Non-interactive inner products in privacy-preserving biometric identification using secure multiparty computation

Introduction

Inner products are fundamental operations in privacy-preserving applications. This report describes an implementation trial of LWE (learning with errors) where inner products are part of. In addition to this report along the code implementation project based on (Couteau & Zarezadeh, Non-Interactive Secure Computation of Inner-Product from LPN and LWE, 2023) my submission for the "Secure and Private Computing" 3 ECTS course at HSG also includes a presentation about biometric identification, a presentation about the status of the code and various reviews on other papers that were presented by peers.

Non-Interactive Key Exchange

When two parties decide to communicate among each other, they first must compute shared private key K. Let's make an example how two parties (Alice and Bob) can get a shared private key K:

$$Alice\ uses\ pk_{Bob}, sk_{Alice}\ to\ compute\ K = Key(pk_{Bob}, sk_{Alice})$$

$$Bob\ uses\ pk_{Alice}, sk_{Bob}\ to\ compute\ K = Key(pk_{Alice}, sk_{Bob})$$

The great thing about this interaction pattern is that it avoids n^2 pairwise key exchange among all the parties. Small n refers to the number of parties being involved. So, we have just n public keys. Please refer to Figure 1 for clarification.

Non-interactive secure computing

It would be nice if there exist a pattern to securely exchange encodings that works between n parties. Parties are broadcasting an encoding of their input:

$$Alice \rightarrow Enc(x_1)$$

$$Bob \rightarrow Enc(x_4)$$

Two parties can than compute $f_i(x_i, x_j)$ and $f_j(x_i, x_j)$ from their own state plus the other party's encoding.

$$Alice \rightarrow f_1(x_1, x_4) = Out(sk_1, Enc(x_4))$$

$$Bob \rightarrow f_1(x_1, x_4) = Out(sk_4, Enc(x_1))$$

This interaction pattern avoids as above n^2 pairwise key exchange among all the parties. Please refer again to Figure 1 for clarification.

Non-interactive multi party computation (MPC)

The goal of the implementation is to do non-interactive MPC for shares of inner products. So instead of encoding some x_i or x_j, shares of $< x_i, x_j > over \mathbb{F}$ are calculated.

Protocol Description

Here some questions and answers to understand the protocol which will be implemented. Please note that all pictures in this chapter are from slides from (Couteau, Geoffroycouteau, 2022) and the explanations are founded on (Couteau, Youtube, 2022).

1. What does the protocol do?
   The protocol is used to exchange shares of vectors among different parties. Assuming there are two parties, Bob and Alice, each holding a vector, x_1 and x_4. Then this protocol can be used to securely exchange shares of the locally computed vector encoding.

