Fast and Secure Linear Regression and Biometric Authentication with Security Update

Yoshinori Aono Takuya Hayashi Le Trieu Phong* Lihua Wang

National Institute of Information and Communications Technology (NICT), Japan
{aono, takuya.hayashi, phong, wlh}@nict.go.jp

Abstract. We explicitly present a homomorphic encryption scheme with a flexible encoding of plaintexts. We prove its security under the LWE assumption, and innovatively show how the scheme can be used to handle computations over both binary strings and real numbers. In addition, using the scheme and its features, we build fast and secure systems of

- linear regression using gradient descent, namely finding a reasonable linear relation between data items which remain encrypted. Compared to the best previous work over a simulated dataset of 10^8 records each with 20 features, our system dramatically reduces the server running time from about 8.75 hours (of the previous work) to only about 10 minutes.
- biometric authentication, in which we show how to reduce ciphertext sizes by half and to do the computation at the server very *fast*, compared with the state-of-the-art.

Moreover, as key rotation is a vital task in practice and is recommended by many authorized organizations for key management,

- we show how to do key rotation over encrypted data, without any decryption involved, and yet homomorphic properties of ciphertexts remain unchanged. In addition, our method of doing key rotation handles keys of different security levels (e.g., 80- and 128-bit securities), so that the security of ciphertexts and keys in our scheme can be "updated", namely can be changed into a higher security level.

Keywords: Homomorphic encryption, LWE assumption, privacy preservation, linear regression, biometric authentication.

1 Introduction

1.1 Background

Imaginatively, the storage and computation on the cloud can be seen as storing data and performing computations on a huge and globally available "machine". Formally, a definition of cloud computing is given by NIST in [23] saying that it is a "model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources that can be rapidly provisioned and released with minimal management effort or service provider interaction."

The benefits of cloud computing come with threats, typically one of which is that plain data stored in the cloud may be accessed unwillingly. Promisingly, homomorphic encryption can balance the situation, as it enables the computations over the data even in encrypted form, on which the discussion can go back far to Rivest et al. [29] in 1978. Specifically, a client can store encrypted data on the cloud to enjoy the service, but at the same time can ensure that no useful information is leaked to the storage.

* Corresponding author

1.2 Our contributions

We explicitly explore features of an LWE-based homomorphic encryption scheme, and exploit those features in designing extremely efficient and secure systems in cloud computing. Details are given below.

1. A homomorphic encryption scheme: we make explicit and analyze a homomorphic public key encryption scheme secure under the LWE assumption. By making the scheme explicit, we newly discover that it has a very *flexible* encoding of plaintexts. By "*flexible*", we mean two things: (a) the scheme natively handles binary strings, real numbers, and the computations (additions, multiplication) over them; and (b) in the scheme, the message length can properly vary with applications.

See Section 3 for the scheme and Section 4 for the encoding. The novel encoding of plaintexts is exploited in depth in following secure systems.

2. Fast and secure linear regression: we show in Section 5 that secure linear regression using gradient descent can be accomplished extremely fast in time and modest in communication. In particular, in the scenario of outsourced computation with an honest-but-curious server and a client, our system processes a simulated dataset of 10^8 records each with 20 features in 10 minutes at the server (Xeon E5-2660 v3, 2.60GHz, 20 threads) and 0.38 second (1 thread) at the client, with only 279 kilobytes of communication from the server to the client. These are extremely fast compared with the best previous work using Paillier encryption equipped with garbled circuits (8.75 hours at the server, [26]). Moreover, on real UCI datasets [1], our system is dramatically faster than [26] with significant improvements on the communication costs as shown in Table 5.

As a warm-up for Section 5, we show how to handle real numbers with fixed precision (usually set to 64-bit) in Section 4, and demonstrate the secure computation of standard statistics including the mean, variance, and weighted sum (which is used in secure prediction in Section 5).

3. Fast and secure biometric system: we show that binary strings are processed extremely efficiently by our scheme in biometric authentication, in which two binary templates are compared by XORing. The computation can be done in a secure way where the templates are encrypted, so that it can be performed by an honest-but-curious server. Our ciphertext size is at least a half smaller than previous results by [31, 32], and the computation at the server is dramatically (more than 1000x) improved. See Section 6.

4. Key rotation and security update: this is very unique to our homomorphic encryption scheme, where we show that encrypted data (stored in the server in above systems) can be key-rotated, and generally security-updated in a non-naive way. The task of key rotation, recalled at length in Section 3.3, is recommended by NIST [25], PCI DSS [2], and OWASP [3]. We design algorithms allowing key rotation and security update over encrypted data without any plain data recovery, and hence help to prevent any data breach (on the cloud servers) specifically in above systems. See Section 3.3.

Discussion: LWE vs. Ring-LWE. There are trade-offs between LWE and its variant ring-LWE [22]. As an assumption, the strength of LWE is more understood than ring-LWE. First, all attacks to LWE apply to ring-LWE. Second, LWE problem with dimension n is *classically* reduced to lattice problems with dimension $\sqrt{n}$ [9], and we partially use this fact in claiming the security update in our scheme. Indeed, our security update algorithm can be succinctly described

as transforming ciphertexts from current to higher dimensions, together with a re-randomization in the high dimension.

However, the ring-LWE assumption generally helps saving the public key size, and improving the speed of the scheme basing on it thanks to careful manipulations over polynomial rings. Therefore, many works choose ring-LWE-based schemes in implementations [16, 19, 24, 31, 32], to list just a few.

When building secure systems, we will highlight that the usage of primitives, either LWE-based or ring-LWE-based, plays a key role for overall efficiency. Perhaps surprisingly, via above applications, we demonstrate that *proper* usages of an LWE-based scheme can still yield sufficient efficiency. Having said that, we discuss how to use ring-LWE in Appendix B, pointing out that the ciphertext size increases (compared to the use of LWE) when using in our systems.

1.3 More related works

Graepel et al. [16] mention using homomorphic encryption for regression, using schemes supporting additions and r multiplications where r is the number of steps in gradient descent. On a small dataset of 100 records each with 2 features, the system in [16] needs about 1 minute even when $r = 1$ (at which the result may be not tuned sufficiently). We are different from that work by showing in Section 5 that secure linear regression can be done with only additive homomorphism (and multiplicative one is not needed at all), regardless of the number r.

Regarding secure linear regression, the work [26] (compared with ours above) itself improves [17], whose running time is 2 days over a dataset of 51K records each with 22 features.

Differential privacy [13] aims to show that the change in any single data item will not much affect the output. Our system for secure linear regression can be combined with the functional mechanism of [33] so that the output can be publicly released with the same assurance from differential privacy. We will discuss details later.

Ignoring the key rotation and security update properties, the pairing-based BGN encryption scheme [7, 14] can also be used in our systems. In Appendix C we will show how it can be used in our systems, and approximately compare the efficiencies with the LWE-based ones.

2 Preliminaries

Let $\mathbb{Z}_s$ be the discrete Gaussian distribution over the integers $\mathbb{Z}$, with mean 0 and deviation $s > 0$. The mark $\xleftarrow{\text{g}}$ is for "sampling from a discrete Gaussian" set, so that $x \xleftarrow{\text{g}} \mathbb{Z}_s$ means x appears with probability proportional to $\exp(-\pi x^2/s^2)$. Below, $\xleftarrow{\$}$ means "sampling randomly"; and $\overset{\text{c}}{\approx}$ stands for "computationally indistinguishable". Abusing notation a little, $\mathbb{Z}_q \subset (-q/2, q/2]$ is the set of integers centered modulus q. (In this manuscript, q is always for modulus, and s is always for Gaussian deviation.)

2.1 Learning with Errors

Related to the decision LWE assumption $\text{LWE}(n, s, q)$, where n, s, q depends on security parameter, consider matrix $A \xleftarrow{\$} \mathbb{Z}_q^{m\times n}$, vectors $r \xleftarrow{\$} \mathbb{Z}_q^{m\times 1}$, $x \xleftarrow{\text{g}} \mathbb{Z}_s^{n\times 1}, e \xleftarrow{\text{g}} \mathbb{Z}_s^{m\times 1}$. Then vector $Ax + e$ is

computed over $\mathbb{Z}_q$. Define the following advantage of a poly-time probabilistic algorithm $\mathcal{D}$:

$$\begin{aligned}&\mathbf{Adv}_{\mathcal{D}}^{\mathrm{LWE}(n,s,q)}(\lambda)\\&= \Big|\Pr[\mathcal{D}(A, Ax+e) \to 1] - \Pr[\mathcal{D}(A, r) \to 1]\Big|.\end{aligned}$$

The LWE assumption asserts that $\mathbf{Adv}_{\mathcal{D}}^{\mathrm{LWE}(n,s,q)}(\lambda)$ is negligible as a functions of λ.

2.2 Homomorphic encryption, key rotation, and security update

Definition 1. *Public key homomorphic encryption (PHE) schemes consist of the following (possibly probabilistic) poly-time algorithms.*

- $\mathsf{ParamGen}(1^\lambda, h) \to pp$*: λ is the security parameter, while h is the maximum number of security updates of one ciphertext.*
  The public parameter pp is implicitly fed in following algorithms.
- $\mathsf{KeyGen}(1^\lambda) \to (pk, sk)$*: pk is the public key, while sk is the secret key.*
- $\mathsf{Enc}(pk, m) \to c$*: probabilistic encryption algorithm produces c, the ciphertext of message m.*
- $\mathsf{Dec}(sk, c) \to m$*: decryption algorithm returns message m encrypted in c.*
- $\mathsf{Add}(c, c')$, $\mathsf{AddM}(c, c')$*: In* Add*, for ciphertexts c and c', the output is the encryption of plaintext addition c_{add}. Similarly,* AddM *adds two ciphertexts each was obtained by one multiplication (using* Mul *below).*
- $\mathsf{Mul}(pp, c, c')$*: for ciphertexts c and c', the output is the encryption of plaintext multiplication c_{mul}.*
- $\mathsf{DecA}(sk, c_{\mathrm{add}})$*: decrypting c_{add} to obtain an addition of plaintexts.*
- $\mathsf{DecM}(sk, c_{\mathrm{mul}})$*: decrypting c_{mul} to obtain a multiplication of plaintexts.*

Definition 2. *With respect to a PHE scheme as in Definition 1, consider the following game between an adversary $\mathcal{A}$ and a challenger:*

- **Setup.** *The challenger creates pp and key pairs (pk, sk). Then pp and pk are given to $\mathcal{A}$.*
- **Challenge.** *$\mathcal{A}$ chooses two plaintexts m_0, m_1 of the same length, then submits them to the challenger, who in turn takes $b \in \{0, 1\}$ randomly and computes $C^* = \mathsf{Enc}(pk, m_b)$. The challenge ciphertext C^* is returned to $\mathcal{A}$, who then produces a bit b'.*

A PHE scheme is CPA-secure if the advantage $\mathbf{Adv}_{\mathcal{A}}^{\mathrm{cpa}}(\lambda) = \left|\Pr[b' = b] - \frac{1}{2}\right|$ is negligible in λ.

Definition 3. *A PHE scheme as in Definition 1 has the property of key rotation and security update if there are algorithms:*

- $\mathsf{UKGen}(pp, pk_1, sk_1, pk_2, sk_2)$*: generating an update key $uk_{1\to 2}$ from the public/secret key pairs (pk_1, sk_1) (old) and (pk_2, sk_2) (new).*
- $\mathsf{Update}(pp, c, uk_{1\to 2}, pk_2)$*: outputting a new ciphertext c_{new} from an old c.*

with the following four requirements:

a) Pseudorandom update key: *the update key $uk_{1\to 2}$ is pseudorandom, even given public pk_1, pk_2 and any poly-number of ciphertexts under these keys.*

b) Pseudorandom updated ciphertexts: *the ciphertext c_{new} is secure as those encrypted directly under pk_2, even given $uk_{1\to 2}$, pk_1, pk_2 and any poly-number of ciphertexts under these keys.*

c) Correctness of updated ciphertexts: *Succinctly,*

$$\mathsf{Dec}(sk_2, c_{\text{new}}) = \mathsf{Dec}(sk_1, c)$$

with overwhelming probability, namely the correctness is not affected by key rotation and security update.

d) Homomorphisms: *are preserved as long as the ciphertexts in operations are under the same public key (either pk_1 or pk_2), regardless of whether they are (1) directly formed by* Enc*, or (2) indirectly formed by* Update*.*

2.3 Our vision: secure cloud computing with key rotation and security update

Using a PHE scheme with properties as in Definition 3, a client can store encrypted data on cloud server for computations using homomorphic properties. Moreover, the client can do key rotation for all ciphertexts, namely replacing an old key pair (pk_1, sk_1) by a new one (pk_2, sk_2) by:

1. The client holding (pk_1, sk_1) generates the new key pair (pk_2, sk_2), and then creates $uk_{1\to 2}$ via executing $\mathsf{UKGen}(pp, pk_1, sk_1, pk_2, sk_2)$. The client sends $uk_{1\to 2}$ to the cloud server. Note $uk_{1\to 2}$ is pseudorandom, and hence reveals no information to the cloud server.
2. The cloud server receives $uk_{1\to 2}$ and runs the algorithm $\mathsf{Update}(pp, c, uk_{1\to 2}, pk_2)$ to rotate the key pk_1 encrypting c to pk_2. The output is c_{new}. The cloud server is trusted in deleting c and storing c_{new} properly. This step is repeated for all old ciphertexts c under pk_1 in the storage.

