NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCE
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

POSTGRADUATE STUDIES
“COMPUTER SYSTEMS: SOFTWARE AND HARDWARE”

MASTER THESIS

Privacy Preserving Medical Data Analytics using Secure Multi Party Computation. An End-To-End Use Case.

Athanasios G. Giannopoulos
Dimitris I. Mouris

Supervisors: **Yannis E. Ioannidis,** Professor NKUA
Minos N. Garofalakis, Professor TUC

ATHENS

SEPTEMBER 2018

ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΜΕΤΑΠΤΥΧΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
"ΥΠΟΛΟΓΙΣΤΙΚΑ ΣΥΣΤΗΜΑΤΑ: ΛΟΓΙΣΜΙΚΟ ΚΑΙ ΥΛΙΚΟ"

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Στατιστικές Αναλύσεις με Σεβασμό στην Ιδιωτικότητα Ιατρικών Δεδομένων χρησιμοποιώντας Ασφαλή Υπολογισμό Πολλαπλών Συμμετεχόντων. Μία Ολοκληρωμένη Περίπτωση Χρήσης.

Αθανάσιος Γ. Γιαννόπουλος
Δημήτρης Η. Μούρης

Επιβλέποντες: **Ιωάννης Ε. Ιωαννίδης,** Καθηγητής ΕΚΠΑ
Μίνως Ν. Γαροφαλάκης, Καθηγητής ΠΚ

ΑΘΗΝΑ

ΣΕΠΤΕΜΒΡΙΟΣ 2018

MASTER THESIS

Privacy Preserving Medical Data Analytics using Secure Multi Party Computation. An End-To-End Use Case.

Athanasios G. Giannopoulos **R.N.:** M1529
Dimitris I. Mouris **R.N.:** M1534

Authors had equal contribution and are listed in alphabetical order.

SUPERVISORS: **Yannis E. Ioannidis,** Professor NKUA
Minos N. Garofalakis, Professor TUC

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Στατιστικές Αναλύσεις με Σεβασμό στην Ιδιωτικότητα Ιατρικών Δεδομένων χρησιμοποιώντας Ασφαλή Υπολογισμό Πολλαπλών Συμμετεχόντων. Μία Ολοκληρωμένη Περίπτωση Χρήσης.

Αθανάσιος Γ. Γιαννόπουλος **Α.Μ.:** Μ1529
Δημήτρης Η. Μούρης **Α.Μ.:** Μ1534

Οι συγγραφείς είχαν ίση συνεισφορά και τα ονόματά τους παρατίθενται σε αλφαβητική σειρά.

ΕΠΙΒΛΕΠΟΝΤΕΣ: **Ιωάννης Ε. Ιωαννίδης,** Καθηγητής ΕΚΠΑ
Μίνως Ν. Γαροφαλάκης, Καθηγητής ΠΚ

ABSTRACT

The new era of big data demands high performance computing, since the amount of data published online is growing exponentially. Cloud computing has emerged as a result, providing strong computational power for both individuals and companies. Though cloud computing is the answer to many business models, there are many use-cases where cloud fails to meet the demands of information privacy. For instance, exposing financial and medical information to the cloud may violate the individuals' right to privacy. People are not comfortable sharing their sensitive data, and more importantly, they do not trust any cloud provider with this information; data that are uploaded in the cloud can be exposed to attacks from both the cloud provider and third parties.

Nevertheless, there are many real world use cases that use information from different parties to jointly compute meaningful results, but due to the aforementioned limitations, some are avoided and others do not always respect data privacy. The solution to this is a technique called Secure Multi-Party Computation (SMPC or MPC), which leverages cryptographic primitives to carry out computations on confidential data, computing a function and learning nothing more than what the N parties would have if a separate trusted party had collected their inputs, computed the same function for them, and then return the result to all parties.

Motivated by this wide range of applications, in this thesis we have focused on providing an end-to-end infrastructure for computing privacy-preserving analytics. More specifically, we have developed algorithms specifically tailored to encrypted architectures and in the SMPC scenario, such as secure aggregators and secure decision tree classifiers. Moreover, we have focused on the coordination and communication between all involved parties; those who provide their data, those who perform the secure computation, and finally those that initiate new computations. Our algorithms are not dependent to the application that our systems serves, however, in order to demonstrate it, in this thesis we use hospitals as data providers and we focus on medical research. Our goal is to establish an end-to-end system for discovering useful information with respect to data privacy, and also to provide the building blocks for potentially more elaborate privacy-preserving algorithms.

SUBJECT AREA: Privacy-Preserving Computation, Privacy-Preserving Data Mining, Secure Multi-Party Computation

KEYWORDS: Secure Multi-Party Computation, Privacy-Preserving Data Mining, Histogram, Decision Tree Classifier, Medical Data

ΠΕΡΙΛΗΨΗ

Η νέα εποχή των μεγάλων δεδομένων απαιτεί μεγάλη υπολογιστική ισχύ, αφού το πλήθος των δεδομένων που δημοσιεύονται στο διαδίκτυο μεγαλώνει εκθετικά. Σαν αποτέλεσμα, προέκυψαν τα Νέφη Υπολογιστικών Συστημάτων, παρέχοντας μεγάλη υπολογιστική ισχύ, τόσο για ιδιώτες όσο και για επιχειρήσεις. Παρόλο που τα υπολογιστικά νέφη είναι η απάντηση σε πολλά επιχειρηματικά μοντέλα, υπάρχουν πολλές περιπτώσεις χρήσης όπου τα υπολογιστικά νέφη αποτυγχάνουν να καλύψουν τις απαιτήσεις ιδιωτικότητας των πληροφοριών. Για παράδειγμα, εκθέτοντας οικονομικές και ιατρικές πληροφορίας στο νέφος μπορεί να παραβιάζει το δικαίωμα των ατόμων στην ιδιωτικότητα. Οι άνθρωποι δεν νιώθουν άνετα με το να μοιράζονται τα ευαίσθητα δεδομένα τους, και πιο σημαντικά, δεν εμπιστεύονται κανέναν πάροχο υπολογιστικού νέφους με τις πληροφορίες αυτές. Τα δεδομένα που μεταφορτώνονται στο νέφος μπορεί να εκτεθούν σε επιθέσεις τόσο από τον πάροχο όσο και από τρίτους.

Παρόλα αυτά, υπάρχουν πολλές πραγματικές περιπτώσεις χρήσης που χρησιμοποιούν πληροφορίες από διαφορετικές οντότητες προκειμένου να υπολογίσουν από κοινού ουσιαστικά αποτελέσματα, αλλά λόγω των προαναφερθέντων περιορισμών, κάποιες από αυτές αποφεύγονται και άλλες δεν σέβονται πάντα την ιδιωτικότητα των δεδομένων. Η λύση σε αυτό είναι μία τεχνική που ονομάζεται Ασφαλής Υπολογισμός Πολλαπλών Συμμετεχόντων, η οποία αξιοποιεί θεμελιώδεις κρυπτογραφικές ιδιότητες προκειμένου να εκτελέσει υπολογισμούς πάνω από εμπιστευτικά δεδομένα, υπολογίζοντας μία συνάρτηση και μαθαίνοντας τίποτε παραπάνω σε σχέση με το τι θα μάθαιναν N συμμετέχοντες, εαν μία ξεχωριστή έμπιστη οντότητα είχε συλλέξει τις εισόδους τους, είχε εκτελέσει την ίδια συνάρτηση, και τέλος επέστρεφε το αποτέλεσμα σε όλους τους συμμετέχοντες.

Παίρνοντας κίνητρο από αυτό το ευρύ φάσμα εφαρμογών, στην εργασία αυτή επικεντρωθήκαμε στο να παρέχουμε μία ολοκληρωμένη υποδομή για υπολογισμό στατιστικών αναλύσεων με σεβασμό στην ιδιωτικότητα. Πιο συγκεκριμένα, έχουμε υλοποιήσει αλγορίθμους ειδικά σχεδιασμένους για κρυπτογραφημένες αρχιτεκτονικές, χρησιμοποιώντας το σενάριο του Ασφαλή Υπολογισμού Πολλαπλών Συμμετεχόντων, όπως ασφαλείς συγκεντρωτικούς αλγορίθμους και ασφαλείς κατηγοριοποιητές με δέντρα απόφασης. Ακόμα, συγκεντρωθήκαμε στο συντονισμό και την επικοινωνία μεταξύ όλων των συμμετεχόντων. Αυτών που παρέχουν δεδομένα, αυτών που εκτελούν τον ασφαλή υπολογισμό και τέλος αυτών που ξεκινούν νέους υπολογισμούς. Οι αλγόριθμοί μας δεν εξαρτώνται από την εφαρμογή που εξυπηρετεί το σύστημά μας, παρόλα αυτά, για λόγους παρουσίασης, στην εργασία αυτή χρησιμοποιούμε νοσοκομεία σας παρόχους δεδομένων και επικεντρωνόμαστε στην ιατρική έρευνα. Ο Στόχος μας είναι να ιδρύσουμε ένα ολοκληρωμένο σύστημα με σκοπό την ανακάλυψη χρήσιμης πληροφορίας με σεβασμό στην ιδιωτικότητα, και επίσης να προσφέρουμε τα δομικά στοιχεία για τυχόν πιο πολύπλοκους αλγορίθμους με σεβασμό στην ιδιωτικότητα.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Υπολογισμοί με Σεβασμό στην Ιδιωτικότητα, Εξόρυξη Δεδομένων με Σεβασμό στην Ιδιωτικότητα, Ασφαλείς Υπολογισμοί μεταξύ πολλαπλών Συμμετεχόντων

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: Ασφαλής Υπολογισμός Πολλαπλών Συμμετεχόντων, Εξόρυξη Δεδομένων με Σεβασμό στην Ιδιωτικότητα, Ιστόγραμμα, Κατηγοριοποιητής Δέντρου Απόφασης, Ιατρικά Δεδομένα

This master thesis is dedicated to our parents
Giorgos and Aggeliki
Ilias and Eirini

ACKNOWLEDGEMENTS

To start, we would like to thank our advisor Prof. Yannis E. Ioannidis for the chance he gave us to work on this research project and also for the insightful discussions we had during this thesis.

We are particularly thankful for the help and advice of Minos N. Garofalakis and Omiros Metaxas all these months.

Last but not least, we thank ATHENA Research Center for the support of this thesis in terms of My Health - My Data (MHMD) project. We would also like to express our gratitude to the MHMD team, and especially Christos Nasikas, for the fruitful discussions.

The work in this thesis was partially supported by the European Union's Horizon 2020 research and innovation programme under grant agreement No. 732907 ("My Health, My Data").

September 2018

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

LIST OF SOURCE CODES

1. INTRODUCTION

Arguing that you don't care about the right to privacy because you have nothing to hide is no different than saying you don't care about free speech because you have nothing to say.

Edward Snowden [57]

Big data is growing exponentially, 90 percent of which has been created in the past few years [37]. People are constantly producing and publishing information about themselves. Such data comes from browsing the web, talking to someone online, moving around emitting GPS signal, being registered by cameras or credit card usage, or even wearables and IoT devices. Many companies and organizations, such as Google and Facebook, make profit from big data analytics, collection and storage.

One of the most common examples is personalized advertisements, occurring from the massive data analytics. It is becoming increasingly common for data about our location, music and movies we like, private conversations, and any other online trace we leave behind, to be linked to our purchasing preferences.

The aforementioned tracking and profiling comes from breaching of the individuals' privacy. Privacy is the ability of individuals to have control over how their personal information is collected and used, and thereby express themselves selectively. Nowadays, the need of preserving someone's privacy is more crucial than ever. For instance, financial information can be sensitive. Such information includes a person's holdings, debts and transactions (*e.g.* purchases). This information, if compromised, can lead criminal activity such as fraud or identity theft. Also, one's purchases can be linked to places they visit, people they contact and so on; thus, such data should remain private.

A noteworthy example that renders privacy of critical importance is *medical data*. People may not be comfortable sharing their medical records to others, due to several reasons. For instance, it could affect their employment, their insurance coverages, or people just do not want others to know about their medical or psychological conditions or treatments. Medical data reveals a lot for a patient's personal life and therefore should be protected.

An argument adopted by many, is that there is no need for privacy if you have nothing to hide. This shows a failure to understand that privacy is a human right. There is no need to justify why such a right is needed. The burden of justification falls on the one seeking to violate this right. Even when a right is not useful to you, you can't give away the right of others.

1.1 Privacy Issues in the Cloud & Multi-Party Computing

This rapid growth of information has resulted in the consistently growing popularity of cloud computing, which offers strong computational power for both individuals and companies. At the same time, all data that are uploaded in the cloud can be exposed to attacks from both the cloud provider and third parties. However, in the case of financial and medical data, people are not comfortable sharing their sensitive data, and more importantly, they do not trust any third party with this information.

There are many real world use cases and business models that use information from different parties to compute jointly meaningful results, but due to the aforementioned limitations, some are avoided and others do not always respect data privacy. The solution to this, is technique called secure multi-party computation (SMPC or MPC) [28, 63], which leverages cryptographic primitives to carry out computations on confidential data. Having N parties with private inputs, the goal is to compute a function and learn nothing more than what they would have if a separate trusted party had collected their inputs, computed the same function for them, and then return the result to all parties.

Real world examples are unlimited; for instance, Sharemind [13] – a platform for secure computations – mentions the example of satellite collision [36], since the number of satellites orbiting the planet is growing and thus the danger of collisions is also growing. Indeed, two satellites crashed in 2009. Satellite owners are not willing to make the orbits of their satellites public. However, this – and future – collisions could be avoided by sharing information about the satellites orbits. Using MPC, the parties can cooperate and learn whether a collision is going to happen and nothing else. No information about the actual orbits would leak, since computations are carried out on encrypted data.

Another interesting example is presented in [41], where in the late 1990s, the Canadian Government maintained a massive federal database that pooled citizen data from a number of different government ministries, with aim to implement governmental research that would arguably improve the services received by citizens. This database became known as the “big brother” database, despite that was officially called the Longitudinal Labor Force File. Fortunately, the people protested and the project was discontinued due to privacy concerns. As in the example of medical research, here, individuals data privacy would have been exposed, rendering the need of privacy-preserving algorithms of crucial importance.

1.2 Our Contribution

In this thesis, our primarily concern is to create an end-to-end infrastructure for computing privacy preserving analytics such as [2, 41]. We have developed algorithms specifically tailored to encrypted architectures and in the SMPC scenario, but also we have focused on the coordination and communication between all involved parties; those who provide their data, those who perform the secure computation, and finally those that initiate new computations. In our view, this thesis provides an end-to-end system for discovering useful information with respect to data privacy. In our system, we have developed some essential analytics algorithms – such as aggregators and decision trees. Our goal is to provide the building blocks for potentially more elaborate algorithms to be implemented with respect to data privacy.

In the context of this thesis, our study is focused on medical data, which has been a popular data mining topic of late [18, 25]. However, the primary reason that we focus on medical data is that the privacy protection of medical records is taken more seriously than other data mining tasks [9]. Medical records are related to humans, which renders privacy of critical importance. Thus, medical data constitute an example that demonstrates the necessity of privacy preserving algorithms and also for a comprehensive infrastructure that incorporates and facilitates all participating parties.

Although in this thesis we have focused on medical data, our end-to-end infrastructure is oblivious to the type of data that it processes. The same analytics will be applied whether it would perform medical research on hospitals data, or highly classified statistics for governments data, or even private computation for preventing satellite collisions. This variety of applications that can be served through our system, are mainly tied to two data types – continuous[1] and categorical[2] data. Examples of the former category include weight, price, profits, etc, where some categories of the latter type of data include product-type, gender, age-group, etc. This heterogeneity of data types has separated our privacy-preserving algorithms in two corresponding categories, since different data types are managed in different ways. The algorithms we have developed for privacy-preserving analytics can deal with both quantitative and categorical data.

