Toward Efficient Homomorphic Encryption for Outsourced Databases through Parallel Caching

OLAMIDE TIMOTHY TAWOSE, University of Nevada, Reno, USA
JUN DAI, California State University, Sacramento, USA
LEI YANG, University of Nevada, Reno, USA
DONGFANG ZHAO, University of Nevada, Reno, USA

Many applications deployed to public clouds are concerned about the confidentiality of their outsourced data, such as financial services and electronic patient records. A plausible solution to this problem is homomorphic encryption (HE), which supports certain algebraic operations directly over the ciphertexts. The downside of HE schemes is their significant, if not prohibitive, performance overhead for data-intensive workloads that are very common for outsourced databases, or database-as-a-serve in cloud computing. The objective of this work is to mitigate the performance overhead incurred by the HE module in outsourced databases. To that end, this paper proposes a radix-based parallel caching optimization for accelerating the performance of homomorphic encryption (HE) of outsourced databases in cloud computing. The key insight of the proposed optimization is caching selected radix-ciphertexts in parallel without violating existing security guarantees of the primitive/base HE scheme. We design the radix HE algorithm and apply it to both batch- and incremental-HE schemes; we demonstrate the security of those radix-based HE schemes by showing that the problem of breaking them can be reduced to the problem of breaking their base HE schemes that are known IND-CPA (i.e. Indistinguishability under Chosen-Plaintext Attack). We implement the radix-based schemes as middleware of a 10-node Cassandra cluster on CloudLab; experiments on six workloads show that the proposed caching can boost state-of-the-art HE schemes, such as Paillier and Symmetria, by up to five orders of magnitude.

CCS Concepts: • **Security and privacy → Management and querying of encrypted data**; • **Computing methodologies → Distributed algorithms**.

Additional Key Words and Phrases: cloud computing, distributed computing, encrypted database, homomorphic encryption

ACM Reference Format:
Olamide Timothy Tawose, Jun Dai, Lei Yang, and Dongfang Zhao. 2023. Toward Efficient Homomorphic Encryption for Outsourced Databases through Parallel Caching. *Proc. ACM Manag. Data* 1, 1, Article 66 (May 2023), 23 pages. https://doi.org/10.1145/3588920

1 INTRODUCTION

1.1 Background and Motivation

While increasingly more applications are deployed on the public clouds, one of the biggest challenges lies in confidentiality, especially for those applications that usually touch on sensitive data in the fields such as public health [37], bioinformatics [65], and financial services [35]. Although various encryption schemes (e.g., AES [47], RSA [54]) can be applied before the data are sent to the cloud,

Authors' addresses: Olamide Timothy Tawose, University of Nevada, Reno, USA, otawose@nevada.unr.edu; Jun Dai, California State University, Sacramento, USA, jun.dai@csus.edu; Lei Yang, University of Nevada, Reno, USA, leiy@unr.edu; Dongfang Zhao, University of Nevada, Reno, USA, dzhao@unr.edu.

2836-6573/2023/5-ART66 $15.00
https://doi.org/10.1145/3588920

it would defeat the purpose of cloud computing if the users must download and decrypt the encrypted data for processing: the cloud in this case works merely as remote storage with no *computing* functionalities. One plausible solution to the above confidentiality problem is adopting specific encryption schemes such that the ciphertexts stored on the cloud can perform certain computations, which are known as *homomorphic encryption* (HE). Although most HE schemes support only primitive arithmetic operations such as addition and multiplication, it turns out that many commonly-used operations (e.g., comparison) can be constructed on top of circuits of additions and multiplications [28]. However, a scheme supporting both addition and multiplication over ciphertexts, namely *fully homomorphic encryption* (FHE), usually incurs a much higher performance overhead than (partial) HE, or PHE schemes by orders of magnitude. These PHE schemes can be categorized into two types depending on how the key is distributed.

The first type of PHE schemes, e.g., Symmetria [57], is implemented as a symmetric operation for the scenarios where a secret key can be securely shared among parties. In order to ensure high security, Symmetria introduces a randomization component in the ciphertext that keeps growing, which might cause significant performance overhead. Seabed [49] is another symmetric PHE cryptosystem but only supports primitive additions (e.g., no subtraction or negation).

The second type of PHE scheme, e.g., Paillier [48], is implemented as an asymmetric operation with a pair of public and private keys. An asymmetric scheme employs hard mathematical problems in number theory and group theory to safely distribute the public keys, rendering it orders of magnitude slower than a symmetric scheme. Although a hybrid scheme can be used with symmetric key for encryption and asymmetric operation for key distribution, key distribution is needed per session in database-as-a-service (DaaS), implying that asymmetric operations would be invoked routinely.

Although PHE is much more efficient than FHE, PHE still cannot meet the performance requirements for data-intensive workloads in DaaS. As we will show later in this paper (§**??**), the state-of-the-art PHE scheme, Symmetria [57], can only encrypt data at a rate of 3 Mbps—much lower than the commodity network bandwidth (cf. Fig. 11) that is in the order of tens of Mbps or even Gbps. That being said, the performance bottleneck of data-intensive applications, such as video analysis [21, 32], lies at the encryption subsystem.

Our long-term goal is to improve the performance of homomorphic encryption applied to large volumes of outsourced data; this paper attains the above goal, as the first step, by proposing a new caching approach to reduce the computational overhead in both symmetric- and asymmetric-PHE schemes for outsourced databases or DaaS in cloud computing. It is our hope that data-intensive applications would better exploit the high security and low overhead of PHE schemes by incorporating the proposed technique.

1.2 Contributions

The key insights of our proposed caching technique include: (i) precomputing and caching some homomorphic ciphertexts before encrypting the large volume of plaintexts; (ii) expanding a requested plaintext into a summation of additive radix entries; (iii) constructing the ciphertexts with randomized homomorphic addition, without touching on encryption primitives; and (iv) enabling incremental encryption based on the extended entries of the cached ciphertexts.

Formally, we claim the following technical contributions.

- Firstly, we propose an algorithm to reconstruct the ciphertext using radixes in the context of homomorphic encryption (HE). We name the new algorithm *radix homomorphic encryption*, or RHE. We conduct a thorough analysis of parametrization for RHE. (§3)

- Secondly, we design a full-fledged protocol called Radix-additive caching for homomorphic encryption (Rache), which adopts RHE to securely encrypt a large volume of data. We articulate the security goal, threat model, and security assumptions, under which the RHE protocol is proven secure. (§4)
- Thirdly, we extend Rache into an incremental protocol that allows for efficient homomorphic encryption of data streams. We also demonstrate the provable security of this incremental protocol. (§5)

2 PRELIMINARIES AND RELATED WORK

2.1 Confidentiality of Outsourced Data

We review four important techniques to ensure the confidentiality of outsourced data: encrypted storage, encrypted tuples, encrypted fields, and secure multi-party computation.

Encrypted Storage. The database instance from the cloud vendor is considered as storage of encrypted data and the client is responsible for nontrivial queries. This solution is viable only if (i) the relations touched on by the query are small enough that the network overhead of transmitting those relations is acceptable, and (ii) the user has the capability (both computation and storage) to execute the query locally. We stress that this solution might defeat the purpose of outsourcing the database service to the cloud.

Encrypted Tuples. Every tuple of the original relation R is encrypted into a ciphertext that is stored in column T of a new relation R^s. For each attribute A_i in R, there is a corresponding attribute A_i^s in R^s, whose value is the index of $R.A_i$. The index is usually assigned by a random integer based on some partitioning criteria and can be retrieved with the metadata stored on the *client*, i.e., the user's local node. As a result, the schema stored at the cloud provider is $R^s(T, A_1^s, \ldots, A_i^s, \ldots)$. When the user submits a query Q, the client splits Q into two subqueries Q_s and Q_c. Q_s serves as a filter to eliminate those unqualified tuples based on the indices in R^s and transmits the qualified tuples (in ciphertexts) to the client. Q_c then ensures that those false-positive tuples are eliminated after the encrypted tuples are decrypted using the secret key presumably stored on the client. This approach involves both the client (i.e., the user) and the server (i.e., the cloud provider) when completing a query, often referred to as *information hiding* approaches [30].

Encrypted Fields. The third approach aims to minimize the involvement of clients when processing the query over the encrypted data stored at the cloud vendor. The idea is to encrypt the relations at a finer granularity—each attribute of a relation is separately encrypted. The key challenge of this approach lies in its expressiveness, e.g., how to apply arithmetic or string operations over the encrypted fields. While fully homomorphic encryption (FHE) [28] can support a large set of computing problems, the performance of current FHE implementations cannot meet the requirements of practical database systems [50, 51]. An alternative solution is partially homomorphic encryption (PHE) schemes [26, 48], which are orders of magnitude faster than FHE but only support a single algebraic operation. Traditional PHE schemes are designed for public-key (asymmetric) encryption, which is desirable for straightforward key distribution over insecure channels but significantly more expensive than secret-key (symmetric) encryption. However, in the context of DaaS, the user usually serves as both the sender and the receiver and there is no need to distribute the key. To this end, symmetric (partially) homomorphic encryption (SHE), was proposed [49, 57].

Secure Multi-Party Computation (MPC). In addition to HE-based methods, another widely-used technique for data privacy is secure *multi-party computation* (MPC), which originated from [64] and has been mostly built upon oblivious transfer [29, 39], threshold homomorphic encryption [17, 19], and secret sharing [52, 59]. MPC has been applied in multiple machine learning frameworks, such as DeepSecure [55], SecureML [45], and ABY [23].

2.2 Homomorphic Encryption

The term *homomorphic* or *homomorphism* originates from group theory, which depicts such a function that can be applied either before or after the operations conducted in the domain or the image. Formally, we have the following mathematical definition.

Definition 1 (Homomorphism). *Given two groups* $(F, \oplus)$ *and* $(G, \otimes)$*, a function* $h : F \rightarrow G$ *is called a homomorphism if* $h(f_1 \oplus f_2) = h(f_1) \otimes h(f_2), \forall f_1, f_2 \in F$.

There are many examples of homomorphism. The following is a simple one we have seen in basic mathematics.