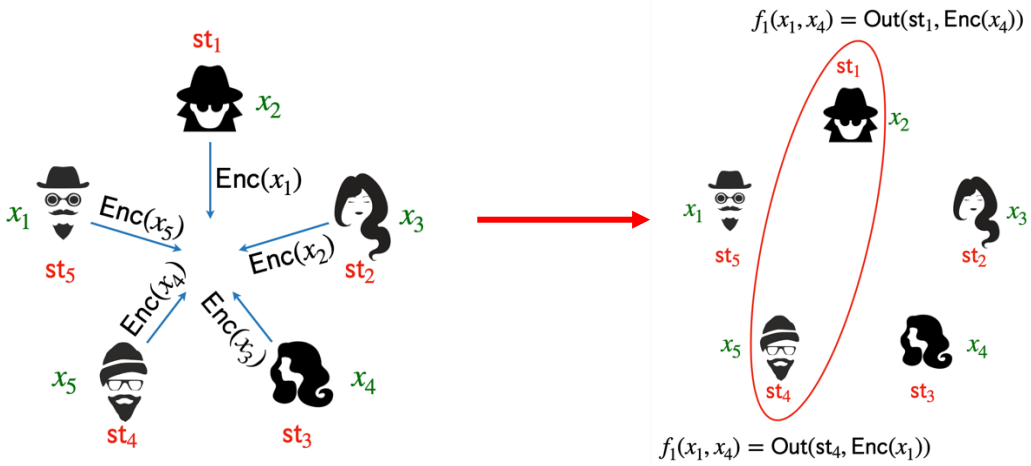

Figure 1: Non-Interactive-Key Exchange and non-interactive secure computing

2. How does the protocol work?
   The protocol's exact mechanism is quite intricate. It is based on Alekhnovich's key exchange (AKE), which inherently involves an inner product calculation. Can this framework embed an inner product computation between x_1 and x_4 within AKE? Absolutely! The protocol operates by performing an inner product computation along with some noise term e. Essentially, it enables a secure inner

product computation layered on top of AKE, which itself relies on inner product computation. Please refer to Figure 2.
Additionally, it supports the homomorphic property. This means that encrypting the product of two elements is feasible when one element is in plaintext, and the other is encrypted. For instance, consider the following equation where x is in plaintext and n is an integer value.

$$Enc(x)^n = Enc(n * x)$$

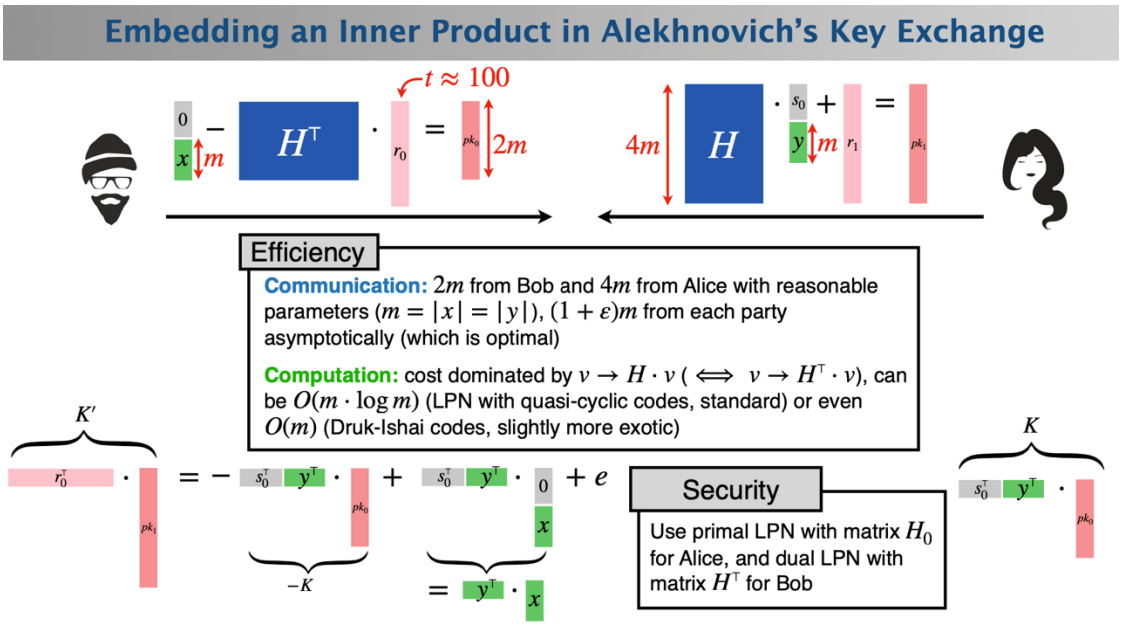

Figure 2: Alekhnovich's key exchange (AKE)

The correctness of the protocol can be demonstrated by examining AKE. Specifically, when $t^2/n << 1$ (where t denotes the Hamming weight and n the dimension of a vector), it holds true. In MPC, correctness errors can lead to information leakage when a "detectable" error occurs, though the system remains secure albeit with some loss. However, using LWE makes the error negligible, which is why LWE is preferred. Please refer to Figure 3. It's important to note that **security** is guaranteed only with appropriate parameters. Due to the challenges and complexities in determining the correct parameters for security, the focus was placed on **correctness**. Selecting LWE parameters is a nontrivial task, as there isn't a straightforward formula for it. A script is needed to estimate runtime against state-of-the-art cryptoanalysis.

