Hochschule Wismar
University of Applied Sciences
Technology, Business and Design
Fakultät für Ingenieurwissenschaften

Master's Thesis

Privacy-Preserving Machine Learning in the Cloud: An Evaluation of Garbled Circuits for Secure Multi-Party Computation

Submitted on: September 23, 2024

by: Kristin Dahnken

First Supervisor: Prof. Dr.-Ing. habil. Andreas Ahrens
Second Supervisor: Prof. Dr. Matthias Kreuseler

Kurzfassung

Machine Learning hat sich zu einem wichtigen Werkzeug verschiedenster Forschungsbereiche entwickelt, wie etwa der Medizin oder der Informationssicherheit. Dennoch bringt die Verwendung von Cloud-Diensten zum Machine Learning noch einige Probleme mit sich, beispielsweise bei der Verwendung von sensiblen Trainingsdaten, da die Kontrolle über den Schutz dieser oft in den Händen des Cloud-Anbieters liegt. Auch sollen nicht alle Parteien die an einem kollaborativen Training eines Modells beteiligt sind, die Daten der jeweils anderen Parteien einsehen können.

Um diesem Problem zu begegnen und sensible Daten während des Trainingsprozesses zu schützen, stellt diese Masterarbeit einen Proof of Concept vor, der Garbled Circuits in diesen Prozess integriert. Ziel ist es, sowohl die Machbarkeit als auch die Sicherheit dieser Integration anhand einer beispielhaften Implementierung eines einfachen linearen Regressionsmodells zu zeigen. Darüber hinaus werden auch Limitierungen, Vorteile und mögliche zukünftige Anwendungen dieser Implementierung diskutiert.

Abstract

Machine learning has become an important tool in research across various domains, from medicine to cybersecurity. However, the use of cloud services for machine learning poses challenges when dealing with sensitive data, as control over data privacy is in the hands of the cloud provider. Additionally, not all parties involved in developing or training a model should have access to the full training data.

To address this problem and protect sensitive data during the training process of a machine learning model, this master's thesis presents a proof of concept that incorporates garbled circuits into the aforementioned process. The aim is to demonstrate that the usage of garbled circuits is both feasible and secure by detailing an exemplary implementation of a simple linear regression model. This work will also discuss limitations, advantages and possible future applications.

Table of Contents

1 Motivation

Over the last few years, machine learning has become an important tool in research, e.g. for medicine, the environment (in terms of sustainability) or cybersecurity (defence against attacks on IoT systems). As growing volumes of data and sometimes highly complex problems can no longer be handled by humans, machine learning is often used in such cases.

Both of these factors are also making the use of cloud services increasingly attractive, as this primarily results in cost savings because there no longer is any need to operate a dedicated data centre. In addition, cloud services enable location-independent, collaborative work and can be scaled depending on the intended use. [8]

The use of cloud services for machine learning tends to be less suitable for sensitive data (e.g. KRITIS, healthcare), as control over the data's privacy is in the hands of the cloud provider, with potentially unrestricted access.

Additionally, in some instances not all parties involved in the development or training of a model should have insight into the training data.

If one wants to utilise the advantages of cloud services (for example, to be able to use all available data for the most precise training possible without disclosing sensitive parts), there has to be a way to work with encrypted data without distorting the results. It is not enough to simply store the training data in encrypted form; any calculations performed must also be secure, as said data may be read by unauthorised parties during this process.

To counter this problem and protect sensitive data from prying eyes even during individual computing operations - i.e. the training of a machine learning model - and thus also avoid potential GDPR breaches, the use of garbled circuits is a good solution. [29]

Briefly explained, garbled circuits are a form of secure multi-party computation that allows two (or more) mistrusting parties to compute a function on given data without having to reveal their individual inputs to each other. Within the garbled circuit the function is represented as a boolean circuit consisting of a number of boolean

gates. This circuit is “garbled” (a special encryption procedure) and afterwards it is run on the encrypted input data, yielding the result of the function’s evaluation without ever disclosing the individual input data to the parties involved. [3, 9]

One possible scenario in which the use of garbled circuits could be beneficial, would be the creation of a dementia screening model based on real patient data from various hospitals and care facilities.

2 Cryptography

This chapter is intended to give a short historical overview of cryptography, as well as some of its basic concepts, algorithms and protocols. It will also touch briefly on the topic of cryptographic attacks.

2.1 The Historical Evolution of Cryptography

Cryptography - derived from the Greek words *kryptos*, meaning 'hidden' and *graphein*, meaning 'to write' - is the discipline of hiding the meaning of a message. In order to achieve this, protocols to both securely encode and decode the information need to be used, which have to be agreed beforehand between the sender and the recipient of said message.

Variations of these agreements have been practised throughout centuries and can be traced back as far as to the use of hieroglyphics in ancient Egypt. The Greeks also employed different forms of secret communication, from wax tablets that appeared to be blank until the wax was scraped off, to the usage of scytale, cylinders with strips of parchment wrapped around them. The messages written on these strips could then only be deciphered by someone in the possession of a rod of the same diameter as the one used by the sender.

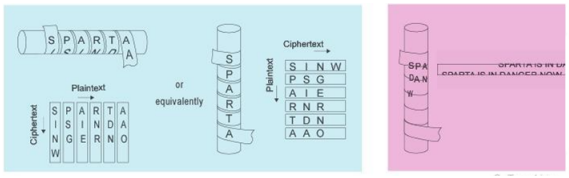

Figure 1: The scytale as a transposition cipher [27]

The aforementioned scytale is a form of transposition, one of the two branches of cryptography. In transposition, the individual letters of a message are rearranged to hide its meaning. This is only effective when the transposition follows a specific rule both sender and recipient have previously agreed upon and also kept secret.

The second branch of cryptography is known as substitution. One of the most prominent examples of substitution is the Caesar Cipher, which is based on a cipher alphabet that is shifted by a given number of places relative to the plain text alphabet and credited to have been developed by Julius Caesar.

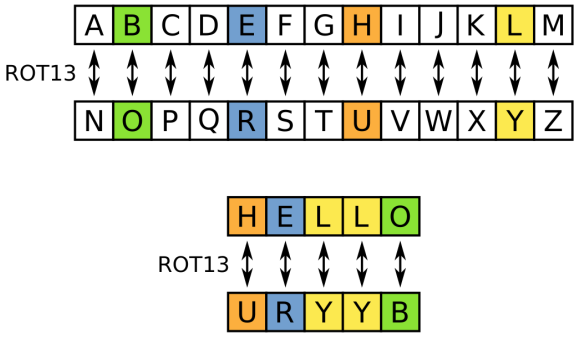

Figure 2: The ROT-13 cipher, a prominent example of the Caesar cipher [5]

While cryptography in the Western World almost vanished during the Middle Ages (largely due to the fall of the Roman Empire), some significant advancements have been made by Arab scholars who were not just employing ciphers but also capable of breaking them and thus inventing the field of cryptanalysis.

It was not until the Renaissance in fifteenth-century-Europe that the interest in - and also the need for - cryptography resurfaced. It saw the development of polyalphabetic ciphers because by this time it was already widely known that monoalphabetic ciphers (like the Caesar cipher) were vulnerable to frequency analysis, meaning that only shifting the alphabet could not hide the frequency in which certain letters appeared in a given language. Developed in the 1850s, the Vigenère Cipher is one of the earliest known polyalphabetic ciphers but also one of the most complex. It uses a series of Caesar ciphers with different shifts for each letter, determined by the letter in the same position of a given keyword.

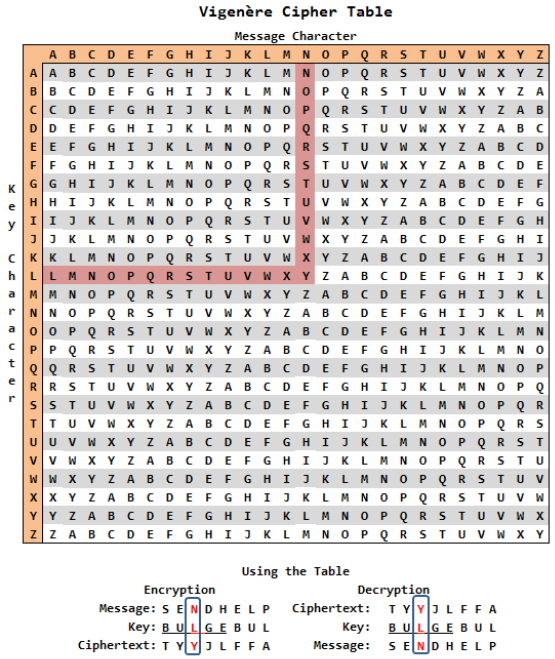

Figure 3: A Vigenère Cipher Table and how to use it [24]

The 20th century marked a crucial turning point in the history of cryptography due to the development of cipher machines like the Enigma, an electromechanical device that uses rotors (or scramblers) and plugboard connections to create a set of polyalphabets to do both encryption and decryption. Messages encrypted with the Enigma machine were nearly impossible to crack without the corresponding decryption key, which was determined by the machine's initial setting (the scramblers' starting positions, which were changed daily).

Figure 4: A Military Model Enigma I, in use from 1930 [17]

The era of modern cryptography began in the second half of the 20th century. Programmable computers were now available and were soon employed to not only break all sorts of ciphers but also to develop increasingly complex ones. New algorithms such as the Data Encryption Standard (DES) and its successor, the Advanced Encryption Standard (AES) were created in the 1970's and 1990's, respectively.

In the 1970s, cryptographers Whitfield Diffie and Martin Hellman came up with the first asymmetric-key cipher that defined what is known today as Public Key Cryptography. This concept eliminates the problem of key distribution that all symmetric-key ciphers suffer from. Like the name suggests, symmetric-key ciphers - where the decryption process is simply the opposite of the encryption process - use the same key for both actions. In asymmetric-key ciphers on the other hand the encryption and decryption keys are not identical and only the latter needs to be kept secret.

Based on the findings of Diffie and Hellman, Ron Rivest, Adi Shamir and Leonard Adleman developed the RSA Algorithm (ca. 1977), the first implementation of an asymmetric-key cipher, which is based on the difficulty of factoring the product of two large prime numbers. Another prominent example of an asymmetric-key cipher is Elliptic Curve Cryptography (ECC).

Building on these principles, the concept of digital signatures - which uses a combination of encryption and public-key cryptography to provide authentication, integrity, and non-repudiation for digital documents and communications - was developed.

During the following years, various cryptographic protocols and standards, such as Transport Layer Security (TLS) and the Public Key Cryptography Standards (PKCS), have been developed to enable secure communication between systems and networks. All of these developments mark crucial milestones of contemporary cryptography, without which secure digital communication would not be possible.

2.2 Quantum & Post-Quantum Cryptography

While everything mentioned so far is practical technology, there are also concepts which to date are mostly of theoretical nature - quantum and post-quantum cryptography.

The emergence of quantum computing can be seen as both a blessing and a curse to the field of cryptography. On one hand it poses a significant threat to classical cryptographic systems, due to the fact that it would be able to solve the problems

that make these systems secure, mainly the difficulty of factoring large numbers and solving the discrete logarithm problem. On the other hand, quantum cryptography would offer cryptographic systems that are virtually unbreakable by classical computing methods.

To take it even further, post-quantum cryptography - or quantum-resistant cryptography - aims to develop cryptographic algorithms that remain secure even against the sheer power of quantum computers. Examples of such algorithms are lattice-based cryptography, code-based cryptography and multivariate-quadratic-equations cryptography.

2.3 Symmetric Cryptography

Classical cryptographic methodologies predominantly centred around the utilisation of a sole confidential element to facilitate secure correspondence between two entities. This confidential element necessitated prior sharing among the concerned parties to prevent interception or inadvertent disclosure. Referred to as symmetric cryptography, both encoding and decoding processes relied on this singular confidential element, commonly known as a key. The primary virtues of such methodologies lie in their rapidity and effectiveness, rendering them the favoured option for securing extensive data volumes.

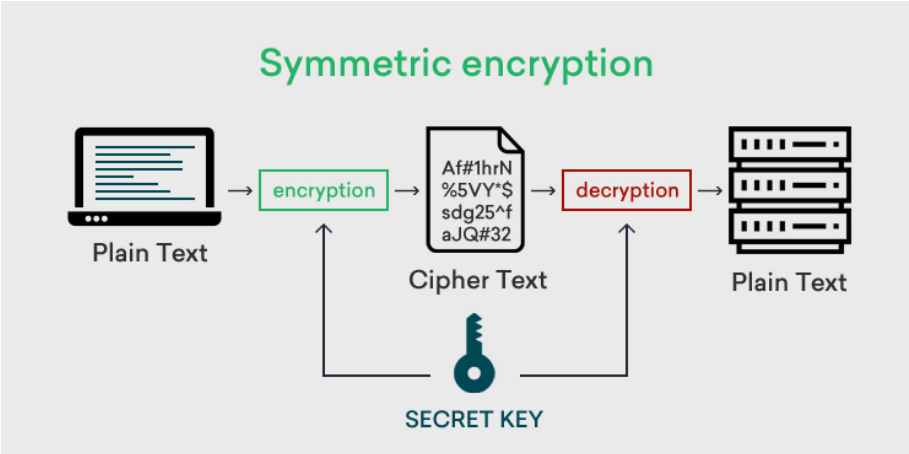

Figure 5: Symmetric Encryption Scheme [4]

One of the earliest, most influential symmetric algorithms is the Data Encryption Standard (DES). It encrypts data in individual 64-bit blocks using a 56-bit key through a series of 16 rounds of transformation, with each round being functionally equivalent. The encryption process uses fixed tables known as S-boxes (substitu-

tion boxes) in each of its rounds to make the encryption more resistant to attacks. Which of these S-boxes are selected depends on the encryption key, or rather the individual subkeys used in each round. Despite still being in use today, e.g. in ATMs or telecommunication protocols, DES has been deemed vulnerable to brute-force attacks due to its relatively small key size. To maintain compatibility but increase security, a more secure variant of DES has been adopted - Triple DES. This variant simply applies the DES algorithm three times with different keys.

The aforementioned vulnerabilities led to the development of the Advanced Encryption Standard (AES), which is commonly used today since its support of key sizes of 128, 192 and 256 bits provides a much higher level of security. It also operates on 128-bit blocks of data, compared to the 64-bit blocks used in DES. The length of the key affects both the number of rounds and the way its subkeys are constructed. Each round but the last one consists of four operations: AddRoundKey, SubBytes, ShiftRows and MixColumns. In the last round MixColumns is replaced with AddRoundKey to increase robustness. To date no practical attacks have been found to break the AES cipher, making it the preferred choice for various applications.

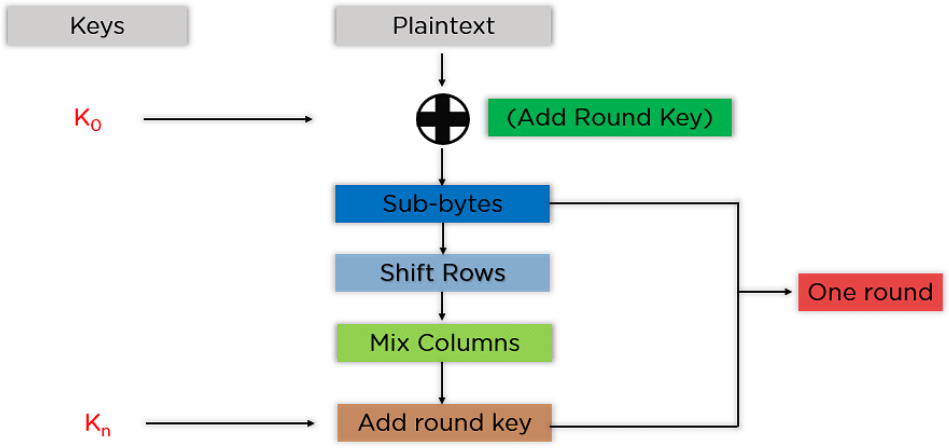

Figure 6: Simplified overview including all steps of the AES algorithm [10]

Another symmetric algorithm worth mentioning is the Blowfish algorithm, which is still used today in applications where speed and flexibility are crucial, like password hashing or network protocols. It essentially works like DES, with a fixed block length of 64-bit but a variable key length ranging from 32 to 448 bits, depending on the desired level of security. While the usage of 64-bit blocks is a disadvantage compared to other algorithms, it is considered secure with a sufficiently long key.

