Homomorphic Encryption and Confidential Computing

Kyle Yates

Clemson University

October 26, 2023

Cryptography

Alice wants to send Bob a message. She does the following.

1. Alice sends the message to Bob.
2. Bob receives and reads the message.

What if Alice doesn't want anyone else reading her message? An unwanted third party could intercept the transmission and read the message!

Instead, Alice will try this. Suppose Alice and Bob possess some shared information called a **secret key**, which only they know.

1. Alice disguises (**encrypts**) the message (**plaintext**) using the secret key.
2. Alice sends the encrypted message (**ciphertext**) to Bob.
3. Bob **decrypts** the message using the secret key.
4. Bob reads the message.

If a third party adversary intercepts the message, they can not read it without the secret key that only Alice and Bob possess!

Homomorphic encryption allows for **addition** and **multiplication** to be performed on ciphertexts without decrypting or possessing the secret key.

Furthermore, it ciphertext computations result in the same output as plaintext computations. That is, for messages m_0, m_1 and secret key k,

$$\mathtt{Enc}(m_0, \mathtt{k}) + \mathtt{Enc}(m_1, \mathtt{k}) = \mathtt{Enc}(m_0 + m_1, \mathtt{k})$$
$$\mathtt{Enc}(m_0, \mathtt{k})\mathtt{Enc}(m_1, \mathtt{k}) = \mathtt{Enc}(m_0 m_1, \mathtt{k})$$

Our goal today:

We are going to build a cryptosystem that allows us to do addition and multiplication on ciphertexts.

To cook this cryptosystem up, we need 4 ingredients:

1. Encryption Algorithm
2. Decryption Algorithm
3. Addition Algorithm
4. Multiplication Algorithm

Notation

Define $\mathbb{Z}_q$ as the ring of centered representatives $\mathbb{Z}_q := \mathbb{Z} \cap (-q/2, q/2]$.

When given an integer x, we denote $[x]_q$ as the reduction of x into the interval $\mathbb{Z}_q$ such that q divides $[x]_q - x$. When x is a polynomial or vector, $[x]_q$ means applying $[\cdot]_q$ to each coefficient.

We define R_n as the ring

$$R_n := \mathbb{Z}[x]/(\Phi(x))$$

where $\Phi(x)$ is an mth cyclotomic polynomial of degree n, a power of two. Namely, $\Phi(x) = x^n + 1$.

$$R_{n,q} := \mathbb{Z}_q[x]/(\Phi(x))$$

Notation

For any polynomial $f(x) = \sum_{i=0}^{k} a_i x^i$ with $a_i \in \mathbb{R}$, the *infinity norm of* $f(x)$ is defined as

$$\|f(x)\|_\infty = \max\{|a_0|, \ldots, |a_k|\}$$

therefore using centered representatives, for any $f(x) \in R_{n,q}$ we have $\|f(x)\|_\infty \leq q/2$.

The symbols $\lfloor \cdot \rfloor$ and $\lceil \cdot \rceil$ will denote floor and ceiling respectively, whereas $\lfloor \cdot \rceil$ will denote rounding to the nearest integer.

For a set S and a given probability distribution χ, we let $\chi(S)$ denote that distribution on S. We will denote $U(S)$ as a uniform distribution on S.

Basic Homomorphic Encryption

Three main schemes: BFV and BGV (exact), and CKKS (approximate).
We will focus on the **BFV** scheme.

Keys and Messages

We'll only look at the private key version of BFV, but a public key version is achievable with some simple modifications.

The secret key is chosen as some $\texttt{sk} = s \in R_{n,3}$, which is a polynomial with coefficients in $\{-1, 0, 1\}$.

The messages are of the form $m_0 \in R_{n,t}$, where t is a positive integer $\ll q$.