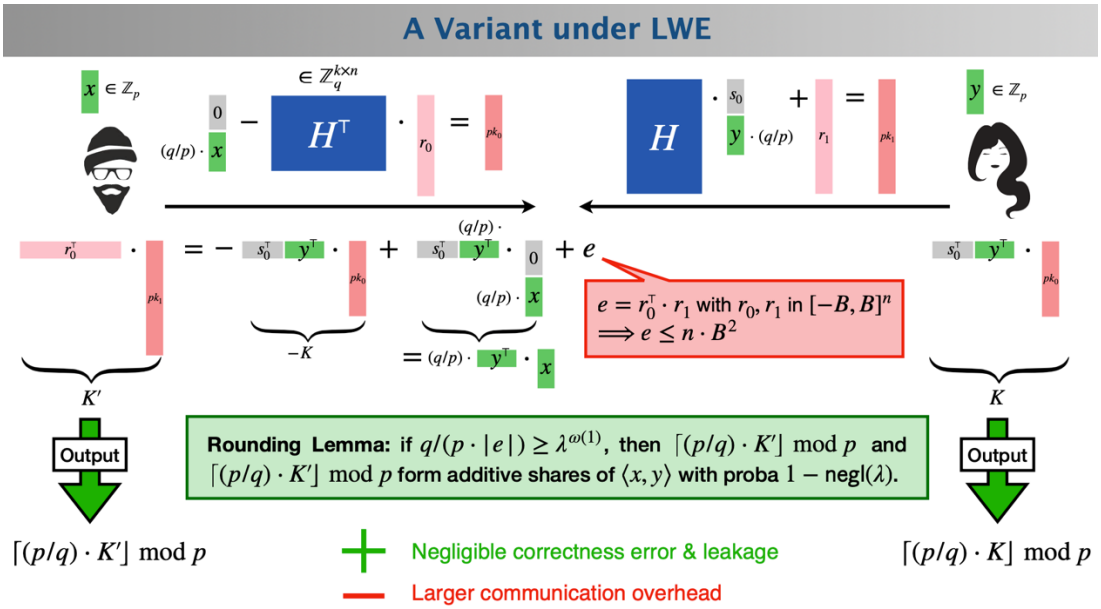

Figure 3: Learning with Error (LWE)

3. Where can such a protocol be used?
   It can for example be used to exchange securely fingerprint, iris scans, any other biometric data or in general secured data. Another example would be authentication from a user to a server as can be depicted from Figure 4.

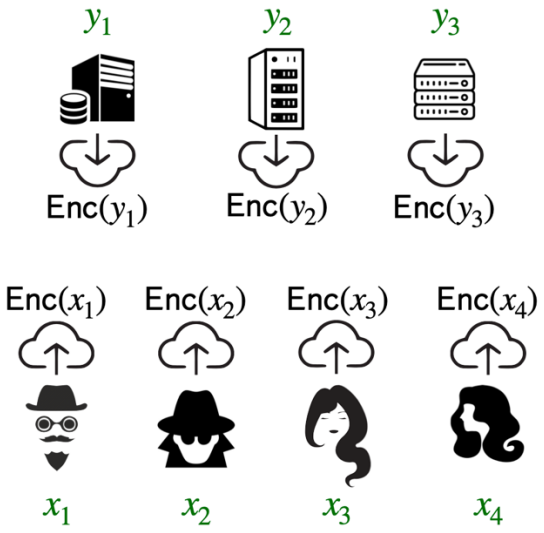

Figure 4: Usage of protocol

4. How can it be used for biometric identification?
   By combining LWE with MPC, biometric identification can be performed in a way that respects the privacy and security of the individual's sensitive biometric data, while still allowing for accurate and secure comparisons. The protocol involves computing the inner product between the stored encrypted biometric data and

the newly provided biometric data (also encrypted). The LWE scheme's homomorphic properties facilitate this computation without revealing the actual data. By relying on cosine similarity for example as stated in (Ernst & Mitrokotsa, 2023) on page 18, biometric matching can easily be computed as an inner product.