Example 1. We can define two groups $F = (\mathbb{R}, +)$ and $G = (\mathbb{R}^+, \times)$ with the regular arithmetic operations, where $\mathbb{R}$ and $\mathbb{R}^+$ denotes real numbers and positive real numbers, respectively. Moreover, we define a function $h(x) = 2^x$, where $x \in \mathbb{R}$. Evidently, the following equation holds: $h(a + b) = 2^{a+b} = 2^a \times 2^b = h(a) \times h(b)$. △

Homomorphic encryption (HE) is a specific type of encryption where certain operations between operands can be performed directly on the ciphertexts in the sense that the result can be decrypted into the same value as if the operations were applied to the plaintexts. If we connect HE to the group-theoretical definition of homomorphism, the encryption function can be thought of the homomorphism, the set of plaintexts as the domain of the homomorphism, and the set of ciphertexts as the image of the homomorphism.

An HE scheme that supports the arithmetic addition over the ciphertexts is called *additive*. That is to say, we can define an addition operation $\oplus$ between two ciphertexts, say $enc(x)$ and $enc(y)$ encrypted by function $enc(\cdot)$, such that

$$dec(enc(x) \oplus enc(y)) = x + y, \tag{1}$$

where $dec(\cdot)$ denotes the decryption function corresponding to $enc(\cdot)$. It should be noted that Eq. (1) does not necessarily imply a mathematical homomorphism as defined in Def. 1; that is, we generally do not require $enc(x) \oplus enc(y) = enc(x+y)$. This is more of a practical security consideration rather than a mathematical one: *randomness* is always required for cryptographic schemes in practice (e.g., to defeat chosen-plaintext cryptanalysis), and therefore, repeated encryption of the same plaintext should look different, i.e., random.

Many encryption schemes in the literature are homomorphic, such as Symmetria [57] and Paillier [48]. Symmetria is a symmetric encryption scheme, meaning that a single secret key is used to both encrypt and decrypt the messages. By contrast, Paillier is asymmetric, where a pair of public and private keys are used for encryption and decryption, respectively. Due to the expensive arithmetical operations performed by the asymmetric encryption, Paillier is orders of magnitude slower than Symmetria. However, Paillier is particularly useful when there is no secure channel to share the secret key among parties.

An HE scheme that supports multiplication is called *multiplicative*. Symmetria [57] is also multiplicative using a distinct scheme than the one for addition. Other well-known multiplicative HE schemes include RSA [54] and ElGamal [26]. A multiplicative HE scheme ensures the following equality,

$$dec(enc(a) \otimes enc(b)) = a \times b,$$

where $\otimes$ denotes the multiplication defined over ciphertexts.

An HE scheme that supports both addition and multiplication is called a *fully HE (FHE) scheme*. This requirement should not be confused with specific addition and multiplication parameters, such as Symmetria [57] and NTRU [34]. That is, the addition and multiplication must be supported

homomorphically under exactly the same scheme:

$$\begin{cases} dec(enc(a) \oplus enc(b)) = a + b \\ dec(enc(a) \otimes enc(b)) = a \times b \end{cases}$$

It turned out to be extremely hard to construct FHE schemes until Gentry [28] demonstrated that such a scheme can be constructed using lattice theory. Indeed, multiple implementations are available today, such as BGV [31], BFV [27], and CKKS [15]. Nonetheless, the performance overhead of FHE implementations still cannot meet the requirement of many real-world applications, especially those data-intensive applications. Two popular open-source libraries of FHE schemes are IBM HElib [33] and Microsoft SEAL [58]. Some more recent implementations are optimized for machine learning and vector computation, such as TenSEAL [11].

A lot of research efforts have been put to optimize the performance of HE schemes. For instance, hardware-based optimization [24, 53, 56] has been heavily exploited. A recent article argues that the current performance bottleneck of HE lies in the memory wall [22]. The notion of *incremental cryptography* was first formalized in 1990s [9, 10], mainly from a theoretical perspective. More recent work on incremental encryption schemes can be found in [5, 40, 44]. Incremental encryption recently draws a lot of research interests for efficient data encoding in the resource-constraint contexts such as mobile computing [12, 38, 63].

2.3 Provable Security

When employing an encryption scheme in an application, it is highly desirable to demonstrate its security in a provable manner. Formally, we need to clearly identify the following three important pieces for provable security of a given encryption scheme: security goal, threat model, and assumptions. The security goal spells out the desired effect when the application is under attack; the threat model articulates what an adversary can do with the attack, such as what information of the plaintext/ciphertext can be collected and the resource/time limitation of the attack; the assumption lists the presumed specifics of the subsystems or components of the cryptographic scheme, which is usually an important building block for the security proof, e.g., reduction. The security goal and threat model are usually called *security definition* collectively.

One well-accepted security definition with a good balance between efficiency and security is that the adversary is able to launch a *chosen-plaintext attack* (CPA), defined as follows.

Definition 2 (Chosen-Plaintext Attack). *Given a security parameter n, i.e., the bitstring length of the key, an adversary can obtain up to $poly(n)$ of plaintext-ciphertext pairs (m, c), where m is arbitrarily chosen by the adversary and $poly(\cdot)$ is a polynomial function on n. With such information, the adversary tries to decrypt a c' that is not included in the polynomial number of known ciphertexts.*

The polynomial requirement is only for practical reasons, as we usually assume that the adversary should only be able to run a polynomial algorithm without unlimited resources. Accordingly, we want to design encryption schemes that are *CPA secure*: even if the adversary $\mathcal{A}$ can obtain those extra pieces of information, $\mathcal{A}$ should not be able to decode the ciphertext better than a random guess up to a very small probability. To quantify the degree of this small probability, *negligible function* is defined as below.

Definition 3. *A function $\mu(\cdot)$ is called negligible if for all polynomials $poly(n)$ the inequality $\mu(n) < \frac{1}{poly(n)}$ holds for sufficiently large n's.*

For completeness, we list the following lemmas for negligible functions that will be used in later sections. We state them without the proofs, which can be found in introductory cryptography or complexity theory texts.

Lemma 1 (Summation of two negligible functions is a negligible function). *Let $\mu_1(n)$ and $\mu_2(n)$ be both negligible functions. Then $\mu(n)$ is a negligible function that is defined as $\mu(n) \stackrel{def}{=} \mu_1(n) + \mu_2(n)$.*

Lemma 2 (Quotient of a polynomial function over an exponential function is a negligible function). *$\frac{poly(n)}{2^n}$ is a negligible function. That is, $\exists N \in \mathbb{N},\ \forall n \geq N :\ \frac{poly(n)}{2^n} < \frac{1}{poly(n)}$, where $\mathbb{N}$ denotes natural numbers.*

3 RHE: RADIX HOMOMORPHIC ENCRYPTION

3.1 Overview

Our key observation is that although a HE encryption operation is costly, the algebraic operation over the ciphertexts is comparatively cheaper. While the concrete performance gap is dependent on how a specific HE scheme is implemented and to which data the scheme is applied, we exemplify such gaps in our experiments: Figure 2 in §6.4.1 shows that the addition of two ciphertexts takes less than 1% time than the encryption of a plaintext in Paillier [48]. With that said, if we convert the expensive encryption operation of a given plaintext into an equivalent set of algebraic operations over existing (i.e., cached) ciphertexts, we may obtain a performance edge. There are two questions, however, in this idea.

First, which ciphertexts should we cache? Evidently, we can always cache only $he(1)$ and then compute $he(m)$ of n-bit plaintext m with $\oplus_{i=1}^{m} he(1)$. However, the accumulative overhead caused by a lot of homomorphic additions would at some point outweigh the encryption cost due to $O(2^n)$ additions. We propose to only cache a set of selective ciphertexts; specifically, let r be a radix (and we will show how to pick r in §3.3), then the ciphertexts of r-power series will be pre-computed: $he(r^i)$, where $r^i \leq 2^n$. By doing so, the target ciphertext will be constructed through $O(n)$ additions. It should be noted that the target ciphertext at this point is merely a deterministic ciphertext with no security.

Second, how to ensure the randomness of the ciphertext? Randomness must be added to the ciphertext to achieve a practical security level, e.g., anti- chosen-plaintext attack (CPA). Informally, the randomness must be probabilistic small, which usually takes the form of picking a piece of data out of an exponential space. From the above discussion, we have n cached ciphertexts; we will use these ciphertexts as ingredients to add a random $he(0)$ to the deterministic ciphertext. The random $he(0)$ is constructed by working on every radix-power r^i: randomly adding radix-power $he(r^i)$ and if so, then subtracting r times of $he(r^{i-1})$. Overall, there are $O(rn)$ homomorphic additions that will result in $he(0)$, which is randomly selected from an exponential space $O(2^n)$. The above radix-wise homomorphic additions can be *parallelized* with the many-core architecture in modern CPUs. Before formalizing the algorithm, we illustrate the idea of Rache in an oversimplified scenario Example 2.

Example 2. Let's try to encrypt number 100 using the Rache encryption scheme. For the sake of simplicity, let $r = 2$, $Ctxt[]$ be the list of cached r-power ciphertexts, and $\oplus$ be the addition on the ciphertexts. Obviously, $100 = 64 + 32 + 4 = r^6 + r^5 + r^2$. Therefore, $Rache(100) = Rache(r^6) \oplus Rache(r^5) \oplus Rache(r^2) = Ctxt[6] \oplus Ctxt[5] \oplus Ctxt[2]$. That is, instead of calculating $Rache(100)$ using sophisticated number-theoretical rules, we can simply construct $Rache(100)$ through two homomorphic additions of cached ciphertexts, which are much simpler and faster. △

3.2 Algorithm

Algorithm 1 formalizes the radix-based procedure. Let n denote the security parameter of the underlying PHE scheme, i.e., the bitstring length of the key k that is usually generated by $k \leftarrow$

Algorithm 1: RHE: Radix Homomorphic Encryption

Input: An array of plaintexts $Ptxt[]$, each being a padded n-bitstring; A homomorphic encryption function $he(\cdot)$ s.t. $\forall a_i \in Ptxt[], \bigoplus_i he(a_i) = he(\sum_i a_i)$; Radix r;

Output: An array of ciphertexts $Ctxt[]$ such that $\forall i$, $he^{-1}(Ctxt[i]) == Ptxt[i]$, where he^{-1} denotes the decryption function;

// Initialization
1 $m \coloneqq 2^n - 1$
2 **for** $i = 0; i <= \lfloor \log_r m \rfloor; i++$ **do**
3 $\quad radixes[i] \coloneqq he(r^i)$
4 **end**
5 $radixes\left[\lfloor \log_r m \rfloor + 1\right] \coloneqq he(0)$