1.3 Thesis Structure

The rest of the thesis is organized as follows: In section 2, we examine some fundamental cryptographic protocols that are essential for the subsequent sections. In section 4 we present the Sharemind secure multiparty computation framework, while in section 5 we elaborate in our end-to-end medical case study. Consecutively, in section 6 we present the basic notion of privacy-preserving algorithms and how they are different from their textbook equivalents. Moreover, we elaborate on details of the algorithms of the two major categories we have developed, secure aggregation and secure classification. In section 7 we delve into the various implementation details of our system. Our experimental evaluation is presented in section 9 and finally, our conclusions and future work goals are summarized in section 10.

[1] *Continuous data* is data where the values can change continuously, rendering uncountable the number of different possible values.

[2] *Categorical data* refer to those aspects of data where there is a distinction between different groups; the number of possible values/categories are small and can be counted.

2. PRELIMINARIES

In this section, we present the basic principles of cryptography and also some fundamental cryptographic protocols that are essential for the subsequent sections.

2.1 Encryption

The fields of cryptography, privacy, computer and information security, design and utilize software, hardware, and human resources to address the issue of sensitive information and private communication. One of the most commonly used ways of preserving data privacy is the use of encryption. Encryption is the process of encoding a message (*plaintext*), such that it is accessible only to authorized parties, and no one else. This is accomplished using an encryption algorithm (*cipher*) that operates on the plaintext and produces a *ciphertext*. The ciphertext needs to be decrypted in order to be read and a decryption algorithm is responsible for this operation. Both algorithms use a *key* in order to operate on the plaintext. The distribution of the key(s) ensures the authorization of the parties involved.

Ultimately, the basic building blocks of an encryption scheme come down to three discrete components. These are the *Key Generation* mechanism and the *Encryption* and *Decryption* algorithms.

2.2 Symmetric-Key Encryption

The simplest form of encryption, is the use of a symmetric key scheme. Such a scheme utilizes a secret key which is used to perform the mathematical operations of both encrypting and decrypting of a message. The communicating parties must share the same key in order to communicate securely. Symmetric-key algorithms are algorithms that use the same key for both encryption of the plaintext and decryption of the ciphertext. The keys represent a shared secret between two or more parties that can be used to maintain a private information channel. The requirement that both parties have access to the secret key is one of the main drawbacks of symmetric key encryption, in comparison to public-key encryption schemes that we expand in section 2.3.

More formally, given a plaintext P and a secret key K, a ciphertext C is produced such that:

$$\begin{aligned} C &= Enc(P, K) \\ P = Dec(C, K) &= Dec(Enc(P, K), K) \end{aligned} \tag{2.1}$$

Where Enc and Dec are the *Encryption* and *Decryption* algorithms respectively.

2.3 Asymmetric-Key Encryption (Public-Key Encryption)

The major problem that traditional symmetric-key algorithms face is that they require the parties to exchange keys, or more specifically to agree to a secret common private key, prior to communicating. This needs a secure (physical or digital) communication channel accessible even for a short period of time, in order for the secret key to be exchanged.

Such a channel could be a piece of paper delivered by hand or by a trusted courier, or a short in person communication.

An asymmetric or public-key cryptosystem is one where different keys are employed for the operations in the cryptosystem (*e.g.*, encryption and decryption), and where one of the keys can be made public without compromising the secrecy of the other key [35]. That way there is no need for a secure channel or prior agreement.

Usually three algorithms are needed to define a public key cryptosystem. These are a *Key Generation* algorithm which produces the public and private keys (based on a security parameter η), an *Encryption* and a *Decryption* algorithm.

In the classic model such an encryption scheme involves a *public key* which is used for encryption and a *private key* which is used for decryption. More formally, given a plaintext P, a public key $PubKey$, and a private key $PrivKey$ we can produce a ciphertext C such that:

$$\begin{aligned} C &= Enc(P, PubKey) \\ P &= Dec(C, PrivKey) \end{aligned} \tag{2.2}$$

Where Enc and Dec are the *Encryption* and *Decryption* algorithms respectively.

If Alice wants to send a message to Bob, she needs Bob's public key which could be publicly available (*e.g.* on Bob's website). Alice encrypts her message with Bob's public key and sends the resulted ciphertext to Bob. Bob then needs his private key which is only in his possession in order to decrypt the ciphertext Alice sent and retrieve the original message. If Bob wishes to reply, he needs Alice's public key etc. Generally, the public (encryption) key could be shared to any party that wants to communicate with the owner of that public key, while the private (decryption) key should be kept secret and used by the recipient of the ciphertext to recover the original plaintext.

The ability for two users to establish a shared secret over an insecure communication channel, despite having no prior communication or information exchange was first proposed in 1976 by Whitfield Diffie and Martin Hellman. That method is named Diffie-Hellman (DH) key exchange [23] after its authors. The security of the algorithm is based on the difficulty of solving the Discrete Log Problem (DLP). The DLP is stated as follows: Given g, p and $g^k \pmod{p}$, find k.

Some of the most widely used public key cryptosystems are the RSA (Rivest – Shamir – Adleman) [54], the El Gamal [24] and the Paillier [50] Cryptosystems.

2.3.1 RSA

One if the first and most widely adopted public key cryptosystem is RSA . The algorithm's difficulty reduces to the difficulty of factoring the product of two large prime numbers. Its three basic algorithms are described below.

- *Key Generation*
  - Randomly select two large primes p and q.
  - Calculate modulus $n = p \cdot q$.
  - Calculate $\phi(n) = (p - 1) \cdot (q - 1)$.
  - Randomly select $e : gcd(e, \phi(n)) = 1$.

  - Calculate reverse $d = e^{-1} \pmod{\phi(n)}$.
    $d \cdot e = 1 \pmod{\phi(n)}$.
  - The public key is (n, e), while the private key is d. p, q and $\phi(n)$ must also be kept private.

- *Encryption*
  - Given a plaintext message m, we get a ciphertext $c = m^e \pmod{n}$.

- *Decryption*
  - Given a ciphertext c, we get back the plaintext m as follows: $c^d \pmod{n} = m^{ed} \pmod{n} = m$.

2.3.2 ElGamal

The ElGamal encryption system is a public-key cryptosystem based on the Diffie-Hellman key exchange. Its three basic algorithms are described below.

- *Key Generation*
  - Select two large primes p and $q : q \mid (p-1)$.
  - Select a generator g of group $\mathbb{G}$ which is a large enough order-q subgroup of the multiplicative group $\mathbb{Z}_p^*$ of integers between 1 and $p-1$.
  - Randomly select $x \in_R \mathbb{Z}_q$.
  - Calculate $y = g^x \pmod{p}$.
  - The public key is y, while the private key is x.

- *Encryption*
  - Given a plaintext message m, we get a ciphertext c as follows.
  - Randomly select $r \in_R \mathbb{Z}_q$.
  - Calculate $G = g^r \pmod{p}$.
  - Calculate $M = m \cdot y^r \pmod{p}$.
  - Return $c = (G, M)$

- *Decryption*
  - Given a ciphertext $c = (G, M)$, we get back the plaintext m as follows:
    $m = M/G^x \pmod{p}$.

2.3.3 Paillier

The Pallier cryptosystem is a public-key cryptosystem that relies its security upon the decisional composite residuosity assumption [50].

- *Key Generation*

- Randomly and independently select two large primes p and $q : gcd(p-1, q-1) = 1$ If both primes have the same length this property holds.
- Calculate the RSA modulus $n = p \cdot q$.
- Calculate $\lambda = lcm(p-1, q-1)$.
- Select generator $g \in \mathbb{Z}^*_{n^2}$ such that the order of g is a non zero multiple of n.
- Calculate $\mu = (L(g^\lambda \pmod{n^2}))^{-1} \pmod{n}$, where $L(x) = \frac{x-1}{n}$.
- The public key is (n, g), while the private key is (λ, μ).

• *Encryption*

- Given a plaintext message m, we get a ciphertext c as follows.
- Encode m into $\mathbb{Z}_n$.
- Randomly select $r \in_R \mathbb{Z}^*_n$.
- Return $c = g^m \cdot r^n \pmod{n^2}$.

• *Decryption*

- Given a ciphertext $c \in \mathbb{Z}_{n^2}$, we get back the plaintext m as follows: $m = L(c^\lambda \pmod{n^2}) \cdot \mu \pmod{n}$.

2.4 Homomorphic Encryption

Some encryption schemes are inherently “malleable”, which means that it is possible to transform a ciphertext into another ciphertext which decrypts to a related plaintext. Homomorphic encryption is a malleable encryption scheme that allows operations directly on encrypted data so that the results after decryption would correspond to applying matching operations on unencrypted data. For example, having an encryption mechanism Enc, the product of any two ciphertexts is equal to a ciphertext of the sum of the two corresponding plaintexts (M_1 and M_2). Or more generally the application of a function to the ciphertexts corresponds to another function on the plaintexts, as follows:

$$Enc(M_1) \otimes Enc(M_2) = Enc(M_1 \oplus M_2) \tag{2.3}$$

for some operations $\otimes$ and $\oplus$.

Homomorphic encryption enables outsourcing computations to a third party, as for example on the cloud. Cloud computing is designed to deal with difficult operations and with computationally demanding algorithms. However, if user's data are unencrypted they can be exposed to attacks from both the cloud provider and third parties (hackers, government agencies, data breaches, etc.). Even if the data are stored *(data at rest)* and transferred *(data in transit)* securely, when they are in use they can be exposed to security risks, such as side-channel attacks [67] and hardware Trojans [6, 59]. *Data at rest* implies data that is stored physically in any digital form, while *data in transit* means data traversing the network. Active data under constant change which is stored in a non-persistent digital state typically in computer random access memory (RAM), CPU caches, or CPU registers are called *data in use*.

It is possible, to construct an entire system to work over encrypted data, manipulating and returning encrypted results. This system would be based on homomorphic encryption, which allows to apply a function to the ciphertexts that corresponds to another function on the plaintexts, and in general enables computation on encrypted data. The final results can then be obtained by a single decryption. For instance, given $Enc(M_1)$ and $Enc(M_2)$ (the encryption of M_1 and M_2), you can compute $Enc(M_1 + M_2)$ without knowing M_1, M_2 nor the decryption key. The only point in the process where data would be decrypted is when the user wants to see the result, and that would presumably happen in the application or client software, not in the database server in the cloud, rendering the host incapable of leaking any type of information.

One challenge that homomorphic encryption schemes, and in general every encrypted computation framework, face is the inability to make runtime decisions based on encrypted data. Using homomorphic operations requires special care to ensure that branching does not reveal any sensitive data by observing side-channel information (*e.g.* the branch target). For instance, the host is unable to perform operations like "`if (x > 0) return;`" when `x` is an encrypted variable. This is known as the "termination problem", introduced in [15], since performing runtime branch decisions is not possible and therefore rendering the algorithms implemented on top of those systems by design more complex.

To address this problem traditional algorithms must be changed to their homomorphic equivalents, a not straightforward process. Some work has been done in [47] by developing benchmarks targeted for computer architectures based on homomorphic operations. Those benchmarks avoid termination problems while maintaining data privacy.

There is not a sole type of homomorphic encryption, different schemes have been proposed for different types of applications.

2.4.1 Partially Homomorphic Encryption

The most widely used type of homomorphic encryption is partially homomorphic encryption (PHE), with schemes like RSA, ElGamal, Benaloh and Paillier. PHE has been expanded with applications such as Helios (e-voting PHE based system) [1], Cryptoleq (Heterogeneous abstract machine for both encrypted and unencrypted computation based on PHE) [45], and others. All those systems, perform manipulations directly on encrypted data without decrypting them, thus no information leakage is possible. When the result is eventually decrypted, it will be the same as applying the same manipulations on plaintexts.

The two most common cases (but not the only ones) of partially homomorphic cryptosystems are *additively homomorphic* and *multiplicatively homomorphic*.

- Additively homomorphic systems enable computation over ciphertexts (encrypted data) that result in the encryption of the sum (addition) of two plaintexts. More formally, a cryptosystem is considered to be additively homomorphic iff:

$$Enc(M_1) \otimes Enc(M_2) = Enc(M_1 + M_2) \tag{2.4}$$

  for some operation $\otimes$.

- Multiplicatively homomorphic systems enable computation over ciphertexts that decrypt to the product (multiplication) of two plaintexts. More formally, a cryptosystem is considered to be multiplicatively homomorphic iff:

$$Enc(M_1) \otimes Enc(M_2) = Enc(M_1 \cdot M_2) \tag{2.5}$$

for some operation $\otimes$.

Below we describe some widely known cryptosystems that are partially homomorphic.

2.4.1.1 RSA Cryptosystem

Plain RSA encryption is multiplicatively homomorphic. Consider the encryption algorithm described in section 2.3.1. We know that $Enc(m) = {m_1}^e \pmod n$. Given two ciphertexts $Enc(m_1)$ and $Enc(m_2)$ of plaintexts m_1 and m_2 respectively, we can see that the following holds.

$$\begin{gathered} Enc(m_1) \cdot Enc(m_2) = {m_1}^e \pmod n \cdot {m_2}^e \pmod n = \\ (m_1 \cdot m_2)^e \pmod n = Enc(m_1 \cdot m_2) \end{gathered} \tag{2.6}$$

So the product of the two ciphertexts corresponds to the ciphertext of the product of the two plaintexts.

2.4.1.2 ElGamal Cryptosystem

The ElGamal cryptosystem is also multiplicative homomorphic. Based on the algorithm described in 2.3.2 we know that $Enc(m) = (G, M) = (g^r \pmod p), m,$ for some random $r \in_R \mathbb{Z}_q$. Consider we have two ciphertexts $Enc(m_1)$ and $Enc(m_2)$ of plaintexts m_1 and m_2, respectively. We can see that the following holds.

$$\begin{gathered} Enc(m_1) \cdot Enc(m_2) = (G1, M1) \cdot (G2, M2) = \\ (g^{r_1} \pmod p), m_1 \cdot y^{r_1} \pmod p)) \cdot (g^{r_2} \pmod p), m_2 \cdot y^{r_2} \pmod p)) = \\ (g^{r_1+r_2} \pmod p), (m_1 \cdot m_2) \cdot y^{r_1+r_2} \pmod p)) = Enc(m_1 \cdot m_2) \end{gathered} \tag{2.7}$$

We can see again that the product of the encryptions of two plaintexts results to the encryption of the encryption of the product of the two plaintexts.

2.4.1.3 Paillier Cryptosystem

The Paillier cryptosystem is additively homomorphic. According to 2.3.3, we know that $Enc(m) = g^m \cdot r^n \pmod{n^2}$, for some random $r \in_R \mathbb{Z}_n^*$. Given two ciphertexts $Enc(m_1)$ and $Enc(m_2)$ of plaintexts m_1 and m_2 respectively, we can see that the following holds.

$$\begin{gathered} Enc(m_1) \cdot Enc(m_2) = (g^{m_1} \cdot {r_1}^n \pmod{n^2}) \cdot (g^{m_2} \cdot {r_2}^n \pmod{n^2}) = \\ (g^{m_1} \cdot {r_1}^n) \cdot (g^{m_2} \cdot {r_2}^n) \pmod{n^2} = g^{m_1+m_2} \cdot (r_1 \cdot r_2) \pmod{n^2} = \\ Enc(m_1 + m_2) \end{gathered} \tag{2.8}$$

We can see that the product of the two ciphertexts will decrypt to the sum of their corresponding plaintexts.