Plain data is never recovered in the above steps, which helps preventing any data breach. The discussion of why key rotation is important appears in [2, 3, 25] and Section 3.3.

3 Proposed scheme

Our proposed homomorphic encryption scheme is depicted in Figure 1. It naturally supports multiple additions and one multiplication. While relying on the idea of [20] for reducing the public key size, its full description is not explicitly appeared in the literature to the best of our knowledge, not to mention its power in application. In particular,

- there are rooms for choosing parameters l and p in the message space $\mathbb{Z}_p^l$ to fit each specific application, as they are not tightly related to parameters deciding security (q, n, s).
  For example, as l (message length) and n (security level) can be different, we can set $(l, n) = (64, 3530)$ for the secure computation of mean in Section 4.2, or $(l, n) = (16128, 3530)$ for linear regression as in Section 5.
  When $p = 2$ and l is arbitrary (e.g., $l = 2048$), the scheme can handle bit vectors in $\mathbb{Z}_2^{1\times l}$ as shown in Section 6 on biometrics. (When $p = 2$ and $l = 1$, we essentially obtain a variant of the Regev encryption scheme [28].)
- the addition and outer product of plaintexts in $\mathbb{Z}_p^{1\times l}$ gives rise to the addition and multiplication of real numbers, via a new encoding developed in Section 4.1.

PKE part

$\underline{\mathsf{ParamGen}(1^\lambda)}$:
Fix $q = q(\lambda) \in \mathbb{Z}^+$
Fix $l \in \mathbb{Z}^+$
Fix $p \in \mathbb{Z}^+$, $\gcd(p, q) = 1$
Return $pp = (q, l, p)$

$\underline{\mathsf{KeyGen}(1^\lambda, pp)}$:
Take $s = s(\lambda, pp) \in \mathbb{R}^+$
Take $n = n(\lambda, pp) \in \mathbb{Z}^+$
$R, S \xleftarrow{g} \mathbb{Z}_s^{n\times l}$, $A \xleftarrow{\$} \mathbb{Z}_q^{n\times n}$
$P = pR - AS \in \mathbb{Z}_q^{n\times l}$
Return $pk = (A, P, n, s)$, $sk = S$

$\underline{\mathsf{Enc}(pk, m \in \mathbb{Z}_p^{1\times l})}$:
$e_1, e_2 \xleftarrow{g} \mathbb{Z}_s^{1\times n}$, $e_3 \xleftarrow{g} \mathbb{Z}_s^{1\times l}$
$c_1 = e_1 A + pe_2 \in \mathbb{Z}_q^{1\times n}$
$c_2 = e_1 P + pe_3 + m \in \mathbb{Z}_q^{1\times l}$
Return $c = (c_1, c_2)$

$\underline{\mathsf{Dec}(S, c = (c_1, c_2))}$:
$\overline{m} = c_1 S + c_2 \in \mathbb{Z}_q^{1\times l}$
$m = \overline{m} \bmod p$
Return m

Homomorphic part

$\underline{\mathsf{Add}(c, c')}$:
Return $c + c' \mod q$

$\underline{\mathsf{AddM}(c_{\text{mul}}, c'_{\text{mul}})}$
Return $c_{\text{mul}} + c'_{\text{mul}} \mod q$

$\underline{\mathsf{DecM}(sk, c_{\text{mul}})}$:
Let $sk = S \in \mathbb{Z}_q^{n\times l}$
$\overline{m} = \begin{bmatrix} S \\ I_l \end{bmatrix}^{\mathsf{T}} (c_{\text{mul}}) \begin{bmatrix} S \\ I_l \end{bmatrix} \in \mathbb{Z}_q^{l\times l}$ (I_l: the identity matrix)
Return $\overline{m} \in \mathbb{Z}_p^{l\times l}$

$\underline{\mathsf{Mul}(pp, pk, c, c')}$:
Return $c_{\text{mul}} = c^{\mathsf{T}} c' \in \mathbb{Z}_q^{(n+l)\times(n+l)}$

$\underline{\mathsf{DecA}(sk, c_{\text{add}})}$:
identical to Dec

Fig. 1. Our PHE scheme. Above, $\mathbb{Z}^+$ and $\mathbb{R}^+$ are correspondingly the sets of positive integers and real numbers, $\xleftarrow{\$}$ means "sampling randomly" while $\xleftarrow{g}$ is for "sampling from the discrete Gaussian" set.

3.1 Correctness and homomorphic property

Correctness. We check that directly-formed ciphertexts can be decrypted correctly. Indeed, in the decryption Dec algorithm,

$$\begin{aligned} c_1 S + c_2 &= (e_1 A + pe_2)S + e_1 P + pe_3 + m \\ &= e_1(AS + P) + pe_2 S + pe_3 + m \\ &= p(e_1 R + e_2 S + e_3) + m \in \mathbb{Z}_q^{1\times l} \end{aligned} \tag{1}$$

will yield correct m in decryption if the noise $p(e_1 R + e_2 S + e_3)$ is small enough. The decryption of added ciphertexts via algorithm DecA can be checked similarly.

For ciphertexts $c \in \mathbb{Z}_q^{1\times(n+l)}$ and $c' \in \mathbb{Z}_q^{1\times(n+l)}$, consider $c_{\text{mul}} = c^{\mathsf{T}} c' \in \mathbb{Z}_q^{(n+l)\times(n+l)}$, where the T stands for matrix transpose. The decryption of this product works as follows. First, over $\mathbb{Z}_q$, and I_l is the identity matrix of size l,

$$\begin{bmatrix} S \\ I_l \end{bmatrix}^{\mathsf{T}} (c^{\mathsf{T}} c') \begin{bmatrix} S \\ I_l \end{bmatrix} = \left(c \begin{bmatrix} S \\ I_l \end{bmatrix} \right)^{\mathsf{T}} \left(c' \begin{bmatrix} S \\ I_l \end{bmatrix} \right) \tag{2}$$

$$= (m + pu)^{\mathsf{T}} (m' + pu') \in \mathbb{Z}_q^{l\times l} \tag{3}$$

so that the result becomes $(m + pu)^{\mathsf{T}}(m' + pu')$ over the integers $\mathbb{Z}$ if q is sufficiently larger than the incurred noise, which grows quadratically in each component. Then taking component-wise modulo p will yield $m^{\mathsf{T}} m' \in \mathbb{Z}_p^{l\times l}$, which is the outer product of two message vectors.

Quadratically homomorphic property. Succinctly, the scheme in Figure 1 can evaluate the following formulas on ciphertexts

$$CT_1 + \cdots + CT_{N_{\text{add}}} \in \mathbb{Z}_q^{1\times(n+l)} \tag{4}$$

$$CT_1^{\mathsf{T}} \cdot CT'_1 + \cdots + CT_{N_{\text{add}}}^{\mathsf{T}} \cdot CT'_{N_{\text{add}}} \in \mathbb{Z}_q^{(n+l)\times(n+l)} \tag{5}$$

in which (4) is a special case of (5) and yet the noise added is smaller. The decryption of (4) and (5) uses the algorithms DecA and DecM respectively, yielding

$$m_1 + \cdots + m_{N_{\text{add}}} \in \mathbb{Z}_p^{1\times l}$$
$$m_1^{\mathsf{T}} \cdot m'_1 + \cdots + m_{N_{\text{add}}}^{\mathsf{T}} \cdot m'_{N_{\text{add}}} \in \mathbb{Z}_p^{l\times l}$$

where m_i and m'_i are the plaintext vectors in CT_i and CT'_i.

3.2 Security analysis

We prove security of the proposed scheme according to Definition 2.

Theorem 1. *The scheme in Figure 1 is CPA-secure under the LWE assumption. Specifically, for any poly-time adversary $\mathcal{A}$, there is an algorithm $\mathcal{D}$ of essentially the same running time such that*

$$\mathbf{Adv}_{\mathcal{A}}^{\text{cpa}}(\lambda) \le (l+1) \cdot \mathbf{Adv}_{\mathcal{D}}^{\text{LWE}(n,s,q)}(\lambda).$$

Proof. The proof is standard and follows [20] closely. First, we modify the original game in Definition 2 by providing $\mathcal{A}$ with a random $P \xleftarrow{\$} \mathbb{Z}_q^{n\times l}$. Namely $P = pR - AS \in \mathbb{Z}_q^{n\times l}$ is turned to random. This is indistinguishable to $\mathcal{A}$ thanks to the LWE assumption with secret vectors as l columns of S. More precisely, we need the condition $\gcd(p,q) = 1$ to reduce $P = pR - AS \in \mathbb{Z}_q^{n\times l}$ to the LWE form. Indeed, $p^{-1}P = R + (-p^{-1}A)S \in \mathbb{Z}_q^{n\times l}$. As A is random, $A' = -p^{-1}A \in \mathbb{Z}_q^{n\times n}$ is also random. Therefore, $P' = p^{-1}P \in \mathbb{Z}_q^{n\times l}$ is random under the LWE assumption which in turn means P is random as claimed.

Second, the challenge ciphertext $c^* = e_1[A|P] + p[e_2|e_3] + [0|m_b]$ is turned to random. This relies on the LWE assumption with secret vector e_1. Here, the condition $\gcd(p,q) = 1$ is also necessary as above. Thus b is perfectly hidden after this change. The factor $l+1$ is due to l uses of LWE in changing P and 1 use in changing c^*. □

3.3 Key rotation and security update

Suppose encryption under (pk_1, sk_1) has security of level n_1, while that under (pk_2, sk_2) has level n_2[1]. The question is how to turn *old* ciphertexts of security level n_1 into *new* ones having level n_2 while keeping the underlying plaintext the same. Moreover, homomorphic operations can be still performed after the transformation.

Two possibly interesting cases are as follows:

- (**Key rotation**) When $n_1 = n_2$, the above is the well-known *key rotation* problem (applied to public-key homomorphic encryption), which is often raised in practice as the process of re-keying encrypted data from an old key to a new one.
- (**Security update**) When $n_1 < n_2$, the problem can be described succinctly as turning security-weakened ciphertexts and the related secret key into ones with higher security assurance.

[1] Concretely, in our scheme, n_1 and n_2 correspond to LWE dimensions, which are related to dimensions in lattice problems.

Indeed, on the federal side, it is recommended by NIST [25] via the concept of cryptoperiods of keys, namely the time spans during which they are used. As suggested in [25, Table 1], cryptoperiods are in the order of 1-2 years depending on the considered primitives. On the industrial side, it is required by the Payment Card Industry Data Security Standard [2, Requirement 3.6.4], and is recommended by the Open Web Application Security Project [3], specifying that "*key rotation is a must as all good keys do come to an end either through expiration or revocation. So a developer will have to deal with rotating keys at some point – better to have a system in place now rather than scrambling later.*" The same arguments apply to the case $n_1 < n_2$ as attacks advance.

