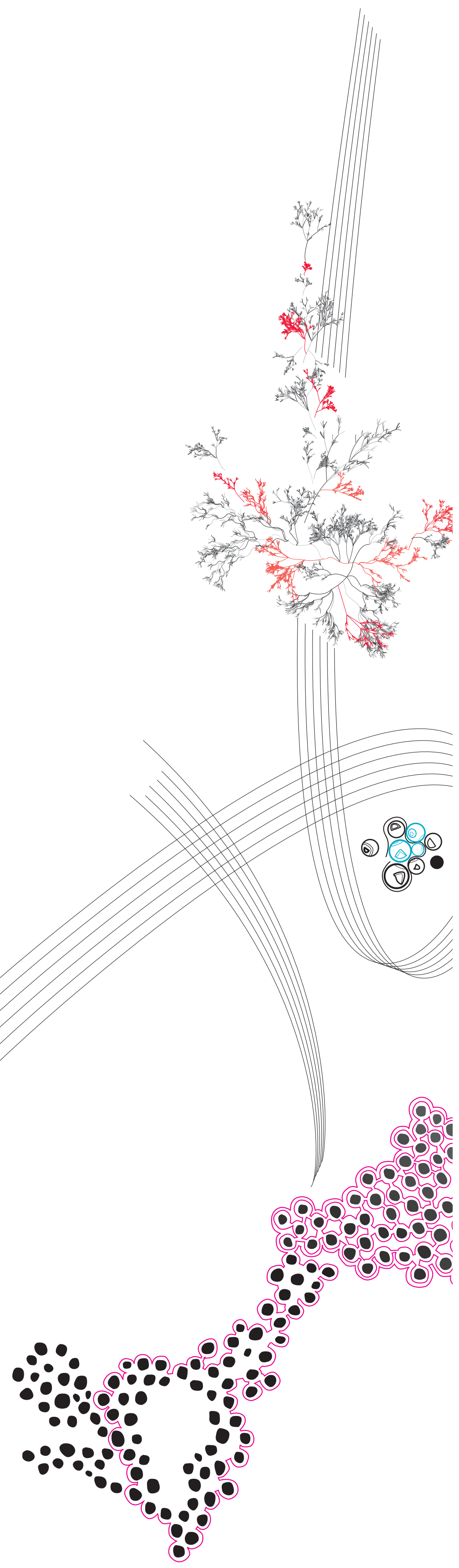

My delighted master experience

Assignment 2 - SMPC & HE

M.M.A.P. van Iperen s2386658
B. Ifkovics s3445674

Group: group 3

Course: PET bootcamp
Course code: 202300046

January, 2025

Faculty of Electrical Engineering,
Mathematics and Computer Science

UNIVERSITY OF TWENTE.

Part A: Weighted Sum Protocol

Approach and Implementation

The weighted sum protocol uses additive secret sharing over a finite field. The protocol allows n parties to compute $\sum_{i=1}^{n} \alpha_i \cdot x_i$ without revealing individual inputs.

Protocol Design

The implementation uses a network-based communication model where each party runs as a separate process. The first process acts as the master node for coordination. Each party's input is protected by distributing random shares that reveal no information about the secret value. However, the protocol is vulnerable to collusion attacks, where n-1 parties combine their shares, and reconstruct the remaining party's input.

Security protocol

Each party P_i with input x_i performs the following steps:

Setup

- A prime modulus $p = 2^{31} - 1$ (Mersenne prime) is established
- Public weights α_i are distributed to all participants
- Each party generates their secret input x_i

Share Generation

- Party P_i splits x_i into n shares: $x_i = \sum_{j=1}^{n} s_{ij} \bmod p$
- Shares s_{ij} are generated randomly except the last one
- Each share s_{ij} is sent to party P_j

Local Computation

- Each party P_i receives shares s_{ji} from all parties
- Computes partial sum: $y_i = \sum_{j=1}^{n} \alpha_j \cdot s_{ji} \bmod p$

Result Reconstruction

- All parties broadcast their partial sums y_i
- Final result is computed as $\sum_{i=1}^{n} y_i \bmod p$

We included a script that orchestrates processes to make testing easier. Otherwise, the first process orchestrates the communication. Alice and Bob's functionality is defined in one file for easier development and consistency in participant executables. Further instructions are detailed in the README files.

Part B: Secure Inner Product Computation

The secure inner product protocol combines Paillier homomorphic encryption with secret sharing using Beaver triplet.

Security protocol

The protocol combines Paillier homomorphic encryption with secret sharing using Beaver triplets to compute secure inner products between two parties (Alice and Bob) without revealing their input vectors.

Setup

Alice generates Paillier keypair (n, g), shares public key, and keeps private parameters. Both parties generate vectors corresponding in size to the inputted dimension parameter.