2.4.2 Fully Homomorphic Encryption

Another form of homomorphic encryption is Fully homomorphic encryption (FHE), first invented by Gentry in [27]. Due to the fact that FHE enables *arbitrary computation* on ciphertexts, is far more powerful than PHE (which was just called Homomorphic Encryption before FHE systems were discovered) and its appearance sparked the academic interest. Consecutively a lot of FHE schemes arise, but unfortunately they come along with a huge performance overhead. This overhead has been a concern and this is the reason that there are not many applications that leverage FHE, despite the wide range of that can benefit from FHE schemes. Some implementations are the HElib [29] and the TFHE [19].

2.5 Secure Multi-party Computation (SMPC)

Secure multi-party computation (SMPC)[3] or Secure Function Evaluation (SFE) (in the two-party setting) is a field of cryptography aiming to create methods that enable distinct parties, to jointly compute a function over their private inputs. Only the outcome of that function is made public and the parties don't learn anything more than their own input except whatever can be learned from the output of the function.

Secure multi-party computation was introduced in 1982 by Andrew Yao. Its first form was that of secure two-party computation (2PC) with the so-called Millionaire's Problem [63]. The problem states that there are two millionaires wishing to know who is richer. However, they should not find out any additional information about each other's wealth.

In the general case we have N parties $P_1, P_2, \ldots, P_N$ with private inputs $x_1, x_2, \ldots, x_N$ respectively. The goal is to compute a function $f(x_1, x_2, \ldots, x_N)$ and learn nothing more than what they would have if a separate trusted party had collected their inputs, computed function f for them, and the return the result to all parties.

There are two important requirements on any protocol for secure computation, namely *privacy* and *correctness* [43]. The privacy requirement suggests that nothing but what is absolutely essential should be learned. More specifically, all parties should learn nothing more but the computation output. The correctness requirement states that every party should receive the correct computation output, that is an adversary should not be able to cause the result of the computation to be different than the outcome of the function the parties agreed to compute.

Plenty of tasks can be modeled using SMPC. The tasks that can be modeled using SMPC include simple ones such as coin tossing, as well as far more complex such as electronic voting, electronic auctions, anonymous transactions, anonymous chatting and private information retrieval systems.

Take electronic voting as an example. The privacy requirement ensures that no party can learn the individual votes of other parties, while still learning the election outcome. The correctness requirement ensures that no party can affect the outcome with means other than casting their one individual vote. Similarly, in the electronic auction example, the privacy requirement ensures that no party learns the bids of other parties while still learning the winning bid. The correctness requirement ensures that no party can bias the auction outcome and that in fact the winning party has places the highest bid.

[3]Both SMPC and MPC abbreviations are used for secure multi-party computation; in this thesis we use them interchangeably.

2.5.1 Millionaire's Problem

The first problem that was modeled using SMPC is the Millionaire's Problem introduced by Yao, and it is a basic building block for such secure computations.

Here the two parties P_1, P_2 are the two millionaires. Their private inputs are each one's wealth x_1 and x_2 respectively. The function which they wish to jointly compute can be formulated as

$$f(x_1, x_2) = \begin{cases} x_1, & \text{if } x_1 > x_2. \\ x_2, & \text{otherwise.} \end{cases} \tag{2.9}$$

A lot of solutions have been proposed to the Millionaire's problem starting with the one from Yao himself [63] as well as multiple others including [30, 40].

2.5.2 Oblivious Transfer

Another example for the two-party case and a basic building block of SMPC systems is Oblivious Transfer (OT) firstly introduced in 1981 by Michael O. Rabin [53]. Its simple flavor is 1-2 oblivious transfer or “1 out of 2 oblivious transfer”.

In the 1-2 oblivious transfer case we have two parties, the sender S and the receiver R. The sender has two messages, m_0 and m_1, and the receiver has a bit b. The receiver wishes to learn m_b, without learning anything about m_{1-b} and without the sender learning b.

A solution for the above problem has been implemented using the protocol of Even, Goldreich, and Lempel in [26]. Below we describe the otherwise generic protocol using RSA as en encryption scheme.

- S (who holds messages m_0 and m_1) generates an RSA key pair. Recall from section 2.4.1.1 that the public key is (n, e) and the private key is d.
- S randomly selects two values x_0, x_1.
- S sends x_0, x_1, N, e to R.
- R picks $b \in \{0, 1\}$
- R randomly selects a value k and computes $v = x_b + k^e \pmod{n}$.
- R sends v to S.
- S computes $k_0 = (v - x_0)^d \pmod{n}$ and $k_1 = (v - x_1)^d \pmod{n}$. One of k_0, k_1 will be equal to k randomly selected be R.
- S sends to $m'_0 = m_0 + k_0$ and $m'_1 = m_1 + k_1$ to R.
- R can compute exactly one of the messages, as $m_b = m'_b - k$.

2.5.3 Garbled Circuits

There are also generic constructions for building SMPC systems. Generic protocols implement secure computation for any probabilistic polynomial time function. One of the most commonly known such generic protocol is Garbled (or Encrypted) Circuits (GC). The most simple version of SMPC using Garbled Circuits can be found in the two-party case. The first protocol for such computations was introduced by Yao in [64].

Assume that we have two parties, Alice and Bob, with private inputs x and y, respectively. They wish to compute function f over their private inputs, and nothing more than $f(x, y)$ should be learned.

The GC protocol works by expressing f as an combinatorial circuit. The circuit contains logical gates that implement any function $g : \{0, 1\} \times \{0, 1\} \to \{0, 1\}$. These include for instance simple `AND`, `OR`, `XOR` and `NOT` gates. This circuit takes as inputs the bitwise representation of x and y and has output the bitwise representation of the value $f(x, y)$. The protocol is based on evaluating an encrypted version of this circuit. A simple description of Yao's protocol can be seen below. A complete description as well as security proof of this protocol can be found in [42].

Let's assume without loss of generality that Alice is the so called *garbled circuit generator*, or *garbler*, and Bob the *evaluator*. The circuit is known to both parties.

2.5.3.1 Circuit Encoding

- Alice "hardwires" her input into the circuit. Thus the circuit now computes $f(x, \cdot)$.
- Alice assigns to each wire i of the circuit two random string values (W_i^0, W_i^1) which are called *labels* that correspond to the Boolean values $0/$ `false` and $1/$ `true` respectively. These labels should be of adequate length (usually 128 bits) as they will be used as symmetric keys in an encryption scheme.
- For each gate g in the circuit Alice prepares the truth table of g. In that truth table the values $0, 1$ are replaced with the corresponding labels that were generated in the previous step. The output column is then encrypted [4] using as keys the labels from the two input columns. For example the transformation of the truth table of an `AND` gate with input wires A, B and output wire C can be seen below.

$$
\begin{array}{cc|c}
A & B & C \\
\hline
0 & 0 & 0 \\
0 & 1 & 0 \\
1 & 0 & 0 \\
1 & 1 & 1
\end{array}
\xrightarrow{\text{becomes}}
\begin{array}{cc|c}
A & B & C \\
\hline
W_A^0 & W_B^0 & W_C^0 \\
W_A^0 & W_B^1 & W_C^0 \\
W_A^1 & W_B^0 & W_C^0 \\
W_A^1 & W_B^1 & W_C^1
\end{array}
\xrightarrow{\text{becomes}}
\begin{array}{cc|c}
A & B & C \\
\hline
W_A^0 & W_B^0 & E_{W_A^0}(E_{W_B^0}(W_C^0)) \\
W_A^0 & W_B^1 & E_{W_A^0}(E_{W_B^1}(W_C^0)) \\
W_A^1 & W_B^0 & E_{W_A^1}(E_{W_B^0}(W_C^0)) \\
W_A^1 & W_B^1 & E_{W_A^1}(E_{W_B^1}(W_C^1))
\end{array}
$$

- As a final step Alice generates a random permutation of the truth table's rows, and the result is a *garbled* table.

2.5.3.2 Data Transfer

- Alice sends the garbled truth table for every gate g of the circuit to Bob.

[4]The notation $E_K(M)$ means an encryption of the value M with key K.

- Alice also sends the randomly generated labels corresponding to her input. For example if Alice's input a is represented by the bits $a_4a_3a_2a_1a_0 = 01101$ then she will send the labels $W^0_{a_4}$, $W^1_{a_3}$, $W^1_{a_2}$, $W^0_{a_1}$, $W^1_{a_0}$ to Bob.
- Bob also needs the labels corresponding to his input bits. For example if Bob's input b is represented by the bits $b_4b_3b_2b_1b_0 = 10100$ then he will need the labels $W^1_{b_4}$, $W^0_{b_3}$, $W^1_{b_2}$, $W^0_{b_1}$, $W^0_{b_0}$ To obtain these labels, Alice and Bob run an 1 out of 2 oblivious transfer protocol, for each bit (each input wire) of Bob's input b. Using this 1-2 OT protocol Bob learns only the labels corresponding to his input bits, and Alice learns nothing about Bob's input. If bob wants to obtain the label for input bit $b_4 = 1$, he will ask Alice between $W^0_{b_4}$ and $W^1_{b_4}$. Bob will learn only $W^1_{b_4}$ and not $W^0_{b_4}$, while Alice will not learn the value of b_4.

2.5.3.3 Circuit Evaluation

- Bob now has the garbled truth tables for each gate of the circuit as well as all input labels. Having one garbled value/label per input wire and the truth table, Bob will try to decrypt every value in the output column of the table, but will succeed only once. The decryption will succeed only in the row for which Bob has the input labels so he can use them as keys in the decryption. The decryption result will be the output label of that gate (the garbled version of the gate's output value). This label will be used as an input for the next gate in the combinatorial circuit.
- Bob repeats the process for each gate g of the circuit, until he reaches to the output gate(s) and acquires the output label(s).

2.5.3.4 Output Revealing

- Alice knows The Boolean value corresponding to the output label(s), so Bob and Alice communicate in order for them to learn the computation result.
- Either Alice will share her information about the original values of the output label(s) with Bob, or Bob will share the output to Alice and she could respond accordingly so that one or both of them learn the output.

2.5.3.5 Overhead

This protocol inevitably inserts a notable overhead. An overview of the most substantial factors as described in [43] can be found below.

- Alice and Bob involve in a 1 out of 2 oblivious transfer protocol for each input wire related with Bob's input. These oblivious transfers can dominate the computation overhead – at least for relatively small circuits – since they require modular exponentiations.
- Alice sends to Bob a garbled truth table for every gate g of the circuit. Thereby, this operation's cost is proportional to the circuit size.
- Bob decrypts a constant number of encrypted values for each gate g of the circuit. This is also linear with respect to the size of the circuit.

Multiple attempts have been made to improve the performance of Yao's protocol. Many optimizations have been introduced like the ones found in [5, 7, 39, 48, 66].

Another generic protocol for Secure multi-party computation can be constructed using *Secret Sharing*, a technique which we will describe in section 2.6.

2.6 Secret Sharing

Secret sharing is a method for distributing a secret between N different parties, where a secret message m is divided into parts, giving each participant its own unique part $(s_1, s_2, \ldots, s_N)$. Combining some of the parts or all of them are needed in order to reconstruct the original secret. The most common type of secret sharing is a scheme with one dealer and N different players. Initially, the dealer splits the secret to shares and gives each player a split. Only with all – or most of the shares $k \geq T$ (T is a threshold) – someone is able to reconstruct the original secret. The dealer accomplishes this by giving each player a share in such a way that any group of T or more players can together reconstruct the secret but no group of fewer than T players can. Such a system is called a (T, N) - threshold scheme.

A naive secret sharing scheme between a dealer and two parties is to apply bitwise `XOR` to the secret and a random string of the same length. For the sake of simplicity, let us assume that the secret message m is a string of 5 bits (either 0 or 1), *e.g.* 01101. Then, the dealer generates a random string of 5 bits s_1 (*e.g.* 10011) and performs the `XOR` operation between s_1 and m; a string $s_2 = 11110$ arises. Finally, the dealer gives the s_1 and s_2 to the two parties. Neither one of them is able to retrieve the secret message m without the other half share, and also nor is able to retrieve any information about the original message.

There exist more complex and elaborate systems for more than two parties, such as Shamir's [55], Blakley's [10], or Additive [38] secret sharing schemes.

2.6.1 Shamir Secret Sharing

Secret sharing was proposed in [55] by Shamir, and it was the first method to enable distribution of a secret to N parties. A secret message m is divided into N parts, giving each participant its own unique part, where any T subset of N can recover the secret, but no $T-1$ element subset can.

Adi Shamir's threshold scheme is based on the idea of polynomial interpolation – the interpolation of a given dataset by the polynomial of lowest possible degree that passes through the points of the dataset –, using Lagrange coefficients. Shamir's secret sharing scheme constructs a polynomial P of degree $N-1$, where N is the number of players, as follows.

2.6.1.1 Algorithm

- Dealer chooses a random polynomial of degree $N-1$ so that $P(0) = m$, where m is the secret message, and also the number T of sufficient subsets that can reconstruct the secret.

For example[5], let us suppose our secret message $m = 1234$ and $N = 6$ parties, but any subset of $T = 3$ is sufficient to reconstruct the secret. Then our polynomial should have degree 2, $f(x) = ax^2 + bx + m$. The dealer chooses $N - 1$ random numbers, *i.e.* $a = 94$ and $b = 166$, thus our polynomial is $f(x) = 94x^2 + 166x + 1234$.

- Dealer distributes N pairs $(x_i, P(x_i))$, $x_i \neq 0$

  First the dealer creates the N pairs. *For example*:
  $D_0 = (1, 1494)$, $D_1 = (2, 1942)$, $D_2 = (3, 2578)$, $D_3 = (4, 3402)$, $D_4 = (5, 4414)$, $D_5 = (6, 5614)$ and consecutively he/she distributes them to the N parties.

- N players can reconstruct the polynomial P with their pairs, however $N - 1$ cannot

  In order to reconstruct the secret any T points will be enough, *for instance*, D_1, D_3 and D_4. Computing the Lagrange polynomials:

  $$l_0 = \frac{x - x_1}{x_0 - x_1} \times \frac{x - x_2}{x_0 - x_2} = \frac{x - 4}{2 - 4} \times \frac{x - 5}{2 - 5} = \frac{1}{6}x^2 - \frac{3}{2}x + \frac{10}{3}$$

  $$l_1 = \frac{x - x_0}{x_1 - x_0} \times \frac{x - x_2}{x_1 - x_2} = \frac{x - 2}{4 - 2} \times \frac{x - 5}{4 - 5} = \frac{1}{2}x^2 + \frac{7}{2}x - 5$$

  $$l_2 = \frac{x - x_0}{x_2 - x_0} \times \frac{x - x_1}{x_2 - x_1} = \frac{x - 2}{5 - 2} \times \frac{x - 4}{5 - 4} = \frac{1}{3}x^2 - 2x + \frac{8}{3}$$

  Therefore:

  $f(x) = \sum_{i=0}^{2} y_i \times l_i(x) = 94x^2 + 166x + 1234$

  Each party can compute $f(0)$ in order to obtain the secret, in this case $f(0) = 1234$.

It is evident, that in Shamir's scheme two points are sufficient to define a line $f(x) = ax+b$, three points are sufficient to define a parabola $f(x) = ax^2 + bx + c$, four points to define a cubic curve $f(x) = ax^3 + bx^2 + cx + d$ and so forth. That is, it takes T points to define a polynomial of degree $T - 1$, where $T - 1$ will be the sufficient number of parties that can reconstructed the secret message.

2.6.1.2 Threat Model

The Shamir secret sharing scheme can tolerate both passive and active adversaries, however in the latter case it has more strict bounds. Shamir's sharing algorithm is secure against a passive adversary when $T < \frac{N}{2}$, while it achieves information-theoretic security for $T < \frac{N}{3}$ with an active adversary. This means that even if the adversary has unbounded computational power, they cannot learn any information about the secret underlying a share. The BGW [14] protocol, which defines how to compute addition and multiplication on secret shares, is often used to compute functions with Shamir secret shares.

