. 2017), 76–85.

[141] Yehuda Lindell and Benny Pinkas. 2000-08-20. Privacy preserving data mining. In *Advances in Cryptology—CRYPTO 2000 (Lecture Notes in Computer Science).* Springer, Berlin, 36–54.

[142] Jian Liu, Mika Juuti, Yao Lu, and N. Asokan. 2017. Oblivious neural network predictions via MiniONN transformations. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (Dallas, Texas, USA) *(CCS'17).* ACM, New York, NY, 619–631.

[143] Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. 2012. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In *Proceedings of the 44th Annual ACM Symposium on Theory of Computing* (New York, New York) *(STOC'12).* ACM, New York, NY, 1219–1234.

[144] Wen-Jie Lu, Yoshiji Yamada, and Jun Sakuma. 2015-12-21. Privacy-preserving genome-wide association studies on cloud environment using fully homomorphic encryption. *BMC Med. Inf. Decis. Making* 15, 5 (2015-12-21), S1.

[145] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. 2013. On ideal lattices and learning with errors over rings. *J. ACM* 60, 6, Article 43 (Nov. 2013), 35 pages.

[146] Xu Ma, Xiaofeng Chen, and Xiaoyu Zhang. 2019. Non-interactive privacy-preserving neural network prediction. *Inf. Sci.* 481 (May 2019), 507–519.

[147] Bradley Malin. 2006. Re-identification of Familial database records. *AMIA Annual Symposium Proceedings* 2006 (02 2006), 524–8.

[148] Paulo Martins, Leonel Sousa, and Artur Mariano. 2017. A survey on fully homomorphic encryption: An engineering perspective. *ACM Comput. Surv.* 50, 6 (Dec. 2017), 83:1–83:33.

[149] Daniele Micciancio and Jessica Sorrell. 2018. Ring packing and amortized FHEW bootstrapping. *Cryptology ePrint Archive, Report 2018/532*. https://eprint.iacr.org/2018/532.

[150] Seonwoo Min, Byunghan Lee, and Sungroh Yoon. 2017. Deep learning in bioinformatics. *Briefings Bioinf.* 18, 5 (2017), 851–869.

[151] Riccardo Miotto, Fei Wang, Shuang Wang, Xiaoqian Jiang, and Joel T. Dudley. 2018. Deep learning for healthcare: Review, opportunities and challenges. *Briefings Bioinf.* 19, 6 (2018), 1236–1246.

[152] Payman Mohassel and Peter Rindal. 2018. ABY^3: A mixed protocol framework for machine learning. In *ACM Conference on Computer and Communications Security*. ACM, 35–52.

[153] P. Mohassel and Y. Zhang. 2017-05. SecureML: A system for scalable privacy-preserving machine learning. 19–38.

[154] Ciara Moore, Maire O'Neill, Elizabeth O'Sullivan, Yarkin Doroz, and Berk Sunar. 2014. Practical homomorphic encryption: A survey. In *2014 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, Melbourne VIC, Australia, 2792–2795.

[155] David W. Mount. 2004. *Bioinformatics: Sequence and Genome Analysis* (2nd ed.). Cold Spring Harbor Laboratory Press.

[156] David Naccache and Jacques Stern. 1998. A new public key cryptosystem based on higher residues. In *Proceedings of the 5th ACM Conference on Computer and Communications Security* (San Francisco, California) *(CCS'98)*. ACM, New York, NY, 59–66.

[157] Michael Naehrig, Kristin Lauter, and Vinod Vaikuntanathan. 2011. Can homomorphic encryption be practical? In *Proceedings of the 3rd ACM Workshop on Cloud Computing Security Workshop* (Chicago, Illinois) *(CCSW'11)*. ACM, New York, NY, 113–124.

[158] Muhammad Naveed, Erman Ayday, Ellen W. Clayton, Jacques Fellay, Carl A. Gunter, Jean-Pierre Hubaux, Bradley A. Malin, and Xiaofeng Wang. 2015. Privacy in the genomic era. *ACM Computing Surveys* 48, 1 (Sept. 2015).

[159] Yu. E. Nexterov. 1983. A method of solving a convex programming problem with convergence rate $O\left(\frac{1}{k^2}\right)$. *Dokl. Akad. Nauk SSSR* 269, 3 (1983), 543–547.

[160] NuCypher. 2018. *NuCypher Fully Homomorphic Encryption (NuFHE)*. NuCypher. https://github.com/nucypher/nufhe.

[161] Ziad Obermeyer and Ezekiel J. Emanuel. 2016-09-29. Predicting the future - Big data, machine learning, and clinical medicine. *The New England Journal of Medicine* 375, 13 (2016-09-29), 1216–1219.

[162] U.S. Department of Health & Human Services. 1996. *The Health Insurance Portability and Accountability Act of 1996 (HIPAA)*. Retrieved April 13, 2020 from https://aspe.hhs.gov/report/health-insurance-portability-and-accountability-act-1996.

[163] Tatsuaki Okamoto and Shigenori Uchiyama. 1998. A new public-key cryptosystem as secure as factoring. In *EUROCRYPT (Lecture Notes in Computer Science)*, Vol. 1403. Springer, 308–318.