// Encoding
6 **for** *i = 0; i < Ptxt.size(); i++* **do**
7 $\quad$ **for** $j = 0; j <= \lfloor \log_r m \rfloor; j++$ **do**
8 $\quad\quad idx[j] \coloneqq (Ptxt[i] \,/\, r^j) \,\%\, r$
9 $\quad$ **end**
$\quad$ // $Ptxt[i] = \sum_j idx[j] \times r^j$
10 $\quad Ctxt[i] \coloneqq \bigoplus_{k=0}^{\lfloor \log_r m \rfloor} \bigoplus_{j=1}^{idx[k]} radixes[k]$

$\quad$ // Randomization
11 $\quad$ isSwap $\xleftarrow{\$} \{0, 1\}$
12 $\quad$ **if** *1 == isSwap* **then**
13 $\quad\quad Ctxt[i] \coloneqq Ctxt[i] \oplus radixes\left[\lfloor \log_r m \rfloor + 1\right]$
14 $\quad$ **end**
15 $\quad$ **for** $j = 1; j < \lfloor \log_r m \rfloor; j++$ **do**
16 $\quad\quad$ isSwap $\xleftarrow{\$} \{0, 1\}$
17 $\quad\quad$ **if** *1 == isSwap* **then**
18 $\quad\quad\quad Ctxt[i] \coloneqq Ctxt[i] \oplus radixes\,[j]$
19 $\quad\quad\quad$ **for** $k = 0; k < r; k++$ **do**
20 $\quad\quad\quad\quad Ctxt[i] \coloneqq Ctxt[i] \ominus radixes\,[j-1]$
21 $\quad\quad\quad$ **end**
22 $\quad\quad$ **end**
23 $\quad$ **end**
24 **end**

$Gen(1^n)$, where $Gen()$ is a pseudorandom generator. For the sake of clarity, we assume that the original plaintext value can be converted into a bitstring of length n or smaller; this should not be a technical limitation in practice, as we can always split a large value into multiple *blocks* of n-bits, each of which is encrypted with randomization. In other words, we construct a *block cipher* using Algorithm 1. If there are identical blocks, the security is nonetheless guaranteed because Algorithm 1 is randomized (Lines 11 and 16).

Lines 1–5 initialize the reused entries of the integral powers of radix r for future construction of ciphertexts. Specifically, Line 5 precomputes the homomorphic encryption of plaintext 0, which will be used for the base case during the randomization (Lines 11–14). Lines 6–24 encode the plaintexts, each of which is computed directly over the encoded radixes that are initialized at the beginning of the protocol. For each plaintext, Lines 11–14 randomize the radix summation of ciphertexts such that repeated plaintexts will result in distinct ciphertexts. The idea of the randomization is to iterate every precomputed ciphertext $radixes[i]$ and randomly add it to the ciphertext; if the addition happens, we subtract ciphertext $radixes[i-1]$ repeatedly r times.

The correctness of Algorithm 1 can be verified by straightforward algebraic computation. We skip the full computation here due to space constraints.

3.3 Parameterization

3.3.1 Heuristic Radix Selection. This section will discuss heuristic methods to decide the radix value r in practice. The discussion will remain mostly informal as there are unlimited factors in real-world applications; a more rigorous approach to be presented in the next section (§3.3.2) focuses on the worst-case scenario, where we can make more assumptions of the factors that allow us to conduct a more quantitative analysis.

In practice, the initialization cost can be thought of a constant cost because it can be amortized by a large number of follow-up computations. As a result, the key trade-off lies at the cost of $\oplus$'s and that of encrypting the plaintext message m. Let g denote the ratio of computational costs of ciphertext addition over homomorphic encryption:

$$g \stackrel{\text{def}}{=} \frac{Time(Ctxt[i] \oplus Ctxt[j])}{Time(Rache(m))},$$

where $Time()$ denotes the time function and $Ctxt[]$ denotes the list of cached ciphertexts. Evidently, the bottom line is to ensure the average cost of $\sum_{k \in K} Ctxt[k]$ for a requested ciphertext is lower than that of $Rache(m)$, or $g|K| < 1$, because otherwise there is no performance improvement from caching the ciphertext. In a specific HE scheme, g can be estimated using some benchmarks; for example, Figure 2 shows that ciphertext addition is two orders of magnitude faster than encryption in Paillier: $g = 0.01$. This implies that, on average, $|K|$ should be smaller than 100. With radix r, the maximal possible upper bound would be r^{100}. Therefore, we need to pick r to ensure that the maximal value of the plaintext set is smaller than r^{100} in Paillier. If M is the maximal message, then we require $M < r^{100}$ or $r > M^{\frac{1}{100}}$. If the plaintext space is a set of 256-bit strings, then $M = 2^{256}$ and $r > (2^{256})^{\frac{1}{100}} > 2^{2.56} \approx 5.9$. Therefore, r can be set to 6.

3.3.2 Optimal Radix in the Worst Case. This section will investigate the optimal radix in the worst case. Let $m \geq 2$ denote the maximal value to be encrypted in the application. Let $r \geq 2$ denote the radix or base of the homomorphic encryption. Obviously, given an arbitrary number x, where $0 \leq x \leq m$, there are $k+1$ radix entries: $r^0, r^1, \ldots, r^k$, where $k = \lfloor \log_r^m \rfloor$. Let $0 \leq \kappa \leq k$. In the worst case, each r^κ radix-entry incurs $r-2$ times of homomorphic addition, i.e., when computing $(r-1) \cdot x^\kappa$. Since one more homomorphic addition needs to be taken for the summation of each radix, the overall times of homomorphic addition, in the worst case when m is one less than the next integral power of r (i.e., $\lfloor \log_r^m \rfloor = \log_r^{m+1} - 1$), is

$$f(r) = (r-2)(k+1) + k = (r-1)\log_r^{m+1} - 1.$$

Our goal is therefore to find out the optimal r that minimizes $f(r)$. This can be achieved by calculating the first-order and second-order derivatives of $f(r)$. We skip the detailed computation here for the sake of space; the following elementary calculus and algebra sketch the procedure to

derive that $r = 2$ leads to the minimum number of homomorphic additions in the worst case.

$$f'(r) = \frac{d}{dr} f(r) = \ln(m+1) \cdot (\ln r)^{-2} \cdot r^{-1} \cdot (r \ln r - r + 1) .$$

The stationary point is therefore the solution to $g(r) = f'(r) = 0$, which yields $r = 1$. Since we require $r \geq 2$, we need to find another qualified radix. First, we calculate $g(2)$:

$$g(2) = 2 \ln 2 - 2 + 1 \geq 2 \times 0.69 - 1 > 0.$$

Then, let $r \geq 3$, therefore $\ln r > 1$, which yields:

$$g(r)|_{r \geq 3} = r \ln r - r + 1 = r(\ln r - 1) + 1 > 0.$$

Note that by definition, the following equation holds:

$$f'(r) = \ln(m+1) \cdot (\ln r)^{-2} \cdot r^{-1} \cdot g(r).$$

If we assume $m \geq 2$, then $\ln(m+1) > 0$. Both $(\ln r)^{-2}$ and r^{-1} factors are obviously positive. Therefore, $f'(r)$ is always positive, meaning that $f(r)$ is a monotonically increasing function. It follows that the minimal qualified radix $r = 2$ leads to the minimum number of homomorphic additions.

4 RACHE: RADIX-ADDITIVE CACHING FOR HOMOMORPHIC ENCRYPTION

4.1 Security Definitions and Assumptions

The security goal of our target outsourced databases is *computational secrecy*, which implies that any adversary cannot differentiate between the encrypted data and a random string with a probability significantly larger than 50%, coined as *indistinguishability*. This means that when an adversary is given a ciphertext, he or she cannot do much better than randomly guessing the corresponding plaintext with reasonable resources. Technically, the degree of closeness is quantified by a negligible function; we refer readers to §2.3 for more technical details. Indeed, if we want to be strict on the 50% requirement, then it is called *perfect secrecy* (information-theoretical secrecy), which is beyond the scope of this paper.

In the context of computational secrecy, we assume that the adversary cannot obtain unlimited computing resources and can only run probabilistic polynomial-time (PPT) algorithms. We also assume that the adversary can launch a chosen-plaintext attack (CPA), meaning that the adversary can obtain $poly(n)$ arbitrary pairs of (plaintext, ciphertext), where n denotes the security parameter and $poly(\cdot)$ denotes a polynomial function. We call a scheme *IND-CPA* if it exhibits *indistinguishability under CPA*.

Finally, we assume the primitive homomorphic encryption schemes, into which radix-caching is integrated, are IND-CPA. This is technically required because we will need this assumption to prove that Rache is IND-CPA. We call those original homomorphic encryption schemes *base schemes*, whose encryption function must not be deterministic—a necessary (but not sufficient) requirement for any scheme to be IND-CPA. In practice, many existing base schemes have been proven IND-CPA; for instance, both base schemes (Paillier [48], Symmetria [57]) used by Rache are IND-CPA.

4.2 Scheme Description

We start with integrating RHE into a symmetric homomorphic encryption scheme. We denote a quadruple

$$\Pi = (Gen, Enc, Dec, \oplus)$$

as a symmetric homomorphic encryption, where *Gen* denotes the function to generate a random key k of length n, *Enc* denotes the encryption function parameterized with k to encode a plaintext

m into a ciphertext c, Dec denotes a decryption function with parameter k to decode c back into m, and $\oplus$ denotes the additive operation over two ciphertexts $Enc(m_1)$ and $Enc(m_2)$ such that

$$Dec_k(Enc_k(m_1) \oplus Enc_k(m_2)) = m_1 + m_2.$$

A symmetric Rache scheme built upon Π is a triple

$$\widetilde{\Pi}(Gen, RHE, Dec), \tag{2}$$

where RHE denotes the procedure defined in Algorithm 1. Note that $RHE(m)$ is equal to $Enc(m)$ up to $O(n)$ random ciphertexts of zeros (out of the overall r^n parameter space):

$$RHE_k(m) \equiv Enc_k(m) \quad \left(\bigoplus_I Enc_k(0)\right),$$

where I is an index set whose cardinality is a polynomial on n. By definition, the equality $Dec_k(RHE_k(m)) = m$ holds.

An asymmetric Rache scheme can be similarly built upon an asymmetric base HE scheme, except for the keys for Enc and Dec: two random keys—public key pk and private key sk—are generated by Gen. For instance, we now require the following equality holds when RHE is built upon an asymmetric base scheme:

$$Dec_{sk}(RHE_{pk}(m_1) \oplus RHE_{pk}(m_2)) = m_1 + m_2.$$

Because RHE touches on only the encryption function, there is no need to differentiate between symmetric and asymmetric base schemes. Therefore, in the following discussion, we assume the underlying base scheme is symmetric for more succinct notations.