2.6.2 Additive Secret Sharing

Another form of secret sharing, and probably the simplest one, is additive sharing. Again, a secret message m is divided into N parts, where each participant has its own unique

[5]This example is based on a similar example in `https://en.wikipedia.org/wiki/Shamir's_Secret_Sharing` Wikipedia link.

part. In contrast with Shamir's sharing algorithm, all parts are necessary in order to recover the original secret.

2.6.2.1 Algorithm

- Dealer chooses a randomly $N-1$ numbers such that $x_i \in \mathbb{Z}, i \in [1, N-1]$
- Computing x_N from the random numbers is pretty trivial; $x_N = m - x_1 - x_2 - \cdots - x_{N-1}$
- Finally, the dealer distributes the N secrets ($x_1, \ldots x_N$)
- In order to reconstruct the original message m, the parties have to combine their secrets: $m = x_1 + x_2 + \cdots + x_N$

Seeing any $N-1$ values (*i.e.* $x_1, \ldots x_{N-1}$) does not give any clue about what m could be, since the remaining x_i may change the final result to any element of $\mathbb{Z}$ with equal probability. More elaborate schemes than the $(\mathbb{Z}, +)$ have been introduced, in the form of $(A, \oplus)$ (which means in group A using the operation $\oplus$), such as linear secret sharing.

There are many systems that have implemented various forms of SMPC with secret sharing schemes. The most popular is SPDZ [21, 22], which implements MPC with additive secret shares and is secure against active adversaries. Another popular system that implements SMPC using additive secret shares is the Sharemind Secure Computing Platform, which we survey in section 4.

2.6.2.2 Threat Model

In contrast to Shamir's secret sharing scheme, that can tolerate up to $T < \frac{N}{2}$ passive adversaries or $T < \frac{N}{3}$ active, as mentioned in 2.6.1.2, additive secret sharing schemes can tolerate the adversary controlling all but one party, that is $T < N$. As we already mentioned, in order to reconstruct the original message m, all parties have to add their secrets ($m = x_1 + x_2 + \cdots + x_N$). It has become evident, that if one x_i is missing, the initial message m cannot be reconstructed.

2.6.3 Secret Sharing Homomorphism

In [8], Benaloh observed that many secret sharing schemes have a homomorphic property which allows multiple secrets to be combined by direct computation on shares.

Suppose we have a secret m that can be computed from T sub-secrets $\{m_i\}_{i=1}^{T}$ using a functions f as follows $m = f(m_1, \ldots, m_T)$. Let $\oplus$ and $\otimes$ be binary functions on elements of the secret domain S and of the share domain T, respectively. We say that a (k, N) threshold scheme has the $(\oplus, \otimes)$-homomorphism property (or is $(\oplus, \otimes)$-homomorphic) if for all $f : T^k \to S$, whenever a message

$$m = f(m_1, \ldots, m_k)$$

and another message

$$m' = f(m'_1, \ldots, m'_k)$$

then the composition of the shares are the shares of the composition:

$$m \oplus m' = f(m_1 \otimes m'_1, \ldots, m_k \otimes m'_k)$$

Shamir's polynomial based secret sharing 2.6.1 scheme is $(+, +)$-homomorphic since the sum of the polynomials use to share the sub-secrets is itself a polynomial that can be used to deal shares of the super secret.

3. RELATED PROJECTS

3.1 My Health My Data

My Health My Data project (MHMD) [46] is a Horizon 2020 Research and Innovation Action which aims at fundamentally changing the way sensitive data are shared.

MHMD is poised to be the first open biomedical information network centred on the connection between organizations and individuals, encouraging hospitals to start making anonymised data available for open research, while prompting citizens to become the ultimate owners and controllers of their health data. It is an academic research project, focusing on the use of Privacy-Preserving Data Mining (PPDM) tools in conjunction with Blockchain technologies, to develop an ecosystem where patient health records across several hospitals in Europe can be safely and traceably shared for the purposes of scientific research. MHMD is intended to become a true information marketplace, based on new mechanisms of trust and direct, value-based relationships between EU citizens, hospitals, research centres and businesses.

The MHMD project currently utilizes the Sharemind secure computing platform, as well as many algorithms presented in the current thesis.

3.2 Scalable Oblivious Data Analytics

Scalable Oblivious Data Analytics project (SODA) [58] is also a Horizon 2020 Research and Innovation Action, with similar goals as MHMD project. More specifically, the SODA project aims to tackle exactly data protection and anonymization issue identified by the Big Data Value Association (BDVA) [4], enabling practical privacy-preserving analytics on Big Data by significant improvement of MPC techniques for privacy-preserving Big Data processing.

One of the differences between the MHMD and the SODA project is that in the latter uses FRESCO (a FRamework for Efficient and Secure COmputation) [3, 17] as the underlying MPC engine.

4. SHAREMIND: A SECURE COMPUTING PLATFORM

Sharemind [12, 13] is a general purpose SMPC system, for privacy preserving data processing operating on additively secret-shared values. The idea of Sharemind is to provide a secure infrastructure that is able to host and evaluate privacy preserving algorithms. Sharemind provides an easily programmable and flexible platform that enables non-cryptographers to develop and test privacy preserving algorithms such as Privacy Preserving Data Mining (PPDM). It consists of the computation runtime environment and a programming library for creating private data processing applications Sharemind is provably secure under the semi-honest (honest but curious) setting.

Sharemind's is deployed as a distributed computation platform, that can be used both for data storage and computation. The deployment model consists of (usually three) nodes, the computing nodes that use SMPC through secret sharing to privately process the data. Secret sharing guarantees data confidentiality during storage. All computations are done by the dedicated computing nodes.

Data providers submit their private inputs by sending the corresponding cryptographic shares to the computing nodes. The secret sharing scheme ensures that each share is a random bit string. Each host, computing node, is not able to decrypt the data, or even extract any information about them. Consequently, a node holding that share learns no extra information about the data input (secret), than if they did not hold that share. For that reason, data providers need not trust any of the computing nodes. Instead providers must trust that the nodes as a group obey a set of rules such as that they do not collude during the computations. This can be achieved through physical and organizational security measures as well as software auditing. In practice, the computing nodes will be servers run by independent entities, such as companies or government agencies, ideally having conflicting interests. Data users want to analyze the information given by data providers. Answers to their queries are given through the Sharemind system, and not from data providers directly.

The Sharemind framework provides a privacy preserving instruction set which includes secure addition, multiplication and greater-than-or-equal comparison of two shared values. Multiplication of a shared value with a constant and extraction of its bits as shares is also possible. Bit extraction and arithmetic primitives are sufficient to construct any Boolean circuit with a linear overhead and thus the Sharemind framework is also Turing complete [13]. Many data mining and statistical analysis algorithms use no other mathematical operations than the Sharemind's instruction set, thus it is sufficient for most applications.

4.1 Real World Applications

4.1.1 Satellite Collision Detection

One notable example of a real world application of SMPC using Sharemind was presented in [36]. In this paper, Sharemind engine was utilized to perform satellite collision detection in a secure multiparty setting. The number of satellites orbiting our planet is growing, and so does the danger of collisions. Indeed, two satellites crashed in 2009. Satellite owners, such as governments, do not want to reveal orbital information about their satellites. However, satellites come with extravagant expenses and each satellite owner want to protect their property.

Future collisions could be avoided by sharing information about the satellites orbits. Using SMPC, the parties can cooperate and learn whether a collision is going to happen and nothing else. No information about the actual orbits would leak – no party can see the individual values, since computations are carried out on encrypted data. The SMPC solution provides cryptographic guarantees for the confidentiality of the input trajectories.

4.1.2 Analysing Private Databases

Another worth mentioning application took place in 2015, where the Estonian Center of Applied Research used Sharemind to collect governmental tax and education records with purpose to run a big data study looking for correlations between people who work during their studies and those who fail to graduate in time.

In Estonia – where this research took place, the Ministry of Education and Science keeps track of students and the Tax and Customs Board keeps track of working (by tracking income tax payments). A correlation between working during studies and not graduating in time could be possible to find, if these databases were not private. However, this data cannot be shared because of the Personal Data Protection Act and the Taxation Act and also many European Union regulations, such as General Data Protection Regulation (GDPR) [49]. All these legislation prevents such studies from being performed.

The results of the study were quite surprising. The study found no relation between working during studies and not graduating on time. Instead, it turned out that the population group that was studied (Estonian students of all fields) work an equal amount. Moreover, it showed an obvious the reduction of employment during the financial crisis in 2008. This study would not have been possible without an SMPC engine such as Sharemind, as no research organization can gain access to these private databases due to Data Protection regulations. Nevertheless, this research allowed more personalized follow-up studies to be planned for finding the reasons why students quit.

4.2 SecreC

The programmer of the privacy preserving applications does not necessarily know the underlying security protocols. Application are developed using the SecreC programming language [33]. SecreC is an procedural imperative domain specific language (DSL) that is syntactically similar to C. SecreC guarantees writing privacy-preserving algorithms without any knowledge of the underlying cryptographic protocol. In other words, SecreC distinguishes the application/business logic from the cryptographic protocol.

The language uses a custom type system, which separates private/confidential data from public. Public values are processed as usual, whereas private values, which are in secret-shared form are processed using secure computation. To achieve this, SecreC adds a security type to each fundamental data type. The security type can be either *private* or *public*. If no security type is specified then the public one is used by default. Security types differ from standard data types in the sense that data associated with the public security type will be stored and processed publicly on the Sharemind virtual machine, whereas data associated with the private security type, will be stored in secret-shared form. Each computing node will acquire one share of the actual value.

In SecreC one can have expressions of private or public data. In the case that such two expressions are combined into a single expression, the public data are secret-shared and moved into the private execution environment in order for the composite expression to be evaluated. That way all data are of the private security type and the expression can be evaluated.

Sometimes however it is vital to open a secret. This could be done in order to gain some insight about an algorithm having good intentions in mind. SecreC offers a special DECLASSIFY operator through which is made possible to publish private information. An expression that is declassified has its security type changed from private to public, and the secret-shared value moves into the public execution environment. Since the use of the DECLASSIFY operator is the only way for private data to become public, its use should be tracked in order to reason about data privacy and to avoid potential unwanted privacy leaks. As a rule of thumb, declassification of private data should occur ass little as possible. Ideally, declassification should be used only to publish the final results of an algorithm, or intermediate results that have low privacy risk *i.e.* do not reveal sensitive information.

Typically, if a decision (*e.g.* a branch) is to be made over private data, that would require the data to be published, that is to be converted to the public security type. For instance, consider the following example:

```
if (x) a = b; else a = c;
```

One can rewrite the previous statement using with the following expression:

```
a = b * x + c * (1-x);
```

Using this technique a programmer can replace conditional statements on private data with such oblivious selection clauses. So the control flow dependencies are converted into data dependencies.

```
pd_shared3p uint64 max(pd_shared3p uint64 x, pd_shared3p uint64 y) {
  pd_shared3p uint64 gt = (uint64)(x > y); // 1 if x > y, 0 otherwise
  return x * gt + y * (1-gt); // Oblivious selection of the max value
}

void main(){
  pd_shared3p uint64 x1 = 5; // Private, secret-shared value of input party 1
  pd_shared3p uint64 x2 = 8; // Private, secret-shared value of input party 2
  pd_shared3p uint64 max = max(x1, x2); // Private result
  print(declassify(max)); // Result publishing
}
```

Code 1: Millionaire problem in SecreC

An example of SecreC code can be found in Code Snippet 1. The problem at hand is Yao's millionaire problem [63] as described in section 2.5. Here we use `pd_shared3p` as the security type, which stands for ***p**rotection **d**omain **shared 3 p**arties*, and corresponds to a private security type using secret-sharing between three parties.

4.2.1 SIMD in SecreC

Each SMPC operation in Sharemind is implemented as an assembly instruction. Sharemind also supports vectorized operations which play a significant role in the program's efficiency. With vectorized operations the same protocol is executed but using multiple inputs at once, in parallel. Using such SIMD (Single Input Multiple Data) instructions the protocol performs all operations in parallel by packaging the values in a single communication message.

This significantly reduces the communication overhead between the participating parties, which is usually a dominating factor in a protocol's performance. Thus, developing applications with vectorization in mind, has a substantial contribution to the application's efficiency. We have focused on using vectorization as much as possible when developing our privacy preserving algorithms, which we describe in chapter 6.

Below, in code snippets 2 and 3, we present two versions of array addition in SecreC. The first one works with a typical loop on each of the array elements, while the second one works by performing an SIMD addition between two arrays.

```
void main(){
  uint64 N = 5;
  pd_shared3p uint64 a(N) = {1, 2, 3, 4, 5}; // Array definition of size N
  pd_shared3p uint64 b(N) = {6, 7, 8, 9, 10}; // Array definition of size N
  pd_shared3p uint64 c(N); // Array declaration of size N
  for (uint64 i = 0; i < N; i++){
    c[i] = a[i] + b[i]; // Element-wise addition with sequential access
  }
  print(declassify(c)); // Result publishing
}
```

Code 2: Array addition in SecreC

```
void main(){
  uint64 N = 5;
  pd_shared3p uint64 a(N) = {1, 2, 3, 4, 5}; // Array definition of size N
  pd_shared3p uint64 b(N) = {6, 7, 8, 9, 10}; // Array definition of size N
  pd_shared3p uint64 c(N) = a + b; // SIMD instruction of addition of the two
arrays
  print(declassify(c)); // Result publishing
}
```

Code 3: Array addition in SecreC using SIMD

The two example programs will have the same output but their performance will differ. The first program will perform N message exchanges between the computing parties for the addition, while the second will perform just one. For large enough values of N, the performance difference is tremendous.

4.3 Sharemind Infrastructure

Sharemind consists of three different kinds of parties, input parties, computing parties and result parties. The number of the input parties – data providers – as well as the result parties – analysts/users – is not restricted, however the computing parties are restricted by the SMPC protocol that is been used; for instance the aforementioned "Millionaire problem" shown in (*Source Code* 1) uses the *pd_shared3p* security protocol, which utilizes three nodes.

As stated by Sharemind's author in [12] three is usually the optimal number for the computing nodes configuration in the passively corrupted parties threat model. Three is the lowest number for which an honest majority of parties processing secret shared data can be formed. The number of parties can be less – just two nodes – but it adds more complexity to the protocols, as it requires more expensive cryptographic primitives to be used. The number of computing parties can also be grater than three, increasing however the communication complexity.

An input party can also be the result party, or one of the computing nodes. The data providers use secret sharing to distribute their confidential data between the computing nodes. The analysts make queries and initiate computations, that are performed by the computing nodes leveraging the homomorphic primitives of the secret shares. Finally, the analysts get the results of the private computation without anyone seeing the original confidential data. Sharemind's infrastructure as explained in [60], is presented in figure 1 in more detail.