BFV Private Key Encryption

Consider a message $m_0 \in R_{n,t}$. We then encrypt the message using the secret key $\texttt{sk} = s \in R_{n,3}$, a constant $D = \lfloor q/t \rfloor \in \mathbb{Z}^+$, and a chosen parameter $\rho \in \mathbb{Z}^+$ as follows:

1. Sample $e_0 \leftarrow \chi(R_n)$ such that $\|e_0\|_\infty \leq \rho$.
2. Sample $a_0 \leftarrow U(R_{n,q})$.
3. Compute $b_0 \in R_{n,q}$ via $b_0 = -a_0 s + Dm_0 + e_0 \mod (\Phi(x), q)$.
4. Return $\texttt{ct}_0 = (a_0, b_0)$.

The ordered pair $\texttt{ct}_0 = (a_0, b_0) \in R_{n,q}^2$ is our BFV ciphertext.

Big Property: Our ciphertext $\mathtt{ct}_0 = (a_0, b_0) \in R_{n,q}^2$ satisfies the relationship

$$b_0 + a_0 s \equiv Dm_0 + e_0 \mod (\Phi(x), q)$$

Furthermore, knowing only (a_0, b_0) gives no information on s or m_0 based on the hardness of RLWE.

Before we move on, let's summarize...

- Our original message m_0 lives in $R_{n,t}$ (polynomials degree $< n$, coefficients in $\mathbb{Z}_t$).
- Our ciphertext $\mathtt{ct}_0 = (a_0, b_0)$ lives in $R_{n,q}^2$ (ordered pair of polynomials degree $< n$, coefficients in $\mathbb{Z}_q$). Note $q \gg t$.
- We have that $b_0 + a_0 s \equiv Dm_0 + e_0 \mod (\Phi(x), q)$.
- **Public Information:** $\mathtt{ct}_0 = (a_0, b_0)$, D, q, t, $\Phi(x)$
- **Private Information:** s, m_0, e_0

BFV Decryption

If we possess the secret key $\mathtt{sk} = s$, can we retrieve the message? The following gives the decryption algorithm.

1. Compute $m_0 = \left\lfloor \frac{[b_0+a_0 s]_{\Phi(x),q}}{D} \right\rceil$.
2. Return m_0.

Is this actually our m_0? Note that a BFV ciphertext has the relationship $b_0 + a_0 s \equiv Dm_0 + e_0 \mod (\Phi(x), q)$, so

$$\left\lfloor \frac{[b_0 + a_0 s]_{\Phi(x),q}}{D} \right\rceil = \left\lfloor \frac{Dm_0 + e_0}{D} \right\rceil = \left\lfloor m_0 + \frac{e_0}{D} \right\rceil = m_0 + \left\lfloor \frac{e_0}{D} \right\rceil$$

Decryption is correct **as long as** $\|e_0\|_\infty < D/2$.

We have cooked up a basic cryptosystem with these ingredients

1. Encryption Algorithm
2. Decryption Algorithm

But this dish doesn't have any *pizzazz*. We need to throw in some herbs and spices to cook up something a little bit better.

Operations (AKA the 11 secret herbs and spices)

We want a cryptosystem that is equipped with **addition** and **multiplication** operations in the ciphertext space.

We'll see now if we can find a way to throw these into the mix.

Addition

Goal: We want to construct an addition operation that doesn't need knowledge of s, or any other private information.

Given two BFV ciphertexts, $\mathtt{ct}_0 = (a_0, b_0)$ and $\mathtt{ct}_1 = (a_1, b_1)$ in $R_{n,q}^2$ satisfying

$$\begin{aligned} b_0 + a_0 s &\equiv Dm_0 + e_0 \mod (\Phi(x), q) \\ b_1 + a_1 s &\equiv Dm_1 + e_1 \mod (\Phi(x), q) \end{aligned}$$

We want to find a way to make a **new ciphertext** $\mathtt{ct}_2 = (a_2, b_2)$ that satisfies

$$b_2 + a_2 s \equiv D(m_0 + m_1) + \text{error}$$

Addition

Given two BFV ciphertexts, $\mathtt{ct}_0 = (a_0, b_0)$ and $\mathtt{ct}_1 = (a_1, b_1)$, addition can be done via

1. Compute $\mathtt{ct}_2 = \mathtt{ct}_0 + \mathtt{ct}_1 \mod q$.
2. Return $\mathtt{ct}_2$.

Two questions that we want to answer: *Does this actually work?* and *What happens to the error?*

Does this actually work?