How to do key rotation and security update. For these purposes, we define two additional algorithms UKGen (generating the update key) and Update (doing the key rotation, or security update over ciphertexts). The algorithm UKGen takes two pairs (pk_1, sk_1) and (pk_2, sk_2) and returns a key $uk_{n_1 \to n_2}$. The algorithm Update uses that $uk_{n_1 \to n_2}$ to turn a ciphertext c under pk_1 to a ciphertext c' decryptable under sk_2. The details are depicted in Figure 2 in which we need the functions $\mathsf{Power2}(\cdot)$ and $\mathsf{Bits}(\cdot)$ explained as follows. Let $v \in \mathbb{Z}_q^n$ and $\kappa = \lceil \log_2 q \rceil$, then there are bit vectors $v_i \in \{0,1\}^n$ such that $v = \sum_{i=0}^{\kappa-1} 2^i v_i$. Define

$$\mathsf{Bits}(v) = [v_0 | \cdots | v_{\kappa-1}] \in \{0,1\}^{1 \times n\kappa}.$$

Let $X = [X_1 | \cdots | X_l] \in \mathbb{Z}_q^{n \times l}$ where X_i are columns. Then

$$\mathsf{Power2}(X) = \begin{bmatrix} X_1 & \cdots & X_l \\ 2X_1 & \cdots & 2X_l \\ \vdots & & \vdots \\ 2^{\kappa-1}X_1 & \cdots & 2^{\kappa-1}X_l \end{bmatrix} \in \mathbb{Z}_q^{n\kappa \times l}.$$

It is easy to check that

$$\mathsf{Bits}(v)\mathsf{Power2}(X) = vX \in \mathbb{Z}_q^{1 \times l}.$$

Intuitively, $\mathsf{Bits}(\cdot)$ is used in Update to limit the noise increase, while $\mathsf{Power2}(\cdot)$ is put in UKGen to ensure correctness of updated ciphertexts using above equation.

New use of an old technique. The functions $\mathsf{Bits}(\cdot)$ and $\mathsf{Power2}(\cdot)$ are originated in [8, 10] as part of the dimension switching technique. Historically, the paper [10] only considered dimension-*reduction*, namely $n_2 < n_1$ in our notation, for efficiency issues in FHE. Then subsequent paper [8], while notified that the technique works for arbitrary dimensions n_1 and n_2, only made use of the case $n_2 < n_1$ as in [10]. In this paper, we use $n_1 = n_2$ for key rotation and $n_1 < n_2$ for security update in homomorphic encryption. Therefore, the specific usages of the dimension switching technique for key rotation and security update are new to this work, as summarized in Table 1.

Table 1. Usages of the dimension switching technique.

Dimension switching	Exploited in	Main purpose
(high $\to$ low) $n_2 < n_1$	[8, 10]	efficiency improvement in FHE
(equal) $n_2 = n_1$	[4, 12]	PRE, obfuscation
(equal, or low $\to$ high) $n_1 \leq n_2$	this manuscript	key rotation and security update

Key rotation and security update

$\mathsf{UKGen}(pp, pk_1, sk_1, pk_2, sk_2)$:

Let $pk_i = (A_i, P_i, n_i, s_i)$. Let $sk_i = S_i$ $(i = 1, 2)$

Let $\kappa = \lceil \log_2 q \rceil$. Take $X \xleftarrow{\$} \mathbb{Z}_q^{n_1 \kappa \times n_2}$, $E \xleftarrow{\mathrm{g}} \mathbb{Z}_{s_2}^{n_1 \kappa \times l}$

$Y = -XS_2 + pE + \mathsf{Power2}(S_1) \in \mathbb{Z}_q^{n_1 \kappa \times l}$

Return $uk_{n_1 \to n_2} = (X, Y)$

$\mathsf{Update}(pp, c, uk_{n_1 \to n_2}, pk_2)$:

Let $c = (c_1, c_2) \in \mathbb{Z}_q^{1 \times n_1} \times \mathbb{Z}_q^{1 \times l}$

Let $pk_2 = (A_2, P_2, n_2, s_2)$

Let $uk_{n_1 \to n_2} = (X, Y)$, and $f_1, f_2 \xleftarrow{\mathrm{g}} \mathbb{Z}_{s_2}^{1 \times n_2}$, $f_3 \xleftarrow{\mathrm{g}} \mathbb{Z}_{s_2}^{1 \times l}$

$E_0 = f_1[A_2|P_2] + p[f_2|f_3] \in \mathbb{Z}_q^{1 \times (n_2 + l)}$

$F = [\mathsf{Bits}(c_1)X | \mathsf{Bits}(c_1)Y + c_2] \in \mathbb{Z}_q^{1 \times (n_2 + l)}$

Return $c' = E_0 + F \in \mathbb{Z}_q^{1 \times (n_2 + l)}$

(In E_0 and F, $[\cdots|\cdots]$ is for matrix concatenation.)

Fig. 2. Algorithms for key rotation and security update.

Theorem 2 (Pseudorandom update key). *Given pp, pk_1, pk_2, and even secret sk_1, the update key $uk_{n_1 \to n_2}$ generated as in Figure 2 is computationally random under the* $\mathrm{LWE}(n_2, s_2, q)$ *assumption.*

Proof. Since $uk_{n_1 \to n_2} = (X, Y)$ where X is truly random, it suffices to show that Y is computationally random, which is straightforward since $-XS_2 + pE \in \mathbb{Z}_q^{n_1 \kappa \times l}$ is computationally random under the $\mathrm{LWE}(n_2, s_2, q)$ assumption.

The argument extends even given pk_1, sk_1, pk_2 and any poly-number of ciphertexts under pk_2. First, the matrix $-XS_2 + pE \in \mathbb{Z}_q^{n_1 \kappa \times l}$ is independent of pk_1. Second, the related part in pk_2 is

$$P_2 = pR_2 - A_2 S_2 \in \mathbb{Z}_q^{n_2 \times l}$$

so that both P_2 and Y are computationally random under the $\mathrm{LWE}(n_2, s_2, q)$ assumption. Third, conditioned random P_2, any ciphertext under pk_2 contains the part $e_1[A_2|P_2] + p \cdot [e_2|e_3]$ which is also pseudorandom under $\mathrm{LWE}(n_2, s_2, q)$ since the secret $e_1 \in \mathbb{Z}_q^{1 \times n_2}$ of Gaussian deviation s_2. □

Theorem 3 (Pseudorandom updated ciphertext). *The updated ciphertext c' generated as in Figure 2 is computationally random under the* $\mathrm{LWE}(n_2, s_2, q)$ *assumption.*

Proof. We have $c' = E_0 + F \in \mathbb{Z}_q^{1 \times (n_2 + l)}$. Note that E_0 is an encryption under pk_2 of length l vector $(0, \ldots, 0)$, and hence is computationally random under the $\mathrm{LWE}(n_2, s_2, q)$ assumption by Theorem 1. Therefore c' is computationally random under the $\mathrm{LWE}(n_2, s_2, q)$ assumption.

The argument extends even given $uk_{n_1 \to n_2}$, pk_1, pk_2 and any poly-number of ciphertexts under these keys. More precisely, pk_2 and $uk_{n_1 \to n_2}$ are first changed to random using the $\mathrm{LWE}(n_2, s_2, q)$ assumption. Conditioned on that, $E_0 = \mathsf{Enc}(pk_2, 0_{1 \times l})$ is turned to random also under the $\mathrm{LWE}(n_2, s_2, q)$ assumption as in the proof of Theorem 1. Other ciphertexts and pk_1 are unrelated to E_0, ending the proof. □

Theorem 4 (Correctness of updated ciphertext). *The ciphertext c' generated as in Figure 2 is correctly decrypted using* $\mathsf{Dec}(sk_2, \cdot)$ *of Figure 1.*

Proof. We have $c' = E_0 + F \in \mathbb{Z}_q^{1\times(n_2+l)}$. As E_0 is an encryption under pk_2 of length l vector $(0,\ldots,0)$, its decryption under sk_2 gives the zero vector. The decryption of F under $sk_2 = S_2$ is

$$\begin{aligned}\overline{m} &= \mathsf{Bits}(c_1)XS_2 + \mathsf{Bits}(c_1)Y + c_2 \\ &= \mathsf{Bits}(c_1)(pE + \mathsf{Power2}(S_1)) + c_2 \\ &= p\mathsf{Bits}(c_1)E + c_1S_1 + c_2 \in \mathbb{Z}_q^{1\times l} \end{aligned} \tag{6}$$

which is equal to the decryption of $c = (c_1, c_2)$ under $sk_1 = S_1$ as long as the added noise $\mathsf{Bits}(c_1)E$ is small, which holds with high probability as matrix E containing small, Gaussian-distributed elements. □

Homomorphisms held even after key rotation or security update. Succinctly, formulas (4) and (5) holds even if CT_i, CT_i' $(1 \le i \le N_{\text{add}})$ are altogether encrypted under the same public key, regardless of whether they are directly formed by the Enc algorithm (Figure 1) or are indirectly transformed by Update (Figure 2) from old ciphertexts. Intuitively, this is because (a) the zero encryption E_0 part in updated ciphertexts does not interfere with homomorphisms, and (b) the F part can be correctly decrypted as in (6).

More precisely, consider following cases, where "old" stands for a ciphertext under pk_1 and "new" for a ciphertext under pk_2, and Update(old) for an updated ciphertext from pk_1 to pk_2:

• (old + old) or (new + new): this should be easily seen, as vector addition of two ciphertexts in the same form $e_1[A|P] + p[e_2|e_3] + [0_n|m]$ and $e_1'[A|P] + p[e_2'|e_3'] + [0_n|m']$ under identical public key $[A|P]$ (either old or new) gives us a ciphertext whose decryption will yield $(m + m') \mod p$.

• Update(old) + new: let c_{ud} and c_{nw} be the updated-from-old and under-new public-key ciphertexts correspondingly. The decryption under the new secret key S_2 is

$$\mathsf{Dec}(S_2, c_{ud} + c_{nw}) = \mathsf{Dec}(S_2, c_{ud}) + \mathsf{Dec}(S_2, c_{nw}) \pmod p$$

as the noise increases linearly when doing $c_{ud} + c_{nw}$. The decryption $\mathsf{Dec}(S_2, c_{ud})$ works as in (6) and $\mathsf{Dec}(S_2, c_{nw})$ as in (1), yielding corresponding messages m_{ud} and m_{nw} as expected.

Like the above, one can do the multiplication of (old × old), (new × new), and (Update(old) × new) where × is the outer product of vectors, justifying that formulas (4) and (5) hold even after key rotation or security update.

3.4 Parameters for homomorphisms

In this subsection we establish the relation between the modulus q and the number of additions (after multiplication or key rotation or update) N_{add}. Results in this subsection are in Theorem 5 (theoretical) and Figure 3 (experimental).

Theorem 5. *Let parameters p, q, s be as in the scheme in Figure 1, and $n_1, \ldots, n_h$ $(h \ge 1)$ are the dimensions in key rotation or security updates, and N_{add} the number of additions over multiplied ciphertexts as in (5) where all ciphertexts are with the same n_i, then the correctness of (5) holds with overwhelming probability if $q = N_{\text{add}} \cdot O\Big(p^2 s^4 \sum_{i=1}^{h} n_i + p^2 s^2 (\log_2 q) \sum_{i=2}^{h} n_i\Big)$ in which the hidden constant in the $O(\cdot)$ is small.*

Proof. Given in Appendix A. □

Figure 3 complements Theorem 5 by showing the experimental N_{add} that one can actually obtain. The straight line is the theoretical values of q and N_{add} in Theorem 5, while the dots are experimental results. As seen in the figure, given a fixed q, the experimental N_{add} is always bigger than the theoretical one, which is as expected since the proof of Theorem 5 makes loose bounds for some worst cases. Setting $h = 1, 2, 3$ in Theorem 5 corresponds to 80-, 128-, 256-bit securities in Figure 3, and one can see that N_{add} is reduced after each "jump" from one security level to another as expected. Certainly, one can also do key rotation between keys of the same security levels (e.g., 128-bit), in which case N_{add} will decrease less.

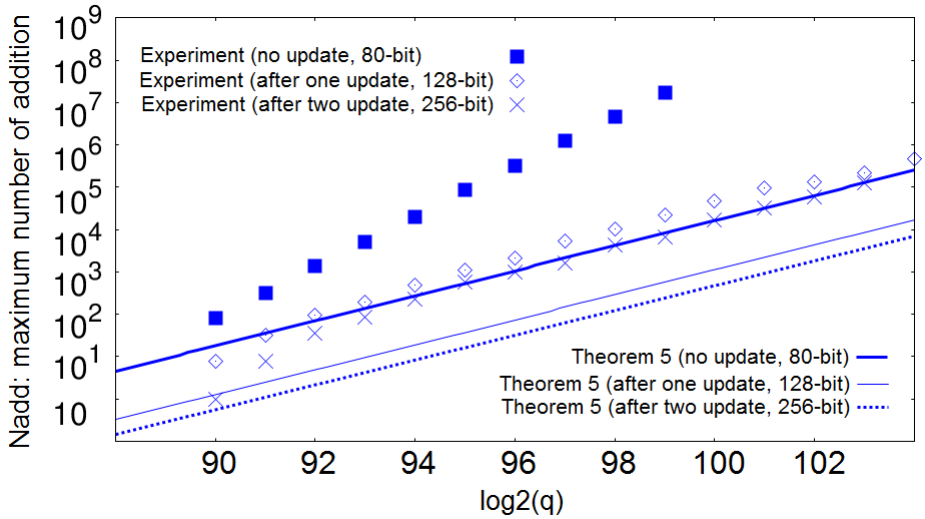

Fig. 3. Number of additions and modulus q. (In the figure, $p = 2^{30} + 1$ so that $\gcd(p, q) = 1$ and (n, s) is set to obtain 80-, 128-, 256-bit securities with a given modulus q.)