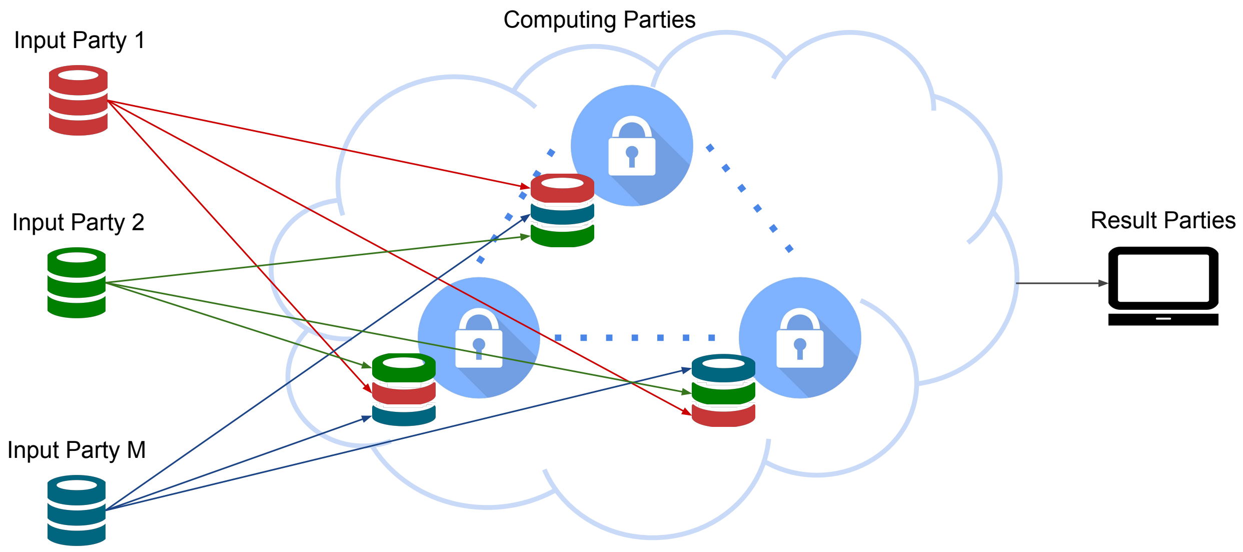


Figure 1: An overview of the architecture of Sharemind

5. A MEDICAL CASE STUDY

In this thesis we primary focus to analyze medical data by using some of the aforementioned techniques from section 2. More specifically we have developed a complete platform built around secure medical data analytics that could be beneficial to patients, doctors and researchers.

The platform can provide insights about some medical datasets that are imported into the platform to anyone who wants to query them, without compromising individual patient's privacy. These analytics which include data aggregation/statistics and classification are implemented through privacy preserving algorithms under the secure multi-party computation scenario.

The platform provides end to end work flow starting from the query selection from a user. Following is the secure data importing, the performing of privacy preserving data analytics algorithms upon those data and finally the visualization of the results to the end user through a friendly user interface (UI).

5.1 A Doctor's view

We consider the use case of a doctor working in a hospital that wants to examine the data stored in that hospital's datasets. He/She could possibly want to evaluate a treatment's outcome. To achieve this, the doctor should have monitored some particular datasets in order to have an overall view of the patients' condition over time. This could allow a comparison between previously treated patients and enable data-driven cues for the treatment. For example, a drug's effect could be evaluated that way.

The doctor could also compare aggregated results from datasets between different hospitals. That way he/she could get an insight of how each patients' condition varies between different hospitals, which could indicate the different effect that different treatments have on patients and identify patterns and differences in these treatments.

Continuous monitoring of patient data is useful for any hospital. This can be achieved through regular supervision of aggregated statistics (*i.e.* histograms, heatmaps, *etc.*) based on the available medical datasets, like those described in section 6.6. Provided with usable and informative tools a doctor could potentially make better diagnoses and take the appropriate action for each case.

5.2 An Individual's view

An individual could also benefit from the insights provided by our platform. We consider the case in which an individual with a certain condition queries the available datasets looking for the same condition (patients-like-me). The individual could locate the hospitals in which patients with the same condition are treated. Also, he/she could identify which hospitals in his area have the best outcome for patients with conditions similar to his/hers.

Another way an individual could be benefited from the privacy preserving medical data analytics is the use of a classification mechanism such as decision trees (section 6.7), using a model trained over patient data. This way one could classify himself to a condition/disease or another chosen attribute for that matter, based on his own data.

5.3 A Researcher's view

We examine the case where an academic or industry researcher queries the datasets to see if they suit their research needs. They could discover if the datasets are providing enough utility/information and decide which ones to use for performing analysis on.

For example, studying aggregate statistical information of the datasets, one could find possible correlations of particular attributes or find patterns of treatments and conditions.

A researcher could also study the relation between certain conditions and different hospitals, which is something that could provide valuable information.

Additionally, a researcher could utilize some of the classification mechanisms mentioned above to build useful machine learning models (*e.g.* classification, prediction, *etc.*) over the patient data. Thus, carry out interesting PPDM studies.

5.4 Our End-to-End Architecture

Our architecture consists of the *SMPC cluster*, a proxy server, dubbed *coordinator*, as well as `N` more servers that are hosted in the hospitals' premises and act as *data providers*.

SMPC cluster: The *SMPC cluster* consists of three servers – three computing nodes – participating in the SMPC protocol. In our case, these nodes do not provide the data for the computation. The only data the computing nodes hold and process, are cryptographic shares of the original data obtained through the secret-sharing mechanism. The cluster performs a distributed computation over these secret-shared data, exploiting their homomorphic properties, in order to produce meaningful results. In GDPR terms, the *SMPC cluster* acts as the *Data Processor*.

Data providers: The *data providers* are hospitals that provide medical datasets upon which the privacy preserving algorithms will execute. These medical datasets get populated from each hospital's patients *i.e.* the *Data Subjects*. The patient data are horizontally distributed among the participating hospitals. When the medical data are transferred from the hospitals to the computing nodes, they get secret-shared.

Coordinator: The *coordinator* handles all private computation requests. The server listens for requests for private query execution, and when such a request arises, the coordinator communicates with the data providers (all `N` hospitals) requesting them to securely import their data to the computing cluster. The data are then secret-shared to the three nodes, with each node acquiring one share of the original value. Then the privacy preserving computation takes place in the SMPC cluster, and the results get visualized and served back to the requesting user through the *coordinator* server. The *coordinator* server acts as the *Data Controller*.

5.4.1 End-to-End Execution Flow

An overview of an end-to-end query execution can be found below.

Step 1: A user makes a privacy-preserving analytics request to the *coordinator*.

Step 2: The *coordinator* server is responsible for orchestrating all the involved parties:

it communicates with the data-providers requesting secure data-import to the SMPC cluster.

Step 3: The data-providers extract the requested data from their datasets and securely import them to the SMPC cluster applying secret-sharing.

Step 4: The SMPC cluster computes the privacy preserving analytics on the requested data and returns the results to the *coordinator*.

Step 5: Finally, the results are returned to the user through the *coordinator*.

All the aforementioned steps are depicted in figure 2. In the subsequent sections we delve into the above end-to-end execution overview in more detail.

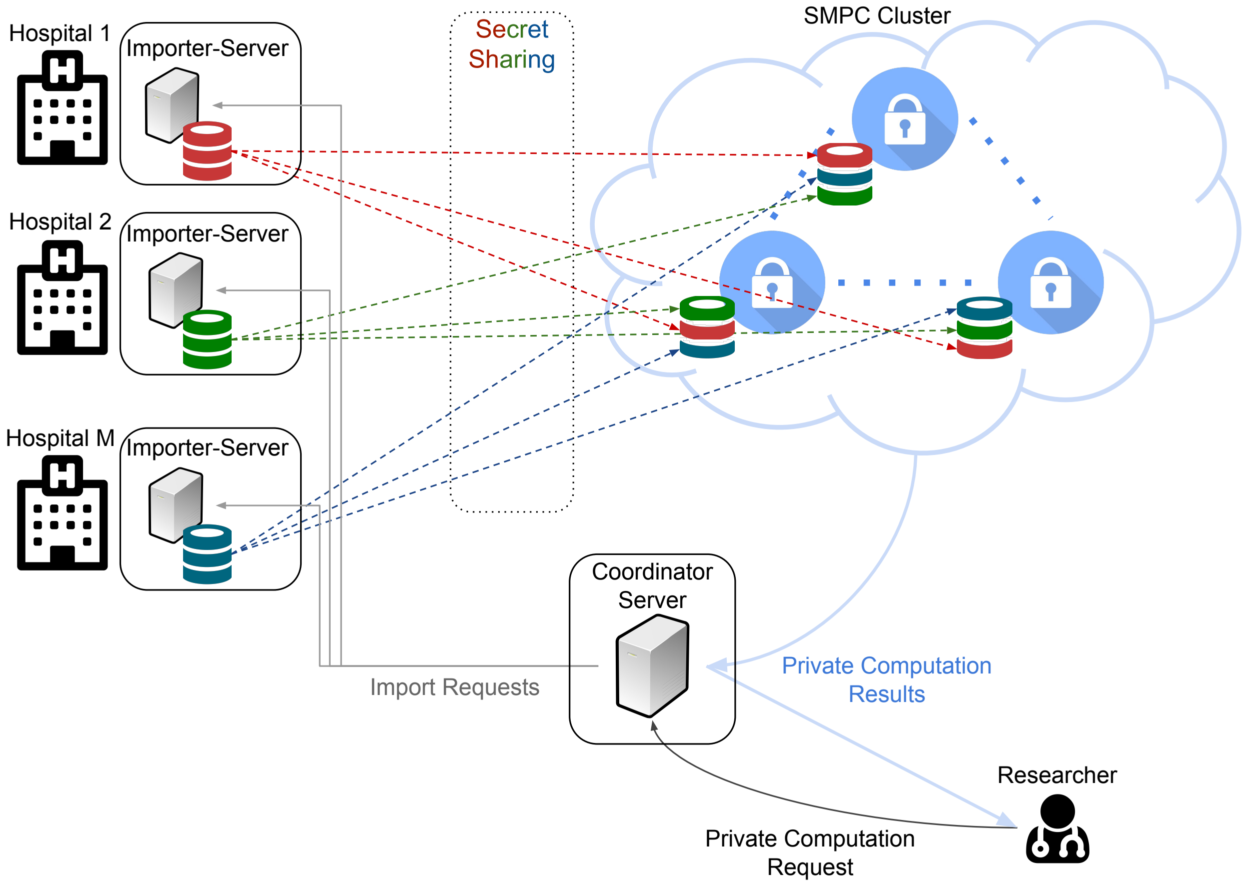


Figure 2: An overview of the architecture of our study

5.4.1.1 Query Initiation

First, a user sends a request to the *coordinator* asking for a private computation. This user can be an analyst/researcher, a doctor, or a patient. This request can specify any one of the supported privacy preserving computations.

Users get informed about the available privacy preserving analytics algorithms through the User Interface (UI). The UI is responsible to inform the user about the supported computations, as well as the available datasets on which these computations can be executed. These datasets are populated from the hospitals that participate in the platform.

The user's request includes the desired private computation to be executed, the selected attributes that are involved in this computation, as well as the selected datasets.

5.4.1.2 Data Import

After the coordinator server collects the user's private computation query, a data importing procedure takes place.

The server will send an importing request to the hospitals that are specified in the user's query. The importing request will include the attributes indicated by the user that are participating in with the private computation.

Upon receiving such an importing request, each hospital involved in this private computation will import a part of its dataset, that is associated with the participating attributes, into the SMPC cluster. The data to be imported will be secret-shared between the nodes participating in the SMPC protocol, using an additive secret-sharing protocol.

The data importing is a procedure of high importance, since the patients' data are transferred outside the hospitals to third party servers. It is easily misinterpreted that since the data are leaving the hospitals' premises, their privacy is being compromised. However, it has been elucidated in section 2.6 that secret sharing is a form of encryption, thus no information leakage is possible while data being in use in the SMPC cluster. For transferring each share from the data providers to the corresponding SMPC computing nodes standard techniques – such as symmetric encryption – are used to protect data in transit. Data are also safe while at rest in the SMPC cluster as each node only stores a cryptographic share of a piece of data, so no information about the original data can be inferred.

5.4.1.3 Data Importing On-the-Fly

Everyday, thousands of people visit their doctors, updating their medical records with new diagnoses. One may wonder, *since the medical data of each hospital are constantly changing, how often should the datasets in the SMPC cluster be updated?*

In our architecture, we have developed a mechanism for data importing *on-the-fly*, as figure 2 portrays. More specifically, when a query is requested for private computation, the coordinator interacts with the hospitals' servers, initiating the data import procedure. Since the individual making the query also selects some specific attributes for data aggregation and/or classification, the importing of the data only happens for the selected attributes.

This importing on-the-fly is beneficial for many reasons. First and foremost, all private computations are evaluated over the most recent data, and not in a outdated/stale version of them. As there are no data stored in the SMPC cluster, every computation is based on fresh data imported directly from the hospitals (data providers) at the time of computation. There is no need to keep redundant copies of the dataset in the SMPC cluster. This eliminates the need to keep the copies regularly updated and reduces the storage requirements. Also, not having the data replicated outside the data providers' servers is less prone to data breaches or similar attacks.

Finally, another reason is that the actual datasets are huge compared to the requested attributes. Each meaningful request does not take into account every possible attribute from the dataset, thus there is no purpose importing everything a priori. The attributes involved in most computations are a subset of the available attributes of the hospitals' datasets. The storage requirements for importing data just for each computation are minor in comparison with the entire dataset.

5.4.1.4 Computation Execution

The next step after the data import is the actual private computation execution. The nodes participating in the SMPC protocol, execute the privacy preserving version of the analytics algorithm that the requesting user has selected.

The algorithm is executed on encrypted (secret-shared) data, thus the computing nodes have no access to the original data that still reside on the hospitals. The execution is oblivious as far as the data is concerned. Any data that gets published is explicitly defined in the privacy preserving algorithms and usually [6] is only the computation output.

5.4.1.5 Result Publishing

The last step of the secure query execution is the publishing of the computation results. These are obtained through computation over private data. The results are now considered public data and can be freely shared with the user that requested the computation or be used in other future computations. One thing to note is that every published result of such a computation should be treated as public data and should not be confused with the confidential input data. With that in mind, whoever initiates a private computation should have a clear view of what data remain private and what gets published.

There are many different approaches for Privacy Preserving Data Mining (PPDM). This thesis aims to investigate the secure computation part, which should not be confused with output-privacy.

Moreover, since some of the private computations may be repeated and also the privacy-preserving algorithms are highly computational intensive, we have implemented a caching mechanism that stores the secure computation results so future requests for that data can be served faster. We elaborate more on the caching mechanism in section (7.1.3).

5.4.2 SMPC Threat Model

The main security goal of every SMPC engine is to protect the values provided by the input parties from all other parties. The data of the input parties will be stored and processed encrypted by the computing parties. The SMPC cluster needs to ensure that no party can learn anything from the information available to them.

Moreover, the final results must not reveal anything except for the actual results of the secure computation performed by the computing parties. Note, that depending on the algorithm and provided data, the output may leak the input of one or more input parties. For instance, without loss of generality let us suppose a SMPC setting with two parties. A malicious adversary can always alter his/her input and define it to be an empty database (or a database with null values). This fact can be very damaging since the final output is the result of the algorithm on the other party's database alone.

In this work we assume that the adversary is *semi-honest* – or similarly, *rational, honest-but-curious*. That is, the adversary correctly follows the protocol specification, but has incentives to eavesdrop sensitive user data. He/She could also attempt to learn additional information by analyzing any data received during the execution, either unencrypted or

[6]Some data (*e.g.* those that will be included in the output) can be published before the algorithm terminates, without violating the prescribed privacy guarantees.

encrypted. In our developed algorithms, we explicitly define which variables are sensitive and should remain encrypted throughout the whole execution to preserve data privacy.

5.4.2.1 Computing Parties Collusion

In the active adversary scenario, where the adversary controls some of the computing nodes, the only requirement is not to control all of them simultaneously. Since the actual data are transferred from the data-providers (hospitals) to the computing nodes through secret-sharing, it is impossible for any of the three nodes to decrypt them, and in general to infer any information, as each computing node possesses only a share of the data. However, meaningful computations can be applied due to the homomorphic properties that the shares have (section 2.6.3).

As we examined in section 2.5, the computing nodes of the SMPC cluster do not need to be trusted nodes. The only requirement of these nodes is not to collude. This should not be confused with trusted servers. A reasonable way to prevent collusion, is to deploy the computing nodes in premises of organizations having conflicting interests.