Roughly speaking, we want our new ciphertext $\texttt{ct}_2 = (a_2, b_2)$ to satisfy the relationship

$$b_2 + a_2 s \equiv D(m_0 + m_1) + \text{error}$$

Recall $\texttt{ct}_2 \equiv \texttt{ct}_0 + \texttt{ct}_1 = (a_0 + a_1, b_0 + b_1)$. We have that

$$\begin{aligned}(b_0 + b_1) + (a_0 + a_1)s &\equiv (b_0 + a_0 s) + (b_1 + a_1 s) \mod (\Phi(x), q) \\ &\equiv (Dm_0 + e_0) + (Dm_1 + e_1) \mod (\Phi(x), q) \\ &\equiv D(m_0 + m_1) + e_0 + e_1 \mod (\Phi(x), q)\end{aligned}$$

Therefore,

$$b_2 + a_2 s \equiv D(m_0 + m_1) + e_0 + e_1 \mod (\Phi(x), q)$$

This is *almost* what we want, but we actually need to reduce $m_0 + m_1$ modulo t since the messages space is $R_{n,t}$.

Let $r \in R_{n,q}$ such that $m_0 + m_1 = [m_0 + m_1]_t + tr$ and $\epsilon = q/t - D$. Then,

$$\begin{aligned}
(b_0 + b_1) + (a_0 + a_1)s &\equiv D(m_0 + m_1) + e_0 + e_1 \mod (\Phi(x), q) \\
&\equiv D[m_0 + m_1]_t + e_0 + e_1 + Dtr \mod (\Phi(x), q) \\
&\equiv D[m_0 + m_1]_t + e_0 + e_1 - \epsilon tr \mod (\Phi(x), q) \\
&\equiv D[m_0 + m_1]_t + e_{\text{add}} \mod (\Phi(x), q)
\end{aligned}$$

where $e_{\text{add}} = e_0 + e_1 - \epsilon tr$. Then,

$$b_2 + a_2 s \equiv D[m_0 + m_1]_t + e_{\text{add}} \mod (\Phi(x), q)$$

What happens to the error?

In our new ciphertext, we obtain the error term

$$e_{\text{add}} = e_0 + e_1 - \epsilon tr$$

Let $\|e_0\|_\infty, \|e_1\|_\infty \leq E$. Recall that $r \in R_{n,q}$ such that $m_0 + m_1 = [m_0 + m_1]_t + tr$ and $\epsilon = q/t - D$. Therefore, $\|r\|_\infty \leq 1$ and $|\epsilon| \leq 1$, which gives

$$\begin{aligned} \|e_{\text{add}}\|_\infty &= \|e_0 + e_1 - \epsilon tr\|_\infty \\ &\leq \|e_0\|_\infty + \|e_1\|_\infty + \|\epsilon tr\|_\infty \\ &\leq 2E + t \end{aligned}$$

For BFV addition, the additive error has an extra growth of at most t per operation.

Multiplication

Multiplication is a little bit more involved. Here is the intuition behind it.

Given two BFV ciphertexts $\mathtt{ct}_0 = (a_0, b_0)$ and $\mathtt{ct}_1 = (a_1, b_1)$, recall they satisfy

$$\begin{aligned} b_0 + a_0 s &\equiv Dm_0 + e_0 \mod (\Phi(x), q) \\ b_1 + a_1 s &\equiv Dm_1 + e_1 \mod (\Phi(x), q) \end{aligned}$$

Step 1. Compute the following:

1. $c_0 = b_0 b_1 \mod (\Phi(x), q)$
2. $c_1 = b_1 a_0 + b_0 a_1 \mod (\Phi(x), q)$
3. $c_2 = a_0 a_1 \mod (\Phi(x), q)$

Why?