2.4 Asymmetric Cryptography

Asymmetric cryptography, also known as public-key cryptography, is a type of encryption that uses a pair of mathematically related keys: a public and a private key.

This approach eliminates the need of secret key exchanges completely since the involved parties can now simply generate a shared secret key. Like the names suggest, the public key can be openly disseminated and used for encryption, while the private key, known solely to the recipient of the encrypted message, is used for decryption. While it would technically be possible to calculate the private key using the public key, the length of both keys is typically chosen to be at least 1024 bits, rendering this plan practically impossible.

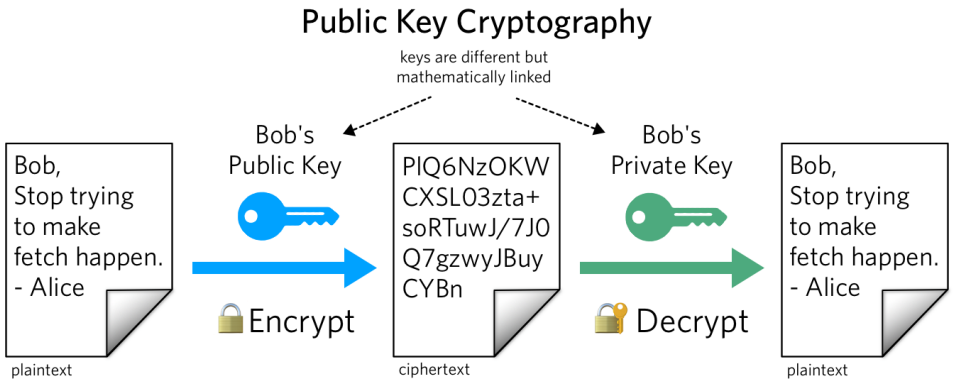

Figure 7: A simple illustration of Public Key Cryptography [23]

In addition to encryption and subsequent decryption, asymmetric algorithms are also used to generate digital signatures, which will be covered in the following section.

The RSA algorithm, named after its creators Rivest, Shamir and Adelman, is probably the most well known and adopted mechanism to date. It relies on the practical difficulty of factoring large prime numbers to ensure its security.

In cases where more efficiency is needed without negatively affecting security levels, Elliptic Curve Cryptography is a popular choice over the RSA algorithm, due to its shorter key lengths and consequently lesser need for computational power. ECC is based on the algebraic structure of elliptic curves and is used in many applications and protocols, such as SSL/TLS or in mobile devices.

Laying the foundation for these algorithms, the Diffie-Hellman key exchange protocol enables secure secret sharing between two parties over insecure channels. Both

parties agree on a large prime number and base and each of them generates a private key and a public value. The latter is then exchanged and enables both to calculate the shared secret key together with their individual private keys.

2.5 Hash Functions & Digital Signatures

In addition to secure communication, there may also be the need to verify that some data has not been tampered with, which can be ensured through hash functions.

Hash functions typically return a hexadecimal number that is unique to each unique input, hence making it virtually impossible to generate the same value from two different inputs. They are deterministic, meaning the same input will always produce the same hash value. If the recipient now calculates the data's hash value and it differs from the original one, they know for a fact that the data they have received has been modified.

Hashes are fast to compute, collision-resistant and very sensitive to small changes in the input data but practically impossible to reverse-engineer, making them an excellent choice for data integrity verification and other use cases, such as storing passwords. Common hash functions include MD5, SHA-1, SHA-256 and SHA-3, which are widely used in modern applications.

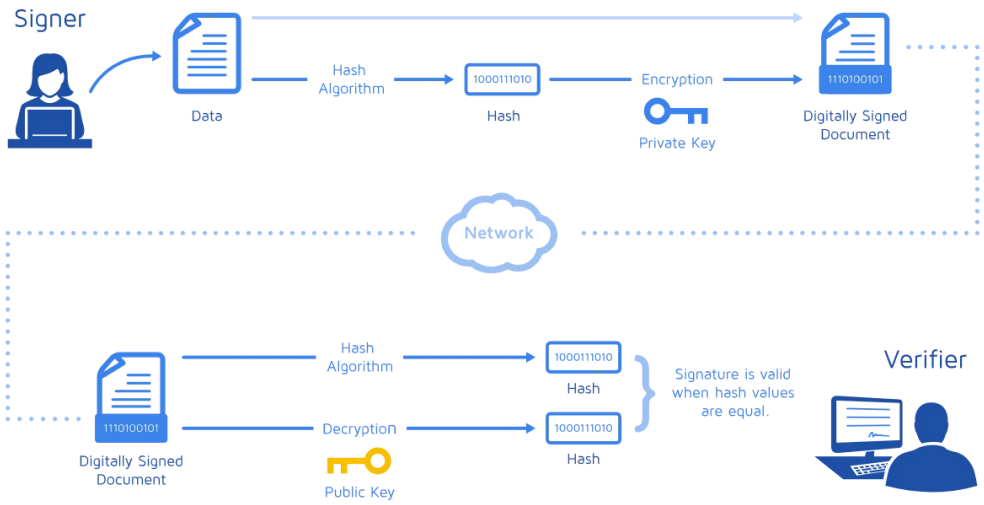

Figure 8: The process of Digital Signing and Verification [25]

Digital signatures integrate both hash functions and asymmetric algorithms for another crucial use case: ensuring authenticity and integrity of messages and their respective senders. First, a unique hash value is generated from the original message. This hash is then encrypted using the sender's private key, resulting in the digital signature. To later verify this signature, the message's recipient simply decrypts it using the sender's public key and then uses the same hash function as the sender on the received message. If both the decrypted hash and the message's hash are the same, the recipient can be sure that the data has not been altered and it also ensures the sender's authenticity.

2.6 Cryptographic Protocols

The previous chapters dealt with so-called 'cryptographic primitives', while well-established, these primitives are low-level algorithms or functions, designed to do a single specific task and thus do not provide enough security in more complex use cases. Cryptographic protocols are either abstract or concrete protocols that use the aforementioned primitives as building blocks for their respective applications. They are essential to communicate securely across a variety of platforms and devices. Two of the most crucial protocols are SSL/TLS and IPsec, without which secure communication on the internet would not be possible and all of the data that travels through it would be exposed to everyone else beyond the sender and the intended recipient.

SSL/TLS stands for Secure Sockets Layer/Transport Layer Security and secures

web connections through HTTPS (alongside other lesser known applications). To achieve this, both client and server use a handshake with an asymmetric algorithm to agree on a session-specific key with which any further communication is encrypted using a symmetric algorithm. The successful conclusion of the handshake starts the secured connection and all messages sent are now encrypted and decrypted using the shared session key.

IPsec - or Internet Protocol Security - is used to authenticate and encrypt data packets sent via IP networks, such as VPNs, for example. IPsec offers two modes of operation, depending on the intended use: transport mode and tunnel mode. In transport mode, only the IP packet's payload is encrypted while the header remains intact, meaning it is primarily used for direct communication between two hosts, e.g. a client and a server. In tunnel mode, the entire packet, including its header, is encrypted and afterwards a new IP header is added to the packet to be able to route it to its destined target. This mode ensures that the original packets are hidden and protected from intermediary devices such as proxy servers. It is primarily used to create VPNs and enables secure network-to-network, host-to-network and also host-to-host communication.

There are also advanced cryptographic protocols that serve purposes beyond ensuring the traditional security goals (confidentiality, integrity and availability), like zero-knowledge proofs or secure multi-party computation. Zero-knowledge proofs allow someone to prove that they are in possession of certain information without revealing the information itself. They play a crucial role in secure voting systems and anonymous transactions. SMPC protocols are considered advanced because they leverage the ideas of traditional protocols to allow two or more parties to jointly compute a function without revealing their individual inputs. Further details on SMPC will be discussed in chapter 3.

2.7 Cryptographic Attacks

Cryptographic systems have always been the target of attacks and there actually are a number of methods that proved to be successful. While symmetric systems can be compromised by sheer brute-force or measures like frequency analysis, breaking certain asymmetric cyphers is a much more complex process.

Symmetric ciphers are not only susceptible to brute-force attacks or frequency analysis, but also to known- and chosen-plaintext attacks. To perform a KPA, an attacker has to be in possession of a plaintext and its ciphertext from which they are then

able to derive the key. For a CPA, an attacker simply chooses a number of plaintexts and sends them to the targeted system. In doing so they are able to amass various corresponding ciphertexts and possibly the encryption key [26].

It is important to realize that brute-force attacks on public-key encryption schemes *can* be successful because they target the same component as in symmetric systems: the key. The shorter the key, the easier it is to guess correctly and vice versa. [7]

The RSA algorithm, for example, relies on the practical difficulty of factoring large prime numbers. If an adversary comes into possession of these numbers they'd be able to calculate the private key. When these prime factors are not large enough to begin with, they can easily be derived from the public key through means of prime factorization, like Fermat's factorization method or Pollard's rho algorithm.

Taking a look at hash functions, one of their crucial properties is collision resistance, which describes the difficulty to find two different messages that yield an equal hash. Ideally, hash functions produce vastly different outputs for slight changes made to an input message. Hash functions that only provide a weak collision resistance are vulnerable to brute-force attacks like a birthday attack, during which random messages are encrypted and stored alongside their hash values in order to find possible collisions. Other attacks on hash functions include dictionary attacks and lookup or rainbow tables [22].

In addition to attacks on the protocol level, the physical environment in which the cryptosystem is implemented can be targeted as well. These attacks are known as side-channel attacks that exploit features such as power consumption, encryption time and/or electromagnetic leaks. Attacks drawing information from power consumption, for example, assume that different encryption keys have different resource demands.
The existence of such side-channel attacks implies that any implementation of a cryptographic protocol may still be vulnerable, even when proven to be secure on protocol level [14].

3 Secure Multi-Party Computation

This chapter is intended to establish a basic understanding of the principles of secure multi-party computation.

3.1 Foundations & Applications

Secure multi-party computation is a tool that enables multiple parties to jointly compute a function without the need to disclose their individual input data. The only value that is shared among them is the eventual output of the given function. Typically, the involved parties do not trust each other and operate independently.

This idea was first introduced by Andrew Yao in 1982 in his paper *Protocols for Secure Computations*, who later also proposed the garbled circuits protocol. To this date, said protocol is still the basis for many SMPC implementations. It was not until the 2000s that practical approaches to multi-party computation have been made, though, due to technical and algorithmic limitations. The first notable implementation, Fairplay, was developed in 2004 and showed that a privacy-preserving program could be compiled into executables using a high-level language. Unfortunately, it was neither scalable nor particularly performant but since then the speed of MPC protocols has improved vastly [30].

Some of the most important applications of MPC protocols include, but are not limited to [6]:

Secure auctions. While in some auctions bidders may openly place their bids, other auctions rely on privacy for bidders and sellers alike. In these auctions none of the bidders should learn any other participant's bid because they would then gain an unfair advantage compared to the previous bidder. An example for such an advantage would be enabling the current bidder to adjust their bid to be just slightly higher than the latest bit and possibly win the auction at a fraction of the intended price.

Secure voting. Privacy is crucial in the context of electronic voting, not only because it is a fundamental civil process closely tied to a country's legislation but also

because knowing the tally allows for manipulation, which in turn could have severe consequences.

Secure machine learning. MPC protocols are a valuable tool for enabling privacy within machine learning systems, for both training and inference. In training, they allow for multiple parties to jointly train a model without exposing their individual data. Oblivious model inference, on the other hand, allows a client to retrieve a prediction from an already trained model on a server - with the model being kept private from the client and the inputs being kept private from the server.

A proof of concept for an exemplary training phase of a machine learning will be detailed in chapter 8.

3.2 Fundamental Protocols

Listed below are a few of the fundamental protocols used in SMPC [6]:

Yao's Garbled Circuits. Yao's GC protocol is arguably the best known MPC technique and a fundamental one at that. Without it many of today's MPC protocols would not exist. For an in-depth explanation of Yao's Garbled Circuit protocol, see chapter 4.

BGW Protocol. Ben-Or, Goldwasser and Wigderson developed one of the first multi-party protocols for secure computation, the BGW protocol, which can be used to evaluate an arithmetic circuit with addition, multiplication and multiplication-by-constant gates.

Information-Theoretic Garbled Circuits. In addition to being shared among parties, secrets can also be shared among a circuit's wires. This form of circuit construction is often preferred in environments that demand unconditional security despite increased costs compared to traditional garbled circuits. In IT GC, less data per communication round is sent but consequently, additional rounds are needed.
Some types of IT GC perform a lot better, though, because they allow the encryption of a bit to again be a single bit rather than a ciphertext.

Oblivious Transfer. OT is a crucial building block for secure computation protocols because it helps prevent involved parties from learning the input data of the other parties during joint computations. An example will be discussed in chapter 4

3.3 Security & Trust

In general, there are three key categories of protocols - *honest, semi-honest* and *malicious* [6].

Honest protocols are often also referred to as *ideal world* protocols because it assumes a completely trusted party that privately computes a function on the inputs provided by the other parties. An adversary in this scenario would not be able to gain any information other than the intended one - which is the output of the computed function - because they cannot take control over the trusted party. While these protocols are theoretical in nature because fully trusted third parties do not exist in the real world.

A semi-honest protocol assumes that all parties follow the rules but that an adversary may try to gain additional information besides the output of the computed function. Adversaries in this model are considered passive because they can only observe the protocol's execution but not intervene or anything because they are obliged to follow the rules. In order for these protocols to be secure, participants are not allowed to deviate from the rules and any information leakage has to be prevented.

In malicious security, adversaries are considered to be active since they themselves can deviate from the protocol as they please and may also corrupt any of the other involved parties to break protocol themselves. These actions could involve controlling, manipulating or injecting inputs into the function. These protocols typically best reflect real-world scenarios where appropriate measures need to be taken against untrusted parties and are considered to guarantee privacy and correctness.

Apart from malicious adversaries, SMPC protocols are also vulnerable to *collusion attacks* and *side-channel attacks*. Collusion attacks are a serious problem in real-world scenarios, especially when multiple malicious parties combine their respective knowledge in order to gain information from other involved parties. Side-channel attacks pose a threat whenever an adversary is able to access additional information about the environment in which the SMPC protocol is executed, such as timing, power consumption or memory access patterns. All of these examples could potentially enable the adversary to gain knowledge about the computed function or its input or intermediately calculated values.

3.4 Limitations

Despite being both adaptable and guaranteeing a high level of security, SMPC protocols still have their limitations. Overhead in terms of computation as well as communication being one of the most prominent ones. Additionally, scalability may cause problems when the computational demands or the complexity of the protocol increase. Many protocols also simply assume a certain degree of trust which might be broken due to insufficient measures or they are tailored for specific use cases and thus limited in terms of generalization [20].

4 Garbled Circuits

4.1 Yao's Garbled Circuits

4.1.1 The Answer to the Millionaires' Problem

In 1982, computer scientist Andrew Yao introduced the *Millionaires' Problem* and alongside it, he proposed a simple solution which is widely thought to have led to the creation of the *garbled circuit protocol*. Said problem described the interest of two millionaires - Alice and Bob - to find out which of them is richer without disclosing their individual wealth.