[164] Pascal Paillier. 1999. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology - EUROCRYPT'99 (Lecture Notes in Computer Science)*. Springer, Berlin, 223–238.

[165] PALISADE. 2020. *The PALISADE Lattice Cryptography Library*. https://palisade-crypto.org/software-library/.

[166] Chris Peikert. 2016. A decade of lattice cryptography. *Foundations and Trends in Theoretical Computer Science* 10, 4 (March 2016), 283–424.

[167] Chris Peikert. 2018. *A Language and Compiler for Homomorphic Encryption Made easY (ALCHEMY)*. Retrieved August 21, 2019 from https://github.com/cpeikert/ALCHEMY.

[168] Le Trieu Phong, Yoshinori Aono, Takuya Hayashi, Lihua Wang, and Shiho Moriai. 2018. Privacy-preserving deep learning via additively homomorphic encryption. *Trans. Info. For. Sec.* 13, 5 (May 2018), 1333–1345.

[169] Vili Podgorelec, Peter Kokol, Bruno Stiglic, and Ivan Rozman. 2002-10-01. Decision trees: An overview and their use in medicine. *J. Med. Syst.* 26, 5 (2002-10-01), 445–463.

[170] Yuriy Polyakov, Kurt Rohloff, and Gerard W. Ryan. 2017. PALISADE Lattice Cryptography Library User Manual.

[171] Anna Poon, Steve Jankly, and Tingting Chen. 2018. Privacy preserving Fisher's exact test on genomic data. In *2018 IEEE International Conference on Big Data (Big Data)*. 2546–2553.

[172] Pyfhel. 2018. *PYthon for Homomorphic Encryption Libraries (Pyfhel)*. Retrieved April 13, 2020 from https://github.com/ibarrond/Pyfhel.

[173] Michael O. Rabin. 1981. *How to Exchange Secrets with Oblivious Transfer*. Technical Report TR-81. Aiken Computation Lab, Harvard University.

[174] Oded Regev. 2005. On lattices, learning with errors, random linear codes, and cryptography. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing* (Baltimore, MD) *(STOC'05)*. 84–93.

[175] Microsoft Research. 2017. *Microsoft SEAL (release 2.3)*. https://github.com/Microsoft/SEAL.

[176] Microsoft Research. 2018. *Microsoft SEAL (release 3.0)*. http://sealcrypto.org.

[177] Microsoft Research. 2019. *Microsoft SEAL (release 3.3)*. https://github.com/Microsoft/SEAL.

[178] M. Sadegh Riazi, Christian Weinert, Oleksandr Tkachenko, Ebrahim M. Songhori, Thomas Schneider, and Farinaz Koushanfar. 2018. Chameleon: A hybrid secure computation framework for machine learning applications. In *AsiaCCS*. ACM, 707–721.

[179] Ronald Rivest, Len Adleman, and Michael Dertouzos. 1978. On data banks and privacy homomorphisms. *Found. Secure Comput.* 4, 11 (1978), 165–179.

[180] R. L. Rivest, A. Shamir, and L. Adleman. 1978. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM* 21, 2 (Feb. 1978), 120–126.

[181] Kurt Rohloff and David Bruce Cousins. 2014. A scalable implementation of fully homomorphic encryption built on NTRU. In *Financial Cryptography and Data Security*, Rainer Böhme, Michael Brenner, Tyler Moore, and Matthew Smith (Eds.). Vol. 8438. Springer Berlin, 221–234.

[182] O. Ronneberger, P. Fischer, and T. Brox. 2015. U-Net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention (MICCAI) (LNCS)*, Vol. 9351. Springer, 234–241.

[183] Bita Darvish Rouhani, M. Sadegh Riazi, and Farinaz Koushanfar. 2018. Deepsecure: Scalable provably-secure deep learning. In *DAC*. ACM, 2:1–2:6.

[184] Theo Ryffel, Andrew Trask, Morten Dahl, Bobby Wagner, Jason Mancuso, Daniel Rueckert, and Jonathan Passerat-Palmbach. 2018. A generic framework for privacy preserving deep learning. *CoRR* abs/1811.04017 (2018). arxiv:1811.04017 http://arxiv.org/abs/1811.04017.

[185] Md. Nazmus Sadat, Md. Momin Al Aziz, Noman Mohammed, Feng Chen, Shuang Wang, and Xiaoqian Jiang. 2017-03-07. SAFETY: Secure gwAs in federated environment through a hYbrid solution with Intel SGX and homomorphic encryption. (2017-03-07). https://arxiv.org/abs/1703.02577v1.

[186] Georgios Sakellariou and Anastasios Gounaris. 2019. Homomorphically encrypted k-means on cloud-hosted servers with low client-side load. *Comput.* 101, 12 (2019), 1813–1836.

[187] Zihao Shan, Kui Ren, Marina Blanton, and Cong Wang. 2018. Practical secure computation outsourcing: A survey. *Comput. Surveys* 51, 2 (Feb. 2018), 1–40.

[188] Reza Shokri and Vitaly Shmatikov. 2015. Privacy-preserving deep learning. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS'15)*. ACM, New York, NY, 1310–1321.