Let $r_m \in R_{n,q}$ such that $m_0 m_1 \equiv [m_0 m_1]_t + t r_m \mod \Phi(x)$. Now, we have

$$\begin{aligned}
c_0 + c_1 s + c_2 s^2 &\equiv b_0 b_1 + (b_1 a_0 + b_0 a_1)s + a_0 a_1 s^2 \mod (\Phi(x), q) \\
&\equiv (b_0 + a_0 s)(b_1 + a_1 s) \mod (\Phi(x), q) \\
&\equiv (D m_0 + e_0)(D m_1 + e_1) \mod (\Phi(x), q) \\
&\equiv D^2 m_0 m_1 + D(m_0 e_1 + m_1 e_0) + e_0 e_1 \mod (\Phi(x), q) \\
&\equiv D^2 [m_0 m_1]_t + D(m_0 e_1 + m_1 e_0) + e_0 e_1 + D^2 t r_m \mod (\ldots \\
&\equiv D^2 [m_0 m_1]_t + e^* \mod (\Phi(x), q)
\end{aligned}$$

This is the intuition to multiplication. However, there are two big issues with this:

- There is now a D^2 term rather than a D term in front of our message.
- The left hand side now uses s and s^2 rather than just s.

Converting D^2 to D.

The general idea here is to scale everything by $t/q \approx D$. Instead of using c_0, c_1, and c_2, we compute $c_0' = \lfloor tc_0/q \rceil, c_1' = \lfloor tc_1/q \rceil$, and $c_2' = \lfloor tc_2/q \rceil$. Then,

$$c_0' + c_1's + c_2's^2 \approx (t/q)(c_0 + c_1 s + c_2 s^2)$$

We also need to convert things into integer equations rather than equations modulo q for this step. Otherwise, scaling by t/q does not make sense!

The algebra to do this while maintaining a reasonably sized error term is ugly ugly.

$$(t/q)(c_0 + c_1 s + c_2 s^2)$$

$$\equiv (tD^2/q)[m_0 m_1]_t + (tD^2/q)2tr_m + (tD/q)(m_0 e_1 + m_1 e_0)$$
$$+ (tq/q)(e_0 r_1 + r_0 e_1) + (t/q)[e_0 e_1]_D + (tD/q)r_e$$
$$+ (tqD/q)(m_0 r_1 + r_0 m_1) + (tq^2/q)r_0 r_1 \mod \Phi(x)$$

$$\equiv ((q - r_t(q))D/q)[m_0 m_1]_t + ((q - r_t(q))D/q)2tr_m$$
$$+ ((q - r_t(q))/q)(m_0 e_1 + m_1 e_0) + (t)(e_0 r_1 + r_0 e_1)$$
$$+ (t/q)[e_0 e_1]_D + ((q - r_t(q))/q)r_e + (q - r_t(q))(m_0 r_1 + r_0 m_1)$$
$$+ (tq^2/q)r_0 r_1 \mod \Phi(x)$$

After much simplification, we can reduce our equations to the following

$$c_0' + c_1' s + c_2' s^2 \equiv D[m_0 m_1]_t + e^* \mod (\Phi(x), q)$$

where e^* is an error term given by

$$e^* = (m_0 e_1 + m_1 e_0) + t(e_0 r_1 + r_0 e_1) + r_e + (-r_t(q))(r_m + m_0 r_1 + r_0 m_1) + r_r - r_a$$

More importantly... The worst-case bound on e^* is given by:

$$\|e^*\|_\infty \leq 2\delta_R t E(1 + \delta_R \|s\|_\infty) + 2\delta_R^2 t^2 (\|s\|_\infty + 1)^2$$

Here, E is the bound on the original error terms e_0, e_1 and δ_R is the expansion factor in $R_{n,q}$. We will return to this bound later when we discuss error improvements.