3.5 Basic timings

Parameter selection. We rely on recent attacks on LWE in [4, 20, 21]. Specifically, we use the following formula to decide the bit security of LWE(n, s, q)

$$\textbf{bit security} = \frac{4.22n - 96.8}{\ln q - 1.26 \ln s} - 65.7 \tag{7}$$

which is extrapolated from the recent cryptanalysis on LWE in [4] employing the RC5-72 testing benchmark[2] published in `distributed.net`. Concretely for $q = 2^{114}, p = 2^{30} + 1, s = 8.0, l = 64$, using (7) we need $n = 2661, 3530, 5847$ respectively for 80-, 128-, 256-bit securities. The message length $l = 64$ is sufficient to handle real numbers of 64-bit precision in computing basic statistics in Section 4.

Gaussian sampling. To generate discrete Gaussian noises, we employ the Knuth-Yao algorithm [18]. For reference, when $s = 10$, our standard desktop computer (having Intel Core i7-950 @

[2] As stated in [4], the relation with time/advantage 2^k of attacking LWE is **bit security** $= \log_2(2^k \cdot 9 \cdot 10^6)$.

3.07GHz) can generate more than $4.1 \cdot 10^4$ Gaussian samples in one millisecond in one thread with 256-bit precision, using only 1.65 megabytes to store a binary tree.

Timings of algorithms in Figure 1. The running times are reported in Table 2. One can see that additive homomorphism Add is extremely fast when compared with the others. This fact is innovatively exploited in applications in Sections 5 and 6.

Table 2. Timings of algorithms in Figure 1.

bit-sec	KeyGen	Enc	Dec	Add	Mul	AddM	DecM
80	1428	63.2	0.92	0.003	35.1	29.3	1313
128	2513	94.7	1.22	0.004	60.8	50.1	2296
256	7249	313	2.05	0.010	164	136	6643

(All times, averaged over 1000 executions, are in milliseconds using 1 thread of a Xeon E5-2660 v3, 2.60GHz machine. Parameters $q = 2^{114}, p = 2^{30} + 1, s = 8.0, l = 64$, and $n = 2661, 3530, 5847$ respectively for 80-, 128-, 256-bit securities using (7).)

Timings of algorithms in Figure 2. The running times are reported in Table 3. As seen in the table, generating an update key requires a few minutes, while updating a ciphertext needs a few seconds. Expectedly, these tasks are done rarely (say, once per year), and the process of ciphertext update can be fully parallelized on the cloud server.

Table 3. Timings of algorithms in Figure 2.

	bit-sec $\to$ bit-sec	UKGen	Update
Key rotation	80 $\to$ 80	165.6	1.1
	128 $\to$ 128	291.4	1.9
	256 $\to$ 256	846.6	5.3
Security update	80 $\to$ 128	238.2	1.5
	128 $\to$ 256	519.8	3.3

(All times are in seconds using the same machine and parameters as in Table 2.)

4 Warming up: encoding real numbers and secure mean and variance

In this section, we show basic techniques to handle real numbers in our proposed scheme in Figure 1, and demonstrate the secure computation of mean, variance, as well as weighted sum.

Previously, many works including [16, 24, 26] encode a real number by two steps: (a) encoding an integer and then (b) using the integer to approximate the real number with a fixed precision. Succinctly, a real number $a \in \mathbb{R}$ is represented via an integer $\lfloor a \cdot 2^\eta \rfloor \in \mathbb{Z}$ where η is the precision.

Below we deviate from the above encoding by showing how to work directly with real numbers.

4.1 Encode, add, multiply, and encrypt real numbers

Encoding. Real numbers of $\mathsf{prec} = (L + \ell + 1)$ bits are encoded via *signed* bit vectors, namely $a, b \in \mathbb{R}$ are expressed as

$$a = \sum_{k=-L}^{\ell} a_k 2^k, \text{ and } b = \sum_{k=-L}^{\ell} b_k 2^k$$

in which all $a_k \in \{0, 1\}$ if $a \geq 0$ and $a_k \in \{0, -1\}$ if $a < 0$, and likewise for b_k.

Define vector $\mathsf{Pow}(2, L, \ell) = [2^{-L} \ \cdots \ 2^0 \cdots \ 2^\ell] \in \mathbb{Z}^{\mathsf{prec}}$ and $\mathsf{Bits}(a) = [a_{-L} \cdots a_0 \cdots a_\ell] \in \{-1, 0, 1\}^{\mathsf{prec}}$ the signed bit vector in the above representation of $a \in \mathbb{R}$.

Addition of real numbers. We have

$$a + b = \sum_{k=-L}^{\ell} (a_k + b_k) 2^k = \mathsf{Pow}(2, L, \ell) \Big(\mathsf{Bits}(a) + \mathsf{Bits}(b) \Big)^{\mathsf{T}}$$

and generally for many $a^{(i)} \in \mathbb{R}$,

$$\sum_{i=1}^{N} a^{(i)} = \mathsf{Pow}(2, L, \ell) \left(\sum_{i=1}^{N} \mathsf{Bits}(a^{(i)}) \right)^{\mathsf{T}} \tag{8}$$

Multiplication of real numbers. Real number multiplication can be expressed by outer product of bit vectors as follows

$$\begin{aligned} ab &= \left([2^{-L} \ \cdots \ 2^\ell] \begin{bmatrix} a_{-L} \\ \vdots \\ a_\ell \end{bmatrix} \right) \left([b_{-L} \ \cdots \ b_\ell] \begin{bmatrix} 2^{-L} \\ \vdots \\ 2^\ell \end{bmatrix} \right) \\ &= [2^{-L} \ \cdots \ 2^\ell] \left(\begin{bmatrix} a_{-L} \\ \vdots \\ a_\ell \end{bmatrix} [b_{-L} \ \cdots \ b_\ell] \right) \begin{bmatrix} 2^{-L} \\ \vdots \\ 2^\ell \end{bmatrix} \end{aligned}$$

or succinctly

$$ab = \mathsf{Pow}(2, L, \ell) \underbrace{\mathsf{Bits}(a)^{\mathsf{T}} \mathsf{Bits}(b)}_{\text{outer product}} \mathsf{Pow}(2, L, \ell)^{\mathsf{T}} \tag{9}$$

where $\mathsf{Pow}(2, L, \ell) = [2^{-L} \ \cdots \ 2^\ell]$ and $\mathsf{Bits}(a)$ is the bit vector in binary representation of $a \in \mathbb{R}$. More generally, for many real numbers $a^{(i)} \in \mathbb{R}$ and $b^{(i)} \in \mathbb{R}$,

$$\begin{aligned} &\sum_{i=1}^{N} a^{(i)} b^{(i)} \\ &= \mathsf{Pow} \underbrace{\sum_{i=1}^{N} \mathsf{Bits}(a^{(i)})^{\mathsf{T}} \mathsf{Bits}(b^{(i)})}_{\text{sum of outer product}} \mathsf{Pow}^{\mathsf{T}} \end{aligned} \tag{10}$$

holds over $\mathbb{R}$, where $\mathsf{Pow} = \mathsf{Pow}(2, L, \ell)$.

Encryption of real numbers. For two real number a and b of precision prec, naturally define their encryptions as

$$E_a = \mathsf{Enc}\Big(\mathsf{Bits}(a)\Big), E_b = \mathsf{Enc}\Big(\mathsf{Bits}(b)\Big) \in \mathbb{Z}_q^{1\times(n+\mathsf{prec})} \tag{11}$$

where Enc is the encryption algorithm in Figure 1. These give rise to vector operations

$$E_a + E_b \in \mathbb{Z}_q^{1\times(n+\mathsf{prec})} \text{ and } E_a^{\mathsf{T}} E_b \in \mathbb{Z}_q^{(n+\mathsf{prec})\times(n+\mathsf{prec})}$$

whose decryption using the DecA and DecM algorithms yields

$$\mathsf{Bits}(a) + \mathsf{Bits}(b) \in \mathbb{Z}_p^{1\times\mathsf{prec}} \text{and } \mathsf{Bits}(a)^{\mathsf{T}}\mathsf{Bits}(b) \in \mathbb{Z}_p^{\mathsf{prec}\times\mathsf{prec}}$$

and then the modulo p is canceled

$$\mathsf{Bits}(a) + \mathsf{Bits}(b) \in \mathbb{Z}^{1\times\mathsf{prec}} \text{ and } \mathsf{Bits}(a)^{\mathsf{T}}\mathsf{Bits}(b) \in \mathbb{Z}^{\mathsf{prec}\times\mathsf{prec}}$$

if $p \geq 3$. These integer vectors, equipped with (9) and (10), give us the results $a+b \in \mathbb{R}$ and $ab \in \mathbb{R}$.

In general, to handle (8) and (10) when all N real numbers $a^{(i)}$ and $b^{(i)}$ are encrypted as in (11), the condition on p becomes

$$N < \frac{p}{2} \tag{12}$$

to remove the plaintext-related modulo p.

4.2 Secure computation of mean, variance, and weighted sum

Mean. To securely compute the mean of $a^{(1)}, \ldots, a^{(N)} \in \mathbb{R}$ using their encryptions $E_{a^{(1)}}, \ldots, E_{a^{(N)}}$ as in (11), just do

$$E_{a^{(1)}} + \cdots + E_{a^{(N)}} \in \mathbb{Z}_q^{1\times(n+\mathsf{prec})}$$

and then use (8) to obtain the sum $P_1 = \sum_{i=1}^N a^{(i)}$ over $\mathbb{R}$. The mean is obtained by dividing the sum by N.

When setting the precision $\mathsf{prec} = 64$ and $N = 1001$, in 128-bit security, one can see that the secure mean computation requires $(N-1) \times \mathbf{T}_{\mathsf{Add}} = 1000 \times 0.004 = 4$ (ms) where $\mathbf{T}_{\mathsf{Add}}$ is the time for one addition given in Table 2.

Variance. Compute

$$E_{a^{(1)}}^{\mathsf{T}} E_{a^{(1)}} + \cdots + E_{a^{(N)}}^{\mathsf{T}} E_{a^{(N)}} \in \mathbb{Z}_q^{(n+\mathsf{prec})\times(n+\mathsf{prec})}$$

whose decryption, equipped with (10), yields the sum of squares $P_2 = \sum_{i=1}^N (a^{(i)})^2 \in \mathbb{R}$. The variance is computed as $P_2/N - (P_1/N)^2$ where P_1 is computed as above.

Mean and variance in one shot. We need to change the encryption format from (11) a little, namely for $a^{(i)} \in \mathbb{R}$, let $E_{a^{(i)}} = (\mathsf{Enc}(\mathsf{Bits}(a^{(i)})), \mathsf{Enc}(\mathsf{Bits}((a^{(i)})^2))) \in \mathbb{Z}_q^{1\times 2(n+\mathsf{prec})}$, and compute $\sum_{i=1}^N E_{a^{(i)}} \in \mathbb{Z}_q^{1\times 2(n+\mathsf{prec})}$ whose decryption, equipped with (8), yields the sum P_1 and P_2 as above.

Again, when setting the precision $\mathsf{prec} = 64$ and $N = 1001$, in 128-bit security, one can see that the secure computation of the mean and variance requires at most $2(N-1)\times\mathbf{T}_{\mathsf{Add}} = 2000\times 0.004 = 8$ (ms) where $\mathbf{T}_{\mathsf{Add}}$ is the time for one addition given in Table 2.