Yao's solution to the aforementioned problem is effectively showing that it is possible to solve this problem, all without the need to rely on third parties (trusted or not). In his paper *Protocols for secure computations*, Yao describes his solution as follows [30]:

For definiteness, suppose Alice has i millions and Bob has j millions, where $1 < i, j < 10$. We need a protocol for them to decide whether $i < j$, such that this is also the only thing they know in the end (aside from their own values). Let M be the set of all N-bit nonnegative integers, and Q_N be the set of all 1-1 onto functions from M to M. Let E_a be the public key of Alice, generated by choosing a random element from Q_N.

The protocol proceeds as follows:

1. Bob picks a random n-bit integer, and computes privately the value of $E_a(x)$; call the result k.
2. Bob sends Alice the number $k - j + 1$;
3. Alice computes privately the values of $y_u = D_a(k - j + u)$ for $u = 1, 2, \ldots, 10$.
4. Alice generates a random prime p of $N/2$ bits, and computes the values $z_u = y_u \pmod{p}$ for all u; if all z_u differ by at least 2 in the mod p sense, stop; otherwise generates another random prime and repeats the process until all z_u differ by at least 2; let p, z_u denote this final set of numbers;

5. Alice sends the prime p and the following 10 numbers to B : $z_1, z_2, \ldots, z_i$ followed by $z_i+1, z_{i+1}+1, \ldots, z_{10}+1$; the above numbers should be interpreted in the $\pmod p$ sense.
6. Bob looks at the j-th number (not counting p) sent from Alice, and decides that $i \geq j$ if it is equal to $x \bmod p$, and $i < j$ otherwise.
7. Bob tells Alice what the conclusion is.

This protocol enables Alice and Bob to correctly determine who is richer without either of them gaining more than this exact information.

Alice does not gain any more knowledge about Bob's wealth except for him sharing the final result with her, which is merely whether it is more or less compared to hers. While this value is derived from one of the values encrypted with Alice's key, it does not leak any information because the key itself is chosen randomly and all values encrypted with it (in this case 10 different values) are equally likely to be chosen by Bob.

Bob knows the random number he has chosen in step 1 of the protocol and its encrypted value, hence he also knows the value of z_j. From this knowledge, however, Bob cannot draw any conclusions in regard to the true values of the other numbers he received from Alice - any $z_u \neq z_j$ may still be z_u or $z_u + 1$.

4.1.2 Oblivious Transfer

Oblivious transfer is a crucial component in evolving Yao's solution into garbled circuits because the latter relies on the exchange of data between all parties involved. The protocol used in garbled circuits is *1-2 oblivious transfer*.

Assuming Alice is the sender who has two secret messages m_0 and m_1, but wants to make sure that Bob, the receiver in this scenario, only learns one of them. Bob on the other hand is in possession of a secret bit b and wants to learn m_b without disclosing b to Alice.

In his thesis on garbled circuits, Ignacio Navarro detailed a simple example of this protocol [19]:

1. Alice generates two asymmetric key pairs $(pub_0, priv_0)$ and $(pub_1, priv_1)$.
2. Bob generates a symmetric key K and, with the help of his secret bit b, encrypts K as $c = Enc_{pub_b}(K)$.

3. Alice decrypts both $c_0 = Dec_{priv_0}(c)$ and $c_1 = Dec_{priv_1}(c)$. One of the two will correctly decrypt K but Alice does not know which, in this example we assume $c_0 = K$.
4. Alice sends $c'_0 = Enc_{c_0=K}(m_0)$ and $c'_1 = Enc_{c_1}(m_1)$.
5. Bob now decrypts both messages $Dec_K(c'_0) = Dec_K(Enc_K(m_0)) = m_0$ and $Dec_K(c'_1)$, with the latter being gibberish and him identifying the first message as the correct one.

Note that this will only work if Alice does not cheat and can thus be considered a trusted party in this scenario.

4.1.3 Boolean Circuits

Boolean Circuits are essentially made up of logic gates. Each gate g can be represented as a function with two input wires and one output wire:

$$g : \{0,1\} \times \{0,1\} \to \{0,1\}$$

Each wire has one of two possible states: $\{0,1\}$

Taking the definition from Bellare *et al.* [2], a boolean circuit can be described as a 6-tuple $f = (n, m, q, A, B, K)$ with $n \geq 2$ inputs to the circuit, $m \geq 1$ being the number of outputs and $q \geq 1$ the number of gates.

Given n inputs and q gates, a boolean circuit r consists of $n + q$ wires, with each input labelled as $I = \{1, \ldots, n\}$ and each gate as $G = \{n + 1, \ldots, r\}$. The set of wires $W = \{1, \ldots, r\}$ is partitioned into input wires $I = \{1, \ldots, r - m\}$ and output wires $O = \{r - m + 1, \ldots, r\}$, resulting in $W = I \cup O$.

Each individual gate's incoming wires can be expressed by the function $A, B : G \to I$, while the gates themselves are described as $K : G \times \{0,1\}^2 \to \{0,1\}$.
$A : G \to W \backslash O$ is a function to identify each gate's *first* incoming wire and $B : G \to W \backslash O$ identifies each gate's *second* incoming wire.

To prevent the occurrence of cyclic graphs, $A(g) < B(g) < g$ is a crucial requirement for any gate $g \in G$. This simply establishes that no wire is used twice as an input to any given gate.

4.1.4 Garbling and Evaluation

A garbled circuit is nothing other than a boolean circuit with obfuscated truth tables. So rather than using the true inputs (0 or 1), each input is assigned a random label. These labels are unique for each possible input of each gate's input wire, with the exception of output wires that serve as input wires for a subsequent gate.

The protocol typically involves a garbler and one or more evaluators. The garbler is responsible for essentially creating and obfuscating the circuit, while the evaluator adds their own input to the already garbled circuit and evaluates it to learn its output.

In the following example, Alice takes on the role of the garbler and Bob takes on the role of the evaluator [19].

Garbling

1. Assuming Alice already converted her function into a circuit, she first needs to encrypt each possible input bit $(0, 1)$ of each of the circuit's wires. The resulting values are called *labels*. All of these labels are randomly created.
2. For Bob to be able to later make sense of the labels and correctly determine the circuit's output label without learning any intermediate values, we need to find a way to traverse the circuit properly solely based on the provided input labels.
   Given two incoming labels $k_a^j \text{and} k_b^k$ and a gate function g, Bob has to determine the gate's output label $k_c^{g(j,k)}$ correctly, even though he does not know any of the true values behind the labels. This can be achieved using the following mapping function:
   $$f : k_a^i \times k_b^j \rightarrow k_c^{g(i,j)}$$
   f must be strictly bijective - meaning that no two different combinations of input labels are allowed to yield the same output label.
   Considering that the output labels are encrypted, going forward f is set to be:
   $$f(k_a^i, k_b^j) = \text{Enc}_{(k_a^i, k_b^j)}(k_c^{g(i,j)})$$
3. Now, Alice computes f for every possible combination of inputs of each individual gate in the circuit. Afterwards she randomly permutes the resulting

truth tables such that the output value cannot be determined from the row and sends the result to Bob.

This is what a garbled `AND` gate looks like when applying f:

$$
\begin{aligned}
c_{10} &= f(k_a^1, k_b^0) = \mathrm{Enc}_{(k_a^1, k_b^0)}(k_c^0) \\
c_{11} &= f(k_a^1, k_b^1) = \mathrm{Enc}_{(k_a^1, k_b^1)}(k_c^1) \\
c_{01} &= f(k_a^0, k_b^1) = \mathrm{Enc}_{(k_a^0, k_b^1)}(k_c^0) \\
c_{00} &= f(k_a^0, k_b^0) = \mathrm{Enc}_{(k_a^0, k_b^0)}(k_c^0)
\end{aligned}
$$

Evaluation

1. Bob now requests the garbled input value (either k_b^0 or k_b^1) for his private input. He does this via Oblivious Transfer so Alice does not learn anything about his true input value.
2. Alongside his own garbled input value Bob also receives Alice's input k_a^i as well as the possible ciphertexts $c_{i,j}$ from the garbled table. He does not know which truth value Alice's garbled input represents, even though he is in possession of the garbled table because the entries are in random order.
3. Due to the fact that the ciphertexts are ordered randomly, to find the correct output Bob needs to decrypt all of them by applying the decryption function of the scheme he agreed on with Alice beforehand. Regardless of the chosen scheme, this is always the inverse of f.
   For example, with k_b^1, k_a^0 as inputs this would equal to:

   $$f^{-1}(c_{i,j})\mathrm{Dec}_{(k_a^0, k_b^1)}(c_{i,j})$$

4. Bob then sends the label he acquired to Alice who in turn informs him about the value it represents. Should the label be an intermediate output of a circuit, Bob uses it as an input for the next gate of the circuit instead of sending it to Alice. Only the final output of a circuit is decrypted and sent back to the garbler.

4.2 Garbling Scheme

Before we can discuss security properties or possible optimizations, it is necessary to formally define what garbled circuits and their functionalities entail. In their

paper *Foundations of Garbled Circuits*, Bellare *et al.* show how to achieve this by establishing so-called garbling schemes [2].

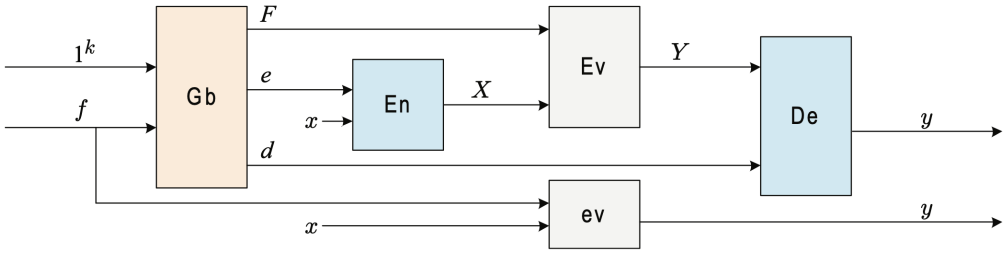

Figure 9: An overview of a garbling scheme's components [2]

A garbling scheme is a five tuple of algorithms $\mathcal{G} = (\texttt{Gb},\texttt{En},\texttt{De},\texttt{Ev},\texttt{ev})$:

- `Gb`: The garbling algorithm takes the original function f (the circuit that should be garbled) and a security parameter $k \in \mathbb{N}$ as inputs. Executing it results in a triple (F, e, d) where F is the garbled function, e is an encoding function and d is a decoding function: $(F, e, d) \leftarrow \texttt{Gb}(1^k, f)$.
- `En`: The encoding function e mentioned above maps each initial input $x \in \{0,1\}^n$ to its respective garbled representation $X = e(x) = \texttt{En}(e, x)$.
- `De`: Similarly, the decoding function d maps each garbled output Y to its final output value $y = d(Y) = \texttt{De}(d, Y)$.
- `Ev`: The evaluation algorithm takes the garbled function F and a garbled input X and maps the latter to a garbled output $Y = F(X) = \texttt{Ev}(F, X)$.
- `ev`: This algorithm represents the evaluation of the circuit (or original function) f as a whole: $\texttt{ev}(f, x) = f(x)$.

4.3 Security Properties

A garbling scheme like the one described in the previous section has three key security properties - *privacy, obliviousness* and *authenticity* [31].

- `Privacy`: Ensures that (F, x, d) does not reveal more information about x than $f(x)$ - to be precise, a simulator S exists that takes $(1^k, f, f(x))$ as input and produces an output that is indistinguishable from (F, X, d)
- `Obliviousness`: (F, X) should reveal no information about x - to be precise, a simulator S exists that takes $(1^k, f)$ as input and produces an output that is indistinguishable from (F, X)

- `Authenticity`: Given only (F, X), no adversary should be able to produce $Y' = \texttt{Ev}(F, X)$ such that $\texttt{De}(d, Y') = y$, except with negligible probability

4.4 Optimizations

Due to the high computational demands, various improvements to the classical garbled circuits protocol have been proposed, from parameters like circuit size to the number of decryptions necessary for each gate. The individual optimization steps will be briefly listed in this section with a few of them being discussed in greater detail in chapter 9.

Circuit size is one of the first parameters that come to mind when thinking about optimizing the protocol. Just reducing a circuit by one gate leads to the decrease of the circuit's garbled table by four entries and therefore reduces the amount of data that needs to be sent to the evaluator. Though decreasing the circuit size might be an advantage considering the communication between the involved parties, doing so might lead to more resource-intensive calculations to be executed by the evaluator. So whenever optimizations to one of these parameters are considered, they need to be as balanced as possible in order to not negatively impact the overall performance of the protocol.

	size per gate		calls to H per gate			
			generator		evaluator	
technique	XOR	AND	XOR	AND	XOR	AND
classical [30]	4	4	4	4	4	4
point-permute [1]	4	4	4	4	1	1
row reduction (GRR3) [21]	3	3	4	4	1	1
row reduction (GRR2) [21]	2	2	4	4	1	1
free XOR + GRR3 [12, 16]	0	3	0	4	0	1
fleXOR [11]	{0, 1, 2}	2	{0, 2, 4}	4	{0, 1, 2}	1
half gates [31]	0	2	0	4	0	2

Table 1: Optimizations of garbled circuits. Size is number of 'ciphertexts' (multiples of k bits).

In their paper *Two Halves Make a Whole*, Zahur, Rosulek and Evans summarized the methods to reduce the data needed to transmit a garbled gate available then and introduced their own method as well [31].

The *point-and-permute* optimization introduces a so-called *select bit* that is appended to each wire label, so that the wire's labels have opposite select bits. This allows for the garbled table to be arranged by these bits without leaking any information about the underlying labels' values. While this does not reduce the amount of ciphertexts per gate, it still reduces the computational cost due to the fact that the evaluator can now directly select the label they need to decrypt.

Garbled row-reduction on the other hand is a form of optimization that reduces the number of ciphertexts per gate. Instead of assigning random labels to each wire, one of its labels is chosen in such a way that the corresponding ciphertext is 0. In doing so, the amount of ciphertexts is reduced to three because the 0-text does not need to be evaluated. This method has later been extended to further reduce each gate to 2 ciphertexts.

Another effective optimization is the *free-XOR* technique which eliminates the need for ciphertexts for all XOR gates in a given circuit. This is achieved by establishing a fixed relationship between the incoming wires and XORing these wires directly instead of using an XOR gate. Because this technique is not compatible with all of the other optimizations, it has been generalized into the *fleXOR* method, which can be combined with GRR2 for AND gates. This combination can significantly reduce the size of a circuit that contains a lot of AND gates.

The new method introduced in the aformentioned paper is known as the *half-gates* technique. It is compatible with free-XOR and allows for any AND gate to be garbled with only 2 ciphertexts. Half-gates are defined as AND gates for which one of the involved parties knows one of the inputs - effectively cutting the amount of ciphertexts needed in half by leveraging the individual parties' knowledge.

4.5 Summary

Considering that Yao's protocol is one of the foundational cornerstones of secure multi-party computation, it comes with some great advantages. Arguably, the most important one is its strong privacy guarantee in semi-honest settings because none of the involved parties can learn anything about the other parties' inputs. The protocol can also be generalized to serve a variety of different scenarios, the only

limiting factor being that these scenarios need to be representable as a boolean circuit. Originally intended for two-party secure computation, Yao's protocol has been extended and improved over the years and can now also be used in multi-party contexts and malicious settings.

The obvious disadvantages of Yao's protocol have mostly stayed the same since the beginning, although the impact of some of these disadvantages has lessened over the years. There still are concerns about the protocol's efficiency because its computational effort increases significantly for more complex applications that require very large circuits. This also leads to a higher communication cost between the involved parties. Another drawback is the fact that any circuit created can only be used once. As soon as new inputs are added, new circuits have to be generated in order to prevent the exposure of anything else than the circuit's final output. Taking a single AND-gate with inputs a and b as an example, when executing the circuit twice with differing values for a but an unchanged value for b, an adversary would obtain different outputs for $b = 1$. This is in direct violation of the protocol's security guarantee and thus cannot be allowed.