Getting rid of s^2.

Remember, even after this we have something of the form

$$c_0' + c_1' s + c_2' s^2 \equiv D[m_0 m_1]_t + e^* \mod (\Phi(x), q)$$

when we need something of the form

$$\tilde{c_0} + \tilde{c_1} s \equiv D[m_0 m_1]_t + \tilde{e}^* \mod (\Phi(x), q)$$

Without going into detail: there are techniques to find this $\tilde{c_0}$ and $\tilde{c_1}$ by introducing a small amount of extra error.

Okay, let's pause for a moment and think about what we've done.

We built a cryptosystem with the following algorithms.

1. Encryption Algorithm
2. Decryption Algorithm
3. Addition Algorithm
4. Multiplication Algorithm

There are still plenty of questions to be answered. For instance, *what happens when our error gets too large? How can we deal with this?*

Modulus Reductions (Error Control)

For the following, let $D_Q = \lfloor Q/t \rfloor$ and $D_q = \lfloor q/t \rfloor$.

Input: $Q \in \mathbb{Z}^+$ an integer, $q \in \mathbb{Z}^+$ an integer, and $\mathtt{ct}_0 = (a_0, b_0) \in R_{n,Q}^2$ BFV ciphertext such that $b_0 + a_0 s \equiv D_Q m_0 + e_0 \mod (\Phi(x), Q)$.

1. Compute $a_0' = \lfloor \frac{q a_0}{Q} \rceil$ and $b_0' = \lfloor \frac{q b_0}{Q} \rceil$.
2. Return $\mathtt{ct}_0' = (a_0', b_0') \in R_{n,q}^2$ such that $b_0' + a_0' s \equiv D_q m_0 + e_{\mathrm{MR}}$ mod $(\Phi(x), q)$ for some e_{MR}.

We have the same two questions as before: *Does this actually work? What happens to the error?*

Let $\epsilon_Q = Q/t - D_Q$, $\epsilon_q = q/t - D_q$, $\epsilon_{a_0} = qa_0/Q - a_0'$, and $\epsilon_{b_0} = qb_0/Q - b_0'$. By assumption, we first note that $(a_0, b_0) \in R_{n,Q}$ is a BFV ciphertext. That is,

$$b_0 \equiv -a_0 s + D_Q m_0 + e_0 \mod (\Phi(x), Q)$$

Therefore, there is some integer $r_Q \in \mathbb{Z}$ such that $b_0 + a_0 s \equiv D_Q m_0 + e_0 + Qr_Q \mod \Phi(x)$. Then,

$$\begin{aligned} b_0' &= \frac{qb_0}{Q} - \epsilon_{b_0} \\ &\equiv -\frac{qa_0 s}{Q} + \frac{qD_Q}{Q} m_0 + \frac{qe_0}{Q} - \epsilon_{b_0} + qr_Q \mod \Phi(x) \end{aligned}$$

Note that as $D_Q = Q/t - \epsilon_Q$, we have that $qD_Q/Q = q/t - q\epsilon_Q/Q$. Since $q/t = D_q + \epsilon_q$, we have $qD_Q/Q = D_q + \epsilon_q - q\epsilon_Q/Q$.

Therefore,

$$\begin{aligned}
b_0' &\equiv -\frac{qa_0 s}{Q} + \frac{qD_Q}{Q} m_0 + \frac{qe_0}{Q} - \epsilon_{b_0} + qr_Q \mod \Phi(x) \\
&\equiv -a_0' s + \epsilon_{a_0} s + D_q m_0 + (\epsilon_q - \frac{q\epsilon_Q}{Q}) m_0 + \frac{qe_0}{Q} - \epsilon_{b_0} + qr_Q \mod \Phi(x) \\
&\equiv -a_0' s + D_q m_0 + e_{\mathrm{MR}} \mod (\Phi(x), q)
\end{aligned}$$

where

$$e_{\mathrm{MR}} = \frac{qe_0}{Q} + (\epsilon_q - q\epsilon_Q/Q) m_0 - \epsilon_{b_0} + \epsilon_{a_0} s$$

Therefore, $b_0' + a_0' s \equiv D_q m_0 + e_{\mathrm{MR}} \mod (\Phi(x), q)$.