Weighted sum. For real numbers $\theta^{(i)} \in \mathbb{R}$, compute $E_{\theta^{(i)}}$ using (11), and

$$E_{\theta^{(1)}}^{\mathsf{T}} E_{a^{(1)}} + \cdots + E_{\theta^{(N)}}^{\mathsf{T}} E_{a^{(N)}} \in \mathbb{Z}_q^{(n+\mathsf{prec})\times(n+\mathsf{prec})}$$

whose decryption, equipped with (10), yields the weighted sum $\sum_{i=1}^{N} \theta^{(i)} a^{(i)} \in \mathbb{R}$. This is later used for secure prediction in linear regression in Section 5.5.

5 Application 1: secure linear regression

5.1 The model of secure outsourced computation

We work in the scenario of outsourced computation using homomorphic encryption: a client outsources its data to a cloud server for computation and storage, but does not want to leak any information to the cloud server. This model is standard to show applications of homomorphic encryption, as in [15, 29]. In the following we recap the details.

Model outline. The general picture of the protocol is in Figure 4. We use $\mathbf{E}_{pk}(data^{(i)})$ to represent the encryption of the data under a public key pk from the client, and possibly $\mathbf{E}_{pk}(data^{(j)})$, $\mathbf{E}_{pk}(data^{(k)})$ from various geographically-distributed data contributors. After receiving the encrypted data, using homomorphic property of $\mathbf{E}_{pk}$, the computing server does necessary computations and sends the output $\mathbf{E}_{pk}(\Theta)$ to the data analyst, from which Θ is recovered by decryption, and the final result θ^* is obtained (from Θ).

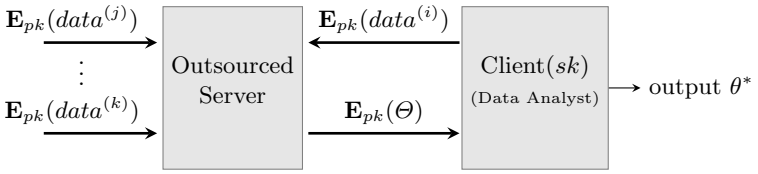

Fig. 4. Basic model of secure outsourced computation.

In medical research, the client can be some research institute, while data contributors can be many geographically distributed hospitals, e.g., as in the iDash system (`https://idash.ucsd.edu/`).

Key generation. The client generates the public and secret key pair (pk, sk) and publicly distributes pk.

Data encryption. Data from the client or many contributors is encrypted and sent to the outsourced server. We assume that these encryption and uploading processes are always correctly executed.

Threat and protection goal. The outsourced server is assumed *honest-but-curious*: it is curious on any information from the data, and yet is honest in instructed computations. This curious nature

of the server is considered a threat. The protection goal of our protocol in Figure 4 is to hide any information of the data from the server.

This honest-but-curious assumption is reasonable to model an economically motivated cloud service provider: it wants to provide excellent service for a successful business, but would be interested in any extra available information. On the order hand, a malicious cloud service provider can mishandle calculations, delete data, refuse to return results, collude with other parties etc. Nevertheless, it is likely to be caught in most of these malicious behaviors, and hence harms its reputation in business. Therefore, we will stick to the assumption of honest-but-curious server.

Non-threat. The client with the secret key can see all data in the plain, and this is not considered a threat in our context of outsourced computation.

5.2 Supporting multiple clients via key rotation

In previous Sections 2.3 and 3.3, we show how to use the key rotation property regarding security. In this section, again making use of that property, we show the model in Figure 4 can be extended to support many clients. The extension is in Figure 5, illustrating the case of two clients utilizing encrypted data in a cloud server.

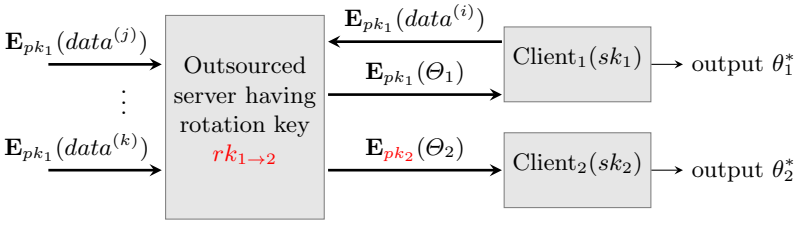

Fig. 5. Extended model with multiple clients.

In Figure 5, all data items from geographically-distributed contributors are encrypted under pk_1 and sent to the cloud server. The cloud server has a rotation key $rk_{1\to 2}$ generated via $\mathsf{UKGen}(pp, pk_1, sk_1, pk_2, sk_2)$ as in Figure 2. While the UKGen algorithm uses both the secret keys, it is unnecessary that Client$_1$ and Client$_2$ do the generation off-line. Instead, by examining the UKGen algorithm, it suffices to have Client$_2$ generate and send matrices $(X, -XS_2 + pE)$ to Client$_1$ via a secure channel. Then Client$_1$ can generate $rk_{1\to 2} = (X, Y)$ as in UKGen and sends it to the cloud server.

Client$_1$ can use the encrypted data and the cloud server as in the basic model in Figure 4. Client$_2$, with the help of $rk_{1\to 2}$, can do the same. Namely, it can have the cloud server to transform all or parts of the encrypted data under pk_1 into pk_2 and stores them on the cloud; it can outsource the computation over the ciphertexts under its key pk_2 just like Client$_1$ does.

5.3 Linear regression and some tweaks

Background. Let us start by recalling the standard (batch) gradient descent, succinctly described in Table 4. Data dimension, namely the number of features, is denoted by d. The number of data items, namely the training set size, is denoted by N_{data}. Each training data item is denoted as

Table 4. Gradient descent for linear regression.

Initialize $\theta = (\theta_0, \ldots, \theta_d) = (0, \ldots, 0) \in \mathbb{R}^{d+1}$.
Set $x_0^{(i)} = 1$ and fix a proper learning rate $\alpha \in \mathbb{R}$.
`Repeat`{

$$\theta_0 \leftarrow \theta_0 - \frac{\alpha}{N_{\text{data}}} \sum_{i=1}^{N_{\text{data}}} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x_0^{(i)}$$

$$\vdots$$

$$\theta_d \leftarrow \theta_d - \frac{\alpha}{N_{\text{data}}} \sum_{i=1}^{N_{\text{data}}} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x_d^{(i)}$$

} `until` $J_{\text{cost}}(\theta)$ at (13) `is minimized`
Return $\theta^* = \theta$ as the output.

$(x^{(i)}, y^{(i)})$ for $1 \le i \le N_{\text{data}}$ in which $x^{(i)} \in \mathbb{R}^d$ and $y^{(i)} \in \mathbb{R}$. As $x^{(i)} \in \mathbb{R}^d$, we can explicitly write it as $x^{(i)} = (x_1^{(i)}, \ldots, x_d^{(i)})$. The following concepts are vital to the gradient descent algorithm.

• Linear hypothesis is a linear function $h_\theta : \mathbb{R}^d \to \mathbb{R}$. Namely for variable $X = (X_1, \ldots, X_d) \in \mathbb{R}^d$,

$$h_\theta(X) = \sum_{j=1}^{d} \theta_j X_j + \theta_0.$$

• The cost function over training data

$$J_{\text{cost}}(\theta) = \frac{1}{2N_{\text{data}}} \sum_{i=1}^{N_{\text{data}}} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2. \tag{13}$$

The goal of the gradient descent algorithm in the left of Table 4 is to find $\theta = (\theta_0, \ldots, \theta_d)$ which minimizes $J_{\text{cost}}(\theta)$ in (13). In practice, one may be content when $J_{\text{cost}}(\theta)$ is sufficiently close to the exact minimum, and hence stops the algorithm when $J_{\text{cost}}(\theta)$ is not much smaller compared to the previous step.

Tweaks in computation. Referring to Table 4, for all $j = 0, \ldots, d$, the sum in updating each θ_j can be expressed as follows

$$\begin{aligned}
&\sum_{i=1}^{N_{\text{data}}} \left(h_\theta\left(x^{(i)}\right) - y^{(i)} \right) x_j^{(i)} \\
&= \sum_{i=1}^{N_{\text{data}}} \left(\sum_{k=0}^{d} \theta_k x_k^{(i)} - y^{(i)} \right) x_j^{(i)} \\
&= \sum_{i=1}^{N_{\text{data}}} \sum_{k=0}^{d} \theta_k x_k^{(i)} x_j^{(i)} - \sum_{i=1}^{N_{\text{data}}} y^{(i)} x_j^{(i)} \\
&= \sum_{k=0}^{d} \theta_k \underbrace{\sum_{i=1}^{N_{\text{data}}} x_k^{(i)} x_j^{(i)}}_{A_{k,j} \in \mathbb{R}} - \underbrace{\sum_{i=1}^{N_{\text{data}}} y^{(i)} x_j^{(i)}}_{B_j \in \mathbb{R}}
\end{aligned} \tag{14}$$

Similarly, the cost function in (13) can be expanded as

$$J_{\text{cost}}(\theta) = \frac{1}{2N_{\text{data}}} \left(\sum_{k,k'=0}^{d} (\theta_k \theta_{k'}) A_{k,k'} - 2 \sum_{k=0}^{d} \theta_k B_k + C \right) \tag{15}$$

where $A_{k,k'}$ and B_k are as in (14), and

$$C = \sum_{i=1}^{N_{\text{data}}} \left(y^{(i)} \right)^2. \tag{16}$$

Observation: heavy works to cloud, light works to client. To proceed linear regression using gradient descent, the client just needs to know the values $A_{k,j}, B_j, C$ for all $0 \leq k, j \leq d$ in which presumably $d \ll N_{\text{data}}$. Therefore, the work of the client is $O(d^2)$. In contrast, the work of the cloud server is $O(N_{\text{data}})$. This observation is vital to the following sections handling encrypted data.

Extension to ridge regression. The above backgrounds and tweaks can be straightforwardly applied to ridge regression, where the cost function is defined as $J_{\text{cost}}(\theta) + \mu ||\theta||_2^2$ for a parameter $\mu \in \mathbb{R}$. When $\mu = 0$, ridge regression is exactly linear regression. When $\mu > 0$, a solution θ with small norm is preferable. As the added term $\mu ||\theta||_2^2$ does not vary with data items, the following subsections remain almost unchanged even with ridge regression. (The only change for ridge regression is in the client side, and we will note that in place later.)

5.4 Secure linear regression

Below we give details on securely outsourced linear regression under the model outlined in Figure 4.

Data format. Each data item i ($1 \leq i \leq N_{\text{data}}$) is represented as

$$data^{(i)} = (x_1^{(i)}, \ldots, x_d^{(i)}, y^{(i)}) \in \mathbb{R}^{d+1}.$$

Via normalization, without loss of generality, we assume that $-1 \leq x_1^{(i)}, \ldots, x_d^{(i)}, y^{(i)} \leq 1$ which is standard to improve the convergence speed of the gradient descent.

Encryption at client (and data sources). The encryption $\mathbf{E}_{pk}(data^{(i)})$ from each data source is done as follows

1. Compute $u_{k,j}^{(i)} = x_k^{(i)} x_j^{(i)}, v_j^{(i)} = y^{(i)} x_j^{(i)}, w^{(i)} = (y^{(i)})^2 \in \mathbb{R}$ for all $1 \leq k, j \leq d$, which is of $l_d = d(d+1)/2 + d + 1$ real numbers.
2. Consider prec bit representations of real numbers in $data^{(i)}$, and all $u_{k,j}^{(i)}$ and $v_j^{(i)}$, concatenate them as one vector in $\{-1, 0, 1\}^{(d+1+l_d) \cdot \mathsf{prec}}$, and do the encryption

$$E^{(i)} = \mathsf{Enc} \begin{pmatrix} \mathsf{Bits}(x_1^{(i)}), \ldots, \mathsf{Bits}(x_d^{(i)}), \mathsf{Bits}(y^{(i)}), \ldots, \\ \mathsf{Bits}(u_{k,j}^{(i)}), \ldots, \mathsf{Bits}(v_j^{(i)}), \ldots, \mathsf{Bits}(w^{(i)}) \end{pmatrix}$$

where Enc is the encryption algorithm in Figure 1. The ciphertext result is a vector in $\mathbb{Z}_q^{1 \times (n + (d+1+l_d) \cdot \mathsf{prec})}$, and is sent to the server.

Outsourced server. Having all ciphertexts $E^{(i)}$, the cloud server computes the sum

$$E = \sum_{i=1}^{N_{\text{data}}} E^{(i)} \in \mathbb{Z}_q^{1\times(n+(d+1+l_d)\cdot\mathsf{prec})} \tag{17}$$

and sends the result to the client.