While the protocol is secure in semi-honest settings, there still are potential vulnerabilities that may be exploited. In malicious environments, it can easily be compromised when adversaries intentionally deviate from the protocol. A malicious garbler could simply construct a circuit that leaks information about the involved parties' inputs, while a malicious evaluator could try to reuse circuits with different inputs. Information may also be leaked accidentally, for example due to faults in the used OT protocol that may reveal the values chosen by the evaluator. Adversaries may also derive information about the computed function from the circuit's size or structure, as well as from the time it takes to evaluate the circuit. The latter is known as a side-channel attack.

5 Machine Learning

The following chapter provides a brief overview over the principles of machine learning, its main paradigms and some algorithms that might be suitable to be trained using garbled circuits.

5.1 Introduction

Machine learning systems offer an innovative approach to problem-solving through their ability to automatically learn and improve from experience. Over the years these systems became invaluable tools in tackling complex tasks across various domains. Machine learning algorithms can generalize from given samples, which in turn enables them to handle previously unseen data and adapt to new situations. They are primarily used in dealing with problems too complex to solve through conventional programming approaches or when the volume of data accompanying a given problem is too large to be handled by humans.

5.2 Paradigms

Usually, the topic of machine learning is divided into three core paradigms: *supervised learning, unsupervised learning* and *reinforcement learning*. The main difference between these three lies in their ability to solve specific tasks and in their representation of the resulting data. The most commonly used paradigm is supervised learning because the other two can often not be applied to the task at hand. In some cases, however, it is possible to simply choose one's favourite method, as well as use different paradigms together.

Supervised Learning

The main principle of supervised learning, as the name suggests, is to guide the system's learning process, which is why it not only receives a set of inputs but also the corresponding outputs. These sets of data are called *labeled examples*. The goal

here is to train a predictive model that is able to guess the output of a previously unseen input, based on the examples provided to it.

Unsupervised Learning

The second most used paradigm is unsupervised learning, which in contrast to supervised learning receives neither input nor output data, instead it draws from a given set of examples. This method is commonly utilized for tasks that revolve around *clustering* with the goal of dividing the given set into groups that feature similar characteristics.

Reinforcement Learning

The third paradigm is called reinforcement learning, which enables autonomous agents to learn from interacting with an environment provided to them. This differs from the previous learning paradigms because the environment is not a fixed set of data but rather an external system. Starting out, the agent interacts randomly with the environment and gradually learns from its experience in order to perform better. The agent's behaviour is usually reinforced by rewarding it when it performs a given task well (hence the name). Reinforcement learning is often used to teach robots how to interact with the real world through simulated real world environments.

5.3 Suitable Algorithms

When it comes to picking a suitable algorithm to be used in combination with garbled circuits, the main prerequisite is that any calculations must be performed exclusively on values that have a binary representation, e.g. numerical values like integers. In order to not unnecessarily increase the complexity of the planned implementation, we should pick a simple task with as few steps as possible.

The first task that comes to mind is *regression*, which is a supervised learning task with the goal of predicting a numeric value. Numerous real world problems like housing prices, salary growth and population sizes can be described as a regression problem. The simplest (and also oldest) of these methods is *linear regression*, which has been widely adopted for both statistics and machine learning. It estimates the relationship between a dependent variable (scalar response) and either one or more independent variables (regressor).

The proof of concept will focus on training a *simple linear regression* model with a single independent variable. An in-depth explanation of the simple linear regression model will be provided in chapter 7.1.

Some other models that might be suitable for an implementation using garbled circuits include, but are not limited to:

1. *Logistic Regression*: These models involve linear operations that could be trained similarly to a linear regression model, followed by a logistic function for classification. The latter could be approximated by mapping the Newton-Raphson method to garbled circuits because it utilizes operations that can entirely be represented by arithmetic logic circuits.
2. *Decision Trees and Random Forests*: Decision trees are a form of information mapping based on comparisons. A single tree is made up of branches for each comparison (or logical decision) which can be represented with garbled circuits. With proper templates in place, the recursions during the training process could be handled efficiently as well. This could be elevated to random forests since they are an aggregation of multiple decision trees.
3. *Neural Networks*: Basic neural networks might also be suitable for garbled circuits, since they primarily involve linear operations. The limiting factor here would be transforming the activation functions of the given network - while some of them can be approximated, others cannot and their transformation could easily become too costly, especially within deeper networks.

6 Cloud Computing

The following chapter gives a brief overview of cloud computing, its technologies, architecture and service models. Trust and confidentiality are also touched upon since they are one of the focal points of this thesis.

6.1 What is Cloud Computing?

The purpose of cloud computing is to provide network access to resources as well as services on-demand and scalable to the individual requirements of users. Said resources or services include (but are not limited to) servers, storages and applications.

According to the **NIST Definition of Cloud Computing**, the cloud model consists of the five essential characteristics below [15]:

- *On-demand self-service:* Customers can utilize the cloud's computing capabilities without the need of human interaction with the provider.
- *Broad network access:* Cloud services can be accessed easily through standard platforms like workstations, laptops or mobile phones.
- *Resource pooling:* The provider's resources are pooled to serve multiple customers with varying demands at once. These resources are assigned and reassigned dynamically as needed.
- *Rapid elasticity:* Capabilities can be both provisioned and released elastically according to customer demand, often automatically.
- *Measured service:* Resource usage can be monitored, controlled and reported to provide transparency for both customers and providers alike.

These characteristics alone already highlight a few key advantages of cloud computing. Cloud services can be accessed worldwide and from basically any device, instantly providing a user with their desired resources. Since cloud providers typically operate on a large scale, they offer their services at a relatively low cost. Customers only have to pay for the resources they actually use and don't need to

invest in on-premise solutions that might require large upfront investments or high maintenance costs. Cloud services are especially valuable to users who need a lot of resources either irregularly or at short notice.

6.2 Architecture and Deployment Models

Cloud computing architecture encompasses different deployment models based on the levels of control, scalability and security required by customers (primarily organizations) [15].

- *Private cloud:* Intended for exclusive use by a single organization, usually consisting of various customers. Private clouds are either owned, managed and operated by the organizations themselves or a third party, on or off premises.
- *Community cloud:* This deployment model is a form of private cloud intended for use by a group of customers from one or more organizations with shared concerns.
- *Public cloud:* Provisioned for use by the general public. Even though public clouds are owned, managed and operated by academic, business or government organizations they exclusively exist on the premises of the cloud provider.
- *Hybrid cloud:* This cloud infrastructure is a combination of two or more of the previously mentioned deployment models, that, while remaining unique, are bound together by some form of technology that enables both data and application portability - otherwise it would not be possible to combine clouds that differ in architecture. Hybrid deployment models are often used for load balancing large amounts of data or requests between clouds.

To optimize resource utilization and enhance flexibility, cloud providers leverage virtualization technologies that allow multiple virtual machines to run on the same hardware. In order to properly scale and distribute physical resources, clouds typically employ *Hypervisors* that assign these physical resources depending on the individual virtual machines' demands.

6.3 Service Models

Cloud computing employs three different service models, based on the individual capabilities provided to the consumer. All of these models have one thing in common, though: the provider always controls their deepest layers, which include the networking infrastructure, physical servers, storage infrastructure and hypervisors.

They are typically categorized as follows [15]:

- *Infrastructure as a Service (IaaS):* Provides virtualised computing resources over the internet, allowing users to manage operating systems, storage and deployed applications while the provider maintains the physical hardware.
- *Platform as a Service (PaaS):* Offers a development environment where users can create, test and deploy applications without managing the underlying infrastructure.
- *Software as a Service (SaaS):* Provides access to applications run on cloud infrastructure, eliminating the need for installation and maintenance on local devices.

These service models enable organisations to choose the level of control and responsibility that best suit their needs, from fully managed software solutions to more flexible infrastructure options.

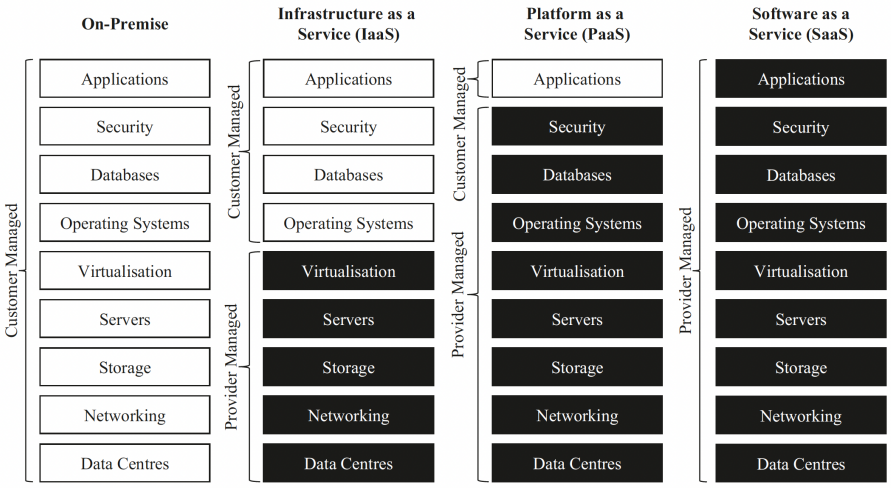

Figure 10: An overview of the different cloud services [13]

6.4 Trust and Confidentiality

The level of data confidentiality is strongly tied to the chosen cloud's deployment model [28].

Theoretically, a public cloud provider could access all of the data that is stored or processed on their hardware, regardless of the employed service model, leaving users with no guarantee of confidentiality. The same is true for private cloud providers,

with one minor (but significant) difference: private cloud providers are deemed trustworthy - internal attacks not taken into account - because they typically operate on a non-commercial basis. Using a private cloud ensures a high level of confidentiality for customers.

When it comes to hybrid cloud environments, it highly depends on which tasks are chosen for execution on a public cloud. If these tasks do not handle any confidential data, they can be executed without the need to worry whether the data is handled securely but if they do, the hybrid cloud's level of confidentiality essentially drops down to that of a public one.

7 Privacy-Preserving Machine Learning with Garbled Circuits

The following chapter will demonstrate an integration of garbled circuits into a suitable machine learning algorithm.

For this purpose the linear regression model has been chosen, since it is one of the easiest algorithms to be broken down into smaller steps which can then be translated into individual binary circuits.

7.1 The Linear Regression Model

A linear regression model describes the dynamics between a dependent variable y_i and one (or multiple) independent variables x_i. It is a statistical model used to make predictions on these sets of data through estimating the coefficients of the underlying linear equation. Linear regression fits a straight line to minimize errors between given and predicted outputs, where the best fit line contains the least errors.

The simplest linear regression calculation uses the *mean squared error (MSE)* function to find the best fit for a provided set of data. The value of the dependent variable is then estimated from the independent variables.

Some applications of this model include forecasting effects or trends (predictions on the dependent variable) and determining the strength of given predictors (assessing the influence of individual independent variables on the dependent variable).

The equation assuming only one independent and the corresponding dependent variable is called *Simple Linear Regression* and is described as follows:

$$\hat{y}_i = \beta_0 + \beta_1 x_i$$

- $\hat{y}_i$ is the predicted output of the dependent variable
- β_0 is the intercept or constant

- β_1 is the slope or regression coefficient
- x_i is the independent variable

The aim is to find the best values for β_0 and β_1, in order to achieve the best-fit regression line on the given data. This is done using a so-called *cost-function* (or *loss-function*) - in this case, the aforementioned *mean squared error (MSE)* function which simply calculates the average of the squared errors between the actual and predicted values.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2$$

The cost function has to be differentiated and calculated for both slope and intercept (initialised to 0, randomly chosen or calculated from x_i and y_i) and both values need to be updated iteratively until minimum cost is reached (a method called *Gradient Descent*). The MSE indicates how accurately the model is able to predict the dependent value, the lower the MSE, the better the fit and thus, the model's accuracy.

7.2 Linear Regression in Machine Learning

Even though linear regression is mainly a statistical model, it plays a pivotal role in machine learning, especially in the field of predictive modelling since it focuses on making the most accurate predictions. In linear regression, the relationship between the dependent and independent variables is straightforward and clearly showing the influence of the used predictors. Because of their simplicity and training speed, linear regression models are also often used as baseline models or extended to more complex models such as multiple linear regression, logistic regression or neural networks.

7.3 Breaking down the model

To use garbled circuits during the training phase of a simple linear regression model, it has to be broken down into smaller parts that can be represented as binary circuits. This step is necessary because garbled circuits are constructed using logic gates.

Since all calculations have to be performed bit-wise, it requires more operations the more bits the input values have. In order to keep the breakdown simple, this

section will not include details of the circuit creation or its requirements. Both will be discussed in following sections.

Assuming a simple dataset with one independent variable x_i and corresponding dependent variable y_i:

$$(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$$

The first important step is to determine the starting values for β_0 and β_1 using the *Ordinary Least Squares* method, thus directly training the model with a minimised error.

$$\beta_1 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2} \qquad \beta_0 = \bar{y} - \beta_1 \bar{x}$$

0 or random values would also be valid but might necessitate more iterations to improve the model's accuracy later. Ideally, the number of iterations is kept to a minimum when using garbled circuits because this process is quite resource-intensive.

Both formulas of the OLS method require the means of x_i and y_1, which are calculated as follows:

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \qquad \bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$$

All values of the respective variables are added up and the result is divided by the number of values. Even though this is a fairly straightforward calculation, implementing it with binary circuits requires splitting it up into simpler operations that are carried out sequentially:

1. Add the first two values of x_i
2. Add the next value to the previous result
3. Repeat step 2 for all remaining values of x_i
4. Once the above steps are completed, divide the acquired sum by the number of values in x_i

Repeat for y_i.

The mean calculation can be performed using *binary adder-subtractor circuits*. In fact, when implemented correctly, all operations can be carried out using these circuits, since multiplication is repeated addition and division is repeated subtraction. The proof of concept will make use of this through templates derived from an initial adder-subtractor circuit.

Next, we calculate the numerator of β_1, $\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})$:

1. Separately subtract the mean $\bar{x}$ from each value of x_i
2. Separately subtract the mean $\bar{y}$ from each value of y_i
3. Multiply the resulting sets of steps 1 and 2
4. Sum up the resulting values of step 3

The denominator $\sum_{i=1}^{n}(x_i - \bar{x})^2$ only requires two additional steps, since $x_i - \bar{x}$ has already been calculated:

1. Reuse the previously calculated result of $x_i - \bar{x}$ and multiply it by itself
2. Sum up the resulting values of step 1

With both the numerator and the denominator calculated, we can determine the slope β_1 by dividing them through repeated subtraction.

Lastly, the slope β_0 can also be calculated in another two simple steps:

1. Multiply $\bar{x}$ by β_1
2. Subtract the result of step 1 from $\bar{y}$

This concludes the breakdown of a single iteration of training. We have discovered that all basic arithmetic operations are required to complete the training process. The construction of the respective circuits needed for training will be detailed in later sections.

Now the slope and intercept have been determined, the linear regression function can be applied to arbitrary test sets (or input values) as a means to make predictions.

$$\hat{y}_i = \beta_0 + \beta_1 x_i$$

Due to its simplicity, this function only needs to be split into two parts which can later be translated into binary circuits - one for the multiplication of the slope by the

input set and one for the addition of the intercept - proving that not only training but also prediction is possible in the given context.

Now all required operations have been identified, we will have a look at the binary circuits that will serve as the building blocks for our proof of concept.

7.4 Constructing the Binary Circuits

The previous section showed that all main arithmetic operations (addition, subtraction, multiplication, division) have to be translated into binary circuits to allow training, prediction and optimisation of a linear regression model.