Reducing the modulus reduces the error!

At the end of the day, we get an error bound of the following for our new ciphertext after modulus recduction.

$$\|e_{\mathrm{MR}}\|_\infty \leq \frac{q}{Q}E + \frac{t+1}{2} + \frac{\delta_R \|s\|_\infty}{2}$$

Here, E is our initial error bound and $Q > q$. We also have $Q, q, t, \delta_R, \|s\|_\infty$ as either constants or predetermined parameters.

Improvements and Results

Lemma

Suppose that $t|q-1$ and $\delta_R \geq 16$. For two BFV ciphertexts with modulus q and error bound E, their product has error e_{mult} satisfying

$$\|e_{mult}\|_\infty \leq 4Et\delta_R^2 \|s\|_\infty^2$$

Lemma

Suppose we have a BFV ciphertext with error bounded by E. Let e_{MR} be the error term resulting from performing a modulus reduction from Q to q. If $t|Q-1$, $t|q-1$, and $Q/q > \frac{2E}{\delta_R\|s\|_\infty - 2}$, then $\|e_{\mathrm{MR}}\|_\infty < \delta_R \|s\|_\infty$.

Lemma

Suppose $q_i > 10kt\delta_R^2 \|s\|_\infty^2$. Then, for two vectors of ciphertexts $\mathbf{u} = (u_1, \ldots, u_k) \in (R^2_{n,Q_i})^k$ and $\mathbf{v} = (v_1, \ldots, v_k) \in (R^2_{n,Q_i})^k$, each with error bounded by $\delta_R \|s\|_\infty$, we can choose to homomorphically compute exactly one of the following:

1. $\langle \mathbf{u}, \mathbf{v} \rangle$, *or*
2. $(\sum u_i)(\sum v_i)$

Furthermore, reducing the modulus by q_i immediately after this computation always results in a ciphertext with error bounded by $\delta_R \|s\|_\infty$.

For the optimized case, this is just $q_i > 10ktn^2$.

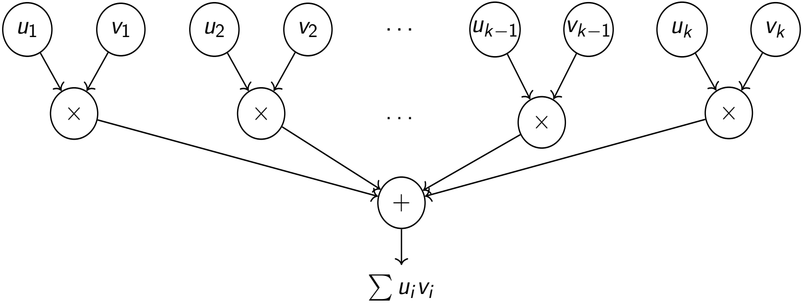


Figure: Homomorphic inner product for each q_i

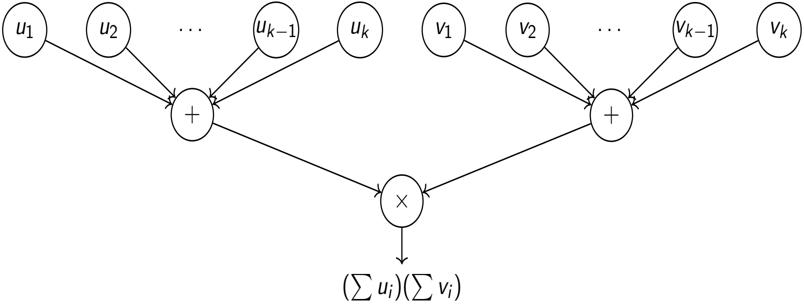


Figure: Homomorphic product of sums for each q_i

The big question...