Client (data analyst). The client decrypts E using algorithm DecA in Figure 1 to obtains the following sums

$$\sum_{i=1}^{N_{\text{data}}} \mathsf{Bits}(x_1^{(i)}), \ldots, \sum_{i=1}^{N_{\text{data}}} \mathsf{Bits}(x_d^{(i)}), \sum_{i=1}^{N_{\text{data}}} \mathsf{Bits}(y^{(i)}) \in \mathbb{Z}_p^{1\times\mathsf{prec}}$$
$$\text{and } \sum_{i=1}^{N_{\text{data}}} \mathsf{Bits}(u_{k,j}^{(i)}), \sum_{i=1}^{N_{\text{data}}} \mathsf{Bits}(v_j^{(i)}), \sum_{i=1}^{N_{\text{data}}} \mathsf{Bits}(w^{(i)}) \in \mathbb{Z}_p^{1\times\mathsf{prec}}$$

from which the client obtains the following real numbers

$$\sum_{i=1}^{N_{\text{data}}} x_1^{(i)}, \ldots, \sum_{i=1}^{N_{\text{data}}} y^{(i)} \in \mathbb{R}$$
$$\sum_{i=1}^{N_{\text{data}}} u_{k,j}^{(i)}, \sum_{i=1}^{N_{\text{data}}} v_j^{(i)} \in \mathbb{R}, \sum_{i=1}^{N_{\text{data}}} w^{(i)} \in \mathbb{R}$$

for all $1 \le k, j \le d$ via (8) if $N_{\text{data}} < p/2$. These are exactly $A_{k,j}, B_j, C$ for all $0 \le k, j \le d$ in (15) and (16). Having these coefficients is equivalent to having $J_{\text{cost}}(\theta)$ and its gradients at (13) and (14), the client then finds $\theta^* = \text{argmin}_\theta J_{\text{cost}}(\theta)$. (One concrete and simple way is to proceed as in Table 4 and outputs the result of that algorithm. For ridge regression, $\theta^* = \text{argmin}_\theta(J_{\text{cost}}(\theta) + \mu||\theta||_2^2)$ at the client side.)

Communication cost between server and client. The cost of sending the computed result is only the size of E at (17), which is in bits

$$\Big(n + (d+1+l_d)\mathsf{prec}\Big)\log_2 q = \Big(n + O(d^2)\mathsf{prec}\Big)\log_2 q. \tag{18}$$

Moreover, in the store phase, as $E^{(i)}$ has the same size of E at (17), the cost of each store is also (18).

5.5 Secure prediction

Section 5.4 showed how to compute $\theta^* = \text{argmin}_\theta J_{\text{cost}}(\theta)$ using encrypted data. Here we show how to use those coefficients for secure prediction in the sense that

- $\theta^* = (\theta_0^*, \ldots, \theta_d^*)$ can be made public, and
- data used in prediction $(x_1, \ldots, x_d)$ is encrypted,

and yet the output $h_{\theta^*}(x) = \sum_{i=1}^d \theta_i^* x_i + \theta_0^*$ can be computed. This is exactly the secure computation of the weighted sum described in Section 4.2 with the "weights" $(\theta_0^*, \ldots, \theta_d^*)$.

5.6 Outputs with differential privacy

Outputs with differential privacy intuitively ensure that the change in any single data item will not much affect the output.

First, it is worth noting that directly adding Laplace noise to the output θ^*, namely directly using the Laplace mechanism [13], may work heuristically. Nevertheless, it seems hard to determine parameters for the noise due to complexity of the gradient descent algorithm.

Instead, we prefer to make use of the functional mechanism [33]. Namely, it suffices to add noises with Laplace distribution into the coefficients $A_{k,k'}$, B_k, and C of the cost function at (15). In that way, differential privacy of the output can be obtained, as shown in [33].

Below gives the details. Recall that, a noise $x \in \mathbb{R}$ has $\mathrm{Lap}(\sigma)$ distribution if its probability density function is $\frac{1}{2\sigma}\exp(-|x|/\sigma)$. To obtain ϵ-differential privacy, the only change we need is the computation of the server given at (17). Namely, the server generates Laplace noises $e_1, \ldots, e_{d+1+l_d} \in \mathbb{R}$ from the $\mathrm{Lap}(n_d/\epsilon)$ distribution, where $n_d = O(d^2)$, and does the following computation in the replacement of (17)

$$E_\epsilon = \mathsf{Enc}\Big(\mathsf{Bits}(e_1), \ldots, \mathsf{Bits}(e_{d+1+l_d})\Big) \in \mathbb{Z}_q^{1\times(n+(d+1+l_d)\cdot\mathsf{prec})}$$
$$E = E_\epsilon + \sum_{i=1}^{N_{\mathrm{data}}} E^{(i)} \in \mathbb{Z}_q^{1\times(n+(d+1+l_d)\cdot\mathsf{prec})}.$$

The effect of the change is that the client obtains real values

$$\overline{A}_{k,j} = A_{k,j} + \mathrm{Lap}(n_d/\epsilon) \in \mathbb{R}$$
$$\overline{B}_j = B_j + \mathrm{Lap}(n_d/\epsilon) \in \mathbb{R}$$
$$\overline{C} = C + \mathrm{Lap}(n_d/\epsilon) \in \mathbb{R}$$

for all $0 \le k, j \le d$ in (15) and (16). Having them, the client again proceeds as in Section 5.4.

5.7 Experiments and comparisons

To compare with the best previous work regarding speed and size [26], we use the same UCI datasets [1]. The concrete figures on server-client communication costs and server timings are given in Table 5.

Regarding the communication costs, megabytes in [26] are reduced to kilobytes by our system. For example, in the **flare2** dataset, 200 megabytes in [26] become 44 kilobytes (0.044 megabytes) in our system.

Regarding the computational costs, seconds in [26] are reduced to milliseconds by our system. For example, with the same **flare2** dataset, 115 seconds become 22.3 milliseconds (0.0223 seconds) in our system.

The above differences are due to many reasons including: the model of computation, the choice of primitives, and the design gluing those together. Specifically, we use the standard model of secure outsourced computation using homomorphic encryption, while [26] combines homomorphic encryption with Yao garbled circuits [30]. The use of garbled circuits causes megabytes of communication in [26]. In turn, the server in [26] needs to solve a garbled linear equation system with d equations and variables. In contrast, in our system, the server only does the additions over ciphertexts; the

Table 5. Comparisons using the UCI [1] datasets.

Name	n	d	Comm. (MB)		Time (s)	
			[26]	**Ours**	**[26]**	**Ours**
automobile	205	14	189	0.032	100	0.0227
autompg	398	9	39	0.016	21	0.0086
challenger	23	2	2	0.005	2	0.0014
communities	2215	20	234	0.050	130	0.0276
computerhardware	209	7	21	0.012	15	0.0035
concreteslumptest	103	7	15	0.009	10	0.0027
concreteStrength	1030	8	27	0.017	17	0.0074
forestFires	517	12	83	0.025	46	0.0111
insurance	9822	14	102	0.036	55	0.0617
flare1	323	20	170	0.035	92	0.0201
flare2	1066	20	200	0.044	115	0.0223

(Server: Xeon E5-2660 v3, 2.60GHz x 2 CPU, 20 threads. Client: same machine with 1 thread. The number of bits for fractional part of real numbers is set identically in each comparison. Our reported time already counts both of the server and the client.)

client decrypts to obtain the coefficients of the cost function at (15) and proceeds with gradient descent, so that our approach scales very well even when the number d of features in the data gets large (e.g., $d = 20$ and beyond).

6 Application 2: secure biometric authentication

For binary vectors $R, A, A' \in \mathbb{Z}_2^{1\times l}$ $(= \{0,1\}^l)$, consider the encryption using the Enc algorithm in Figure 1 with $p = 2$ and $\gcd(2, q) = 1$,

$$c^{(1)} = \mathsf{Enc}(R + A) \in \mathbb{Z}_q^{n+l}, c^{(2)} = \mathsf{Enc}(A') \in \mathbb{Z}_q^{n+l}$$

and define

$$c^{\text{add}} = c^{(1)} + c^{(2)} \in \mathbb{Z}_q^{n+l} \tag{19}$$

so that by the additive homomorphism

$$\mathsf{Dec}_{sk}\left(c^{\text{add}}\right) = R + A + A' \in \mathbb{Z}_2^l \tag{20}$$

which gives us binary vector $R \in \mathbb{Z}_2^l$ if $A = A'$.

The computations at (19) and (20) can be readily applied to biometric authentication, as follows. The following passively secure protocol has been known in [11].

Biometric authentication protocol [11]. There are three parties in the protocol: users, a server, and a key manager.

- **Key generation:** The key manager runs algorithms $\mathsf{ParamGen}(1^\lambda)$ and $\mathsf{KeyGen}(1^\lambda, pp)$ in Figure 1 to obtain pp and (pk, sk) respectively. The key manager keeps sk secret and makes $pp = (q, l, p = 2)$, and $pk = (A, P, n, s)$ public. As $p = 2$ we have $\mathbb{Z}_p = \mathbb{Z}_2 = \{0, 1\}$.

- **Enrollment phase:** a user with identity id and biometric representation $A \in \mathbb{Z}_2^l$ takes a random binary linear code $R \in \mathbb{Z}_2^l$ (with error correcting functionality), computes and sends the tuple

$$id, H(R), \mathsf{Enc}(R + A)$$

to the server, in which H is a hash function, and $\mathsf{Enc}(\cdot)$ is the encryption algorithm $\mathsf{Enc}(pk, \cdot)$ in Figure 1, and the addition $R + A$ is done over $\mathbb{Z}_p^l = \mathbb{Z}_2^l$. The server stores the above tuple.
- **Verification phase:** this consists of following two steps.
Computation at server: A user sends $id, \mathsf{Enc}(A')$ to the server. The server, using the identity id, restores $H(R)$ and $\mathsf{Enc}(R + A)$ and then computes as in (19)

$$CT = \mathsf{Enc}(R + A) + \mathsf{Enc}(A') \in \mathbb{Z}_q^{n+l}$$

which is $\mathsf{Enc}(R + A + A')$ by additive homomorphism. The server then sends id, $H(R)$, and the added ciphertext CT to the key manager.
Checking at key manager: The key manager, owning secret key sk, computes the decryption $\mathsf{Dec}_{sk}(CT)$ as in (20) to obtain

$$R' = R + A + A' \in \mathbb{Z}_p^l \ (= \mathbb{Z}_2^l)$$

and it checks $H(R) = H(\mathsf{EC}(R'))$, in which EC is the error correcting algorithm of the binary code. The key manager outputs OK only if the check passes.

Correctness. As seen above, if $A = A'$, then the verification will pass as $A + A'$ becomes the zero vector in $\mathbb{Z}_2^l$. Moreover, even if a few bits in A' are flipped, the verification will still pass due to the error correcting code, which captures the situation that some errors may occur in the representation of biometric information.

Passive security. On the security side, an honest-but-curious server will not obtain any information on the biometric representation, as all are stored encryptedly and the hash value $H(R)$ is random computationally. The information recovered at the key manager R' (and hence $R = \mathsf{EC}(R')$) is independent of any biometric templates and is deleted after the check. (We assume that the key manager is honest.)

An interesting feature of the protocol is as follows. Assuming the key manager is honest, yet sk is unfortunately leaked to an adversary who controls the database. Even in that case, no biometric template can be recovered. The reason is that the adversary can only see $H(R)$ and $R + A \in \mathbb{Z}_2^l$. As both $H(R)$ and R are independently random if the hash function H is "ideal", A is still perfectly hidden.

Comparisons. In Table 6, (q, n, s) decides the bit security of LWE, while l is the message length in bits. Since these values can be independent, and in particular n is smaller than l, the modulus q is also smaller compared to [31,32] (in which $q \approx 2^{61}$). Therefore, our size of ciphertexts, while relying on the LWE assumption, is half smaller than [31,32] (31 Kbytes, under the ring-LWE assumption). It is also worth noting that if Paillier encryption [27] is used to encrypt 2048 bits in a bit-by-bit manner, then the ciphertext size can be even worse as shown in [31].

The computation at the server, in our case is 4.56 (μs). In contrast, it is 5.31 (ms) in [31] so that we are more than 1000x faster. The reason is the protocol we use above do not require ciphertext multiplication at the server. The difference in speed shows that the *usage* of a primitive is as important as the primitive itself.