[189] Victor Shoup. 2020. *NTL: A Library for doing Number Theory*. Retrieved April 13, 2020 from https://www.shoup.net/ntl/.

[190] Alice Silverberg. 2013. Fully homomorphic encryption for mathematicians. In *Contemporary Mathematics*, Chantal David, Matilde Lalín, and Michelle Manes (Eds.). Vol. 606. American Mathematical Society, 111–123.

[191] N. P. Smart and F. Vercauteren. 2010. Fully homomorphic encryption with relatively small key and ciphertext sizes. In *Public Key Cryptography - PKC 2010 (Lecture Notes in Computer Science)*, Phong Q. Nguyen and David Pointcheval (Eds.). Springer Berlin, 420–443.

[192] N. P. Smart and F. Vercauteren. 2014. Fully homomorphic SIMD operations. *Designs, Codes and Cryptography* 71, 1 (2014), 57–81. Preliminary version in ePrint Report 2011/133.

[193] Damien Stehlé and Ron Steinfeld. 2010. Faster fully homomorphic encryption. In *Advances in Cryptology - ASIACRYPT 2010 (Lecture Notes in Computer Science)*, Masayuki Abe (Ed.). Springer Berlin, 377–394.

[194] Damien Stehlé and Ron Steinfeld. 2011. Making NTRU as secure as worst-case problems over ideal lattices. In *Proceedings of the 30th Annual International Conference on Theory and Applications of Cryptographic Techniques: Advances in Cryptology* (Tallinn, Estonia) *(EUROCRYPT'11)*. 27–47.

[195] X. Sun, P. Zhang, J. K. Liu, J. Yu, and W. Xie. 2018. Private machine learning classification based on fully homomorphic encryption. *IEEE Trans. Emerging Top. Comput.* (2018). Early access.

[196] Sonia M. Suter. 2010. All in the family: Privacy and DNA familial searching. *Harv. J. Law Technol.* 23, 2 (2010), 309–399.

[197] Tsuyoshi Takagi and Thomas Peyrin (Eds.). 2017. In *Proceedings of the 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3–7, 2017, Part I, Advances in Cryptology (ASIACRYPT'17)*. Lecture Notes in Computer Science, Vol. 10624. Springer.

[198] Jaideep Vaidya, Murat Kantarcıoğlu, and Chris Clifton. 2008-07-01. Privacy-preserving Naïve Bayes classification. *VLDB J.* 17, 4 (2008-07-01), 879–898.

[199] V. Vaikuntanathan. 2011. Computing blindfolded: New developments in fully homomorphic encryption. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*. 5–16.

[200] Shuang Wang, Yuchen Zhang, Wenrui Dai, Kristin Lauter, Miran Kim, Yuzhe Tang, Hongkai Xiong, and Xiaoqian Jiang. 2016. HEALER: Homomorphic computation of ExAct Logistic rEgRession for secure rare disease variants analysis in GWAS. *Bioinf. (Oxford, England)* 32, 2 (Jan. 2016), 211–218.

[201] W. Wang, Y. Hu, L. Chen, X. Huang, and B. Sunar. 2012-09. Accelerating fully homomorphic encryption using GPU. In *Proceedings of the 2012 IEEE Conference on High Performance Extreme Computing*. 1–5.

[202] David J. Wu, Tony Feng, Michael Naehrig, and Kristin Lauter. 2016. Privately evaluating decision trees and random forests. In *Proceedings on Privacy Enhancing Technologies* 2016, 4 (2016), 335–355.

[203] Yang Yang, Xindi Huang, Ximeng Liu, Hongju Cheng, Jian Weng, Xiangyang Luo, and Victor Chang. 2019. A comprehensive survey on secure outsourced computation and its applications. *IEEE Access* 7 (2019), 159426–159465.

[204] Andrew C. Yao. 1982. Protocols for secure computations. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science (SFCS'82)*. IEEE Computer Society, Washington, DC, 160–164.

[205] Masaya Yasuda, Takeshi Shimoyama, Jun Kogure, Kazuhiro Yokoyama, and Takeshi Koshiba. 2013. Secure pattern matching using somewhat homomorphic encryption. In *Proceedings of the 2013 ACM Workshop on Cloud Computing Security Workshop* (Berlin, Germany). ACM, 65–76.

[206] Qingchen Zhang, Laurence T. Yang, and Zhikui Chen. 2016. Privacy preserving deep computation model on cloud for big data feature learning. *IEEE Trans. Comput.* 65, 5 (2016), 1351–1362.

[207] Yuchen Zhang, Wenrui Dai, Xiaoqian Jiang, Hongkai Xiong, and Shuang Wang. 2015-12-21. FORESEE: Fully outsourced secuRe gEnome study basEd on homomorphic encryption. *BMC Med. Inf. Decis. Making* 15, 5 (2015-12-21), S5.

[208] R. A. Wagner and M. J. Fischer. 1974. The string-to-string correction problem. *Journal of the Association for Computing Machinery* 21, 1 (Jan. 1974), 168–173.

Received August 2019; revised April 2020; accepted April 2020