5. What advantage does the protocol provide (over other protocols) for biometric identification?
   It provides post quantum security. It is very efficient. The error tolerance makes the protocol more robust. Ahead of time, each party can publish the shares of inner product. That makes the protocol more efficient.

Implementation Overview

Refer to https://github.com/taschoebli/SPC_implementation_spring_24 to find the implementation code. Start in lwe.py file.

Table 1: Parameter description for implementation

Security parameter	λ
Upper case letters denote matrices	M
Bold lower-case letters denote row vectors	$\boldsymbol{v}$
Transpose on row vectors denote column vectors	$\boldsymbol{v}^T$
Horizontal concatenation of horizontal vectors	$\boldsymbol{u}^T \| \| \boldsymbol{v}^T$
Vertical concatenation of vertical vectors	$\boldsymbol{u} // \boldsymbol{v}$
x is uniformly sampled from the set X	$x \xleftarrow{\$} X$
Finite set S uniform distribution is denoted by	$\mathbb{U}(S)$
Bernoulli distribution with parameter	Ber_τ

In Table 1 the parameters are described. Please note: $e \sim Ber_\tau$ means that the random variable e evaluates to 1 with probability τ and to 0 with probability $1 - \tau$. More generally, $Ber_\tau(\mathbb{F})$ denotes the distribution that outputs a uniformly random element of $\mathbb{F}$ with probability τ, and 0 otherwise.

According to (Couteau & Zarezadeh, Non-Interactive Secure Computation of Inner-Product from LPN and LWE, 2023) in section 3, there exists a "general non-interactive protocol for securely computing the inner product between two vectors over $\mathbb{F}^n$, with ε correctness error (independent of the value of the inputs)."The great thing is that the protocol can either be instantiated under LPN (learning parity with noise, the error will be noticeable but arbitrary small) or under the LWE (learning with errors, the error can be made negligible) assumption. As described above and depicted in Figure 3 focus will be on LWE.

But what data is been used? The tutors provided two arrays (.npy) with shape 13'232x128 (data) and 13'232x1 (labels). After having quickly looked at the arrays and with some research done, it turns out that the data must be LFW (labeled faces in the wild). The LFW dataset contains 13,233 images of faces collected from the web. This dataset consists of the 5749 identities with 1680 people with two or more images.[1]

Implementation in Detail

Implementation is done according to (Couteau & Zarezadeh, Non-Interactive Secure Computation of Inner-Product from LPN and LWE, 2023) in section 3.3 (non-interactive inner product from LWE) as described in Figure 5.

$\mathsf{Setup}(1^\lambda)$: sample $H \xleftarrow{\$} \mathsf{Code}^{\perp}(m, k+n, \mathbb{Z}_q)$ and output $\mathsf{crs} = H$.
$\mathsf{Encode}(\mathsf{crs}, b, \mathbf{u})$: parse crs as H and sample $\mathbf{r}_b \xleftarrow{\$} \chi^m$. If $b = 0$, output $\mathsf{pk}_0 \leftarrow (q/p) \cdot (\mathbf{u} /\!\!/ \mathbf{0}) - H^\intercal \cdot \mathbf{r}_0$ and $\mathsf{sk}_0 \leftarrow \mathbf{r}_0$. If $b = 1$, sample $\mathbf{s} \xleftarrow{\$} \mathbb{Z}_q^k$, and output $\mathsf{pk}_1 \leftarrow H \cdot (\mathbf{u} /\!\!/ \mathbf{s}) + \mathbf{r}_1$ and $\mathsf{sk}_1 \leftarrow (\mathbf{u} /\!\!/ \mathbf{s})$.
$\mathsf{Decode}(\mathsf{crs}, \mathsf{pk}, \mathsf{sk}')$: output $\lceil (p/q) \cdot \mathsf{pk}^\intercal \cdot \mathsf{sk}' \rfloor \bmod p$.