4.3 Provable Security

This subsection proves that the Rache scheme is IND-CPA. We first explain the intuition why Rache is CPA-secure and then give the formal proof.

Recall that Rache precomputes and caches $\log_r 2^n$ radix entries. If we assume the system picks the optimal $r = 2$ in the worst case, then the scheme will simply cache n radix entries. Therefore, those ciphertexts cached by Rache should not significantly help the adversary—who presumably runs a probabilistic polynomial-time (PPT) Turing machine—as the overall space is exponential (Lines 11–23, Algorithm 1).

Technically, we want to *reduce* the problem of breaking the base homomorphic encryption scheme Π to the problem of breaking its Rache extension $\widetilde{\Pi}$. That is, if a PPT adversary $\mathcal{A}$ takes an algorithm alg to break $\widetilde{\Pi}$, then $\mathcal{A}$ can efficiently (i.e., in polynomial time) construct another algorithm alg' that calls alg as a subroutine to break Π as well (simulating alg' with alg). However, we assume that the base scheme is IND-CPA, so the above cannot happen—leading to a contradiction. We formalize the above in the following proposition.

Proposition 1. *If HE scheme Π is IND-CPA, then its Rache-extension $\widetilde{\Pi}$ defined in Eq.*(2) *is IND-CPA.*

Proof. Let $CPA_X^{\mathcal{A}}$ denote the indistinguishability experiment with scheme X. The probability for $\mathcal{A}$ to successfully break Π and $\widetilde{\Pi}$ are $Pr\left[CPA_{\Pi}^{\mathcal{A}} = 1\right]$ and $Pr\left[CPA_{\widetilde{\Pi}}^{\mathcal{A}} = 1\right]$, respectively. By assumption, the following inequality holds:

$$Pr\left[CPA_{\Pi}^{\mathcal{A}} = 1\right] \leq \frac{1}{2} + \epsilon, \tag{3}$$

where ϵ is a negligible probability. By comparing Π and $\widetilde{\Pi}$, the latter yields n additional pairs of plaintexts and ciphertexts out of the total 2^n possible pairs in the worst case. Therefore, the

following inequality holds:

$$Pr\left[CPA^{\mathcal{A}}_{\widetilde{\Pi}} = 1\right] - Pr\left[CPA^{\mathcal{A}}_{\Pi} = 1\right] \leq \frac{poly(n)}{2^n}. \tag{4}$$

Combining Eq. (3) and Eq. (4) yields the following inequality:

$$Pr\left[CPA^{\mathcal{A}}_{\widetilde{\Pi}} = 1\right] \leq \frac{1}{2} + \epsilon + \frac{poly(n)}{2^n}.$$

Now, we only need to show that the summation of the last two terms, $\epsilon + \frac{poly(n)}{2^n}$, is negligible. According to Lemma 1 and Lemma 2 (§2), this is indeed the case. Therefore, the probability for the adversary $\mathcal{A}$ to succeed in the $CPA^{\mathcal{A}}_{\widetilde{\Pi}}$ experiment is only negligibly higher than $\frac{1}{2}$, proving that Rache is IND-CPA, as claimed. □

5 INCREMENTAL RACHE

5.1 Overview

While Rache can effectively precompute and cache those selected ciphertexts given an upper bound of the plaintexts, the principle cannot be applied to data streams where the maximal value is unknown a priori. To that end, we propose to dynamically precompute those r powers when a newly seen maximum is observed. The key idea is straightforward: whenever the cipher encounters a plaintext that is *significantly* larger than the largest (cached) value, we submit a request to expand the list of cached values by adding *a few* precomputed ciphertexts that are closer to the new large plaintext. The remaining job is then to quantify the meaning of *significantly* and *a few*, which will be elaborated on in the remainder of this section. Before the formal discussion, we illustrate the high-level idea of incremental Rache by extending Example 2 into the following Example 3: recall that we have a good set of cached ciphertexts now for up to r^6, where $r = 2$.

Example 3. Now let's assume that a new value 200 is being encrypted. In theory, we could compute $Rache(200) = Rache(r^6) \oplus Rache(r^6) \oplus Rache(r^6) \oplus Rache(r^3)$; however, this naive approach would not scale: at some point the cost of many $\oplus$'s would outweigh that of the original encryption. An alternative is to precompute some larger ciphertexts and append them into $Ctxt[]$: $Ctxt[7] = Rache(r^7) = Rache(128)$. As a result, we can compute $Rache(200) = Rache(r^7) \oplus Rache(r^6) \oplus Rache(r^3) = Ctxt[7] \oplus Ctxt[6] \oplus Ctxt[3]$, which saves one $\oplus$ in this example; but for larger plaintexts, the saving would look much more significant. △

5.2 Definitions and Notations

We begin by defining two important building blocks of incremental Rache, *pivot* and *nuance*.

Definition 4 (Pivot). *A pivot in incremental Rache is one plaintext whose ciphertext is precomputed and cached.*

By definition, the preimage of every entry of the $radixes[]$ array discussed in Alg. 1 is a pivot. However, the converse is not true in general for incremental Rache: we might optionally choose to cache more "important" ciphertexts in addition to those in $radixes[]$.

Definition 5 (Nuance). *A nuance in incremental Rache is a pair $(\xi, RHE(\xi))$, where ξ is a plaintext and $RHE(\xi)$ is the Rache ciphertext of ξ.*

We use $p = \Theta(poly(n))$ to denote the asymptotic number of pivots that will be preprocessed. Common values for p include n^c, $1 \leq c \leq 5$ [6]. Similarly, we use $d = \Theta(poly(n))$ to denote the asymptotic number of nuances that will be cached. We assume the plaintext can be encoded with the security parameter n. Again, we can pad shorter ones or break longer ones into blocks to ensure the aligned lengths. We denote by m the number of plaintexts (thus $m \leq 2^n$).

5.3 Scheme Description

To make it more concrete, we slightly extend the triple expression of an HE scheme into a quintuple by considering the spaces of plaintexts and ciphertexts. Formally, we denote by quintuple $\Pi = (\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$ an HE scheme, where $\mathcal{P}$ is the set of plaintexts, $\mathcal{C}$ is the set of ciphertexts, $\mathcal{K}$ is the set of secret keys (for succinctness assuming the scheme is symmetric), $\mathcal{E}$ and $\mathcal{D}$ are sets of keyed encryption and decryption functions that satisfy the following predicate,

$$\forall K \in \mathcal{K},\ \forall x \in \mathcal{P},\ \exists e_K \in \mathcal{E},\ \exists d_K \in \mathcal{D},\ d_K(e_K(x)) = x.$$

An incremental Rache is a septuple extended from Π:

$$\widetilde{\Pi} = (\mathcal{P}, \mathcal{C}, \mathcal{K}, \widetilde{\mathcal{E}}, \mathcal{D}, \mathcal{B}, \mathcal{N}), \tag{5}$$

where $\mathcal{B}$ is a function from plaintexts to the set of the indexed pivots, $\mathcal{N}$ is a nuance function from a polynomial number of plaintexts to their ciphertexts, and $\widetilde{\mathcal{E}}$ is the set of keyed functions for incremental encryption. While $\mathcal{P}$, $\mathcal{C}$, $\mathcal{K}$, and $\mathcal{D}$ inherit the same semantics from Π, others need more explanation. We elaborate on $\mathcal{B}$, $\mathcal{N}$, and $\widetilde{\mathcal{E}}$ as follows.

We start with $\mathcal{B}$. Recall that we assume the size of the current data set is m, implying its index $m-1$ (counting from 0). The newly added data point, therefore, has index m. The value of function $\mathcal{B}(m)$ is calculated as the encryption of the largest pivot that is smaller than the new data point. If we sort the pivots P_i's in an increasing order ($P_0 \le P_1 \le P_2 \le \dots$), then we can formally define $\mathcal{B}$ as follows:

$$\mathcal{B}(m) \stackrel{\text{def}}{=} e_K(P_i),$$

where $P_i \le m < P_{i+1}$ and i denotes the pivot index.

The nuance function $\mathcal{N}$ maps a logarithmic distance from P_i to its encryption:

$$\mathcal{N} : \left[1, \left\lceil \frac{P_{i+1} - P_i}{2} \right\rceil\right] \to \mathcal{C},$$
$$\xi \mapsto e_K(\xi),$$

where $\xi \in \left\{2^j : \forall j \in \mathbb{N},\ 2^j \le \left\lceil \frac{P_{i+1}-P_i}{2} \right\rceil\right\}$ and $e_K \in \widetilde{\mathcal{E}}$. By convention, we use $dom(\mathcal{N})$ to denote the *domain* of function $\mathcal{N}$, i.e., the set of nuance plaintexts between two adjacent pivots. It is evident to see that the new data point, denoted $Ptxt[m]$, can be calculated as follows:

$$Ptxt[m] = P_i + \sum_{j=1}^{|dom(\mathcal{N})|} \{0,1\} \times 2^j.$$

We are now ready to define $\widetilde{\mathcal{E}}$. Let $RHE_K^{inc} \in \widetilde{\mathcal{E}}$ with key K, then an incremental encryption function in $\widetilde{\mathcal{E}}$ is defined as follows:

$$\begin{aligned} RHE_K^{inc}(m) &\stackrel{\text{def}}{=} e_K\left(P_i + \sum_{j=1}^{|dom(\mathcal{N})|} \{0,1\} \times 2^j\right) \\ &= e_K(P_i) \oplus e_K\left(\sum_{j=1}^{|dom(\mathcal{N})|} \{0,1\} \times 2^j\right) \\ &= e_K(P_i) \oplus \bigoplus_{j=1}^{|dom(\mathcal{N})|} e_K\left(\{0,1\} \times 2^j\right) \\ &= \mathcal{B}(m) \oplus \bigoplus_{\xi \in dom(\mathcal{N})} \mathcal{N}(\xi) \times \{0,1\}. \end{aligned}$$

5.4 Provable Security

We will demonstrate that incremental Rache is IND-CPA. We formalize the proof in the following proposition.