Table 6. Timing of computations and sizes in biometric authentication (128-bit security).

$q = 2^{32} - 1$, $n = 921$, $s = 8$, $p = 2$, $l = 2048$

Computation of (19) (Addition)	Computation of (20) (Decryption)
4.56 (μs)	2.15 (ms)

Enc.	Ciphertext	pk	sk
3.05 (msec)	12 Kbytes	11 Mbytes	1.18 Mbytes

Timings are averaged over 1000 executions using 1 thread on a Xeon E5-2660 v3, 2.60GHz machine. The sizes of ciphertexts and pk and sk are computed as $(n + l) \log_2 q$ and $n(n + l) \log_2 q$ and $nl \log_2(4s)$ in bits.

References

1. UCI Machine Learning Repository, `http://archive.ics.uci.edu/ml`.
2. `https://www.pcisecuritystandards.org/documents/Prioritized_Approach_V2.0.pdf`.
3. `https://www.owasp.org/index.php/Cryptographic_Storage_Cheat_Sheet`.
4. Y. Aono, X. Boyen, L. T. Phong, and L. Wang. Key-private proxy re-encryption under LWE. In G. Paul and S. Vaudenay, editors, *INDOCRYPT*, volume 8250 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2013.
5. W. Banaszczyk. New bounds in some transference theorems in the geometry of numbers. *Mathematische Annalen*, 296(1):625–635, 1993.
6. W. Banaszczyk. Inequalities for convex bodies and polar reciprocal lattices in $\mathbb{R}^n$. *Discrete & Computational Geometry*, 13(1):217–231, 1995.
7. D. Boneh, E. Goh, and K. Nissim. Evaluating 2-DNF formulas on ciphertexts. In J. Kilian, editor, *Theory of Cryptography, Second Theory of Cryptography Conference, TCC 2005, Cambridge, MA, USA, February 10-12, 2005, Proceedings*, volume 3378 of *Lecture Notes in Computer Science*, pages 325–341. Springer, 2005.
8. Z. Brakerski, C. Gentry, and V. Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. In S. Goldwasser, editor, *ITCS*, pages 309–325. ACM, 2012. Available at `https://eprint.iacr.org/2011/277.pdf`.
9. Z. Brakerski, A. Langlois, C. Peikert, O. Regev, and D. Stehlé. Classical hardness of learning with errors. In D. Boneh, T. Roughgarden, and J. Feigenbaum, editors, *STOC*, pages 575–584. ACM, 2013.
10. Z. Brakerski and V. Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In R. Ostrovsky, editor, *FOCS*, pages 97–106. IEEE, 2011.
11. J. Bringer and H. Chabanne. An authentication protocol with encrypted biometric data. In S. Vaudenay, editor, *Progress in Cryptology - AFRICACRYPT 2008, First International Conference on Cryptology in Africa, Casablanca, Morocco, June 11-14, 2008. Proceedings*, volume 5023 of *Lecture Notes in Computer Science*, pages 109–124. Springer, 2008.
12. N. Chandran, M. Chase, F. Liu, R. Nishimaki, and K. Xagawa. Re-encryption, functional re-encryption, and multi-hop re-encryption: A framework for achieving obfuscation-based security and instantiations from lattices. In H. Krawczyk, editor, *Public-Key Cryptography - PKC 2014 - 17th International Conference on Practice and Theory in Public-Key Cryptography, Buenos Aires, Argentina, March 26-28, 2014. Proceedings*, volume 8383 of *Lecture Notes in Computer Science*, pages 95–112. Springer, 2014.
13. C. Dwork and A. Roth. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, 9(3-4):211–407, 2014.
14. D. M. Freeman. Converting pairing-based cryptosystems from composite-order groups to prime-order groups. In H. Gilbert, editor, *Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, French Riviera, May 30 - June 3, 2010. Proceedings*, volume 6110 of *Lecture Notes in Computer Science*, pages 44–61. Springer, 2010.
15. C. Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009. `crypto.stanford.edu/craig`.
16. T. Graepel, K. E. Lauter, and M. Naehrig. ML confidential: Machine learning on encrypted data. In T. Kwon, M. Lee, and D. Kwon, editors, *Information Security and Cryptology - ICISC 2012 - 15th International Conference, Seoul, Korea, November 28-30, 2012, Revised Selected Papers*, volume 7839 of *Lecture Notes in Computer Science*, pages 1–21. Springer, 2012.

17. R. Hall, S. E. Fienberg, and Y. Nardi. Secure multiple linear regression based on homomorphic encryption. *Journal of Official Statistics*, 27(4):669, 2011.
18. D. E. Knuth and A. C. Yao. The complexity of non-uniform random number generation. *Algorithms and Complexity, Academic Press, New York*, pages 357–428, 1976.
19. K. E. Lauter, A. López-Alt, and M. Naehrig. Private computation on encrypted genomic data. In D. F. Aranha and A. Menezes, editors, *Progress in Cryptology - LATINCRYPT 2014 - Third International Conference on Cryptology and Information Security in Latin America, Florianópolis, Brazil, September 17-19, 2014, Revised Selected Papers*, volume 8895 of *Lecture Notes in Computer Science*, pages 3–27. Springer, 2014.
20. R. Lindner and C. Peikert. Better key sizes (and attacks) for LWE-based encryption. In A. Kiayias, editor, *CT-RSA*, volume 6558 of *Lecture Notes in Computer Science*, pages 319–339. Springer, 2011.
21. M. Liu and P. Q. Nguyen. Solving BDD by enumeration: An update. In E. Dawson, editor, *CT-RSA*, volume 7779 of *Lecture Notes in Computer Science*, pages 293–309. Springer, 2013.
22. V. Lyubashevsky, C. Peikert, and O. Regev. On ideal lattices and learning with errors over rings. *J. ACM*, 60(6):43, 2013.
23. P. Mell and T. Grance. The NIST definition of cloud computing. `http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf`.
24. M. Naehrig, K. E. Lauter, and V. Vaikuntanathan. Can homomorphic encryption be practical? In C. Cachin and T. Ristenpart, editors, *Proceedings of the 3rd ACM Cloud Computing Security Workshop, CCSW 2011, Chicago, IL, USA, October 21, 2011*, pages 113–124. ACM, 2011.
25. National Institute of Standards and Technology (NIST). Recommendation for Key Management: Part 1: General (Revision 3). `http://csrc.nist.gov/publications/nistpubs/800-57/sp800-57_part1_rev3_general.pdf`. Accessed: 2014, January 16.
26. V. Nikolaenko, U. Weinsberg, S. Ioannidis, M. Joye, D. Boneh, and N. Taft. Privacy-preserving ridge regression on hundreds of millions of records. In *2013 IEEE Symposium on Security and Privacy, SP 2013, Berkeley, CA, USA, May 19-22, 2013*, pages 334–348. IEEE Computer Society, 2013.
27. P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In J. Stern, editor, *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding*, volume 1592 of *Lecture Notes in Computer Science*, pages 223–238. Springer, 1999.
28. O. Regev. On lattices, learning with errors, random linear codes, and cryptography. In H. N. Gabow and R. Fagin, editors, *STOC*, pages 84–93. ACM, 2005.
29. R. L. Rivest, L. Adleman, and M. L. Dertouzos. On data banks and privacy homomorphisms. *Foundations of secure computation*, 4(11):169–180, 1978.
30. A. C.-C. Yao. How to generate and exchange secrets. In *Proceedings of the 27th Annual Symposium on Foundations of Computer Science*, SFCS '86, pages 162–167, Washington, DC, USA, 1986. IEEE Computer Society.
31. M. Yasuda, T. Shimoyama, J. Kogure, K. Yokoyama, and T. Koshiba. Practical packing method in somewhat homomorphic encryption. In J. García-Alfaro, G. V. Lioudakis, N. Cuppens-Boulahia, S. N. Foley, and W. M. Fitzgerald, editors, *Data Privacy Management and Autonomous Spontaneous Security - 8th International Workshop, DPM 2013, and 6th International Workshop, SETOP 2013, Egham, UK, September 12-13, 2013, Revised Selected Papers*, volume 8247 of *Lecture Notes in Computer Science*, pages 34–50. Springer, 2013.
32. M. Yasuda, T. Shimoyama, J. Kogure, K. Yokoyama, and T. Koshiba. Secure pattern matching using somewhat homomorphic encryption. In A. Juels and B. Parno, editors, *CCSW'13, Proceedings of the 2013 ACM Cloud Computing Security Workshop, Co-located with CCS 2013, Berlin, Germany, November 4, 2013*, pages 65–76. ACM, 2013.
33. J. Zhang, Z. Zhang, X. Xiao, Y. Yang, and M. Winslett. Functional mechanism: Regression analysis under differential privacy. *PVLDB*, 5(11):1364–1375, 2012.

A Proof of Theorem 5

We will use following lemmas, whose proofs can be derived from [5,6]. Below $\langle \cdot, \cdot \rangle$ stands for inner product. Writing $||\mathbb{Z}_s^n||$ is a short hand for taking a vector from the discrete Gaussian distribution of deviation s and computing its Euclidean norm.

Lemma 1. *Let $c \geq 1$ and $C = c \cdot \exp(\frac{1-c^2}{2})$. Then for any real $s > 0$ and any integer $n \geq 1$, we have*

$$\Pr\left[||\mathbb{Z}_s^n|| \geq \frac{c \cdot s\sqrt{n}}{\sqrt{2\pi}}\right] \leq C^n.$$

Lemma 2. *For any real $s > 0$ and $T > 0$, and any $x \in \mathbb{R}^n$, we have*

$$\Pr\left[||\langle x, \mathbb{Z}_s^n\rangle|| \geq Ts||x||\right] < 2\exp(-\pi T^2).$$

Proof (of Theorem 5). For now suppose $h = 2$, we check the decryption by secret key S_2 in dimension n_2 of updated ciphertexts. Using the notations as in Figure 2, let $E_0 = [E_1|E_2]$ and $F = [F_1|F_2]$ for $E_1, F_1 \in \mathbb{Z}_q^{1\times n_2}$, and $E_2, F_2 \in \mathbb{Z}_q^{1\times l}$, so $c' = E_0 + F = [E_1 + F_1|E_2 + F_2]$, and the decryption of c' by S_2 becomes

$$\begin{aligned}
&(E_1 + F_1)S_2 + E_2 + F_2\\
&= \big(f_1A_2 + pf_2 + \mathsf{Bits}(c_1)X\big)S_2 + f_1P_2 + pf_3\\
&\quad + \mathsf{Bits}(c_1)Y + c_2\\
&= f_1A_2S_2 + pf_2S_2 + \mathsf{Bits}(c_1)XS_2 + f_1(pR_2 - A_2S_2)\\
&\quad + pf_3 + \mathsf{Bits}(c_1)Y + c_2\\
&= pf_2S_2 + \mathsf{Bits}(c_1)XS_2 + pf_1R_2 + pf_3\\
&\quad + \mathsf{Bits}(c_1)\big(-XS_2 + pE + \mathsf{Power2}(S_1)\big) + c_2\\
&= pf_2S_2 + pf_1R_2 + pf_3\\
&\quad + p\mathsf{Bits}(c_1)E + \mathsf{Bits}(c_1)\mathsf{Power2}(S_1) + c_2\\
&= \underbrace{p(f_2S_2 + f_1R_2 + f_3 + \mathsf{Bits}(c_1)E)}_{\text{noise incurred after one Update}} + c_1S_1 + c_2 \in \mathbb{Z}_q^{1\times l}.
\end{aligned}$$

In the worst case, $\mathsf{Bits}(c_1)$ contains all 1's, so that the noise added after one update is

$$p(f_1R_2 + f_2S_2 + f_3 + \mathbf{1}_{1\times n_1\kappa} \cdot E).$$

Generally, the noise added after h updates corresponding to key dimension n_i $(2 \leq i \leq h)$ is a sum of form

$$p\sum_{i=2}^{h}\left(f_1^{(i)}R_2^{(i)} + f_2^{(i)}S_2^{(i)} + f_3^{(i)} + \mathbf{1}_{1\times n_i\kappa} \cdot E^{(i)}\right).$$