Finally, the only information that gets published is the output of the privacy preserving computation. This is returned to the user who initiated the query via the *coordinator* server.

5.5 Supported Computations

Our focus is to create an end-to-end infrastructure for computing privacy preserving analytics. In the field of data mining there have been developed numerous algorithms for data classification, however a little work has been done with respect to data privacy. Since we deal with medical data in this thesis, our primary concern is to preserve data privacy and in the same time provide meaningful information about the data to doctors and researchers. Notably, our goal is to built a complete platform around secure medical data analytics that could be beneficial to a wide range of users. Therefore, the analytics produced from our system should not be complex and convoluted for the average user.

One of the simplest and widely used ways to visualize and comprehend a dataset is histograms. Histograms are bar-graphs that depict frequency distribution, or even statistical approximations, of a dataset. Decision trees are prediction/classification machine learning models, that help to yield conclusions about an item, based on certain observations. The decision tree classification technique is quite intuitive and can offer useful insights about a particular dataset. Generally, decision trees are tree-like structures that classify individuals based on a dataset. Each internal node represents a “test” on an attribute, each branch represents the outcome of the specific test, and each leaf node represents a class label (decision taken after computing all attributes).

Although histograms’ and decision trees’ outcomes are more easily understood than more elaborate algorithms, such as neural networks, SVM or other classifiers, they both provide very insightful information about a dataset. Our goal is to provide an end-to-end architecture with essential analytics algorithms that will provide the building blocks for more elaborate algorithms that are implemented with respect to data privacy. We discuss in more detail histograms and decision trees in sections 6.6 and 6.7 respectively.

6. PRIVACY PRESERVING ALGORITHMS

6.1 Challenges in Privacy Preserving Algorithms

A private computation scheme allows the user to compute any arbitrary function, while revealing no information to any individual server about the identity of that function. Computation over encrypted data can be utilized in a wide variety of use-cases. For instance, the problem of how to use encrypted database fields in search queries, or the problem of testing for membership in a set. Another very common use case is the private set intersection, *i.e. "Department of homeland security (DHS) wants to check its list of terrorist suspects against the passenger manifest of a flight operated by a foreign airline. Neither party is willing to reveal its information, however, if there is a (non-empty) intersection, DHS will not give the flight permission to land".*

Despite the variety of applications that encrypted computation can assist, the translation of any function to its privacy preserving equivalent is not a trivial process. As mentioned in section 2.4, termination problems are introduced and the avoidance of them is not straightforward. First, one has to identify the corresponding termination channels that needs protection against side-channels and leakage; that is, any branch decision which is based on encrypted data (*i.e.* `if` $(Enc(X)$ `is True) then statement`). Consecutively, they have to translate the termination channels using oblivious computation in the encrypted domain. Commonly, the `if` statement is replaced by a loop over all possible values and an oblivious selection of the desired value. Early termination conditions, in general, eventually leak sensitive information, which is why the execution time should only depend on the length of the inputs, not their plaintext values (*i.e.*, execution requires maximum iterations independent of the input). Below we examine the privacy preserving equivalents of some simple and widely used algorithms in order to clarify the termination problems and the translation process.

6.2 Branching Oracles

Although we cannot take branch decisions that are based on encrypted values, recent research [45] has implemented special constructions – dubbed *BRanching Oracles* (*BROs*) – which obliviously evaluate the branch outcomes. *BROs* are treated as decision-making black boxes and enable an alternative solution to branching on private information. All the algorithms presented in the following sections suppose that the underlying encrypted architecture supports a privacy-preserving *BRO*. More specifically, *BROs* enable comparison of encrypted values and obliviously return an encrypted boolean variable (true, false).

For instance, the following example illustrates the usage of *BROs*:

```
if (x > 1) a = b; else a = c;
```

Since `x` is encrypted, the host is unable to make the branch decision whether `a` should take the value of `b` or `c`. Using a branching oracle, an encryption of true or false will emerge from the comparison `(x > 1)` (*i.e.* `gt = (x > 1)`). Then, one can compute obliviously the branch outcome using the result of the private comparison as follows:

```
a = b * gt + c * (1-gt).
```

Using this technique a programmer can replace conditional statements on private data with such oblivious selection clauses. Therefore, the control flow dependencies are converted into data dependencies.

6.3 Notation

Our notation for encrypted values and for operations between encrypted variables consists of a variation of the classical mathematical operations. Namely, $\widetilde{X}$ corresponds to the encryption of number X, while $\hat{+}$, $\hat{-}$, $\star$ and $\hat{\div}$ represent homomorphic addition, subtraction, multiplication and division respectively, as shown in equation 6.1. In a like manner, $\hat{=}$ (eq. 6.2), $\hat{\neq}$ (eq. 6.3) $\hat{<}$ (eq. 6.4) and $\hat{>}$ (eq. 6.5) represent private equality and private comparison, returning an encryption of one (that is $\widetilde{1}$) if the operands map to equal plaintexts (or respectively the first is less than the second), or an encryption of zero ($\widetilde{0}$) otherwise. All those aforementioned comparisons are utilizing the branching oracle of the underlying encrypted architecture.

$$\begin{aligned} Enc(X) \hat{+} Enc(Y) &= Enc(X+Y) \\ Enc(X) \hat{-} Enc(Y) &= Enc(X-Y) \\ Enc(X) \star Enc(Y) &= Enc(X \cdot Y) \\ Enc(X) \hat{\div} Enc(Y) &= Enc(X \div Y) \end{aligned} \tag{6.1}$$

$$Enc(X) \hat{=} Enc(Y) : \begin{cases} \widetilde{1}, & \text{if } X = Y. \\ \widetilde{0}, & \text{otherwise.} \end{cases} \tag{6.2}$$

$$Enc(X) \hat{\neq} Enc(Y) : \begin{cases} \widetilde{1}, & \text{if } X \neq Y. \\ \widetilde{0}, & \text{otherwise.} \end{cases} \tag{6.3}$$

$$Enc(X) \hat{<} Enc(Y) : \begin{cases} \widetilde{1}, & \text{if } X < Y. \\ \widetilde{0}, & \text{otherwise.} \end{cases} \tag{6.4}$$

$$Enc(X) \hat{>} Enc(Y) : \begin{cases} \widetilde{1}, & \text{if } X > Y. \\ \widetilde{0}, & \text{otherwise.} \end{cases} \tag{6.5}$$

Also, in the algorithms described below, the variables in red correspond to private values, while the variables in black represent public data.

6.4 Transforming Algorithms to their Privacy Preserving Equivalent

In this section we will examine three simple algorithms (namely Sum, Max, PIR) and how to develop them in a way to preserve data privacy. Firstly, algorithm 1 computes the sum of an array of numbers and it is quite similar to the private sum algorithm. The latter obliviously updates the encrypted sum with each value of the array, utilizing the homomorphic properties of the encrypted variables.

Algorithm 1 Textbook & Privacy Preserving Sum of an Array

```
1: procedure $\textsc{Sum}(array[N])$                    ▷ Sum of a public array
2:     $sum \leftarrow 0$
3:     for $i \in \{0, \ldots, N-1\}$ do
4:         $sum \leftarrow sum + array[i]$
5:     end for
6:     return $sum$
7: end procedure
Private Vars: $array, sum$
8: procedure $\textsc{Sum}(array[N])$                    ▷ Sum of a private array
9:     $sum \leftarrow \widetilde{0}$
10:    for $i \in \{0, \ldots, N-1\}$ do
11:        $sum \leftarrow sum \hat{+} array[i]$
12:    end for
13:    return $sum$
14: end procedure
```

On the other hand, algorithm 2 finds and returns the maximum number of an array. In the private $\textsc{Max}$ algorithm, the host is not able to determine for each value of the array if it is greater from the current value of max. However, it can obliviously update the value of max, based on the encrypted comparison outcome (alg. 2 line 13, 14). In the cases that the value of the $array$ is less than the current max the gt flag will have the value $\widetilde{0}$, otherwise $\widetilde{1}$. Thus the multiplexer operation in line 14 each time will obliviously "keep" the greatest value and set the variable max to it.

Algorithm 2 Textbook & Privacy Preserving Max of an Array

```
1: procedure $\textsc{Max}(array[N])$                    ▷ Find max of a public array
2:     $max \leftarrow -\infty$
3:     for $i \in \{0, \ldots, N-1\}$ do
4:         if $array[i] > max$ then
5:             $max \leftarrow array[i]$
6:         end if
7:     end for
8:     return $max$
9: end procedure
Private Vars: $array, max, gt$
10: procedure $\textsc{Max}(array[N])$                   ▷ Find max of a private array
11:    $max \leftarrow \widetilde{-\infty}$
12:    for $i \in \{0, \ldots, N-1\}$ do
13:        $gt \leftarrow (array[i] \hat{>} max)$  ▷ $gt$ obliviously gets either $\widetilde{1}$ if $array[i] > max$, or $\widetilde{1}$ otherwise
14:        $max \leftarrow (gt \star array[i]) \hat{+} ((\widetilde{1} \hat{-} gt) \star max)$  ▷ Obliviously update the value of $max$
15:    end for
16:    return $max$
17: end procedure
```

Both textbook algorithms' time complexity (algorithms. 1 and 2) is $\mathcal{O}(n)$ and the transformation to their privacy preserving equivalents did not affect their complexity. However, the privacy-preserving algorithm is inevitably more computationally intensive, since the encrypted operations require many additional CPU cycles.

Algorithm 3 Textbook & Private Information Retrieval

```
 1: procedure $\mathrm{IR}(table[N][2], key)$
 2:     $value \leftarrow 0$
 3:     for $i \in \{0, \ldots, N-1\}$ do
 4:         if $table[i][0] = key$ then
 5:             $value \leftarrow table[i][1]$
 6:             return $value$        ▷ Return the value that corresponds to the requested key
 7:         end if
 8:     end for
 9:     return $value$        ▷ If key not found return $0$
10: end procedure
Private Vars: $table, key, sum, value, eq$
11: procedure $\mathrm{PIR}(table[N], key)$
12:     $sum \leftarrow \widetilde{0}$
13:     for $i \in \{0, \ldots, N-1\}$ do
14:         $value \leftarrow table[i][1]$
15:         $eq \leftarrow (table[i][0] \mathrel{\hat{=}} key)$ ▷ $eq$ obliviously gets either $\widetilde{1}$ if $table[i][0] = key$, or $\widetilde{1}$ otherwise
16:         $sum \leftarrow (eq \star value) \hat{+} ((\widetilde{1} \hat{-} eq) \star sum)$        ▷ Obliviously update the value of $sum$
17:     end for
18:     return $sum$
19: end procedure
```

The third and final example is the Information Retrieval algorithm, where a user maintains a database and desires to retrieve some data based on a key. The simplest textbook algorithm searches each consecutive position on the array to find a key match, and immediately returns the corresponding value. In contrast, the privacy preserving algorithm (PIR) [20] is not able to return before searching the whole database without revealing any information about the key and the result. In this pair of examples the worst case complexity is also $\mathcal{O}(n)$, however in the average case, the former will search the half database.

The idea here is to brute-force all possible table positions and obliviously find and keep the value for the requested key. It has become obvious that it is not straightforward to make more sophisticated algorithms privacy preserving. However, similar techniques are applied for simple and more elaborate algorithms, in order to transform them to their privacy preserving equivalents, sacrificing performance for privacy.

Motivated by privacy concerns and from the challenges that arise in securing any function, in this thesis we concentrate on developing an end-to-end infrastructure for privacy preserving analytics and incorporate some essential algorithms for aggregation statistics and classification.

6.5 Algorithms for Two Types of Data: Categorical & Numerical

Due to the limitless applications that our system can serve, the input data can have many different types. We have separated the data in two broad categories – categorical and continuous, therefore our algorithms are also logically separated for those two different kinds of data. Categorical variables are those that can take on one of a limited, and usually fixed, number of possible values. For instance, blood type, gender, the age-group and the country that a person lives in. On the other hand, numerical/continuous data can take any real number, rendering uncountable the number of different possible values. Examples of the latter category include weight, price, profits, etc.

Medical data, of course, appear in both ways, depending on the "nature" of the attributes. One more reason that this distinction should be made is that even for the same attribute (for instance "Age"), a doctor may either classify the exact age, or the birth date, or even classify the patient in one group (*i.e.* "child" or "adult").

Due to these two different kinds of data that exist in medical datasets, we have separated our algorithms in two categories respectively, categorical and numerical.

6.6 Privacy Preserving Histograms

Despite the challenges presented in sections 6.1 and 6.4, the majority of algorithms can be transformed to their privacy preserving equivalent. However, this is not a straightforward translation , as we examined in section 6.4, and in most cases it adds a complexity overhead to every algorithm that depends its control flow decisions in private data.

Histograms is a practical and notable example of algorithms that are widely used and the complexity of their privacy preserving version remains in computationally feasible levels, comparing to the textbook algorithm. But first of all, what is a histogram?

As stated in [31], histograms initially conceived as a visual aid to statistical approximations. Webster's defines a histogram as "a bar graph of a frequency distribution in which the widths of the bars are proportional to the classes into which the variable has been divided and the heights of the bars are proportional to the class frequencies". A histogram is generally a form of classifying and representing data in some categories of a specific range; the range is an individual "base" element associated with each column.

More specifically, a histogram on some attributes $\{A, B, \ldots, Z\}$ is constructed by partitioning the data distribution of those attributes into some ranges which are mutually disjoint subsets called buckets and approximating the frequencies and values in each bucket. For the sake of simplicity let us suppose that the number of ranges (β) of all attributes are equal ($\beta \geq 1$).

In figures 3 and 4 we present two histograms, the first one-dimensional over the attribute "Patient Age" and the second is a two-dimensional over the attributes "Patient Age" and "Heart rate". In both histograms $\beta = 4$, since the range of values for each attribute has been partitioned in four mutually disjoint subsets. In the 1-dimensional histogram 3 the $y - axis$ corresponds to the total number of occurrences of values that belong to each bucket. In the 2-dimensional histogram 4 since $y - axis$ corresponds to the ranges for the buckets for the second attribute, the occurrences are depicted with different colors.

Similarly, in figures 5 and 6 we present two more histograms, but with categorical values. The attribute "Age Groups" in 5 is separated in four disjoint categories; "Infant", "Adolescent", "Child" and "Adult". In 6 we present demographic information (attribute "Continental Population Groups") in relation with "Age Groups".