Proposition 2. *If a homomorphic encryption* Π *is IND-CPA, then its corresponding incremental Rache-extension* $\widetilde{\Pi}$ *defined in Eq.* (5) *is IND-CPA.*

Proof. The probability for an adversary $\mathcal{A}$ to successfully break Π and $\widetilde{\Pi}$ are $Pr\left[CPA_{\Pi}^{\mathcal{A}} = 1\right]$ and $Pr\left[CPA_{\widetilde{\Pi}}^{\mathcal{A}} = 1\right]$, respectively. By assumption, the following inequality holds:

$$Pr\left[CPA_{\Pi}^{\mathcal{A}} = 1\right] \leq \frac{1}{2} + \epsilon, \tag{6}$$

where ϵ is a negligible probability. By comparing Π and $\widetilde{\Pi}$, the latter yields $p + d$ additional pairs of plaintexts and ciphertexts out of the total 2^n possible pairs in the worst case. Therefore, the following inequality holds:

$$Pr\left[CPA_{\widetilde{\Pi}}^{\mathcal{A}} = 1\right] - Pr\left[CPA_{\Pi}^{\mathcal{A}} = 1\right] \leq \frac{p+d}{2^n}. \tag{7}$$

Combining Eq. (6) and Eq. (7) yields the following inequality:

$$Pr\left[CPA_{\widetilde{\Pi}}^{\mathcal{A}} = 1\right] \leq \frac{1}{2} + \epsilon + \frac{p+d}{2^n} = \frac{1}{2} + \epsilon + \frac{poly(n)}{2^n},$$

where the last equality comes from the simple fact that the summation of two polynomials is also a polynomial:

$$\forall x, y \in poly(n) : \ (x + y) \in poly(n).$$

Now, we only need to show that the summation of the last two terms, $\epsilon + \frac{poly(n)}{2^n}$, is negligible. According to Lemma 1 and Lemma 2 (§2), this is indeed the case. Therefore, the probability for the adversary $\mathcal{A}$ to succeed in the $CPA_{\widetilde{\Pi}}^{\mathcal{A}}$ experiment is only negligibly higher than $\frac{1}{2}$, proving that incremental Rache is IND-CPA, as claimed. □

6 EVALUATION

6.1 Objectives

We aim to answer the following questions experimentally:

- What is the performance overhead of encryption in outsourced databases? (§6.3)
- How does Rache perform comparing with state-of-the-art HE schemes in term of computational time and scalability? (§6.4)
- How does incremental Rache help reduce the performance overhead of encrypting data streams? (§6.5)

Specifically, in §6.3, we report the performance overhead of homomorphic encryption schemes, i.e., Cassandra performance with and without data encryption. In §6.4, we report the performance of Rache from three perspectives: comparison on three micro benchmarks (§§6.4.1–6.4.3), comparison on three real-world applications (§§6.4.4–6.4.6), and scalability on the number of parallel cores and input sizes (§6.4.7). In §6.5, we report the performance of incremental Rache from the following three perspectives. The performance and overhead are reported in sections §§6.5.1–6.5.2. The effectiveness of incremental encryption for aggregation functions is reported in §6.5.3. Lastly in §6.5.4, we show that incremental Rache outperforms Symmetria even for an arbitrary message with the original cache.

6.2 Experimental Setup

6.2.1 Systems and Implementation. We implement Rache (both the batch and the incremental versions) upon two base schemes, an asymmetric scheme Paillier [48] and a symmetric one Symmetria [57]. Both base schemes have proven to be IND-CPA [48, 57]. Our implementation follows the same spirit of CryptDB [51], which leaves the vanilla database unchanged but plugs in the cryptographic subsystem as a middleware. As a result, we integrate Rache into Cassandra [41] through the DataStax Java driver [20].

The project is managed by Maven 3.6.3 and compiled with Java 11. The parallelization (e.g., randomized radix additions on Lines 15–23, Algorithm 1) is implemented with OpenMPI 4.0.3 [46]. At the time of writing this paper, the implementation consists of 29,584 lines of code.

We deploy the Rache-enabled Cassandra on a 10-node cluster hosted at CloudLab [25]. Each node is equipped with two 36-core Intel Xeon Platinum 8360Y CPUs, 256 GB ECC DDR4-2666 memory, and two 1 TB SSDs. The operating system image is Ubuntu 20.04.3 LTS. All servers are connected via a 1 Gbps control link (Dell D3048 switches) and a 10 Gbps experimental link (Dell S5048 switches). We only use the experimental links for our evaluation.

6.2.2 Configurations. Some of the most important parameters of Cassandra are as follows. The replica factor is set to three. Hinted handoff is enabled globally. The maximum throttle of each thread is the default 1,024 KB. The internal buffers are flushed to disk every 10 seconds. The partitioner is the default Murmur3Partitioner. There is one seed node (i.e., `node 0`) with the SimpleSeedProvider class (implementing the SeedProvider interface). The concurrency of reads and writes (including materialized view writes) is set to 32. The full specification can be found in the `cassandra.yaml` file in the source code.

6.2.3 Workloads. We have tested the system prototype with six workloads, all of which are publicly available. These workloads include three micro-benchmarks and three real-world applications.

The first benchmark is a micro-benchmark to quantify the cost of homomorphic encryption and homomorphic addition, respectively. For the former, a sequence of integers [0, 32,768) are homomorphically encrypted; for the latter, the ciphertexts stored at radix entries are homomorphically summed up in a round-robin fashion 32,768 times.

The second benchmark is TPC-H ver. 3.0.0 [62], a standard relational database benchmark. TPC-H allows the user to specify the scales of the generated data; in this paper we set the scale as one, resulting in about one gigabyte of data. We will focus on the *part* table, which consists of 200,000 tuples.

The third benchmark is a dynamic set of random numbers for homomorphic encryption. This benchmark is mainly used for the purpose of weak scaling, allowing for the scalability test ranging between 1,024 and 32,768 numbers.

The first application is the U.S. national COVID-19 statistics from April 2020 to March 2021 [16]. The data set has 341 days of 16 metrics, such as *death increase*, *positive increase*, and *hospitalized increase*.

The second application is the human genome reference 38 [36], commonly known as *hg38*, which includes 34,424 rows of singular attributes, e.g., *transcription positions*, *coding regions*, and *number of exons*, last updated in March 2020.

The third application is the history of Bitcoin trade volume [13] since it was first exchanged in the public in February 2013. The data consists of the accumulated Bitcoin exchange on a 3-day basis from February 2013 to January 2022, totaling 1,086 large numbers.

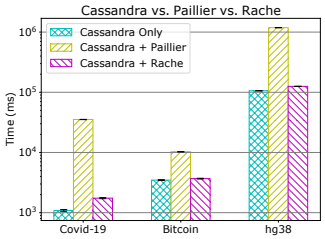

Fig. 1. Performance with and without encryption schemes.

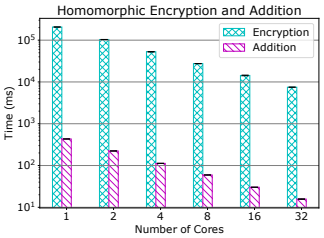

Fig. 2. Homomorphic encryption and addition in Paillier.

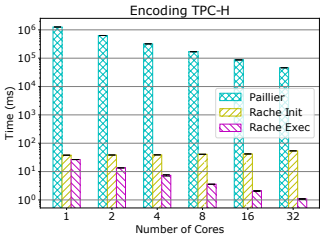

Fig. 3. Performance comparison on the TPC-H benchmark.

6.3 Performance with and without Homomorphic Encryption

We record the execution time of Cassandra when inserting three real-world data sets. The configuration of Cassandra and data set specification can be found in the previous section §6.2. We repeat the experiments three times and report both the average and the standard variation in the figure. To eliminate the possible caching effect, we truncate the table every time before starting the timer for the execution.

Figure 1 reports the results, which clearly shows that Rache significantly improves the performance of homomorphic encryption. For the Covid-19 data set (left column), the original Paillier scheme incurs 20× overhead while Rache only incurs about 2×. For the Bitcoin data set (center column), the originial Paillier scheme incurs 4× overhead while Rache's overhead is negligible. Similarly, for the hg38 data set (right column), Paillier incurs about 10× overhead and Rache's overhead is marginal.

6.4 Batch Rache

6.4.1 Encryption vs. Addition. Fig. 2 shows that the homomorphic addition is a much cheaper operation than homomorphic encryption in Paillier. Regardless of the number of available cores, homomorphic encryption takes more than two orders of magnitude time than homomorphic addition.

6.4.2 TPC-H. We report Rache's performance of encoding the TPC-H [62] data in Fig. 3. We report the execution time of initializing the radixes and that of encoding with Rache, respectively. The former is referred to as *Rache Init* and the latter as *Rache Exec* in the figure. The initialization time of Rache is roughly flattened, showing a marginal increase when more cores are involved due to the inter-process communication (IPC) overhead. It should be noted that, however, the *Rache Init* overhead is a one-time cost. Specifically, the *Init* cost is the execution time to construct the $Ptxt[]$ vector, which stores the radix values for future additive computation over ciphertexts. We observe that Rache outperforms Paillier by four orders of magnitude at all scales.

In general, the overhead incurred by Rache on different number of cores comes from the coordination of multiple processes and threads, such as *MPI_Reduce* that aggregates the partial summations over ciphertexts. The overhead discrepancy of different workloads, however, largely depends on the maximal value in the message space (assuming the radix r is fixed). As we will see soon in the following sections, the Rache initialization overhead (i.e., `Rache Init`) is lower than others (i.e., Figures 4–7). This can be best explained by the fact that the `Part` relation in TPC-H has its maximal numeric values in the order of thousands, which are much smaller than other benchmarks. Because the maximal value is smaller in TPC-H, Rache needs to precompute and cache fewer ciphertexts during the initialization phase, which results in smaller overhead than other benchmarks. This

observation also explains why the overhead stays roughly constant from one core to 32 cores: each core precomputes the same set of cached ciphertexts that are determined by radix r and the maximal plaintext message, both of which are the same on 1–32 cores.

6.4.3 Random Numbers. In this benchmark, n random numbers are generated in a uniform distribution by modular n. We report the results of Rache and Paillier in Fig. 4. The Rache overhead stays roughly constant for different numbers of cores, but not as low as TPC-H. Despite the overhead, we observe that Rache's encoding time is about two orders of magnitude lower than Paillier's at all scales.

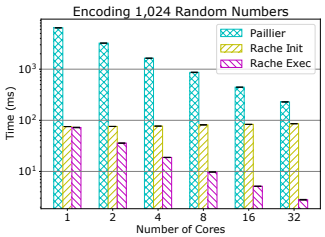

Fig. 4. Encoding performance on random numbers.

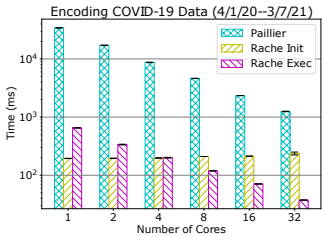

Fig. 5. Encoding the U.S. COVID-19 statistics.