Beaver Triplet Generation

For each component i in $\vec{a}$ (Alice) and $\vec{b}$ (Bob):

1. Alice generates random x_{Ai}, $y_{Ai} \mod n$, encrypts them and sends them to Bob
2. Bob generates random x_{Bi}, $y_{Bi} \mod n$, using homomorphic properties, he then calculates: $[\![x_{Ai} \cdot y_{Bi} + y_{Ai} \cdot x_{Bi} + r_i]\!]$ and sends it to Alice
3. Alice's triplet: $x_i = x_{Ai}$, $y_i = y_{Ai}$, $z_i = x_{Ai} \cdot x_{Ai} + (x_{Ai} \cdot y_{Bi} + y_{Ai} \cdot x_{Bi} + r_i) \mod n$
4. Bob's triplet: $x_i = x_{Bi}$, $y_i = y_{Bi}$, $z_i = x_{Bi} \cdot y_{Bi} - r_i \mod n$

This way of creating distributed beaver triplets that reveal nothing about the to-be-used x, y and z values. The full process of distributing the beaver triplets was described on the reference paper of the assignment.

Public Parameter calculation

For each component i in $\vec{a}$ (Alice) and $\vec{b}$ (Bob):

1. Share encrypted values $d_{ip} = a_i - x_{Ai}$ and $e_{ip} = b_i - y_{Bi}$
2. Alice can then decrypt $[\![b_i - y_{Bi}]\!]$ and deduct her share y_{Ai}, learning e_i
3. Bob then using the homomorphic properties calculates $[\![a_i - x_{Ai}]\!] \cdot [\![(-1) \cdot x_{Bi}]\!] = [\![a_i - x_{Ai} - x_{Bi}]\!]$ and sends it back to Alice, not learning anything
4. Alice then by decrypting Bob's message learns public parameter d_i and shares it with Bob along e_i

It is important, that no participant learns the partial masking parameters and that components stay uncompromised. Bob has no decryption access to what Alice sends and Alice can deduct no useful information from decrypting $[[b_i - y_{Bi}]]$. The partial masking parameters are enough to obfuscate respective vector values.

Inner Product Computation

For each component i in $\vec{a}$ (Alice) and $\vec{b}$ (Bob):

1. Alice calculates: $w_{Ai} = d_i \cdot e_i + d_i \cdot y_{Ai} + e \cdot x_{Ai} + z_{Ai} \mod n$
2. Bob calculates: $w_{Bi} = d_i \cdot y_{Bi} + e \cdot x_{Bi} + z_{Bi} \mod n$

After iterating through all components Bob sends his final $w_B = \sum_{i=1}^{dim} w_{Bi} \mod n$ share to Alice, who then calculates $w_B + \sum_{i=1}^{dim} w_{Ai} \mod n$.

Scaling Analysis

The protocol's computational complexity can be analyzed in two parts:

Beaver Triplet Generation

- Time complexity: $O(n)$ where n is vector dimension
- Each triplet requires:
  - 2 Paillier encryptions
  - 1 homomorphic multiplication
  - 1 decryption
- Communication: $O(n)$ constant number of messages for each component

Inner Product Computation

- Time complexity: $O(n)$ for n vector components
- Each component requires:
  - 2 Paillier encryptions (public)
  - 1 homomorphic multiplication (public)
  - 2 Paillier decryptions (public)
  - 1 multiplication (public)
  - constant number of multiplications and additions w_i
- Communication: $O(n)$ constant number of messages for each component

Security Analysis

The inner product protocol combines Paillier homomorphic encryption with secret sharing through Beaver triples, creating a hybrid security model. Its security relies on both the decisional composite residuosity assumption (for Paillier encryption) and the information-theoretic properties of secret sharing.

1 Information Leakage

1. Parties learn only the final inner product $\langle \vec{a}, \vec{b} \rangle$
2. No individual vector components are revealed

2 Potential Vulnerabilities

1. Malicious parties could generate beaver shares that are not random to manipulate the outcome
2. If one party's vector is not completely random, the other party can learn partial information. This can happen for example with sparse vectors (many zeros in the components)
3. Multiple protocol runs with related input vectors can leak information through a system of equations, where one party can even induce the process by deliberately choosing specific components
4. A man-in-the-middle attack is also possible since there is no validation process through any sort of signature or some sort. A malicious party can impersonate Bob and send encrypted messages to Alice to disrupt the computation.
5. Participants can solve uniquely for b using modular arithmetic if $gcd(a, n) = 1$, by computing $b \equiv c \cdot a^{-1} \mod n$ and can still guess if they are not co-primes.
6. If the system has d dimensions then it is undetermined with $n^{(d-1)}$ different solutions, forming a (d-1)-dimensional subspace over n.

3 Security Guarantees

1. Secure against honest-but-curious adversaries
2. Relies on semantic security of Paillier encryption
3. Security of secret sharing for partial products
4. Participants agree on vector length and don't split them up

For further instructions please refer to the README files.

Conclusion

Both implementations demonstrate practical approaches to secure multiparty computation. The weighted sum protocol scales linearly with the number of participants, while the inner product protocol scales linearly with vector dimension but with higher constants due to homomorphic operations.

Disclaimer

This code is intended solely for learning and research purposes and should be treated as a reference implementation rather than a security-hardened solution.

Use of AI Tools

AI-assisted tools, like ChatGPT, were used to enhance readability and document code functionalities in this report, as well as clarify differential privacy concepts and improve structure. All implementation details and logic were developed independently by the team.