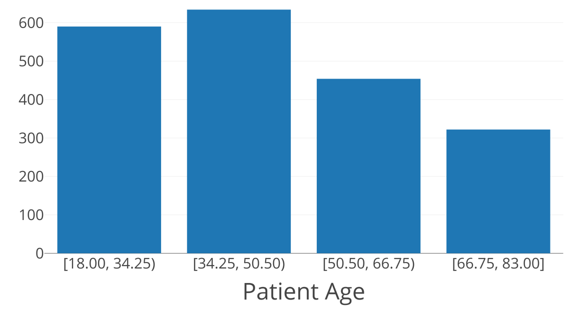


Figure 3: An one-dimensional histogram with $\beta = 4$

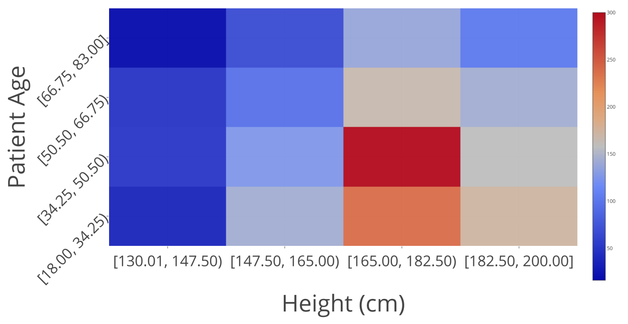


Figure 4: A two-dimensional histogram/heatmap with $\beta = 4$

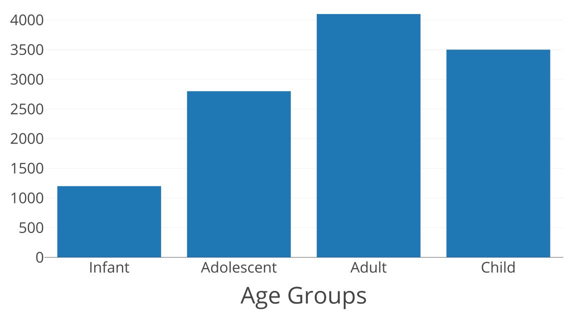


Figure 5: One-dimensional histogram displaying Age Groups

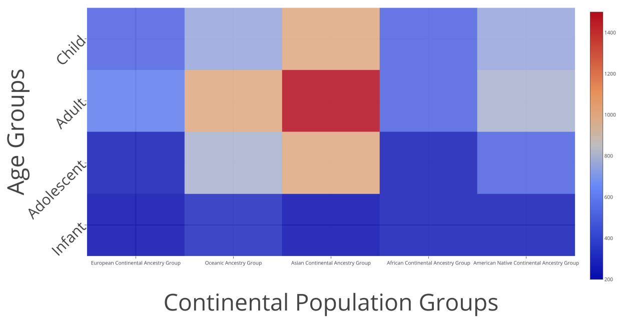


Figure 6: A two-dimensional histogram displaying Age Groups with Continental Population Groups

6.6.1 Algorithms for Privacy Preserving Histograms

Histograms are one of the simplest ways to visualize and easily understand data; rendering them very useful in many data analysis applications. As we already mentioned, histograms consist of bars that are mutually disjoint and their heights are proportional to the counts of values in the corresponding ranges.

Challenges with Privacy-Preserving Histograms: The textbook algorithm splits the dataset to buckets and consecutively for each item it increments the corresponding "bucket-counter". Implementing this algorithm with respect to the privacy of the dataset introduces the problem that for each value it is not trivial to find the bucket that this specific value should be placed.

For instance, let us suppose that we are constructing the histogram in figure 5. In this case $\beta = 4$, thus the algorithm should obliviously split the values into four disjoint buckets. The range of each bucket is computed considering the β parameter, as well as the minimum and maximum age in the dataset. In this specific example, the minimum age is 18, while the maximum is 83. Having $\beta = 4$ results to bucket-range $(83 - 18)/4 = 16.25$. Therefore, the ranges for attribute "Patient Age" are formed as follows: $[18, 34.25)$, $[34.25, 50.50)$, $[50.50, 66.75)$, $[66.75, 83]$. Consecutively, for each patient in the dataset the algorithm calculates the number of bucket that the patient belongs by dividing his/her value with the bucket-range; however, this index is encrypted. Finally, the algorithm iterates through all β buckets and obliviously adding $\widetilde{1}$ if the bucket is the correct one, or $\widetilde{0}$ otherwise.

The aforementioned algorithm works for numerical values; however, the one for categorical values is very similar. Bucket-range is fixed to one, and β equals to all different possible values in the dataset. Also, there is no notion of terms minimum and maximum in categorical values.

6.6.1.1 Privacy Preserving Histogram Computation: A Naive Approach

As we mentioned in section 6.5, we have separated our algorithms in two major categories, for categorical and for numerical data. The procedure described in 6.6.1 is shown in algorithms 4 and 5. These algorithms iterate through all N items and check all possible cells for that item. When the correct cell is found, they increment the corresponding counter. The former is tailored to categorical datasets, while the latter is tailored to numerical/continuous values.

Algorithm 4 Naive Privacy Preserving 1D Histogram for Categorical Values

```
Private Vars: $array, histogram$
1: procedure 1DimSimpleHistogramCategorical($array[N], cells$)
2:     for $i \in \{0, \ldots, N-1\}$ do
3:         for $j \in \{0, \ldots, cells-1\}$ do
4:             $histogram[j] \leftarrow histogram[j] \hat{+} (array[i] \hat{=} j)$    ▷ Add $\tilde{1}$ if $array[i]$ is equal to $j$,
5:         end for                                                                   ▷ $\tilde{0}$ otherwise
6:     end for
7:     return $histogram$
8: end procedure
```

Algorithm 5 Naive Privacy Preserving 1D Histogram for Numerical Values

```
Private Vars: $array, min, max, histogram, cellWidth, cell$
1: procedure 1DimSimpleHistogramNumerical($array[N], cells, min, max$)
2:     $cellWidth \leftarrow (max \hat{-} min) \hat{\div} cells$    ▷ $cellWidth$ gets a float value
3:     for $i \in \{0, \ldots, N-1\}$ do
4:         $cell \leftarrow array[i] \hat{\div} cellWidth$    ▷ $cell$ gets an encrypted value in range $[0, cells-1]$
5:         for $j \in \{0, \ldots, cells-1\}$ do
6:             $histogram[j] \leftarrow histogram[j] \hat{+} (cell \hat{=} j)$    ▷ Add $\tilde{1}$ if $cell$ is equal to $j$,
                                                                             $\tilde{0}$ otherwise
7:         end for
8:     end for
9:     return $histogram$
10: end procedure
```

Both algorithms are straightforward and simple to understand, however, not very efficient. In the following subsections we delve into details for the privacy preserving algorithms that are based in the same idea, but leverage `SIMD` operations.

6.6.1.2 Privacy Preserving Histograms for Categorical Values

One-Dimensional Histograms: In algorithm 6 we present the privacy preserving algorithm of an one-dimensional (1D) histogram for categorical values. In simple words, categorical data means that the values are discrete. Hence, the second parameter in the algorithm (dubbed P) is the number of possible choices that exist in $array[N]$. The input data is given in the form of a private array/vector.

The algorithm creates a boolean array of equal size as the input data, and then for each possible encrypted value of the dataset checks (line 3) and counts (line 4) its occurrences. Method $\textsc{Sum}$ is the same as in algorithm 1. Finally, the counts are gathered into a vector and returned – still encrypted – to the user.

Algorithm 6 Privacy Preserving 1D Histogram for Categorical Values

```
Private Vars: $array, histogram, eq$
1: procedure $\textsc{1DimHistogramCategorical}(array[N], P)$
2:     for $i \in \{0, \ldots, P-1\}$ do        ▷ $P$ is the number of the different (possible) values in $array$
3:         $eq \leftarrow (array \hat{=} i)$        ▷ SIMD; array eq contains $\widetilde{1}$ where $array[j] \hat{=} i$,
                                                  $\widetilde{0}$ otherwise, $j \in \{0, \ldots, P-1\}$
4:         $histogram[i] \leftarrow \textsc{Sum}(eq)$        ▷ Operation to sum eq array
5:     end for
6:     return $histogram$
7: end procedure
```

Multi-Dimensional Histograms: In algorithm 7 we present the privacy preserving algorithm of multi-dimensional histograms for categorical values. As in algorithm 6, here, the third parameter (Ps) is the number of possible choices that exist in $array$. However, since here the input data is a private array with multiple dimensions $array[N][M]$, the possible values should express the possible values for each dimension. Thus, Ps is an array of A slots, where A is the number of attributes.

The algorithm that computes a private multi-dimensional histogram is similar to the one regarding one-dimensional histograms but also addresses the issue of having a histogram of arbitrarily many dimensions. In case we had a fixed (*i.e.* known *a-priori*) number of dimensions the simplest solution would be to use nested loops, as many as the histogram dimensions. Since the number of dimensions is not known, we have to think of something else. We represent the multi-dimensional histogram as a serialized version with an one-dimensional array (a vector) instead of using a multi-dimensional array (a matrix). For example a 2-dimensional 3×4 histogram will be represented as a vector whose length is 12 $(= 3 \cdot 4)$, and a $3 \times 4 \times 5$ 3-dimensional histogram wit a vector of length 60 $(= 3 \cdot 4 \cdot 5)$

When it comes to indexing if we wish to access the 2-dimensional $N \times M$ histogram represented as an one-dimensional at row i and column j, instead of using $h[i][j]$ we use $h[i \cdot M + j]$. Similarly, for the 3-dimensional $L \times N \times M$ histogram, $h[i][j][k]$ becomes $h[i \cdot N \cdot M + j \cdot M + k]$. So there needs to be a computation of a single index based on the multiple dimension indexes. In algorithm 7, the $positions$ vector holds this index for each record of the provided array, as can be seen in lines 6 and 7

Similarly to the 1D algorithm, the multi-dimensional one creates a boolean array (eq) for each possible histogram cell, which counts the occurrences of that cell in the $positions$ vector, as can be seen in lines 12 and 13 of algorithm 7.

Algorithm 7 Privacy Preserving Multi-Dimensional Histogram for Categorical Values

```
Private Vars: $array, column, positions, histogram, eq$
1:  procedure MULTDIMHISTOGRAMCATEGORICAL($array[N][M], attributes[A], Ps[A]$)
2:      $positions \leftarrow [\widetilde{0}, \widetilde{0}, \dots, \widetilde{0}]$    ▷ $positions$ is an array initialized with $\widetilde{0}$
3:      for $a \in \{0, \dots, A-1\}$ do
4:          $attribute \leftarrow attributes[a]$
5:          $column \leftarrow array[:, attribute]$    ▷ Slicing of a specific column of the $array$ matrix
6:          $prod \leftarrow$ PRODUCT($Ps[a+1:]$)    ▷ Product of all $Ps$ elements from index $a+1$ onwards
7:          $positions \leftarrow positions \hat{+} column \star prod$
8:      end for
9:      $length \leftarrow$ PRODUCT($Ps$)
10:     $histogram \leftarrow [\widetilde{0}, \widetilde{0}, \dots, \widetilde{0}]$    ▷ $histogram$ is an array initialized with $\widetilde{0}$
11:     for $j \in \{0, \dots, length-1\}$ do
12:         $eq \leftarrow positions \hat{=} j$    ▷ SIMD; $eq$ gets either $\widetilde{0}$ or $\widetilde{1}$
13:         $histogram[j] \leftarrow$ SUM($eq$)
14:     end for
15:     return $histogram$
16: end procedure

17: procedure PRODUCT($array[N]$)    ▷ Compute and return the product of an array
18:     $product \leftarrow 1$
19:     for $i \in \{0, \dots, N-1\}$ do
20:         $product \leftarrow product \times array[i]$
21:     end for
22:     return $product$
23: end procedure
```

6.6.1.3 Privacy Preserving Histograms for Numerical Values

One-Dimensional Histograms: In algorithm 8 we present the privacy preserving algorithm of one-dimensional (1D) histograms for numerical values. In contrast with the categorical data, numerical data are not discrete, which means that the buckets in which the histogram will separate the dataset should be fixed (the β parameter, as mentioned in section 6.6)

First and foremost, the input data is given in the form of a private array/vector of N positions. The second parameter is open (since the final results will eventually disclose it), and is the number of buckets/cells that the algorithm will create, namely the β factor. The two last parameters are also encrypted and are the minimum and maximum values found in the dataset/array (first parameter of the algorithm). Those two parameters are necessary in order to determine for each element in the array in which cell should be placed.

This algorithm is very similar and in accord with algorithm 6, however does some extra steps. In lines 2 and 3, it first determines the width for each cell and then creates an array of N elements that each one indicates the bucket that the element in the corresponding position of $array$ should be placed. Consecutively, the algorithm creates another boolean array of equal size as the input data, and then for each possible encrypted value of the $cellMap$ checks (line 5) and counts (line 6) its occurrences. Finally, the counts are gathered into a vector and returned – still encrypted – to the user. Line 5 of algorithm 8 is more complex than line 3 of algorithm 6, since the former has to check some corner cases of data that belong to the last bucket.

Algorithm 8 Privacy Preserving 1D Histogram for Numerical Values (Specified Cells)

```
Private Vars: $array, min, max, histogram, cellWidth, cellMap, eq$
1: procedure 1DIMHISTOGRAMNUMERICAL($array[N], cells, min, max$)
2:     $cellWidth \leftarrow (max \hat{-} min) \hat{\div} cells$        ▷ $cellWidth$ gets a float value
3:     $cellMap \leftarrow (array \hat{-} min) \hat{\div} cellWidth$        ▷ SIMD; $cellMap$ has $N$ elements,
                                    each corresponds to an index $idx$ where $\widetilde{0} \leq idx \leq \widetilde{cells}$
4:     for $i \in \{0, \dots, cells - 1\}$ do
5:         $eq \leftarrow (cellMap \hat{=} i) \hat{+} ((cellMap \hat{=} cells) \star (i == cells - 1))$    ▷ SIMD; similar to
                                    line 3 alg. 6, grouping elements to corresponding cell indexes
6:         $histogram[i] \leftarrow \text{SUM}(eq)$
7:     end for
8:     return $histogram$
9: end procedure
```

Multi-Dimensional Histograms: The algorithm that implements multi-dimensional histograms for numerical, *i.e.* continuous data is a straightforward generalization of algorithms 7 and 8. Similarly, for each specified attribute we compute the corresponding cell width (as in the 1-dimensional for numerical values – alg. 8), that depends on the minimum and maximum values and the specified cell number of that attribute. Consecutively, we compute the $positions$ vector containing the histogram cell for each one of the N rows of $array$. Like in algorithm 7, the $positions$ array contains single indexes that correspond to multiple indexes according to the procedure described above in section 6.6.1.2.

6.6.1.4 Filters in Privacy Preserving Histograms

Although histograms are one of the simplest ways to visualize and easily understand data, they also come with some limitations. For instance, it is difficult to visualize in one image more than three different attributes. Even though we can compute histograms of arbitrarily many dimensions, even a 3D representation can sometimes be very obscure and difficult to understand.

Oftentimes, one needs to take into consideration multiple attributes in order to get meaningful results back. However, adding more attributes to the computed histogram is not always an option as for the reasons described above, the output could be obscure or even useless whatsoever.

For all those reasons, we have implemented a *filtering* technique in order to take more attributes into consideration for the secure computation. With filtering, the user is able to select a histogram computation over some attributes, and also specify some extra constraints over the same or different attributes. Each tuple from the dataset has to meet the specified criteria/filters in ordered to be counted in the resulting histogram.

The specified filters are represented as a list of constraints that should be met. Each constraint is defined by the corresponding attribute, an operator (one of $\{<, >, =\}$ for continuous attributes, and just $=$ for categorical ones) and a value. Also a boolean operator (*e.g.* `AND, OR, XOR`) that will be applied between the different constraints is specified.