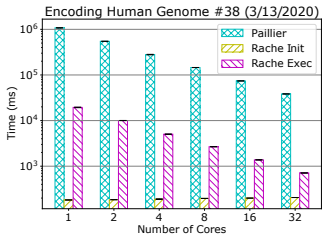

Fig. 6. Encoding the human genome reference 38.

6.4.4 U.S. COVID-19 Statistics. Fig. 5 reports the encoding performance of the U.S. COVID-19 statistics published at [16]. We observe that with few cores (e.g., 1 and 2) the overhead is smaller than the encoding cost, while with more cores (e.g., 16, 32) the per-core encoding is very efficient and takes less time than the overhead. Some of the overhead, i.e., precomputing and caching the large radixes, is unnecessary for those small values, and yet has to exist due to those extremely large values. We stress that the overhead is a one-time thing though: if there were, say, ten years of COVID-19 data, the overhead would look roughly the same and would be outweighed by the increased cost of encoding the data.

6.4.5 Human Genome Reference 38. Fig. 6 reports the encoding performance of Rache and Paillier on a database of human genome [36] (*hg38*) that was last updated in March 2020, under the umbrella of the Augustus gene prediction project [7]. As expected, Rache outperforms Paillier at all scales by orders of magnitude. In sheer contrast to the COVID-19 dataset, the initialization overhead of Rache in *hg38* is much less significant: even at 32-core, the overhead is less than 30%. This is mainly due to a large number of plaintexts (172,120), whose encoding time greatly outweighs the initialization, which is not trivial: 29 radixes for values as large as 248,937,123.

6.4.6 Bitcoin Trade Volume. We apply Rache and Paillier to the historical trade volume of Bitcoin exchanges since 2013 [13]. Fig. 7 shows that Rache outperforms Paillier by more than one order of magnitude, which is consistent with what we have found so far. The notable thing here is the large overhead incurred by Rache: on a single core, the overhead is on par with Rache's encoding time; on 32 cores, the overhead is on par with the Paillier processing time and orders of magnitude larger than Rache's encoding time. This phenomenon is due to two reasons. First, the Bitcoin trade volume consists of very large numbers—most are in the order of millions and the largest one is 4,956,849,516 requiring 34 radixes. Second, the number of plaintexts is relatively small: there are 1,086 plaintexts, each of which records the Bitcoin exchange for the last three days.

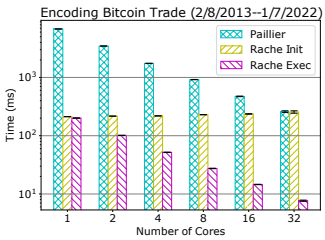

Fig. 7. Encoding the Bitcoin trade volume.

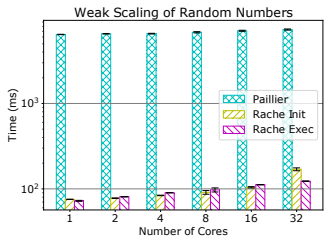

Fig. 8. Weak scaling of the encryption of random numbers.

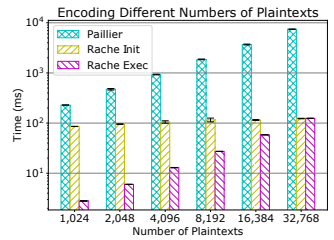

Fig. 9. Encoding a variety of workloads with a fix number of 32 cores.

6.4.7 Scalability. We evaluate the scalability of Rache in this section. We focus on the data sets of random numbers rather than specific benchmarks or applications simply because we can generate arbitrarily large data sets of random numbers. Fig. 8 reports the conventional weak-scaling experiment. We control the workload to be proportional to the number of cores: 1,024 plaintexts for every core. That is, the workloads range from 1,024 to 32,768 plaintexts of uniformly distributed random numbers. In each workload, the maximal value is close to the maximal number due to the uniform distribution.

Rache outperforms Paillier by orders of magnitude at all scales. However, Rache seems to exhibit a higher slope of encoding time. We stress that the absolute values of Rache performance are sub-seconds (and the y-axis is logarithmic), therefore the overhead can be best explained by the IPC overhead. To verify this, we conduct the following experiment, in which we fix the number of cores but increase the workloads.

Fig. 9 shows the encoding time when we fix the number of cores as 32 but increase the number of plaintexts from 1,024 to 32,768. We observe that when the IPC overhead is fixed (for 32 cores), the encoding time is proportionally increased regarding the workload size.

6.5 Incremental Rache

6.5.1 TPC-H. We compare Rache[1] and Symmetria on TPC-H with the option "-s 100"; there are overall 20,000,000 tuples in the Part table. We vary the number of pivots (i.e., p) on the x-axis between 2 and 64. We report the performance of Rache (without the overhead of constructing the pivots p's and nuances d's, which will be reported in the next experiment), and compare it against Symmetria in Fig. 10. Generally speaking, larger p values allow Rache to complete faster because of the finer granularity of the gaps among p's as well as fewer nuances. Notably, Rache is about 3x faster than Symmetria when $p = 32$. If the plaintexts are overly split (e.g., $p = 64$), the extra cost for maintaining the pivots may outweigh the benefit of d dictionaries, causing suboptimal performance.

6.5.2 Random Numbers. We compare the performance of Symmetria and Rache when encrypting 1,024 random numbers of variable lengths in Fig. 11. We on the x-axis vary the (n, p) pairs ranging between 8 and 32, where n indicates the bitstring length and p indicates the number of pivots, respectively. We observe that Rache consistently outperforms Symmetria for all (n, p) pairs by up to 50% reduction in running time, which is aligned with the results of the TPC-H benchmark in Fig. 10.

We measure the time overhead for precomputing pivots and nuances of 2^{32} random values. Note that this experiment has a much larger data set than that in Fig. 11 (i.e., $1{,}024 = 2^{10}$) because we will,

[1]For simplicity, we use Rache to indicate *incremental Rache* in this section.

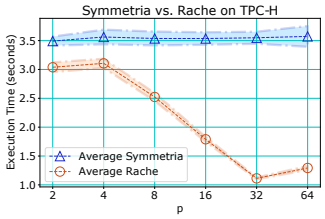

Fig. 10. Performance comparison on TPC-H (scale = 100), 20,000,000 tuples in table Part.

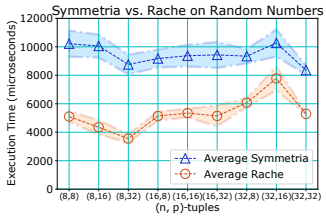

Fig. 11. Performance comparison of Symmetria and Rache on 1,024 random plaintexts.

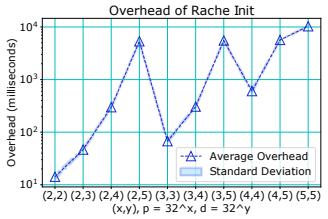

Fig. 12. Performance overhead incurred by pivots and nuances when encrypting 2^{32} random plaintexts.

to a large extent, vary both the number of pivots $p = n^x$, $2 \le x \le 5$ (x is considered as a practical upper bound in complexity theory [6]), and the number of nuances $d = n^y$, $x \le y$. We set $n = 32$, meaning that there are up to 2^{32} distinct values in the underlying data set. The x-axis of Fig. 12 enumerates those (x, y) pairs.

6.5.3 Aggregating Encrypted Fields. For a simple aggregate query shown in Listing 1 (i.e., the average part size), its Rache execution on the scale-10 TPC-H is illustrated in the following equation:

$$e_k\left(\sum_{i=1}^{2{,}000{,}000} s_i\right) = \bigoplus_{i=1}^{2{,}000{,}000} e_K(s_i),$$

where s_i denotes the value of the *P_Size* field of the i-th row of relation Part.

```
-- TPC-H 3.0, "dbgen -s 10", two million tuples
SELECT   AVG(P_Size)
FROM     Part;
```

Listing 1. A simple SQL aggregate query on TPC-H.

Directly adding up $e_K(s_i)$ is more costly than arithmetic operations because $\oplus$ on ciphertexts is number-theoretical. Rache allows us to cache the ciphertexts of both pivot and nuance along with their frequencies in plaintexts. Therefore, we can reduce the frequency of $\oplus$ by arithmetic $\times$ if the HE scheme supports it (Symmetria [57] does) and calculate the result as follows:

$$e_k\left(\sum_{i=1}^{2{,}000{,}000} s_i\right) = freq_i^p \times \bigoplus_{i=1}^{p} e_K(P_i) + freq_j^{\xi} \times \bigoplus_{j=1}^{d} e_K(\xi_j),$$

where p and d are much smaller than 200,000 (e.g., $p = d = 32$), $freq_x^y$ indicates the frequency of the x-th element in the y-container, and $e_K(\cdot)$'s are part of the entries (trees of pivots and dictionaries of nuances) cached in memory.

Fig. 13 reports the time for aggregating 200,000 *Part.P_Size* fields on scale-1 TPC-H, where each step aggregates additional 10,000 encrypted fields. We observe that the one-step cost of Symmetria is not constant: in a later step, it takes more time to aggregate the same number of new ciphertexts. This is concerning because it implies that the batch HE scheme is not scalable and would stop working at some point. To investigate how bad it could become, Fig. 14 reports the same workload on TPC-H of both scales-1 and scale-10; we did not report the scale-100 results because Symmetria finished only 53% (10,550,000 out of 20,000,000) ciphertext additions after 100 hours of execution. We observe that Rache can aggregate 2,000,000 fields within a second while Symmetria takes hours to complete the same workload.

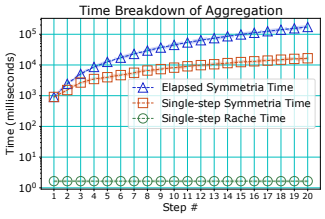

Fig. 13. Time breakdown of aggregating 200,000 tuples of table Part in TPC-H.

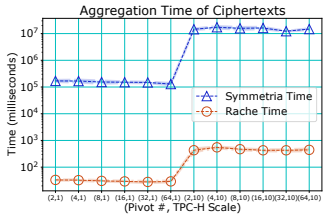

Fig. 14. Aggregating time with different numbers of pivots on different TPC-H scales.