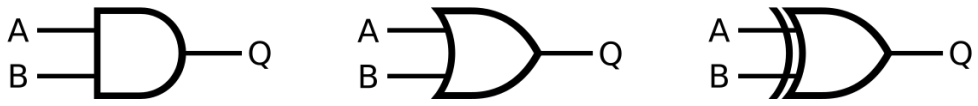

Figure 11: *AND, OR & XOR* ANSI symbols

In order to apply the simplest type of garbling to these circuits later on, all of them are constructed exclusively of **AND**, **OR** and **XOR** gates.

A	B	$A \wedge B$
0	0	0
0	1	0
1	0	0
1	1	1

A	B	$A \vee B$
0	0	0
0	1	1
1	0	1
1	1	1

A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

Table 2: *AND, OR & XOR* truth tables

7.4.1 Addition

The most basic operation generally is the addition of two binary digits (or bits), the sum of which either produces another single digit or - when both summands are 1 - two digits. In the latter result the higher significant bit is called the *carry bit*. To perform this operation, a so-called *half-adder* is needed.

A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Table 3: *Half-adder* truth table

There are multiple ways to implement such a half-adder but since it is a crucial building block of the next type of adder, it is built with an XOR gate (producing the sum bit S) and an AND gate (producing the carry bit C).

$$S = A \oplus B \qquad C = A \cdot B$$

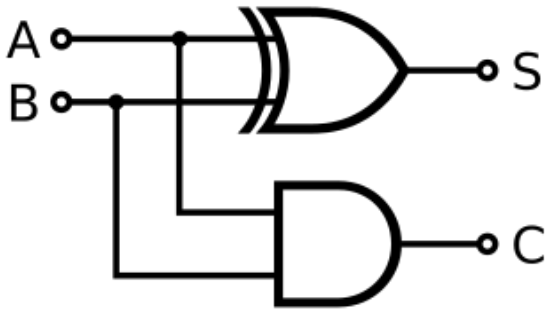

Figure 12: Half-adder schematic

In order to properly deal with carry bits and later even add numbers of arbitrary bit lengths, we have to construct an adder that does not only take the two summand bits but also the carry-out of a possible previous operation. This can be achieved with a *full-adder* that can be implemented with two half-adders and a single OR gate.

A	B	C_{in}	Sum	Carry out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Table 4: *Full-adder* truth table

A single full-adder can only add two bits (and a previous carry), so we need to find an implementation that leverages the usage of full-adders and thus is capable of adding two binary numbers of n bits.

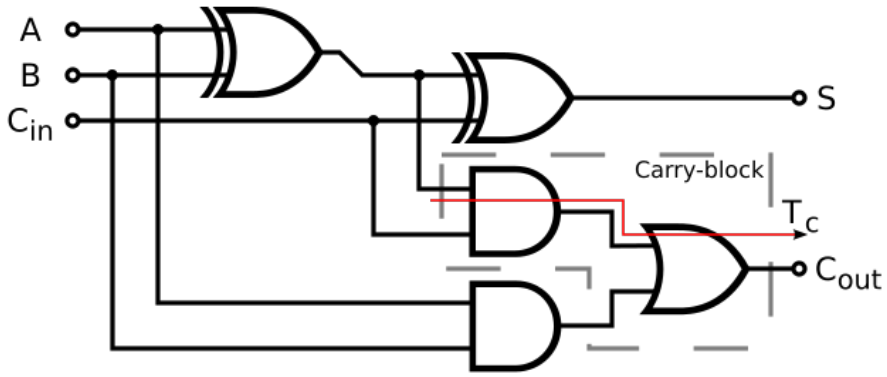

Figure 13: Full-adder schematic

Let's consider two 4-bit numbers as an example. Starting with the least significant bit of both numbers, each pair of bits is added individually through a full-adder. If an addition produces a carry-out it is used as an input when adding the pair of bits one position higher.

Since each pair of bits is added through one full-adder, two n-bit numbers always require n full-adders. A circuit composed of full-adders connected in this way is called a *binary parallel adder* (or *ripple-carry adder*).

7.4.2 Subtraction

The subtraction of two n-bit numbers can conveniently be done through addition, by means of complement. Any subtraction $A - B$ can be performed by taking the *Two's Complement* of B and adding it to A. The Two's Complement can be obtained through inverting each bit of B (also called the *One's Complement*) and adding 1 to the least significant pair of bits.

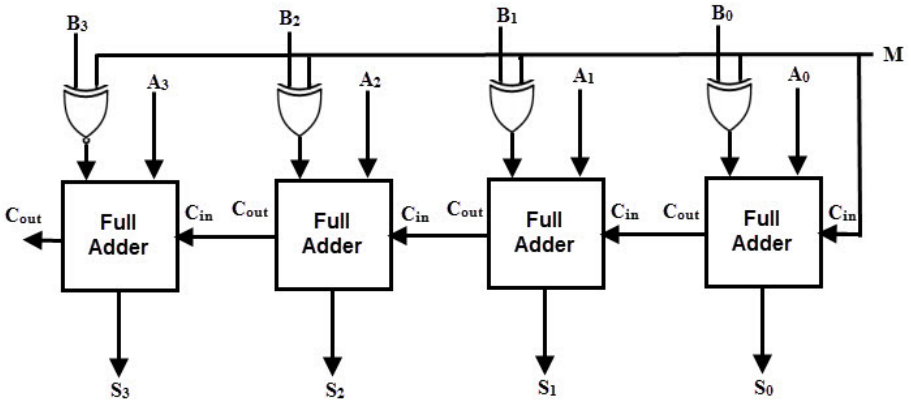

Figure 14: A 4-bit parallel adder-subtractor with a mode input control line

Any binary adder can be upgraded to a *binary adder-subtractor* simply by including an XOR gate with each full-adder. To control the operation carried out, an input bit M (representing the mode) is added to the circuit and each XOR gate receives both M and one of the input bits of B. When $M = 0$, the full-adders receive the value of B with an input carry of 0 and the circuit performs an addition. When $M = 1$, all bits of B are inverted and 1 is added through the input carry, resulting in the circuit adding the Two's Complement of B to A and thus performing a subtraction.

7.4.3 Multiplication

Binary multiplication can be done by successive additions and shifting.

Like the previous operations, multiplication starts with the least significant bit first. If the multiplier bit is 1, the multiplicand bit is copied down, otherwise 0. The numbers in successive lines are each shifted one position to the left compared to the previous one. Finally, the sum of all these numbers results in the product of multiplicand and multiplier. While this form of multiplication is fairly easy, it has to be modified slightly in order to form a proper binary circuit. Instead of storing and

adding all binary numbers representing the 1's of the multiplier, they are calculated iteratively in pairs with the help of a parallel adder.

Furthermore, instead of shifting the multiplicand to the left, the partial product is shifted to the right - which directly results in the correct relative positions of both. If any of the numbers are negative, the operation will use their respective Two's Complement, which will also be the output should the result of the multiplication be negative.

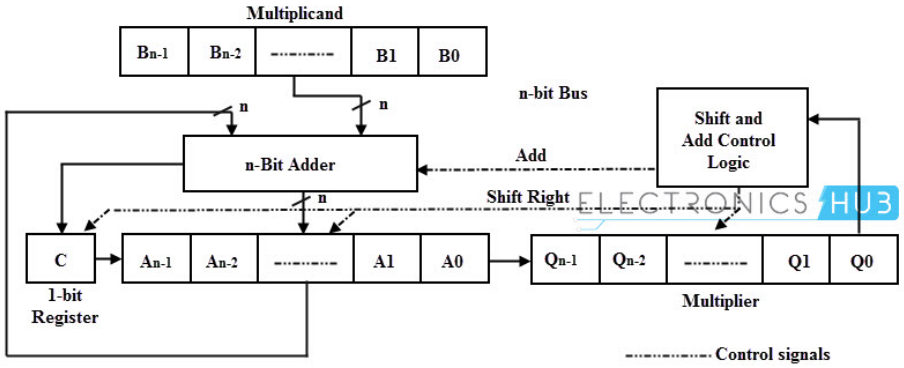

Figure 15: The schematic of a 4-bit parallel multiplier

7.4.4 Division

Binary division is done by repeated subtraction, thus allowing us to build on the already established adder-subtractor. The first step is to determine the signs of both dividend and divisor in order to later determine the sign of the quotient. Then the quotient will be initialised to 0. Negative numbers are converted into their positive equivalents through calculating their Two's Complement.

Next, the divisor is subtracted from the dividend to acquire the first partial remainder and 1 is added to the quotient. If the partial remainder is positive, the divisor is subtracted from it and 1 is added to the quotient to get the next partial remainder. These steps are repeated for every partial remainder until it is either 0 or negative.

At the end of this operation, the quotient equals the number of times the divisor has been subtracted. Should the quotient be negative it has to be converted into its Two's Complement in order to get the correct result of the division.

7.5 Research & Development Design

The following section provides a high-level overview of the implementation plan and the proposed workflow as well as its scope and limitations.

7.5.1 Implementation Plan

The proposed proof of concept utilizes garbled circuits to securely train a linear regression model. Its aim is to ensure the privacy of all given inputs throughout the training process. The linear regression process is broken down into its individual calculation steps, which are then implemented as logic circuits in order to enable garbling.

As a prerequisite, the implementation also includes a data preparation layer, since it is necessary to convert all input data from decimal to signed binary.

- Data Preparation Layer
  - Input Data Conversion
- Binary Calculations
  - Half- and Full-Adders
  - Ripple-Adder-Subtractor
  - n-bit Multiplier
  - n-bit Divisor
- Garbled Circuits
  - Gates
  - Wires
  - Labels
  - Circuit Generation
  - Garbling Methods
  - Evaluation Methods
- Garbled Circuit Templates for Secure Linear Regression
- Exemplary Model Training
- Exemplary Prediction

7.5.2 Proposed Workflow

To keep the implementation effort within the scope and time frame of this thesis, we will not introduce a second party with additional training data. Instead the Garbler simply wants to train their linear regression model in a cloud environment, without risking to expose their data to the cloud provider. In this use case the cloud serves as the evaluator instead of another party. This type of circuit can be described as a *non-interactive garbled circuit* with the following workflow:

1. The Data Preparation Layer converts the Garbler's training data from decimal to signed binary
2. The Garbler translates the linear regression computations into their respective binary circuits
3. They then garble all circuits created in the previous step, producing the respective garbled tables and input labels for each wire (*Note:* when an input is the result of a previous output, the input wire needs to be exactly the same as the output wire to ensure that the circuit can be evaluated correctly)
4. The garbled circuit is then uploaded into the chosen cloud, which serves as the Evaluator
5. The training method is then triggered in order to evaluate the circuit and obtain the trained model after decrypting all final output labels
6. The Garbler can now extract the truth values from the labels to gain access to the trained model

7.5.3 Scope & Limitations

It is important to keep in mind that the implementation is merely a proof of concept to showcase that it is possible to train a machine learning model incorporating garbled circuits into the process. While there will be an example of value prediction, fully trained models or the performance of multiple iterations to increase the model's accuracy are not focal points of this thesis. Instead the focus lies on maintaining the confidentiality of the training data and on evaluating the general suitability of garbled circuits for the aforementioned process.

As mentioned in section 4.1.2, Yao's garbled circuit protocol includes an oblivious transfer protocol as well because it relies on the exchange of data between involved parties. However, the exchange of data and consequently, an implementation of

oblivious transfer, is not required to answer the research question and will therefore not be included.

The proof of concept includes test executions on a local machine that doubles as the cloud environment. This was necessary due to the limited time frame of this thesis. Afterwards, an evaluation is carried out that assesses the overall security of the implementation. Additionally, it will take its performance and complexity into account because both could be valid reasons to not utilize garbled circuits during model training.

One of the biggest limitations encountered while setting up the implementation plan was the fact that existing libraries could not be used due to two main reasons:

- The linear regression model needed to be broken down into smaller pieces than offered by the commonly used frameworks. While some allowed for creating custom models, there would have been no benefit in using them over developing a custom implementation from the start.
- Other libraries, like `Gabes`, and some of their dependencies were simply too old to be compiled on a MacBook with an ARM CPU.

Since `Gabes` was released under the MIT License and proved to be a straightforward implementation of garbled circuits, it served as a blueprint for the corresponding classes included in the proof of concept [18].

8 Proof of Concept

The following chapter details a proof of concept that shows an integration of garbled circuits into the training of a simple linear regression model, as well as touching on a possible implementation of the linear regression equation in order to make predictions. The proposed solution is written in `Python` since it's both easy to understand and widely used for machine learning purposes. The implementation is partly derived from the **Gabes** package because it features implementations of all crucial components of a garbled circuit, which can easily be modified to fit the use case of this proof of concept.

8.1 Implementation

8.1.1 Data Preparation Layer

Following the implementation plan set up in the previous chapter, the first component to be developed is the *Data Preparation Layer*, which handles the input data conversion.

```
1     def convert_training_data(set_x, set_y, bit_len):
2       bin_x = ut.int_array_to_bin_array(set_x.tolist(), bit_len)
3     bin_y = ut.int_array_to_bin_array(set_y.tolist(), bit_len)
4     set_len = ut.int_to_bin_array(len(set_x), bit_len)
5
6     return bin_x, bin_y, set_len
```

Listing 1: Preparing the input data

In order to correctly execute all necessary operations, including negative numbers, all input values, as well as the number of entries in the given test set, have to be converted from decimal to signed binary. During conversion all binary numbers will be set to a predetermined bit length to reduce the need for padding during circuit creation and thus keep the computational cost as low as possible.

At this time, operations on floating point binary numbers are not part of the system, hence all input values need to be integers.

8.1.2 Binary Arithmetic Operations

Next, all necessary binary arithmetic operations have to be implemented. These operations are crucial to prove that training a linear regression model using arithmetic logic circuits is possible in the first place. The following implementations will be used as blueprints for the corresponding garbled circuit templates, whenever possible.

Not detailed here are auxiliary functions like negativity checks, shifting methods or conversion functions. These can be found in the source code attached to this document.

Throughout all functions signed binaries are used, since training sets or intermediate calculation results may include negative values.

As previously explained in chapter 7, all operations are based on addition and for this purpose both a *half-adder* and a *full-adder* are implemented.

```
1 def half_adder(bit1, bit2):
2     sum_ = bit1 ^ bit2
3     carry = bit1 & bit2
4     return sum_, carry
```

Listing 2: Half-adder function

The half-adder simply takes two bits and returns their sum and a possible carry. Since model training requires the calculation of numbers of arbitrary bit length, we have to construct a component that is able to take previous carry overs into account. This is done through a full-adder that's making use of the already established half-adder.

```
1 def full_adder(bit1, bit2, carry_in):
2     sum1, carry1 = half_adder(bit1, bit2)
3     sum_, carry2 = half_adder(sum1, carry_in)
4     carry_out = carry1 | carry2
5     return sum_, carry_out
```

Listing 3: Full-adder function

Now we can simply chain together n full-adders in order to create a *ripple-carry adder* that adds two n-bit numbers.

Knowing that subtraction $A - B$ of n-bit numbers can be done by means of complement, the ripple-carry adder is modified slightly to perform both addition and

subtraction. This modification requires the use of the *Two's Complement* that is obtained through inverting each bit of B and then adding 1 to the least significant pair of bits. Within a ripple-carry adder this can easily be achieved by including an XOR gate with each of the n full-adders and adding a control line M. The latter decides the mode of operation (addition or subtraction).