Each component in $\mathbb{Z}_q$ of the total noise in $\mathbb{Z}_q^{1\times l}$, namely including the noise in the original ciphertext, can be written as the inner product of two vectors of form

$$\begin{aligned}
e &= (f_1^{(2)}, f_2^{(2)}, f_3^{(2)} \quad, \ldots, f_1^{(h)}, f_2^{(h)}, f_3^{(h)},\\
&\quad e^{(2)}, \ldots, e^{(h)}, \quad e_1, e_2, e_3)\\
x &= (r_2^{(2)}, s_2^{(2)}, \mathbf{0}1\mathbf{0}_{1\times l}, \ldots, r_2^{(h)}, s_2^{(h)}, \mathbf{0}1\mathbf{0}_{1\times l},\\
&\quad \underbrace{\mathbf{1}_{1\times n_i\kappa}}_{2\leq i\leq h}, r, r', \mathbf{0}1\mathbf{0}_{1\times l})
\end{aligned}$$

where, for all $2 \leq i \leq h$,

– Vectors $f_1^{(i)}, f_2^{(i)} \xleftarrow{g} \mathbb{Z}_s^{1\times n_i}$, and $f_3^{(i)} \xleftarrow{g} \mathbb{Z}_s^{1\times l}$. Vectors $e^{(i)} \xleftarrow{g} \mathbb{Z}_s^{1\times n_i\kappa}$ represents one column in matrix $E^{(i)}$. Vectors $e_1 \xleftarrow{g} \mathbb{Z}_s^{1\times n_1}, e_2 \xleftarrow{g} \mathbb{Z}_s^{1\times n_1}$, and $e_3 \xleftarrow{g} \mathbb{Z}_s^{1\times l}$ are the noises in the original ciphertext.
– Vectors $r_2^{(i)}, s_2^{(i)} \xleftarrow{g} \mathbb{Z}_s^{1\times n_i}$, and $\mathbf{0}\mathbf{1}\mathbf{0}_{1\times l}$ stands for a vector of length l with all 0's except one 1; $\mathbf{1}_{1\times n_i\kappa}$ for a vector of length $n_i\kappa$ with all 1's. Vectors $r, r' \xleftarrow{g} \mathbb{Z}_s^{1\times n_1}$ represent corresponding columns in matrices R, S.

Here we use the same deviation s for all dimensions to ease the computation. We have

$$e \in \mathbb{Z}_s^{1\times\left(\sum_{i=1}^{h}(2n_i+l)+\sum_{i=2}^{h} n_i\kappa\right)}$$

$$||x|| \leq ||(r_2^{(2)}, s_2^{(2)}, \ldots, r_2^{(h)}, s_2^{(h)}, r, r')|| + \sqrt{\kappa\sum_{i=2}^{h} n_i + h}$$

where $(r_2^{(2)}, s_2^{(2)}, \ldots, r_2^{(h)}, s_2^{(h)}, r, r') \in \mathbb{Z}_s^{1\times\left(2\sum_{i=1}^{h} n_i\right)}$. Applying Lemma 1 for vector of length $2\sum_{i=1}^{h} n_i$, with high probability of

$$1 - C^{2\sum_{i=1}^{h} n_i} (\geq 1 - 2^{-40} \text{ for all choices of parameters})$$

we have

$$||x|| \leq \frac{c\cdot s\sqrt{2\sum_{i=1}^{h} n_i}}{\sqrt{2\pi}} + \sqrt{\kappa\sum_{i=2}^{h} n_i + h}.$$

We now use Lemma 2 with vectors x and e. Let ρ be the error per message symbol in decryption, we set $2\exp(-\pi T^2) = \rho$, so $T = \sqrt{\ln(2/\rho)}/\sqrt{\pi}$. The bound on the noise becomes $pTs||x||$, which is not greater than

$$\frac{ps\sqrt{\ln(2/\rho)}}{\sqrt{\pi}}\left(\frac{c\cdot s\sqrt{2\sum_{i=1}^{h} n_i}}{\sqrt{2\pi}} + \sqrt{\kappa\sum_{i=2}^{h} n_i + h}\right)$$

$$\stackrel{\text{def}}{=} B(\rho, h, s, n_1, \ldots, n_h, p, q). \tag{21}$$

No update ($h = 1$): (21) becomes

$$\frac{ps\sqrt{\ln(2/\rho)}}{\sqrt{\pi}}\left(\frac{c\cdot s\sqrt{2n_1}}{\sqrt{2\pi}}\right) = \frac{pcs^2\sqrt{\ln(2/\rho)\cdot n_1}}{\pi}.$$

which is the upper-bound of noise in each original ciphertext. If original ciphertexts are multiplied as in (3) and then the results are added N_{add} times, the corresponding noise bound is set below $q/2$ for correctness

$$\frac{N_{\text{add}} n_1 p^2 c^2 s^4 \ln(2/\rho)}{\pi^2} \leq \frac{q}{2}. \tag{22}$$

With update ($h \geq 2$): Now we consider (21) with $h \geq 2$. As in (22) we set

$$N_{\text{add}} \cdot B(\rho, h, s, n_1, \ldots, n_h, p, q)^2 \leq \frac{q}{2} \tag{23}$$

becomes the condition for correctness stated in the statement of the theorem, ending the proof. □

B Use of a ring-LWE-based scheme and some comparisons

We recall the known ring-LWE-based homomorphic encryption scheme (described in [16]) in Figure 6. The addition of ciphertexts can be done naturally as $\mathbf{c}+\mathbf{c}' \in R_q^2$. The multiplication is the tensor (aka, outer) product $\mathbf{c} \otimes \mathbf{c}' \in \mathbb{Z}_q^{n_{\text{rlwe}}^2}$ in which $\mathbf{c}$ and $\mathbf{c}'$ are seen as vectors in $\mathbb{Z}_q^{n_{\text{rlwe}}}$.

$\underline{\mathsf{ParamGen}(1^\lambda)}$:	$\underline{\mathsf{KeyGen}(1^\lambda, pp)}$:
Take:	Take $s = s(\lambda, pp) \in \mathbb{R}^+$
$q = q(\lambda) \in \mathbb{Z}^+$	$\mathbf{r}, \mathbf{s} \xleftarrow{g} R_s$, $\mathbf{a} \xleftarrow{\$} R_q$
$n_{\text{rlwe}} = n_{\text{rlwe}}(\lambda) \in \mathbb{Z}^+$	$\mathbf{p} = \mathbf{r} - \mathbf{as} \in R_q$
$p \in \mathbb{Z}^+$	Return $pk = (\mathbf{a}, \mathbf{p}, s)$
Return $pp = (q, n_{\text{rlwe}}, p)$	$sk = \mathbf{s}$
$\underline{\mathsf{Enc}(pk, \mathbf{m} \in R_p)}$:	$\underline{\mathsf{Dec}(\mathbf{s}, \mathbf{c} = (\mathbf{c}_1, \mathbf{c}_2))}$:
$\mathbf{e}_1, \mathbf{e}_2 \xleftarrow{g} R_s$, $\mathbf{e}_3 \xleftarrow{g} R_s$	$\overline{\mathbf{m}} = \mathbf{c}_1\mathbf{s} + \mathbf{c}_2 \in R_q$
$\mathbf{c}_1 = \mathbf{e}_1\mathbf{a} + \mathbf{e}_2 \in R_q$	$\mathbf{m} = \lfloor (p/q) \cdot \overline{\mathbf{m}} \rceil \bmod p$
$\mathbf{c}_2 = \mathbf{e}_1\mathbf{p} + \mathbf{e}_3 + \lfloor q/p \rfloor \mathbf{m} \in R_q$	Return $\mathbf{m} \in R_p$
Return $\mathbf{c} = (\mathbf{c}_1, \mathbf{c}_2) \in R_q^2$	

Fig. 6. The ring-LWE-based PKE in [16]. Ring $R = \mathbb{Z}[x]/f(x)$, $f(x) = x^{n_{\text{rlwe}}} + 1$, and $R_q = R/q$, $R_p = R/p$. The notion R_s stands for polynomials in R with small Gaussian coefficients.

Differing from our scheme in Figure 1, the plaintext space is $R_p = \mathbb{Z}_p[x]/(x^{n_{\text{rlwe}}} + 1)$ so that plaintexts are vectors of length n_{rlwe}. In other words, the parameter n_{rlwe} decides both the security and the plaintext size in the ring-LWE-based scheme. Therefore,

• (**Secure linear regression**) If using the above ring-LWE-based scheme in secure linear regression in Section 5.4, it is necessary to set the dimension n_{rlwe} to handle plaintexts in $\{-1, 0, 1\}^{(d+1+l_d)\cdot \mathsf{prec}}$ where $l_d = d(d+1)/2 + d + 1$ and $\mathsf{prec} = 64$. If the data dimension $d = 20$, the plaintext space becomes $\{-1, 0, 1\}^{16128}$. Naively setting $n_{\text{rlwe}} \approx 16128$ will yield big modulus q to ensure correctness, and hence long ciphertexts of $2n_{\text{rlwe}} \log_2 q$ bits.

A way to reduce the modulus q is to split the message length 16128 to k (e.g., $= 4$) pieces of size $\approx 16128/k$ bits. In this way, $n_{\text{rlwe}} \approx 16128/k$ which helps reducing q. However, as each piece of message is encrypted independently, we have k ciphertexts, each of which is of length $2(16128/k) \log_2 q$, so that the total length is $2k(16128/k) \log_2 q = 2 \cdot 16128 \log_2 q$. In contrast, in Section 5, the ciphertext size is $(n + l) \log_2 q$ where $l = 16128$ and $n \ll l$ so our LWE-based scheme gets a gain in size here.

Table 7. Our parameters vs. [31]'s.

	Message length (bits)	Dimension n_{rlwe} n_{lwe}	$\log_2 q$
[31]	2048	**2048**	≈ 61
Sect.6	2048	**950**	≈ 32

• (**Secure biometric authentication**) If using the ring-LWE-based scheme in the protocol of biometric authentication in Section 6 with biometric templates of size 2048 bits, n_{rlwe} must be set

to 2048 as in [31]. In turn, [31] took $\log_2 q \approx 61$ to ensure about 128-bit security. Each ciphertext is of $2n_{\text{rlwe}} \log_2 q$ bits, which is about 31 Kbytes as computed in [31]. In contrast, as in Table 7, we can take smaller dimension $n_{\text{lwe}} = 950$ thanks to the independence between the LWE's dimension and the message length in our scheme. These yield shorter ciphertexts as claimed in Section 6.

C Use of the Boneh-Goh-Nissim scheme and some comparisons

The BGN scheme originates in [7] using composite-order pairing groups. Its conversion to prime-order pairing group is in [14]. BGN has shorter public key than our scheme. Nonetheless, when using in our systems, it decreases the speeds of the server and the client and increases the ciphertext sizes, argued below. The discussions apply to both versions of BGN.

The message space in BGN encryption can be expressed as $\mathbb{Z}_p$ for some small p (to ensure that the discrete logarithm problem is easy). To encrypt $m = (m_1, \ldots, m_l) \in \mathbb{Z}_p^l$ using BGN, we need to encrypt element-wise, so that the ciphertext is

$$CT = (\mathsf{BGN.Enc}(m_1), \ldots, \mathsf{BGN.Enc}(m_l)) \in \mathbb{G}^l$$

where $\mathbb{G}$ is an elliptic curve over finite field $\mathbb{F}_q$. Therefore, ciphertext addition is done by l point additions over the elliptic curve, which is usually more than 10 multiplications in $\mathbb{F}_q$ (see e.g. `https://hyperelliptic.org/EFD/`). Assume that each multiplication is as fast as 3 additions over $\mathbb{F}_q$, we estimate that each ciphertext addition takes more than $3 \cdot 10 \cdot l = 30l$ additions over $\mathbb{F}_q$.

In contrast, in our scheme, as each ciphertext is a vector in $\mathbb{Z}_q^{n+l}$, each ciphertext addition needs $n + l$ additions over $\mathbb{Z}_q$. At 128-bit security level, $n = 3530$ in our scheme, and as $n + l \ll 30l$ if $l = 2048$ (biometric authentication) or 16128 (linear regression over high-dimensional data), we will not vote for the use of BGN in our systems for speed.

Let us consider ciphertext sizes. At 128-bit security level, encrypting one message in $\mathbb{Z}_p$ producing 3072 bits in the original BGN [7] and 1536 bits in the prime-order version [14]. As a result, CT above is of at least $1536l$ bits, which is $1536l/2^{13}$ kilobytes. When $l = 2048$ and 16128, the size of CT becomes 384 kilobytes and 3024 kilobytes respectively, which are much longer than ours (of 12 and 279 kilobytes respectively). Therefore, we will not vote for the use of BGN in the systems regarding ciphertext sizes.