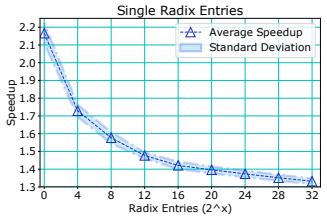

Fig. 15. Rache speedup over Symmetria when computing nuances on-the-fly.

6.5.4 Computing Nuances On-the-fly. The previous experiments assume that there is sufficient memory capacity to accommodate p pivots and d nuances. In certain application scenarios (e.g., edge computing [1, 2], supply chains [60], system-on-chip [14]), we might have limited resources and may not be able to hold, say, 2^{32} nuances. Therefore, the following experiment will investigate the worst-case scenario where we are forced to compute nuances on the fly. We report the performance of adopting a single nuance for a random value in $\left[0, 2^{64}\right)$ in Fig. 15. The worst-case overhead of calculating a single nuance leads to as low as 1.3x speedup over the vanilla Symmetria encryption. In the best case, i.e., when nuance is set to one, the speedup is over 2.1x.

6.6 Summary of Experimental Results

Rache. Both micro benchmarks and real-world applications confirm the efficiency of Rache: Rache incurs insignificant overhead to Cassandra while the conventional Paillier encryption is 2-10 times slower. Rache also exhibits strong scalability on up to 32 cores and 32× larger input data size.

Incremental Rache. Incremental Rache is 2–3× faster than Symmetria and the initialization overhead is as low as 10 ms. In particular, incremental Rache is 3–5 orders of magnitude faster than Symmetria for aggregation workloads that are commonly deployed in outsourced databases. Finally, incremental Rache outperforms Symmetria by 1.3–2.2× speedup even though incremental ciphertexts are not cached.

7 CONCLUSION AND FUTURE WORK

This paper proposes radix-based parallel caching optimization for accelerating the performance of homomorphic encryption (HE) of outsourced databases in cloud computing. The key insight of the proposed optimization is caching selected radix-ciphertexts in parallel without violating existing security guarantees of the original HE scheme. We design the radix HE algorithm and apply it to both batch and incremental HE schemes; we demonstrate the security of those radix-based HE schemes by reducing the Rache-extended problem to the base HE schemes that are known IND-CPA. We implement the radix-based schemes as middleware of a 10-node Cassandra cluster on CloudLab; experiments on six workloads show that the proposed caching significantly improves the performance of state-of-the-art HE schemes.

Our future work will focus on integrating radix-based caching into scientific blockchains [3, 4] such that sensitive scientific data can be shared and verified among the collaborators confidentially. One orthogonal optimization in this context will be to exploit the specific data format used in scientific workflows [43, 61] and array databases [8, 18]. We also plan to apply radix caching in federated learning [42].

ACKNOWLEDGMENTS

This work is supported in part by NSF under Grants IIS-1838024, CNS-1950485, and OIA-2148788. The authors thank the CloudLab team for providing access to their computing clusters. CloudLab is funded by NSF.

REFERENCES

[1] Abdullah Al-Mamun, Jun Dai, Xiaohua Xu, Mohammad Sadoghi, Haoting Shen, and Dongfang Zhao. 2020. Consortium Blockchain for the Assurance of Supply Chain Security. In *27th Annual Network and Distributed System Security Symposium (NDSS)*.

[2] Abdullah Al-Mamun, Haoting Shen, and Dongfang Zhao. 2022. DEAN: A Lightweight and Resource-efficient Blockchain Protocol for Reliable Edge Computing. In *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*.

[3] Abdullah Al-Mamun, Feng Yan, and Dongfang Zhao. 2021. BAASH: Lightweight, Efficient, and Reliable Blockchain-As-A-Service for HPC Systems. In *International Conference on High Performance Computing, Networking, Storage and Analysis (SC)*.

[4] Abdullah Al-Mamun, Feng Yan, and Dongfang Zhao. 2021. SciChain: Blockchain-enabled Lightweight and Efficient Data Provenance for Reproducible Scientific Computing. In *IEEE 37th International Conference on Data Engineering (ICDE)*.

[5] Prabhanjan Ananth, Aloni Cohen, and Abhishek Jain. 2017. Cryptography with Updates. In *Advances in Cryptology – EUROCRYPT 2017*, Jean-Sébastien Coron and Jesper Buus Nielsen (Eds.). Springer International Publishing, Cham, 445–472.

[6] Sanjeev Arora and Boaz Barak. 2009. *Computational Complexity: A Modern Approach* (1st ed.). Cambridge University Press, USA.

[7] Augustus: Gene prediction. Accessed 2022. https://github.com/Gaius-Augustus/Augustus.

[8] Peter Baumann, Dimitar Misev, Vlad Merticariu, Bang Pham Huu, and Brennan Bell. 2018. Rasdaman: Spatio-Temporal Datacubes on Steroids. In *Proceedings of the 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems* (Seattle, Washington) *(SIGSPATIAL '18)*. Association for Computing Machinery, New York, NY, USA, 604–607. https://doi.org/10.1145/3274895.3274988

[9] Mihir Bellare, Oded Goldreich, and Shafi Goldwasser. 1994. Incremental Cryptography: The Case of Hashing and Signing. In *Advances in Cryptology - CRYPTO '94, 14th Annual International Cryptology Conference, Santa Barbara, California, USA, August 21-25, 1994, Proceedings (Lecture Notes in Computer Science, Vol. 839)*, Yvo Desmedt (Ed.). Springer, 216–233. https://doi.org/10.1007/3-540-48658-5_22

[10] Mihir Bellare, Oded Goldreich, and Shafi Goldwasser. 1995. Incremental cryptography and application to virus protection. In *Proceedings of the Twenty-Seventh Annual ACM Symposium on Theory of Computing (STOC)*, Frank Thomson Leighton and Allan Borodin (Eds.).

[11] Ayoub Benaissa, Bilal Retiat, Bogdan Cebere, and Alaa Eddine Belfedhal. 2021. TenSEAL: A Library for Encrypted Tensor Operations Using Homomorphic Encryption. arXiv:2104.03152 [cs.CR]

[12] Tarunpreet Bhatia, A.K. Verma, and Gaurav Sharma. 2020. Towards a secure incremental proxy re-encryption for e-healthcare data sharing in mobile cloud computing. *Concurrency and Computation: Practice and Experience (CCPE)* 32, 5 (2020), e5520. https://doi.org/10.1002/cpe.5520 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/cpe.5520 e5520 CPE-18-0794.R1.

[13] Bitcoin Trade History. Accessed 2022. https://www.blockchain.com/charts/trade-volume.

[14] Subodha Charles and Prabhat Mishra. 2020. Securing Network-on-Chip Using Incremental Cryptography. In *2020 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. 168–175. https://doi.org/10.1109/ISVLSI49217.2020.00039

[15] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yong Soo Song. 2017. Homomorphic Encryption for Arithmetic of Approximate Numbers. In *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 10624)*, Tsuyoshi Takagi and Thomas Peyrin (Eds.). Springer, 409–437. https://doi.org/10.1007/978-3-319-70694-8_15

[16] Covid-19 Data. Accessed 2022. https://covidtracking.com/data/download/national-history.csv.

[17] Ronald Cramer, Ivan Damgård, and Jesper Buus Nielsen. 2001. Multiparty Computation from Threshold Homomorphic Encryption. In *Advances in Cryptology - EUROCRYPT 2001, International Conference on the Theory and Application of Cryptographic Techniques, Innsbruck, Austria, May 6-10, 2001, Proceeding (Lecture Notes in Computer Science, Vol. 2045)*, Birgit Pfitzmann (Ed.). Springer, 280–299. https://doi.org/10.1007/3-540-44987-6_18

[18] P. Cudre-Mauroux, H. Kimura, K.-T. Lim, J. Rogers, R. Simakov, E. Soroush, P. Velikhov, D. L. Wang, M. Balazinska, J. Becla, D. DeWitt, B. Heath, D. Maier, S. Madden, J. Patel, M. Stonebraker, and S. Zdonik. 2009. A Demonstration of SciDB: A Science-Oriented DBMS. *Proc. VLDB Endow.* 2, 2 (aug 2009), 1534–1537. https://doi.org/10.14778/1687553.1687584

[19] Ivan Damgård and Jesper Buus Nielsen. 2003. Universally Composable Efficient Multiparty Computation from Threshold Homomorphic Encryption. In *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings (Lecture Notes in Computer Science, Vol. 2729)*, Dan Boneh (Ed.). Springer, 247–264. https://doi.org/10.1007/978-3-540-45146-4_15

[20] DataStax Java Driver. Accessed 2022. https://github.com/datastax/java-driver.

[21] M. Daum, B. Haynes, D. He, A. Mazumdar, and M. Balazinska. 2021. TASM: A Tile-Based Storage Manager for Video Analytics. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE Computer Society, Los Alamitos, CA, USA, 1775–1786. https://doi.org/10.1109/ICDE51399.2021.00156

[22] Leo de Castro, Rashmi Agrawal, Rabia Yazicigil, Anantha Chandrakasan, Vinod Vaikuntanathan, Chiraag Juvekar, and Ajay Joshi. 2021. Does Fully Homomorphic Encryption Need Compute Acceleration? arXiv:2112.06396 [cs.CR]

[23] Daniel Demmler, Thomas Schneider, and Michael Zohner. 2015. ABY - A Framework for Efficient Mixed-Protocol Secure Two-Party Computation. In *22nd Annual Network and Distributed System Security Symposium, NDSS 2015, San Diego, California, USA, February 8-11, 2015*. The Internet Society. https://www.ndss-symposium.org/ndss2015/aby---framework-efficient-mixed-protocol-secure-two-party-computation

[24] Yarkın Doroz, Erdinc Ozturk, and Berk Sunar. 2015. Accelerating Fully Homomorphic Encryption in Hardware. *IEEE Trans. Comput.* 64, 6 (2015), 1509–1521. https://doi.org/10.1109/TC.2014.2345388

[25] Dmitry Duplyakin, Robert Ricci, Aleksander Maricq, Gary Wong, Jonathon Duerig, Eric Eide, Leigh Stoller, Mike Hibler, David Johnson, Kirk Webb, Aditya Akella, Kuangching Wang, Glenn Ricart, Larry Landweber, Chip Elliott, Michael Zink, Emmanuel Cecchet, Snigdhaswin Kar, and Prabodh Mishra. 2019. The Design and Operation of CloudLab. In *Proceedings of the USENIX Annual Technical Conference (ATC)*. 1–14. https://www.flux.utah.edu/paper/duplyakin-atc19

[26] T. Elgamal. 1985. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory* 31, 4 (1985), 469–472. https://doi.org/10.1109/TIT.1985.1057074

[27] Junfeng Fan and Frederik Vercauteren. 2012. Somewhat Practical Fully Homomorphic Encryption. Cryptology ePrint Archive, Paper 2012/144. https://eprint.iacr.org/2012/144 https://eprint.iacr.org/2012/144.