The resulting adder-subtractor looks as follows:

```
    def binary_adder_subtractor(a, b, control):
    max_len = max(len(a), len(b))
    a = [0] * (max_len - len(a)) + a
    b = [0] * (max_len - len(b)) + b

    carry = control
    result = [0] * max_len

    # Iterate over each bit from LSB to MSB
    for i in range(max_len - 1, -1, -1):
        b_i = b[i] ^ control
        result[i], carry = full_adder(a[i], b_i, carry)

    # If there's a carry-out during addition, add it to the result
    if carry and not control and not all(x == 0 for x in result):
        result.insert(0, carry)

    return result
```

Listing 4: n-bit adder-subtractor

Multiplication is done by successive addition and shifting the partial products acquired through the individual additions. This operation will also utilize the Two's Component should one of the inputs or the result be negative.

```
    def n_bit_multiplier(a, b):
        n = len(a)
    m = len(b)

    # Determine the sign of the result
    multiplicand_sign = a[0]
    multiplier_sign = b[0]
    result_sign = multiplicand_sign ^ multiplier_sign

    # Convert to absolute value
    if multiplicand_sign == 1:
        a = twos_complement(a)

    if multiplier_sign == 1:
```

```
15          b = twos_complement(b)
16
17      product = [0] * (n + m)
18
19      for i in range(m):
20          if b[-(i + 1)] == 1:
21              shifted_multiplicand = a + [0] * i
22              product = binary_adder_subtractor(product[-(m + i):],
                ↪  shifted_multiplicand, 0)
23              product = [0] * (n - len(product)) + product
24
25      # Adjust the sign of the result if necessary
26      if result_sign:
27          product = twos_complement(product)
28
29      return product
```

Listing 5: n-bit Multiplier

Lastly, division is done by repeated subtraction, allowing us again to reuse the adder-subtractor that has already been implemented.

```
1       def binary_division(dividend, divisor):
2           # Division by zero is not possible, just return
3       if is_zero(divisor):
4           return
5
6       """
7       Step 1: Determine if the signs of the dividend and divisor are the same
        ↪  or different. This determines what
8       the sign of the quotient will be. Initialize the quotient to zero.
        ↪  Transform numbers into their uncomplemented form
9       and pad where necessary.
10      """
11      dd_neg = is_negative(dividend)
12      dv_neg = is_negative(divisor)
13      q_neg = dd_neg ^ dv_neg
14
15      max_len = max(len(dividend), len(divisor))
16
17      quotient = [0] * max_len
18
19      if dd_neg:
20          dividend = twos_complement(dividend)
```

```
    else:
        dividend = [0] * (max_len - len(dividend)) + dividend

    if dv_neg:
        divisor = twos_complement(divisor)
    else:
        divisor = [0] * (max_len - len(divisor)) + divisor

    """
    Step 2: Subtract the divisor from the dividend using two's complement addition (final carries are discarded) to get
    the first partial remainder and add 1 to the quotient. If this partial remainder is positive, go to step 3. If the
    partial remainder is zero or negative, the division is complete.
    """
    remainder = binary_adder_subtractor(dividend, divisor, 1)
    quotient = binary_adder_subtractor(quotient, [0, 1], 0)

    if is_zero(remainder) or is_negative(remainder):
        return quotient

    """
    Step 3: Subtract the divisor from the partial remainder and add 1 to the quotient. If the result is positive,
    repeat for the next partial remainder. If the result is zero or negative, the division is complete.
    """
    while True:
        if is_zero(remainder) or is_negative(remainder):
            break
        remainder = binary_adder_subtractor(remainder, divisor, 1)
        quotient = binary_adder_subtractor(quotient, [0, 1], 0)

    if q_neg:
        quotient = twos_complement(quotient)

    return quotient
```

Listing 6: n-bit Division

8.1.3 Garbled Circuits

Using **Gabes** as a blueprint, all components of a garbled circuit, including garbling, evaluation and decryption methods, are implemented.

A `Circuit` is represented as a collection of `Gates` which are connected by `Wires`. Each wire within the circuit is labelled using a `Label` object.

```
1     class Circuit:
2         def __init__(self, i_gates=None):
3         self.gates = [] if i_gates is None else i_gates
4         self.initial_wires = []
5         self.input_labels = {}
6         self.output_wires = {}
7         self.outputs = {}
```

Listing 7: Circuit object

Each `Gate` has two input `Wire` objects (left and right) and one output `Wire` object.

```
1     class Gate:
2         def __init__(self, g_type, create_l_wire=True, create_r_wire=True):
3         self.garbled_table = {}
4         self.gate_type = g_type
5         self.left_wire = Wire() if create_l_wire else None
6         self.right_wire = Wire() if create_r_wire else None
7         self.output_wire = Wire()
```

Listing 8: Gate object

In addition to setters for the input wires and the output identifier, this class handles the garbling process for a given `Gate` instance.

The function implemented here represents the classical approach to garbled circuits, where each output label of the gate's truth table is encrypted using the accompanying inputs as keys. The result is then shuffled to prevent the evaluator from deriving information about the inputs from their respective rows in the truth table.

```
    def garble(self):
        for label1 in self.left_wire:
        for label2 in self.right_wire:
            i1 = int(label1.get_value())
            i2 = int(label2.get_value())
            o = calc_output_value(self.gate_type, i1, i2)
            encrypted_output = encrypt(label1.get_label() +
            ↪  label2.get_label(), self.output_wire[o].get_label())
            self.garbled_table[(i1, i2)] = encrypted_output

    items = list(self.garbled_table.items())
    random.shuffle(items)
    self.garbled_table = dict(items)

    return self.garbled_table
```

Listing 9: Classical garbling function

In order to later learn the correct output of the circuit, each gate needs to be evaluated individually because most output labels are used as inputs to subsequent gates throughout the circuit. During evaluation the circuit is essentially reconstructed from the first gate to the last, based on the inputs the evaluator provides.

```
    def evaluate(self, inputs):
        input_label1, input_label2 = inputs
    for (i1, i2), encrypted_label in self.garbled_table.items():
        if input_label1 == self.left_wire[i1] and input_label2 ==
        ↪  self.right_wire[i2]:
            output_label = self.learn_output_label(encrypted_label, inputs)
            return output_label
    return None
```

Listing 10: Evaluation function

It is imperative to decrypt the output label when evaluating a given `Gate` object, otherwise it cannot serve as an input to the next gate due to the fact that the input values of its garbled truth table are stored as plaintext labels.

```
    def learn_output_label(self, encrypted_output_label, org_inputs):
        input_label1, input_label2 = org_inputs
    output_label = decrypt(encrypted_output_label, input_label1.get_label()
    ↪  + input_label2.get_label())
    for out_label in self.output_wire:
        if out_label.get_label() == output_label:
            return out_label
    return None
```

Listing 11: Decryption method for output labels

As previously mentioned, all gates are connected by wires - each of which has two randomly created labels representing the truth values 0 and 1 respectively, as well as a flag that states whether it is one of the circuit's initial input wires.

An instance of a `Wire` object is defined as follows:

```
1     class Wire:
2         def __init__(self, ident=None):
3         self._index = 0
4         self.identifier = ident
5         self.labels = [Label(False), Label(True)]
6         self.is_initial = False
```

Listing 12: Wire object

Ultimately, a `Label` object stores both the truth value it represents and its randomly created 16 bit label string which will be used during circuit evaluation.

```
1     class Label:
2         def __init__(self, truth_value):
3         self.label = os.urandom(16)
4         self.truth_value = truth_value
```

Listing 13: Label object

8.1.4 Templates for Secure Linear Regression

Now that both binary arithmetic operations and garbled circuits are implemented, we can use them as an outline and create reusable templates for secure linear regression model training.

We need to implement templates for every arithmetic operation previously defined, starting with the half-adder, the smallest of our building blocks and working our way up to the more complex operations. Each template can either be built from a string or from already established wires. The latter enables linking an arbitrary combination of templates and thus facilitates constructing the desired training circuit.

All of these templates are structured similarly, starting out with the creation of the necessary wires should these not be supplied, followed by flagging these wires as the initial inputs to the template at hand. Then all gates of the desired circuit and - where applicable - any subcircuits needed to execute the given operation are created. Each template also provides a method for garbling and evaluation, with

some including subcircuit evaluation whenever intermediate outputs are needed to continue the respective operation.

```
    class HalfAdderTemplate:
        def __init__(self, l_wire=None, r_wire=None):
        # Initialize new circuit
        self.circuit = Circuit()

        # Create input wires when not supplied
        if l_wire is None:
            l_wire = Wire('A')

        if r_wire is None:
            r_wire = Wire('B')

        # Set initial wires
        l_wire.set_as_initial()
        r_wire.set_as_initial()

        # Add XOR gate (sum)
        xor_gate = Gate('XOR', l_wire, r_wire)
        xor_gate.set_out_identifier('S')
        self.circuit.add_gate(xor_gate)

        # Add AND gate (carry)
        and_gate = Gate('AND', l_wire, r_wire)
        and_gate.set_out_identifier('C')
        self.circuit.add_gate(and_gate)
```

Listing 14: The general template structure, applied to the half-adder

Just as described in section 8.1.2 the next component in need of a template is the full-adder, which is making use of the already implemented half-adder template. For better readability, only part of the template is shown - the full code listings can be found in the attachment section of this thesis.

```
# Create first half adder, set output identifiers
ha1 = HalfAdderTemplate(l_wire, r_wire)
ha1.circuit.gates[0].set_out_identifier('SAB1')
ha1.circuit.gates[1].set_out_identifier('CAB1')
self.circuit.extend_circuit(ha1.circuit)

# Create second half adder, set output identifiers
ha2 = HalfAdderTemplate(ha1.circuit.gates[0].output_wire, c_wire)
```

```
35 ha2.circuit.gates[0].set_out_identifier('SAB')  # S = sum
36 ha2.circuit.gates[1].set_out_identifier('CAB2')
37 self.circuit.extend_circuit(ha2.circuit)
38
39 # Create OR gate, set output identifiers
40 or_gate = Gate('OR', ha1.circuit.gates[1].output_wire,
   ↪  ha2.circuit.gates[1].output_wire)
41 or_gate.set_out_identifier('CABO')  # CO = carry_out
42 self.circuit.add_gate(or_gate)
```

Listing 15: The full-adder template making use of the half-adder template

From here we are moving on to initializing an n-bit adder-subtractor that is constructed with n full-adders and n additional XOR-gates (with the latter forming a separate subcircuit) for bit inversion in case of subtraction. Due to creating all these circuits individually, some of the wires need to be manually connected to their respective target gates.

```
105     # Create corresponding XOR gate, set output identifier
106     gate = Gate('XOR', b_wire, m_wire)
107     gate.set_out_identifier(f'B{i}M')
108     self.xor_gates.append(gate)
109
110     if i == 0:
111         fa = FullAdderTemplate(a_wire, gate.output_wire, m_wire)
112     fa.circuit.gates[4].set_out_identifier(f'CO{i}')
113     self.full_adders.append(fa)
114     else:
115         fa = FullAdderTemplate(a_wire, gate.output_wire,
116                                self.full_adders[i -
                                   ↪  1].circuit.gates[4].output_wire)
117     fa.circuit.gates[4].set_out_identifier(f'CO{i}')
118     self.full_adders.append(fa)
119
120     # Connect output of corresponding XOR gate
121     r_wire = self.xor_gates[i].output_wire
122     r_wire.set_as_initial()
123     self.full_adders[i].circuit.gates[0].set_right_wire(r_wire)
124     self.full_adders[i].circuit.gates[1].set_right_wire(r_wire)
125
126     # Create circuit from XOR gates
127     self.xor_circuit = Circuit(self.xor_gates)
```

Listing 16: Creating the required circuits for the n-bit adder-subtractor template and connecting them

Considering that the adder-subtractor circuit consists of multiple subcircuits - with some of which relying on the outputs from other subcircuits - the evaluation method has to be modified accordingly, so the individual circuits receive the correct inputs.

```
137     def evaluate(self, inputs, labels=False):
138         # Set input wires and evaluate XOR circuit
139     self.xor_circuit.set_input_wires()
140     if labels:
141         self.xor_circuit.set_input_labels(inputs.copy())
142     else:
143         self.xor_circuit.set_input_values(inputs.copy())
144     self.xor_circuit.evaluate()
145 
146     # Declare additional inputs for full-adders
147     fa_inputs = self.xor_circuit.outputs.copy()
148     if self.upd_input_labels is not None:
149         fa_inputs.update(self.upd_input_labels)
150 
151     # Evaluate each full-adder individually, otherwise crucial output labels
        ↪ are lost
152     c = 0  # counter for output values
153     for fa in self.full_adders:
154         fa.circuit.set_input_wires()
155         if labels:
156             fa.circuit.set_input_labels(inputs.copy())
157         else:
158             fa.circuit.set_input_values(inputs.copy())
159         fa.circuit.update_input_labels(fa_inputs)
160         fa.circuit.evaluate()
161         fa_inputs.update(fa.circuit.outputs.copy())
162         # Collect outputs
163         self.update_outputs(fa.circuit, c)
164         c += 1
```

Listing 17: Evaluating an adder-subtractor circuit

Before we can tackle multiplication, we need to define a template for the *Two's Complement* in case one (or both) of the numbers involved are negative, which may also result in a negative output that needs to be represented correctly.

If the *most significant bit* (MSB) of a signed binary number equals 1, the number is negative. Unfortunately, it is not possible to determine this using the MSB's label only - we have to reveal the true value behind this label. While this technically breaks with the protocol, the implementation can still be considered secure since this intermediate value is never revealed to the evaluator. We will discuss whether this may weaken the overall security in section 8.2.2.

When creating garbled circuits to perform binary long multiplication (i.e. calculating partial products, shifting them to the left and adding them together) the incoming multiplicand and/or multiplier might have to be modified depending on their sign.

```
101     def prepare_template(n, in_wrs, in_lbls):
102         mc_wi, mp_wi, mc_la, mp_la = split_inputs(in_wrs, in_lbls)
103     mc_n, mp_n = determine_signs(mc_la, mp_la)
104
105     if mc_n:
106         mc_wi, mc_la = transform_into_tc(n, 'A', mc_wi, mc_la)
107
108     if mp_n:
109         mp_wi, mp_la = transform_into_tc(n, 'B', mp_wi, mp_la)
110
111     wrs = mc_wi + mp_wi
112     lbls = mc_la
113     lbls.update(mp_la)
114
115     return wrs, lbls
```

Listing 18: Preparing the inputs of the n-bit multiplier template

Initially, the multiplier template only consists of the AND-gates required to calculate the partial products of the incoming numbers. Since the bit length of the multiplication result may be $2n$, each partial product has to be sign extended to this width.

```
101     def perform_sign_extension(amount, pp_id, pp_wrs, pp_lbls):
102         # Create new wires & labels for given partial product
103     ext_wrs = pp_wrs.copy()
104     ext_lbls = pp_lbls.copy()
105
106     for i in range(amount):
107         new_key = f'A{i + amount}{pp_id}'
108         if new_key not in pp_wrs.keys():
```

```
109             ext_w = Wire(new_key)
110             ext_wrs[new_key] = ext_w
111             ext_lbls[new_key] = ext_w.get_label(0)  # sign extension bits
                ↪  are always 0
112     return ext_wrs, ext_lbls
```

Listing 19: Performing sign extension by creating new wires and labels

During evaluation the partial products are then shifted and added to each other by creating, rewiring and evaluating individual ripple-adder templates. In order to shift each partial product correctly, we create new identifiers for each wire to subsequently be able to provide the correct inputs to the adders.

```
101     def shift_keys(pp_wrs, pp_lbls, shift=0):
102         num_keys = len(pp_wrs)
103     new_keys = [f'B{(i + shift) % num_keys}' for i in range(num_keys)]
104
105     # Update wires and labels with shifted keys
106     sh_wrs = []
107     sh_lbls = {new_key: value for new_key, (_, value) in zip(new_keys,
        ↪  pp_lbls.items())}
108
109     for new_key, (old_key, value) in zip(new_keys, pp_wrs.items()):
110         value.set_identifier(new_key)
111         sh_wrs.append(value)
112
113     return sh_wrs, sh_lbls
```

Listing 20: Shifting the partial products by assigning new wire identifiers

The final template needed is the one for the division operation, which is performed by repeated subtraction for which we simply use the already implemented added-subtractor template. The division template is initially created with an adder-subtractor circuit to subtract the divisor from the dividend and an additional adder-subtractor circuit to determine the first quotient. After evaluating these initial circuits, the sign of the remainder has to be checked because the division algorithm only needs to continue when the remainder is positive - this is done in the same way as for the multiplication.