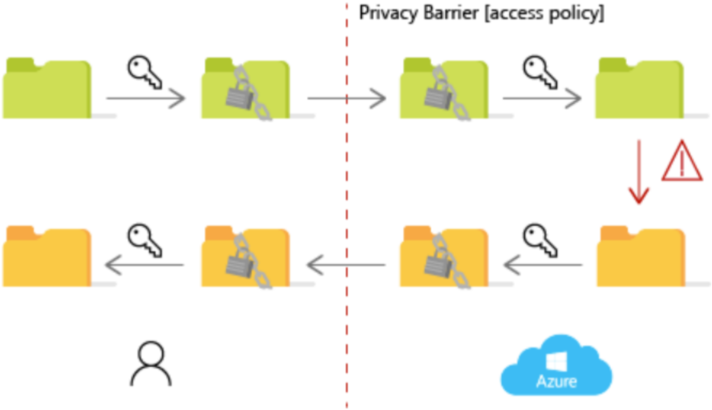
WHO CARES?

Secure Cloud Computing

- Cloud computing has become increasingly popular
    - Amazon Web Services (AWS), Microsoft Azure, Google Cloud Platform (GCP)
- AWS - Encryption services provided in transit and at rest.
- Homomorphic encryption - maintain encryption during computation.

Secure Cloud Computing

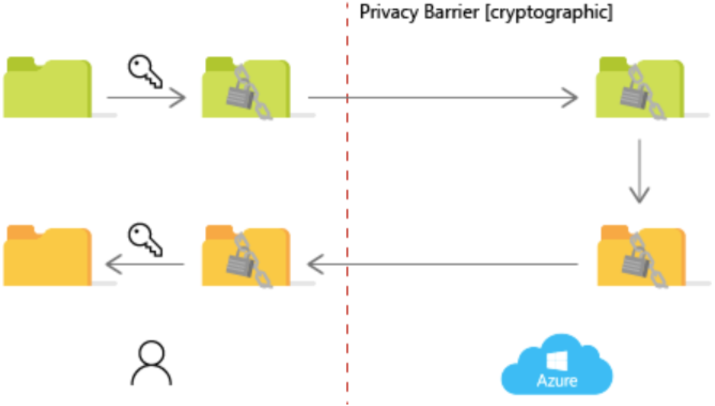
Traditional cloud storage and computation



https://www.microsoft.com/en-us/research/project/microsoft-seal/

Secure Cloud Computing

Microsoft SEAL cloud storage and computation



https://www.microsoft.com/en-us/research/project/microsoft-seal/

Secure Multi-party Computation

1. Suppose there are N parties, each holding a private share. The parties want to jointly compute an outcome of their combined shares, while keeping their individual share secret.
2. Example: E-voting schemes.
3. Naturally, homomorphic encryption has close ties and applications in multi-party computation.

Open Source Libraries (C++)

- PALISADE Library

  `https://palisade-crypto.org/`
- Microsoft SEAL

  `https://www.microsoft.com/en-us/research/project/microsoft-seal/`
- HElib (IBM)

  `https://homenc.github.io/HElib/`
- OpenFHE

  `https://www.openfhe.org/`
- HEAAN

  `https://github.com/snucrypto/HEAAN`

This just scratches the surface of homomorphic encryption. Other content includes:

-Chinese Remainder Theorem. Break down Q into individual coprime pieces and do operations component-wise. Tricky part here is doing multiplication computations that we did in $\mathbb{Q}$ or $\mathbb{R}$.

-Number theoretic transforms and Fast Fourier Transforms for actual ciphertext computation.

-Approximate Homomorphic Encryption.

-Non-arithmetic operations (ie, sgn function).

-Bootstrapping (modulus raising).

Thank You!