[28] Craig Gentry. 2009. Fully Homomorphic Encryption Using Ideal Lattices. In *Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing (STOC)*.

[29] O. Goldreich, S. Micali, and A. Wigderson. 1987. How to Play ANY Mental Game. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing* (New York, New York, USA) *(STOC '87)*. Association for Computing Machinery, New York, NY, USA, 218–229. https://doi.org/10.1145/28395.28420

[30] Hakan Hacigümüş, Bala Iyer, Chen Li, and Sharad Mehrotra. 2002. Executing SQL over Encrypted Data in the Database-Service-Provider Model. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data* (Madison, Wisconsin) *(SIGMOD '02)*. Association for Computing Machinery, New York, NY, USA, 216–227. https://doi.org/10.1145/564691.564717

[31] Shai Halevi and Victor Shoup. 2021. Bootstrapping for HElib. *J. Cryptol.* 34, 1 (jan 2021), 44 pages. https://doi.org/10.1007/s00145-020-09368-7

[32] Brandon Haynes, Maureen Daum, Dong He, Amrita Mazumdar, Magdalena Balazinska, Alvin Cheung, and Luis Ceze. 2021. VSS: A Storage System for Video Analytics. In *Proceedings of the 2021 International Conference on Management of Data* (Virtual Event, China) *(SIGMOD/PODS '21)*. 685–696. https://doi.org/10.1145/3448016.3459242

[33] HElib. Accessed 2022. https://github.com/homenc/HElib.

[34] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. 1998. NTRU: A Ring-Based Public Key Cryptosystem. In *Algorithmic Number Theory, Third International Symposium, ANTS-III, Portland, Oregon, USA, June 21-25, 1998, Proceedings (Lecture Notes in Computer Science, Vol. 1423)*, Joe Buhler (Ed.). Springer, 267–288. https://doi.org/10.1007/BFb0054868

[35] W. Kuan Hon and Christopher Millard. 2018. Banking in the cloud: Part 3 – contractual issues. *Computer Law & Security Review* 34, 3 (2018), 595–614. https://doi.org/10.1016/j.clsr.2017.11.007

[36] Human Genome Databases. Accessed 2022. http://hgdownload.soe.ucsc.edu/goldenPath/hg38/database/.

[37] Tehsin Kanwal, Adeel Anjum, and Abid Khan. 2021. Privacy preservation in e-health cloud: taxonomy, privacy requirements, feasibility analysis, and opportunities. *Clust. Comput.* 24, 1 (2021), 293–317. https://doi.org/10.1007/s10586-020-03106-1

[38] Gang Ke, Shi Wang, and Huan-huan Wu. 2021. Parallel incremental attribute-based encryption for mobile cloud data storage and sharing. *Journal of Ambient Intelligence and Humanized Computing* (01 2021), 1–11. https://doi.org/10.1007/s12652-020-02842-x

[39] Marcel Keller, Emmanuela Orsini, and Peter Scholl. 2016. MASCOT: Faster Malicious Arithmetic Secure Computation with Oblivious Transfer. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (Vienna, Austria) *(CCS '16)*. Association for Computing Machinery, New York, NY, USA, 830–842. https://doi.org/10.1145/2976749.2978357

[40] Louiza Khati and Damien Vergnaud. 2018. Analysis and Improvement of an Authentication Scheme in Incremental Cryptography. In *Selected Areas in Cryptography - SAC 2018 - 25th International Conference, Calgary, AB, Canada,*

August 15-17, 2018, Revised Selected Papers (Lecture Notes in Computer Science, Vol. 11349), Carlos Cid and Michael J. Jacobson Jr. (Eds.). Springer, 50–70. https://doi.org/10.1007/978-3-030-10970-7_3

[41] Avinash Lakshman and Prashant Malik. 2010. Cassandra: A Decentralized Structured Storage System. *SIGOPS Oper. Syst. Rev.* 44, 2 (April 2010).

[42] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. 2017. Communication-Efficient Learning of Deep Networks from Decentralized Data. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, AISTATS 2017, 20-22 April 2017, Fort Lauderdale, FL, USA (Proceedings of Machine Learning Research, Vol. 54)*, Aarti Singh and Xiaojin (Jerry) Zhu (Eds.). PMLR, 1273–1282. http://proceedings.mlr.press/v54/mcmahan17a.html

[43] Parmita Mehta, Sven Dorkenwald, Dongfang Zhao, Tomer Kaftan, Alvin Cheung, Magdalena Balazinska, Ariel Rokem, Andrew Connolly, Jacob Vanderplas, and Yusra AlSayyad. 2017. Comparative Evaluation of Big-Data Systems on Scientific Image Analytics Workloads. In *43rd International Conference on Very Large Data Bases (VLDB)*.

[44] Ilya Mironov, Omkant Pandey, Omer Reingold, and Gil Segev. 2012. Incremental Deterministic Public-Key Encryption. In *Advances in Cryptology – EUROCRYPT 2012*, David Pointcheval and Thomas Johansson (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 628–644.

[45] Payman Mohassel and Yupeng Zhang. 2017. SecureML: A System for Scalable Privacy-Preserving Machine Learning. In *2017 IEEE Symposium on Security and Privacy (SP)*. 19–38. https://doi.org/10.1109/SP.2017.12

[46] MPI. Accessed 2021. https://www.mpi-forum.org/docs/.

[47] National Institute and Technology of Standards. 2001. Advanced Encryption Standard. *NIST FIPS PUB 197* (2001).

[48] Pascal Paillier. 1999. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In *Proceedings of the 17th International Conference on Theory and Application of Cryptographic Techniques* (Prague, Czech Republic) *(EUROCRYPT'99)*. Springer-Verlag, Berlin, Heidelberg, 223–238.

[49] Antonis Papadimitriou, Ranjita Bhagwan, Nishanth Chandran, Ramachandran Ramjee, Andreas Haeberlen, Harmeet Singh, Abhishek Modi, and Saikrishna Badrinarayanan. 2016. Big Data Analytics over Encrypted Datasets with Seabed. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation (OSDI)*. USENIX Association, USA, 587–602.

[50] Rishabh Poddar, Tobias Boelter, and Raluca Ada Popa. 2019. Arx: An Encrypted Database using Semantically Secure Encryption. *Proc. VLDB Endow.* 12, 11 (2019), 1664–1678. https://doi.org/10.14778/3342263.3342641

[51] Raluca Ada Popa, Catherine Redfield, Nickolai Zeldovich, and Hari Balakrishnan. 2011. CryptDB: protecting confidentiality with encrypted query processing. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles (SOSP)*.

[52] T. Rabin and M. Ben-Or. 1989. Verifiable Secret Sharing and Multiparty Protocols with Honest Majority. In *Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing* (Seattle, Washington, USA) *(STOC '89)*. Association for Computing Machinery, New York, NY, USA, 73–85. https://doi.org/10.1145/73007.73014

[53] Dayane Reis, Jonathan Takeshita, Taeho Jung, Michael Niemier, and Xiaobo Sharon Hu. 2020. Computing-in-Memory for Performance and Energy-Efficient Homomorphic Encryption. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 28, 11 (2020), 2300–2313. https://doi.org/10.1109/TVLSI.2020.3017595

[54] 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. https://doi.org/10.1145/359340.359342

[55] Bita Darvish Rouhani, M. Sadegh Riazi, and Farinaz Koushanfar. 2018. Deepsecure: Scalable Provably-Secure Deep Learning. In *Proceedings of the 55th Annual Design Automation Conference* (San Francisco, California) *(DAC '18)*. Association for Computing Machinery, New York, NY, USA, Article 2, 6 pages. https://doi.org/10.1145/3195970.3196023

[56] Nikola Samardzic, Axel Feldmann, Aleksandar Krastev, Srinivas Devadas, Ronald Dreslinski, Christopher Peikert, and Daniel Sanchez. 2021. *F1: A Fast and Programmable Accelerator for Fully Homomorphic Encryption.* Association for Computing Machinery, 238–252. https://doi.org/10.1145/3466752.3480070

[57] Savvas Savvides, Darshika Khandelwal, and Patrick Eugster. 2020. Efficient Confidentiality-Preserving Data Analytics over Symmetrically Encrypted Datasets. *Proc. VLDB Endow.* 13, 8 (April 2020), 1290–1303. https://doi.org/10.14778/3389133.3389144

[58] SEAL 2021. Microsoft SEAL (release 3.7). https://github.com/Microsoft/SEAL. Microsoft Research, Redmond, WA..

[59] Adi Shamir. 1979. How to Share a Secret. *Commun. ACM* 22, 11 (nov 1979), 612–613. https://doi.org/10.1145/359168.359176

[60] Haoting Shen, Shahriar Badsha, and Dongfang Zhao. 2020. Consortium Blockchain for the Assurance of Supply Chain Security. In *27th Annual Network and Distributed System Security Symposium (NDSS)*.

[61] Tong Shu, Yanfei Guo, Justin Wozniak, Xiaoning Ding, Ian Foster, and Tahsin Kurc. 2021. Bootstrapping In-Situ Workflow Auto-Tuning via Combining Performance Models of Component Applications. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)* (St. Louis, Missouri). Article 28, 15 pages. https://doi.org/10.1145/3458817.3476197

[62] TPC-H 3.0.0. Accessed 2022. http://tpc.org/tpc_documents_current_versions/current_specifications5.asp.
[63] Fenghe Wang, Junquan Wang, and Wenfeng Yang. 2021. Efficient incremental authentication for the updated data in fog computing. *Future Generation Computer Systems (FGCS)* 114 (2021), 130–137. https://doi.org/10.1016/j.future.2020.07.039
[64] Andrew C. Yao. 1982. Protocols for secure computations. In *23rd Annual Symposium on Foundations of Computer Science*. 160–164. https://doi.org/10.1109/SFCS.1982.38
[65] Xiaojie Zhu, Erman Ayday, Roman Vitenberg, and Narasimha Raghavan Veeraragavan. 2021. Privacy-Preserving Search for a Similar Genomic Makeup in the Cloud. *IEEE Transactions on Dependable and Secure Computing* (2021). https://doi.org/10.1109/TDSC.2021.3074327

Received July 2022; revised October 2022; accepted November 2022