Unfortunately, the division process requires revealing every single bit of the remainder because the division is also considered complete when the remainder equals 0. Although this might be interpreted as more information leakage, none of this information is visible to the evaluator.

8.1.5 Training and Prediction

Lastly, we need to provide exemplary model training and prediction in order to declare the proof of concept as successful. The individual calculations will be done both in their binary and their garbled circuit versions alongside the original mathematical representation, to show that they yield the same results.

Training

As mentioned in section 7.3, training a model essentially consists of four main calculation steps that are performed on given input values - the independent variable(s) x_i and the dependent variable y_i.

Assuming a simple training set with a single independent variable $x_i = (1, 2, 4, 3, 5)$ and its dependent variable $y_i = (2, 3, 5, 4, 6)$ we first need to determine the individual mean of both variables:

$$\begin{aligned}\bar{x} &= \frac{1}{n}\sum_{i=1}^{n} x_i \\ &= \frac{1}{5}(1+2+4+3+5) \\ &= \frac{1}{5} \times 15 \\ &= 3\end{aligned}$$

$$\begin{aligned}\bar{y} &= \frac{1}{n}\sum_{i=1}^{n} y_i \\ &= \frac{1}{5}(2+3+5+4+6) \\ &= \frac{1}{5} \times 20 \\ &= 4\end{aligned}$$

Utilizing the binary operations defined in section 8.1.2, as well as some auxiliary functions (e.g. summing a list of binaries), calculating the means produces the expected outputs for both x_i and y_i as binary numbers (the decimal values are printed alongside them for better understanding).

```
*** Starting linear regression training with binary circuits ***

Initializing training set:
X = [1 2 4 3 5]
y = [2 3 5 4 6]

8-bit binary representation of X & y:
X_bin = [[0, 0, 0, 0, 0, 0, 0, 1], [0, 0, 0, 0, 0, 0, 1, 0], [0, 0, 0, 0, 0, 1, 0, 0], [0, 0, 0, 0, 0, 0, 1, 1], [0, 0, 0, 0, 0, 1, 0, 1]]
y_bin = [[0, 0, 0, 0, 0, 0, 1, 0], [0, 0, 0, 0, 0, 0, 1, 1], [0, 0, 0, 0, 0, 1, 0, 1], [0, 0, 0, 0, 0, 1, 0, 0], [0, 0, 0, 0, 0, 1, 1, 0]]

Calculated means of X & y:
X_mean = [0, 0, 0, 0, 0, 0, 1, 1] (3)
y_mean = [0, 0, 0, 0, 0, 0, 1, 0, 0] (4)
```

Figure 16: Result of the binary means calculation

The same applies when calculating the means using garbled circuits.

```
*** Starting linear regression training with garbled circuits ***

8-bit binary training set:
X = [[0, 0, 0, 0, 0, 0, 0, 1], [0, 0, 0, 0, 0, 0, 1, 0], [0, 0, 0, 0, 0, 1, 0, 0], [0, 0, 0, 0, 0, 0, 1, 1], [0, 0, 0, 0, 0, 1, 0, 1]]
y = [[0, 0, 0, 0, 0, 0, 1, 0], [0, 0, 0, 0, 0, 0, 1, 1], [0, 0, 0, 0, 0, 1, 0, 1], [0, 0, 0, 0, 0, 1, 0, 0], [0, 0, 0, 0, 0, 1, 1, 0]]
Test set length = [0, 0, 0, 0, 0, 1, 0, 1]

Mean of X = [0, 0, 0, 0, 0, 0, 1, 1] (3)
Mean of y = [0, 0, 0, 0, 0, 1, 0, 0] (4)
```

Figure 17: Result of the garbled circuit means calculation

The next training step covers the calculation of the slope (β_1), using the means we have determined previously.

$$
\begin{aligned}
\beta_1 &= \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2} \\
&= \frac{\sum_{i=1}^{5}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{5}(x_i - \bar{x})^2} \\
&= \frac{(1-3)(2-4) + (2-3)(3-4) + (4-3)(5-4) + (3-3)(4-4) + (5-3)(6-4)}{(1-3)^2 + (2-3)^2 + (4-3)^2 + (3-3)^2 + (5-3)^2} \\
&= \frac{(-2)(-2) + (-1)(-1) + (1)(1) + (0)(0) + (2)(2)}{(-2)^2 + (-1)^2 + (1)^2 + (0)^2 + (2)^2} \\
&= \frac{4+1+1+0+4}{4+1+1+0+4} \\
&= \frac{10}{10} \\
&= 1
\end{aligned}
$$

Thus, the slope of the linear regression line is $\beta_1 = 1$.

```
*** Starting linear regression training with binary circuits ***

Initializing training set:
X = [1 2 4 3 5]
y = [2 3 5 4 6]

8-bit binary representation of X & y:
X_bin = [[0, 0, 0, 0, 0, 0, 0, 1], [0, 0, 0, 0, 0, 0, 1, 0], [0, 0, 0, 0, 0, 1, 0, 0], [0, 0, 0, 0, 0, 0, 1, 1], [0, 0, 0, 0, 0, 1, 0, 1]]
y_bin = [[0, 0, 0, 0, 0, 0, 1, 0], [0, 0, 0, 0, 0, 0, 1, 1], [0, 0, 0, 0, 0, 1, 0, 1], [0, 0, 0, 0, 0, 1, 0, 0], [0, 0, 0, 0, 0, 1, 1, 0]]

Calculated means of X & y:
X_mean = [0, 0, 0, 0, 0, 0, 1, 1] (3)
y_mean = [0, 0, 0, 0, 0, 1, 0, 0] (4)

Calculated slope:
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] (1)
```

Figure 18: Result of the binary slope calculation

Note that the bit length of the slope increases to 16 due to the sign extension that occurs during the multiplication process of two (signed) binary numbers. This is done to avoid loss of information that would lead to wrong results.

```
*** Starting linear regression training with garbled circuits ***

8-bit binary training set:
X = [[0, 0, 0, 0, 0, 0, 0, 1], [0, 0, 0, 0, 0, 0, 1, 0], [0, 0, 0, 0, 0, 1, 0, 0], [0, 0, 0, 0, 0, 0, 1, 1], [0, 0, 0, 0, 0, 1, 0, 1]]
y = [[0, 0, 0, 0, 0, 0, 1, 0], [0, 0, 0, 0, 0, 0, 1, 1], [0, 0, 0, 0, 0, 1, 0, 1], [0, 0, 0, 0, 0, 1, 0, 0], [0, 0, 0, 0, 0, 1, 1, 0]]
Test set length = [0, 0, 0, 0, 0, 1, 0, 1]

Mean of X = [0, 0, 0, 0, 0, 0, 1, 1] (3)
Mean of y = [0, 0, 0, 0, 0, 1, 0, 0] (4)

Slope = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1] (1)
```

Figure 19: Result of the garbled circuit slope calculation

We can now combine both means $\bar{x} = 3$ and $\bar{y} = 4$, and the slope $\beta_1 = 1$ and execute the next step to acquire the intercept β_0 of the linear regression function.

$$\begin{aligned} \beta_0 &= \bar{y} - \beta_1 \cdot \bar{x} \\ &= 4 - 1 \cdot 3 \\ &= 4 - 3 \\ &= 1 \end{aligned}$$

The intercept of the linear regression line is $\beta_0 = 1$, the same as the slope.

```
*** Starting linear regression training with binary circuits ***

Initializing training set:
X = [1 2 4 3 5]
y = [2 3 5 4 6]

8-bit binary representation of X & y:
X_bin = [[0, 0, 0, 0, 0, 0, 0, 1], [0, 0, 0, 0, 0, 0, 1, 0], [0, 0, 0, 0, 0, 1, 0, 0], [0, 0, 0, 0, 0, 0, 1, 1], [0, 0, 0, 0, 0, 1, 0, 1]]
y_bin = [[0, 0, 0, 0, 0, 0, 1, 0], [0, 0, 0, 0, 0, 0, 1, 1], [0, 0, 0, 0, 0, 1, 0, 1], [0, 0, 0, 0, 0, 1, 0, 0], [0, 0, 0, 0, 0, 1, 1, 0]]

Calculated means of X & y:
X_mean = [0, 0, 0, 0, 0, 0, 1, 1] (3)
y_mean = [0, 0, 0, 0, 0, 1, 0, 0] (4)

Calculated slope:
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1] (1)

Calculated intercept:
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1] (1)
```

Figure 20: Result of the binary intercept calculation

Prediction

Now that we have calculated both coefficients, we can apply the linear regression equation on the test set's independent variable to make predictions - y will be calculated for each value of x_i.

```
*** Starting linear regression training with garbled circuits ***

8-bit binary training set:
X = [[0, 0, 0, 0, 0, 0, 0, 1], [0, 0, 0, 0, 0, 0, 1, 0], [0, 0, 0, 0, 0, 1, 0, 0], [0, 0, 0, 0, 0, 0, 1, 1], [0, 0, 0, 0, 0, 1, 0, 1]]
y = [[0, 0, 0, 0, 0, 0, 1, 0], [0, 0, 0, 0, 0, 0, 1, 1], [0, 0, 0, 0, 0, 1, 0, 1], [0, 0, 0, 0, 0, 1, 0, 0], [0, 0, 0, 0, 0, 1, 1, 0]]
Test set length = [0, 0, 0, 0, 0, 1, 0, 1]

Mean of X = [0, 0, 0, 0, 0, 0, 1, 1] (3)
Mean of y = [0, 0, 0, 0, 0, 1, 0, 0] (4)

Slope = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1] (1)

Intercept = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1] (1)
```

Figure 21: Result of the garbled circuit intercept calculation

$$\beta_0 = 1, \quad \beta_1 = 1, \quad x_i = (1, 2, 4, 3, 5)$$

$$y = \beta_0 + \beta_1 \cdot x$$

$$y = 1 + 1 \cdot 1 = 1 + 1 = 2$$

$$y = 1 + 1 \cdot 2 = 1 + 2 = 3$$

$$y = 1 + 1 \cdot 4 = 1 + 4 = 5$$

$$y = 1 + 1 \cdot 3 = 1 + 3 = 4$$

$$y = 1 + 1 \cdot 5 = 1 + 5 = 6$$

The predicted values y_p corresponding to x_i are: $y_p = (2, 3, 5, 4, 6)$

```
*** Starting linear regression training with binary circuits ***

Initializing training set:
X = [1 2 4 3 5]
y = [2 3 5 4 6]

8-bit binary representation of X & y:
X_bin = [[0, 0, 0, 0, 0, 0, 0, 1], [0, 0, 0, 0, 0, 0, 1, 0], [0, 0, 0, 0, 0, 1, 0, 0], [0, 0, 0, 0, 0, 0, 1, 1], [0, 0, 0, 0, 0, 1, 0, 1]]
y_bin = [[0, 0, 0, 0, 0, 0, 1, 0], [0, 0, 0, 0, 0, 0, 1, 1], [0, 0, 0, 0, 0, 1, 0, 1], [0, 0, 0, 0, 0, 1, 0, 0], [0, 0, 0, 0, 0, 1, 1, 0]]

Calculated means of X & y:
X_mean = [0, 0, 0, 0, 0, 0, 0, 1, 1] (3)
y_mean = [0, 0, 0, 0, 0, 0, 1, 0, 0] (4)

Calculated slope:
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1] (1)

Calculated intercept:
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1] (1)

Calculated predictions:
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0] (2)
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1] (3)
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1] (5)
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0] (4)
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0] (6)
```

Figure 22: Result of applying the binary regression equation

```
*** Starting linear regression training with garbled circuits ***

8-bit binary training set:
X = [[0, 0, 0, 0, 0, 0, 0, 1], [0, 0, 0, 0, 0, 0, 1, 0], [0, 0, 0, 0, 0, 1, 0, 0], [0, 0, 0, 0, 0, 0, 1, 1], [0, 0, 0, 0, 0, 1, 0, 1]]
y = [[0, 0, 0, 0, 0, 0, 1, 0], [0, 0, 0, 0, 0, 0, 1, 1], [0, 0, 0, 0, 0, 1, 0, 1], [0, 0, 0, 0, 0, 1, 0, 0], [0, 0, 0, 0, 0, 1, 1, 0]]
Test set length = [0, 0, 0, 0, 0, 1, 0, 1]

Mean of X = [0, 0, 0, 0, 0, 0, 1, 1] (3)
Mean of y = [0, 0, 0, 0, 0, 1, 0, 0] (4)

Slope = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1] (1)

Intercept = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1] (1)

Calculated predictions y for X:
Predicted value y for X0 = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0] (2)
Predicted value y for X1 = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1] (3)
Predicted value y for X2 = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1] (5)
Predicted value y for X3 = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0] (4)
Predicted value y for X4 = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0] (6)
```

Figure 23: Result of applying the garbled circuit regression equation

8.2 Evaluation

This proof of concept intended to demonstrate an example of how garbled circuits could be utilized for secure multi-party machine learning in the cloud. While some aspects were shown to be feasible, others need to be addressed should this implementation be elevated to a viable solution for real-world machine learning scenarios.

8.2.1 Performance, Complexity and Implementation Effort

One of the major concerns when implementing garbled circuits to securely compute a given function is the increased computational demand compared to executing this function publicly. Unfortunately, this is also prevalent in our proof of concept.

Compared to training the simple linear regression model using binary operations, the same training process is significantly slower when carried out with garbled circuits.

Run	Binary Circuits	Garbled Circuits
1	0.0016210079193115234	0.3389451503753662
2	0.0010771751403808594	0.3446238040924072
3	0.0011820793151855469	0.33850693702697754
4	0.0005500316619873047	0.379697322845459
5	0.0006239414215087891	0.3488502502441406

Table 5: Execution duration of Linear Regression training in seconds, using binary and garbled circuits

Even though the execution with garbled circuits takes less than a second, the increase in duration is quite large since it takes approximately 200 times as long as the execution with normal binary circuits. Considering that only a small training set was used, this number is likely to increase exponentially for larger sets or bigger numbers, as well as floating point numbers.

Increasing execution times are also an indicator of higher memory consumption. The larger the circuits get, the more objects have to be created and consequently, more memory is required.

Despite linear regression being one of the simplest machine learning models, combining it with garbled circuits took a comparatively high implementation effort because all necessary circuits have to be constructed and correctly connected to each other on the fly. Depending on the training data, the amount of necessary circuits as well as the combinations with each other may differ significantly.

8.2.2 Security

As already mentioned in chapter 4.5, as the garbled circuit protocol offers a strong privacy guarantee and thus is an inherently secure protocol, the proof of concept can be considered secure on this level as well. Hence, any potential vulnerabilities would stem from the implementation itself.

Two of the templates created for this implementation required the disclosure of intermediate values, more precisely the MSBs of some numbers as well as one instance where an entire number needed to be revealed. While the MSB's contain information about the sign of the given number, there is no way for an adversary to derive the number itself from a single bit, especially without knowing anything about the number's bit length. When we look at the division template that discloses even more information, we find that the remainder is only revealed in full when it equals 0. An adversary would not be able to make much sense of this information without also knowing the context in which the number is used.

While the above can certainly be seen as a deviation from the original protocol because the implementation reveals intermediate outputs, they are only utilized by the system in order to determine its next steps. Due to the fact that none of these intermediate values are ever made available to the evaluator, we could argue that the proof of concept remains secure at the protocol level and that it does not introduce any additional attack vectors on implementation level.

8.2.3 Advantages and Drawbacks

The training sets used for linear regression models may vary greatly, which is why the implementation features circuit templates for all necessary arithmetic operations. This structure gives the system the adaptability to not only work with binary numbers of arbitrary length but also with sets of varying sizes, thus enhancing its scalability.