Figure 5: LWE-based non-interactive inner-product

So the first step is to implement the setup method. The output will be H. Setup is normally done by an trusted party but here it is done in directly in the code. Dimension of H is $\mathbb{F}^{m\ x\ k+n}$. Actually it is an matrix with the dimension 512x384 as listed in Table 2. The security of the setup depends on λ. In Figure 6 the implementation of the setup method is depicted. "low" translates to λ whereas "row" and "col" define the dimension of H. Please note that setup is simplified in order to have a running implementation.

```
6 usages   Luzi Schoeb
def setup(low, row, col):
    return np.random.randint(low, size=(row, col))
```

Figure 6: Setup method

Table 2: Parameter values

Description	Parameter	Value
Generator matrix of linear code	H	512x384
Dimension, eg. row dimension of u	n	128
Dimension	m	4n
Dimension	k	3n
Dimension	q	7793
Dimension	p	3329

[1] https://paperswithcode.com/dataset/lfw

In two further steps, encode and decode are implemented. Plese refer directly to the code. The methods are located in the same file as the setup method. Please note that some matrices had to be padded with zeros or random values according to the describtion of the algorithms in Figure 5.

Validation 1

So the next step will be to apply the encode and decode algorithms accordingly.

Encode takes an input vector $\mathbf{x} \in \mathbb{F}^n$ over some field $\mathbb{F}$, and produces a pair $(\mathsf{pk}_\mathbf{x}, \mathsf{sk}_\mathbf{x})$. $\mathsf{pk}_\mathbf{x}$ is the *public encoding*, and $\mathsf{sk}_\mathbf{x}$ is the secret state. All parties can publicly reveal the encodings $\mathsf{pk}_\mathbf{x}$, since they computationally hide their vectors $\mathbf{x}$.
Decode takes as input a public encoding $\mathsf{pk}_\mathbf{x}$, and a secret state $\mathsf{sk}_\mathbf{y}$, and outputs a value z, such that the following holds: $z = \mathsf{Decode}(\mathsf{pk}_\mathbf{x}, \mathsf{sk}_\mathbf{y})$ and $z' = \mathsf{Decode}(\mathsf{pk}_\mathbf{y}, \mathsf{sk}_\mathbf{x})$ form *additive shares* of the inner product $\mathbf{x}^\intercal \cdot \mathbf{y} = z + z'$ over $\mathbb{F}$.

Figure 7: Application of Encode and Decode

Let's assume we have two parties, X and Y. Now calucaltion of pk_x, sk_x, pk_y and sk_y is performed with Encode function.

We want to validate if $x^T * y = z + z'$. If so, we are done! Please refer to Figure 7

Validation 2

Another possible way to validate according to (Couteau & Zarezadeh, Non-Interactive Secure Computation of Inner-Product from LPN and LWE, 2023) from page 9 as depicted in Figure 8.

$$\mathsf{pk}_0^\intercal \cdot \mathsf{sk}_1 + \mathsf{pk}_1^\intercal \cdot \mathsf{sk}_0 = (q/p) \cdot \mathbf{u}_0^\intercal \cdot \mathbf{u}_1 + \mathbf{r}_1^\intercal \cdot \mathbf{r}_0,$$

Figure 8: Validation as in LPN (Learning Parity with Noise)

In Figure 9 the validation part of the implementation is shown. Unfortunately, both validations are not correct and therefore weather correctness nor security could have been proven. The simplification of the setup method and all the parameters (q and p) helped to have a running implementation. In order to be able to continue, help was asked for from the author of the paper. Here the paraphrased answer:

"Firstly, regarding security, I must acknowledge that I am not an expert in LWE-based cryptography. Thus, for selecting appropriate LWE parameters, I recommend consulting with experts in the field. Tools like LWE parameter estimators are available, with Albrecht's estimator being a commonly used option (accessible here: https://bitbucket.org/malb/lwe-estimator/src/master/). It's important to note that choosing LWE parameters is complex and doesn't follow a straightforward formula.

Instead, scripts that estimate runtime based on the latest cryptanalysis techniques should be used.

For correctness, the primary constraint is derived from the rounding lemma (Lemma 5). For the rounding process to be effective, the expression $(|e| + 1) * p / q$ must be negligible, where $|e|$ represents a bound on the error term in the inner product, calculated as $m * B^2$ (with B being the bound on the maximum value of any entry in $r0$ and $r1$). Since this relates to statistical failure probability, a practical approach is to ensure that $m * B^2 * p / q$ is less than 2^{-40}.

The parameter p can be chosen freely, as it is the field over which inner products are computed. In Section 3.5, I recommend setting m = 3n (where n is the vector size) as a balanced choice, though other values may also be viable. Subsequently, the values of B and q should be determined using an LWE parameter estimator, ensuring that the condition $m * B^2 * p / q$ is less than 2^{-40}."

It would need further investigation in this interesting topic. Maybe this could be achieved in the IMP (integrative masters project) or the master thesis itself.

```python
# Setup phase
H = setup(q, m, (k + n))  # Generator matrix

# Get two values, Colin Powell at indices 0 and 133
x = data_embeddings[:1]
print(f"{data_labels[0]}: {x}")
y = data_embeddings[133:134]
print(f"{data_labels[133]}: {y}")

# Encode phase
pk_x_0, sk_x_0 = encode(x, b: 0)
pk_x_1, sk_x_1 = encode(x, b: 1)
pk_y_0, sk_y_0 = encode(y, b: 0)
pk_y_1, sk_y_1 = encode(y, b: 1)

# Decode phase
z = decode(pk_x_0, sk_y_1)
z_prime = decode(pk_y_1, sk_x_0)

# Variant 1 Validation primitive according to page 2
product_x_y = np.dot(x.T, y)
sum_z_z_prime = z + z_prime
print(f"Product x.T * y: {product_x_y}")
print(f"Sum z + z': {sum_z_z_prime}")

# Variant 2 Validation of correctness according to page 9
left_side = np.dot(pk_x_0.T, sk_y_1) + np.dot(pk_y_1.T, sk_x_0)
temp_a = ((q / p) * np.dot(x.T, y))

r0 = setup(q, m, (m - k))  # see section 2.1, dimension is m x (m-k)
r1 = setup(q, m, (m - k))  # see section 2.1, dimension is m x (m-k)
temp_b = np.dot(r1.T, r0)
right_side = temp_a + temp_b
delta = left_side - right_side
delta_abs = np.abs(delta)
print(f"Delta Error: {delta_abs}")
```

Figure 9: Implementation with validation part

Sources

Couteau, G. (2022, November 01). *Geoffroycouteau*. Récupéré sur Github: https://geoffroycouteau.github.io/presentations/

Couteau, G. (2022, December 07). *Youtube*. Récupéré sur Asiacrypt 2022 Session on Multiparty Computation 1: https://youtu.be/FwmEffEc2-Y?feature=shared&t=1435

Couteau, G., & Zarezadeh, M. (2023). Non-Interactive Secure Computation of Inner-Product from LPN and LWE. *Cryptology ePrint Archive*.

Ernst, J., & Mitrokotsa, A. (2023). A Framework for UC Secure Privacy Preserving Biometric Authentication using Efficient Functional Encryption. *Cryptology ePrint Archive*.