Although we are not reusing any of the circuits created during model training as a whole, the templates offer reusable components like gates or wires. Not creating fully reusable circuits surely increases the overall complexity of the system but in return allows for the intermediate evaluation of subcircuits within the training process, possibly leading to lower computational demands. Due to their modularity, more generic patterns could be generated from these templates, which could then be used for different machine learning algorithms like logistic regression or binary trees.

Due to incorporating a data preparation layer as well, users only need to input their training data - everything else is handled by the system itself, including circuit creation, making it easy to use even for users who are not familiar with the concept of garbled circuits. Not only does the proof of concept show the feasibility of model training, it is able to handle predictions as well.

Despite demonstrating that it is possible to perform linear regression model training, the proof of concept still comes with some drawbacks we have to address. Currently, it is not possible to perform any training (or calculations, for that matter) on floating point numbers because the implementation would have gone beyond the scope of this thesis due to a significant increase in overall complexity. All inputs, intermediate and final results must be integers, otherwise no training can be performed. Due to this restriction the proof of concept utilizes a carefully fabricated data set in order to perform a single training iteration.

So far the system is only capable of the simplest form of garbling and includes none of the crucial practical optimizations necessary to speed up the evaluation of more complex circuits. At the moment there is no possibility for a second party to add their inputs and execute the training on the combined data, which is in contrast to one of the points the implementation intended to prove. The cloud currently serves as the only evaluator and while this obviously limits the system, we can still demonstrate that it guarantees privacy and security when training a linear regression model.

8.2.4 Future Work

Considering the proof of concept, there are a number of improvements that could be made to the current implementation, the most crucial being the upgrade to floating point arithmetic. Due to the present limitation to integers it cannot be used in real-world training scenarios but rather has to be supplied with fabricated test data to avoid any occurrence of floating point numbers. Enabling a second party (or more parties) to add their own training data and consequently take on the role of the Evaluator is the next logical step in the given context. This would allow for joint model training and thus make the system a more attractive option for secure machine learning.

To allow more use cases than just training and making predictions, future work could include the implementation of the model optimization process because there are numerous applications in which optimization is crucial, especially when training is performed with very large datasets.

Optimization is often done by using *Gradient Descent*, an iterative algorithm with which a local minimum of a given function can be determined. When used to optimise a linear regression model, the algorithm will iteratively update its coefficients (the slope β_1 and the intercept β_0) until the best-fit line is found.

As previously mentioned in chapter 7.1, this is done by differentiating and calculating the *MSE* for both β_0 and β_1 to acquire their respective partial derivatives.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2$$

Differentiating the MSE with respect to β_0:

$$\begin{aligned}
MSE'_{\beta_0} &= \frac{\partial MSE(\beta_0, \beta_1)}{\partial \beta_0} \\
&= \frac{\partial}{\partial \beta_0}\left[\frac{1}{n}\left(\sum_{i=1}^{n}(\hat{y}_i - y_i)^2\right)\right] \\
&= \frac{1}{n}\left[\sum_{i=1}^{n} 2(\hat{y}_i - y_i)\left(\frac{\partial}{\partial \beta_0}(\hat{y}_i - y_i)\right)\right] \\
&= \frac{1}{n}\left[\sum_{i=1}^{n} 2(\hat{y}_i - y_i)\left(\frac{\partial}{\partial \beta_0}(\beta_0 + \beta_1 x_i - y_i)\right)\right] \\
&= \frac{1}{n}\left[\sum_{i=1}^{n} 2(\hat{y}_i - y_i)(1 + 0 - 0)\right] \\
&= \frac{1}{n}\left[\sum_{i=1}^{n} (\hat{y}_i - y_i)(2)\right] \\
&= \frac{2}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)
\end{aligned}$$

Differentiating the MSE with respect to β_1:

$$
\begin{aligned}
MSE'_{\beta_1} &= \frac{\partial MSE(\beta_0, \beta_1)}{\partial \beta_1} \\
&= \frac{\partial}{\partial \beta_1} \left[\frac{1}{n} \left(\sum_{i=1}^{n} (\hat{y}_i - y_i)^2 \right) \right] \\
&= \frac{1}{n} \left[\sum_{i=1}^{n} 2(\hat{y}_i - y_i) \left(\frac{\partial}{\partial \beta_1} (\hat{y}_i - y_i) \right) \right] \\
&= \frac{1}{n} \left[\sum_{i=1}^{n} 2(\hat{y}_i - y_i) \left(\frac{\partial}{\partial \beta_1} (\beta_0 + \beta_1 x_i - y_i) \right) \right] \\
&= \frac{1}{n} \left[\sum_{i=1}^{n} 2(\hat{y}_i - y_i)(0 + x_i - 0) \right] \\
&= \frac{1}{n} \left[\sum_{i=1}^{n} (\hat{y}_i - y_i)(2x_i) \right] \\
&= \frac{2}{n} \sum_{i=1}^{n} (\hat{y}_i - y_i) \cdot x_i
\end{aligned}
$$

The updated coefficients serve as inputs for the next iteration of the gradient descent algorithm. As soon as the minimum squared error is achieved, the values will be used to make an updated prediction using the linear regression function.
Both of these functions could be broken down similarly to training and prediction, like we have done in chapter 7.3.

Apart from enhancing its functionality, future work on this project could also include practical optimizations to boost its speed or lower the overall computational cost of circuit construction. There are a few different approaches to circuit optimizations, ranging from computing certain gates without any cost (e.g. *FreeXOR*) to reducing the number of ciphertexts each gate contains (and thus reducing the size of all garbled tables within the circuit; like *Garbled Row Reduction*). Furthermore, the templates created for the individual arithmetic operations can certainly be improved by reducing their complexity and optimizing their overall resource usage.

9 Outlook on Garbled Circuits in Machine Learning

The primary aim of this thesis was to determine whether garbled circuits are a suitable tool for secure training of a machine learning model and develop a proof of concept that demonstrates an exemplary training process. While possible, only the simplest form of both garbling and model have been implemented.

In order to make garbled circuits a more attractive and versatile option for secure machine learning, more of the known optimizations, like *point-permute* or *free XOR* should be implemented.

Despite simple linear regression being a rather simple equation with only a few parameters and calculation steps, the number of circuits that need to be garbled and evaluated is already very high. Therefore, augmenting the training data or calculating binaries with an increasing number of bits would most likely significantly slow down the training process. Within a cloud environment it might not seem like an issue at first glance but in order to prevent this deceleration one would need to increase computation power, which in turn would increase the usage cost.

In general, the outlook on garbled circuits in machine learning is promising even though there still are some challenges that need to be overcome.

References

[1] Donald Beaver, Silvio Micali, and Phillip Rogaway. "The round complexity of secure protocols". In: *Symposium on the Theory of Computing*. 1990. URL: https://api.semanticscholar.org/CorpusID:1578121 (visited on 09/07/2024).

[2] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. "Foundations of garbled circuits". In: *Proceedings of the 2012 ACM Conference on Computer and Communications Security*. CCS '12. Raleigh, North Carolina, USA: Association for Computing Machinery, 2012, pp. 784–796. ISBN: 9781450316514. DOI: https://doi.org/10.1145/2382196.2382279.

[3] Vitalik Buterin. *Garbled Circuits: Basic Scheme and Applications*. 2022. URL: https://hackernoon.com/garbled-circuits-basic-scheme-and-applications (visited on 01/16/2024).

[4] ClickSSL. *What is Symmetric Encryption? Symmetric-Key Algorithms*. 2024. URL: https://www.clickssl.net/blog/what-is-symmetric-encryption (visited on 09/06/2024).

[5] Benjamin D. Esham. *A table of the substitutions achieved with the ROT-13 cipher with the Latin alphabet, and an example.* 2007. URL: https://commons.wikimedia.org/w/index.php?title=File:ROT13%5C_table%5C_with%5C_example.svg%5C&oldid=891208936 (visited on 09/06/2024).

[6] David Evans, Vladimir Kolesnikov, and Mike Rosulek. *A Pragmatic Introduction to Secure Multi-Party Computation*. Now Publishers, 2018. ISBN: 978-1-680-83508-3.

[7] GeeksforGeeks. *Asymmetric Key Cryptography*. 2024. URL: https://www.geeksforgeeks.org/asymmetric-key-cryptography/ (visited on 09/06/2024).

[8] Google. *Benefits of Cloud Computing*. URL: https://cloud.google.com/learn/advantages-of-cloud-computing#section-3 (visited on 01/16/2024).

[9] Matan Hamilis. *Garbled Circuits: A Primer*. 2021. URL: https://hackmd.io/@matan/garbled_circuits#Garbled-Circuits-A-Primer (visited on 01/16/2024).

[10] Baivab Kumar Jena. *AES Encryption: Secure Data with Advanced Encryption Standard*. 2024. URL: https://www.simplilearn.com/tutorials/cryptography-tutorial/aes-encryption (visited on 09/06/2024).

[11] Vladimir Kolesnikov, Payman Mohassel, and Mike Rosulek. "FleXOR: Flexible Garbling for XOR Gates That Beats Free-XOR". In: *Advances in Cryptology - CRYPTO 2014*. Berlin, Heidelberg: Springer, 2014, pp. 440–457. ISBN: 978-3-662-44381-1. DOI: https://doi.org/10.1007/978-3-662-44381-1_25.

[12] Vladimir Kolesnikov and Thomas Schneider. "Improved Garbled Circuit: Free XOR Gates and Applications". In: *Automata, Languages and Programming*. Berlin, Heidelberg: Springer, 2008, pp. 486–498. ISBN: 978-3-540-70583-3. DOI: https://doi.org/10.1007/978-3-540-70583-3_40.

[13] Theo Lynn et al. *Measuring the Business Value of Cloud Computing*. Singapore: Springer Nature, 2020, p. 22. ISBN: 978-3-030-43198-3. DOI: https://doi.org/10.1007/978-3-030-43198-3_2.

[14] Olivier Markowitch, Liran Lerman, and Gianluca Bontempi. "Side channel attack: An approach based on machine learning". In: *Constructive Side-Channel Analysis and Secure Design, COSADE*. Feb. 2011.

[15] Peter Mell and Tim Grance. *The NIST Definition of Cloud Computing*. Gaithersburg: Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology, 2011. DOI: https://doi.org/10.6028/NIST.SP.800-145.

[16] Moni Naor, Benny Pinkas, and Reuben Sumner. "Privacy preserving auctions and mechanism design". In: *Proceedings of the 1st ACM Conference on Electronic Commerce*. EC '99. Denver, Colorado, USA: Association for Computing Machinery, 1999, pp. 129–139. ISBN: 1581131763. DOI: https://doi.org/10.1145/336992.337028.

[17] Alessandro Nassiri. *Enigma (crittografia) - Museo scienza e tecnologia Milano.* 2012. URL: https://commons.wikimedia.org/wiki/File:Enigma%5C_(crittografia)%5C_-%5C_Museo%5C_scienza%5C_e%5C_tecnologia%5C_Milano.jpg (visited on 09/06/2024).

[18] Ignacio Navarro. *Gabes*. 2018. URL: https://github.com/nachonavarro/gabes (visited on 09/16/2024).

[19] Ignacio Navarro. "On Garbled Circuits". Imperial College London, 2018, pp. 4–11. URL: https://www.imperial.ac.uk/media/imperial-college/faculty-of-engineering/computing/public/1718-ug-projects/Ignacio-Navarro-On-Garbled-Circuits.pdf (visited on 09/06/2024).

[20] Umang H Patel. "Secure Multi-Party Computation (SMPC) For Privacy-Preserving Data Analysis". In: *International Journal of Creative Research Thoughts (IJCRT)* 12.4 (2024). ISSN: 2320-2882. URL: https://www.ijcrt.org/papers/IJCRT2404250.pdf (visited on 09/07/2024).

[21] Benny Pinkas et al. "Secure Two-Party Computation Is Practical". In: *Advances in Cryptology - ASIACRYPT 2009*. Berlin, Heidelberg: Springer, 2009, pp. 250–267. ISBN: 978-3-642-10366-7. DOI: https://doi.org/10.1007/978-3-642-10366-7_15.

[22] Jennifer Ritz. "Post-Quantum Kryptographie - Evaluation Quantencomputer-resistenter Public-Key-Verfahren basierend auf der Implementierung auf klassischen Computern". Master's Thesis. Hochschule Wismar, 2020, pp. 26–27.

[23] Kelley Robinson. *What is Public Key Cryptography?* 2018. URL: https://www.twilio.com/en-us/blog/what-is-public-key-cryptography (visited on 09/06/2024).

[24] Ayush Saha. *Understanding the Vigenère Cipher*. 2023. URL: https://dev.to/cognivibes/understanding-the-vigenere-cipher-16g5 (visited on 09/06/2024).

[25] Meruja Selvamanikkam. *Digital Signature Generation*. 2018. URL: https://meruja.medium.com/digital-signature-generation-75cc63b7e1b4 (visited on 09/06/2024).

[26] Jagjit Singh. *Known-plaintext attacks, explained*. 2023. URL: https://cointelegraph.com/explained/known-plaintext-attacks-explained (visited on 09/16/2024).

[27] Dharmendra Thirunavukkarasu. *Cracking the Code: A Deep Dive into the Scytale Cipher and Its Modern-Day Counterpart, Homoglyph Attacks*. 2023. URL: https://www.linkedin.com/pulse/serpentine-connections-how-griekenland-scytale-modern-dharmendra/ (visited on 09/06/2024).

[28] Linus Töbke. "Quantenresistente vertrauliche Cloud-Nutzung". Master's Thesis. Hochschule Wismar, 2022, p. 40.

[29] Yongge Wang, Qutaibah M. Malluhi, and Khaled MD Khan. "Garbled computation in cloud". In: *Future Generation Computer Systems* 62 (2016), pp. 54–65. ISSN: 0167-739X. DOI: https://doi.org/10.1016/j.future.2015.11.004.

[30] Andrew C. Yao. "Protocols for secure computations". In: *23rd Annual Symposium on Foundations of Computer Science (sfcs 1982)*. 1982, pp. 160–164. DOI: 10.1109/SFCS.1982.38.

[31] Samee Zahur, Mike Rosulek, and David Evans. "Two Halves Make a Whole". In: *Advances in Cryptology - EUROCRYPT 2015*. Berlin, Heidelberg: Springer, 2015, pp. 220–250. ISBN: 978-3-662-46803-6. DOI: https://doi.org/10.1007/978-3-662-46803-6_8.

List of Figures

List of Tables

List of Listings

Attachments

- `/mtkd_files` ..... root folder
  - `code` ..... the proof of concept source code
    - `binary_calc` ..... classes for model training with binary circuits
    - `garbled_circuits` ..... classes for garbled circuit construction
    - `linear_regression` ..... the templates for secure model training
    - `templates` ..... template classes for secure model training
  - `kd_thesis.pdf` ..... pdf file containing this master's thesis

Due to the size all source code mentioned but not added to this document is contained in the accompanying folder. The secure training templates mentioned throughout this thesis can be found in the `templates` subfolder.

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe. Die Stellen der Arbeit, die anderen Quellen im Wortlaut oder dem Sinn nach entnommen wurden, sind durch Angaben der Herkunft kenntlich gemacht. Dies gilt auch für Zeichnungen, Skizzen, bildliche Darstellungen sowie für Quellen aus dem Internet.

Ich erkläre ferner, dass ich die vorliegende Arbeit in keinem anderen Prüfungsverfahren als Prüfungsarbeit eingereicht habe oder einreichen werde.

Die eingereichte schriftliche Arbeit entspricht der elektronischen Fassung. Ich stimme zu, dass eine elektronische Kopie gefertigt und gespeichert werden darf, um eine Überprüfung mittels Anti-Plagiatssoftware zu ermöglichen.

Ort, Datum Unterschrift