For example, one could want to see the correlation of age and height but only for a subset of the dataset. Let us suppose two examples, in the first we want the subset that include

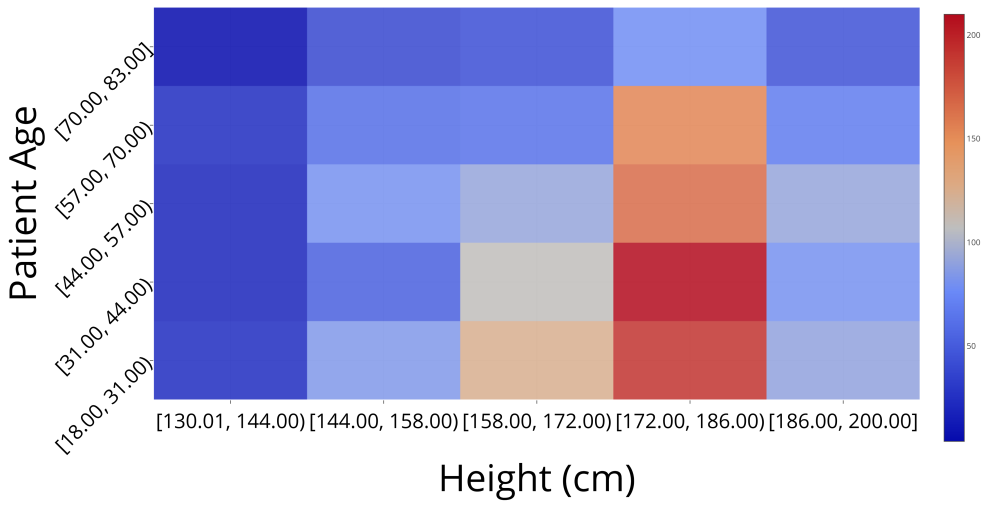


Figure 7: Histogram on *Patient Age* and *Height (cm)* with no filters

all patients of age above 45, while in the second subset we want to include all patients of age above 45 and that in the same time weigh less than 90 kg. This can be achieved with two requests of two-dimensional histograms on the attributes *Patient Age* and *Height (cm)* that also includes the filters *Patient Age* > 45, and the second additionally have the limitation that *Weight (kg)* < 90, and also the boolean operator `AND` between them. These will result to two 2-dimensional histograms (heatmap) with axes the *Patient Age* and *Height (cm)* attributes, but at the same time the results will be restricted by the selection of these filters. The described example from above, is depicted in figures 7, 8 and 9, in which the impact of the filters is obvious.

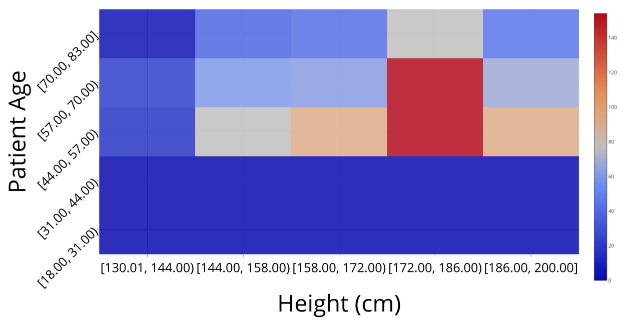


Figure 8: Histogram on *Patient Age* and *Height (cm)* with filters on *Patient Age*

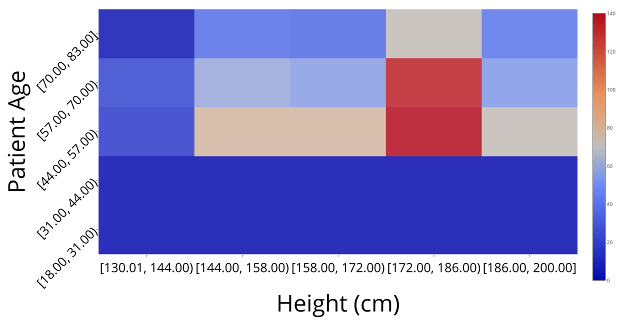


Figure 9: Histogram on *Patient Age* and *Height (cm)* with filters on *Patient Age* and *Weight (kg)*

Another example can be an one-dimensional histogram on Body mass Index (BMI) on the subset of patients that weigh more than 90 kg or are less than 170 cm in height. This can be achieved withl a request of an one-dimensional histogram on the attribute *BMI (kg/msq)* that also includes the filters *Weight (kg)* > 90, *Height (cm)* < 170, and also the boolean operator `OR` between them. The effect of this example can be found in figures 10 and 11.

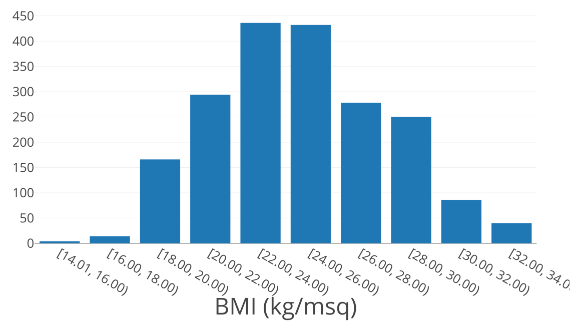


Figure 10: Histogram on *BMI (kg/msq)* with no filters

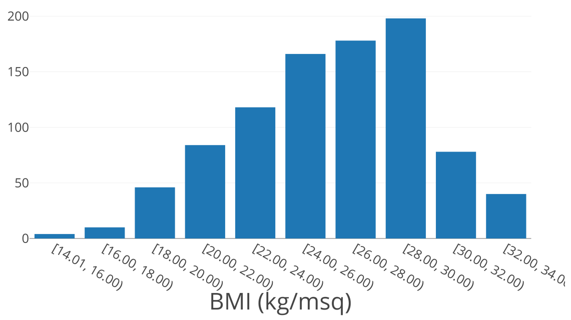


Figure 11: Histogram on *BMI (kg/msq)* with filters on *Height (cm)* and *Weight (kg)*

Algorithm 9 Privacy Preserving Constraint Mask

```
Private Vars: $array, constraintMask, eq, constraintAttribute$
 1: procedure ConstraintMask($array[N][M], operation, constraintAttributes[C],$
                              $constraintOperators[C], constraintValues[C]$)
 2:     if $operation = AND$ then
 3:         $constraintMask \leftarrow [\widetilde{1}, \widetilde{1}, \dots, \widetilde{1}]$        ▷ array initialized with $\widetilde{1}$
 4:     else if $operation = OR$ then
 5:         $constraintMask \leftarrow [\widetilde{0}, \widetilde{0}, \dots, \widetilde{0}]$        ▷ array initialized with $\widetilde{0}$
 6:     else if $operation = XOR$ then
 7:         $constraintMask \leftarrow [\widetilde{0}, \widetilde{0}, \dots, \widetilde{0}]$        ▷ array initialized with $\widetilde{0}$
 8:     end if
 9:     for $c \in \{0, \dots, C-1\}$ do        ▷ $C$ is the number of the different constraints
10:         $constraintIndex \leftarrow constraintAttributes[c]$
11:         $constraintAttribute \leftarrow array[:, constraintIndex]$
12:         $constraintValue \leftarrow constraintValues[c]$
13:         $constraintOperators \leftarrow constraintOperators[c]$
14:         if $constraintOperators = GreaterThan$ then
15:             $eq \leftarrow constraintAttribute \hat{>} constraintValue$        ▷ SIMD; $eq$ vector gets $\widetilde{1}$ at
                    positions where the constraint holds, $\widetilde{0}$ otherwise
16:         else if $constraintOperators = LessThan$ then
17:             $eq \leftarrow constraintAttribute \hat{<} constraintValue$
18:         else if $constraintOperators = Equal$ then
19:             $eq \leftarrow constraintAttribute \hat{=} constraintValue$
20:         end if
21:         if $operation = AND$ then
22:             $constraintMask \leftarrow constraintMask \star eq$        ▷ Applying the boolean operator
                    between the constraints. SIMD
23:         else if $operation = OR$ then
24:             $constraintMask \leftarrow constraintMask \hat{+} eq$
25:         else if $operation = XOR$ then
26:             $constraintMask \leftarrow constraintMask \hat{+} (1 - eq) + (1 - constraintMask) * eq$
27:         end if
28:     end for
29:     return $constraintMask$
30: end procedure
```

The way that *filters* have been implemented is by constructing a vector of size N (*i.e.* as long as the number of tuples in the dataset), that works as a boolean mask corresponding to the specified constraints. Each position of that vector corresponds to a data tuple from the dataset. If that tuple satisfies all the specified constraints, then the vector should have the (encrypted) value `True` in that position, or (encrypted) `False` otherwise. This vector then is taken into consideration when we build the histogram. Only tuples which have “scored” `True` are counted into the histogram.

In algorithm 9, the aforementioned vector is called $constraintMask$. In lines 14-20 we build this boolean mask for each constraint into the eq vector, and in lines 21-27 we combine these vectors to build the total one, according to the specified boolean operator applied between them.

The algorithm for computing histograms for either categorical or numerical attributes utilizes this $constraintMask$ vector. After having computed the constraints, in order to join them with the data computed by the algorithms, a private multiplication with the N-size vector $constraintMask$ is needed. This multiplication can be done with SIMD operation, which saves a lot of communication overhead between the computing nodes.

As the multi-dimensional histogram computation algorithms for numerical and categorical data, constraint mask can be used for as many dimensions as requested. Since the constraint-mask is dependent to the number of tuples in the dataset, the dimensions of the requested histogram can be as many as the user requests.

6.7 Decision Trees

Decision trees are tree-like structures in which each internal node represents a “test” on an attribute, each branch represents the outcome of the specific test, and each leaf node represents a class label (decision taken after computing all attributes). More specifically, decision trees depict models of decisions and their possible results. They are also a simple way to display an algorithm that only contains conditional control statements. Starting from the root node and following the paths to leaves, each path represents classification rules.

For instance, the attribute “Gender” would have two edges leaving it, one for “Male” and one for “Female”. Each edge could point to a new attribute (for example “Age Groups”), and so on. The leaves of the tree contain the expected class value for transactions matching the path from the root to that leaf. Given a decision tree, one can predict the class of a new transaction just by following the “correct” path, which will result in a class label. He/She can predict the class of a transaction by viewing only the non-class attributes (*i.e.* the “constructed” decision tree).

For instance, two friends wish to decide if they will go to play tennis outside, depending the weather conditions. They also remember the previous days that some weather conditions did not allow them to play. Thus, they decide to create a decision tree to assist them. First they define the class attribute to be “Play Tennis”, and three more attributes that will help them decide: “Outlook” (“Sunny”, “Overcast”, “Rainy”), “Windy” (“High”, “Low”) and “Humidity” (“High”, “Low”). Then they construct a decision tree like figure 12 that corresponds to the data that the have collect from the past days.

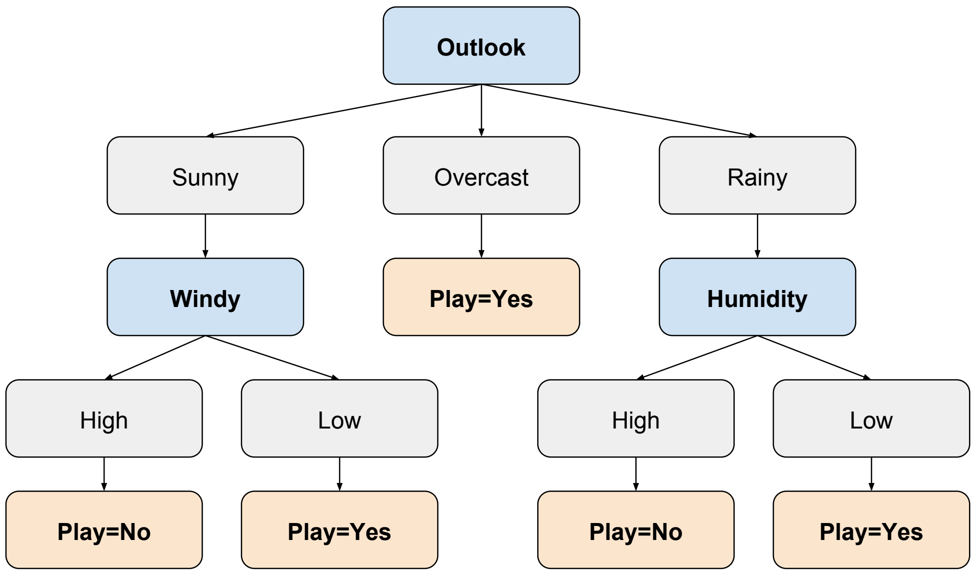


Figure 12: Play-Tennis decision tree example

From figure 12, the two friends are able to extract a set of rules, listed in Code 4, that will help them decide if they should go play tennis given the weather conditions.

```
IF (Outlook = Sunny) AND (Windy = Low) THEN Play=Yes

IF (Outlook = Sunny) AND (Windy = High) THEN Play=No

IF (Outlook = Overcast) THEN Play=Yes

IF (Outlook = Rainy) AND (Humidity = High) THEN Play=No

IF (Outlook = Rainy) AND (Humidity = Low) THEN Play=Yes
```

Code 4: Rules for decision tree from figure 12

Of course, remembering the weather conditions of more days in the past and if they had played tennis those days – having an extended dataset – could probably result to more specific rules. Decision tree rules, help covering all possible future decisions-transactions and can be used to classify them. There are many useful examples and it has become evident why this type of learning has become so popular. The algorithms according to which those tress are generated vary and lead to possible different trees.

One famous algorithm for decision tree construction is ID3 (Iterative Dichotomiser 3). ID3 is the precursor to the C4.5 algorithm, and is typically used in the machine learning and natural language processing domains. Both algorithms build decision trees from a set of training data, using the concept of information entropy, however, C4.5 results in better classification utilizing a more efficient splitting criterion.

6.7.1 Textbook ID3

The ID3 algorithm was first introduced in [51] and assumes that each attribute is categorical, such as the aforementioned attribute “Age Groups” which is separated in four disjoint categories; “Infant”, “Adolescent”, “Child” and “Adult”.

ID3 algorithm constructs the classification tree top-down in a recursive fashion. The idea is to find the *best* attribute that classifies the transactions. At start, the algorithm searches and chooses the best attribute for the root node, and consecutively the remaining transactions are partitioned by it. On each partition, ID3 is then recursively called.

The *best* attribute is defined as the attribute that has the smallest entropy – or in other words, the best information gain. Let T be a set of transactions, C the class attribute and A some non-class attribute. On each iteration, ID3 iterates through every unused attribute A of the dataset and calculates the entropy $H_C(T)$ (equation 6.6 and algorithm 10) for every subset T that results from splitting the dataset on attribute A.

Algorithm 10 Entropy Textbook Algorithm

```
Global Vars: $classAttribute$
1: procedure ENTROPY($examples$)    ▷ Entropy (H(S) equation 6.6), is a measure of the amount of uncertainty in the dataset
2:     $entropy \leftarrow 0$
3:     for each possible value $v_i$ of $classAttribute$ do
4:         $count \leftarrow$ number of examples that $examples[classAttribute] = v_i$
5:         $percentage \leftarrow count \div$ LENGTH($examples$)
6:         if $percentage \neq 0$ then
7:             $entropy \leftarrow entropy - (percentage * log_2(percentage))$
8:         end if
9:     end for
10:    return $entropy$
11: end procedure
```

$$H_C(T) = \sum_{i=1}^{l} -\frac{| T(c_i) |}{| T |} log \frac{| T(c_i) |}{| T |} \tag{6.6}$$

The idea is to identify the class of a transaction t, given that the value of A has been obtained. Let A obtain values $a_1, \ldots, a_m$ and let $T(a_j)$ be the transactions obtaining value a_j for A. Then, the conditional information of T given A, is given from equation 6.7.

Algorithm 11 Information Gain Textbook Algorithm

```
1: procedure InformationGain($examples, attribute$)    ▷ Information gain (equation 6.8) is
       the measure of the difference in entropy from before to after a dataset is split on an attribute
2:     $gain \leftarrow$ Entropy($examples$)
3:     for each possible value $v_i$ of $attribute$ do
4:         $subset \leftarrow \{example \in examples \mid example[attribute] = v_i\}$
5:         $percentage \leftarrow$ Length($subset$) $\div$ Length($examples$)
6:         if $percentage \neq 0$ then
7:             $gain \leftarrow gain - (percentage *$ Entropy($subset$)$)$
8:         end if
9:     end for
10:    return $gain$
11: end procedure
```

Algorithm 12 Best Textbook Algorithm

```
1: procedure Best($examples, attribute$)    ▷ Find the best attribute; the one with maximum
                                              information gain – or similarly, minimum entropy
2:     $maxGain \leftarrow -\infty$
3:     $best \leftarrow -1$
4:     for $attribute \in attributes$ do
5:         $gain \leftarrow$ InformationGain($examples, attribute$)
6:         if $gain > maxGain$ then
7:             $maxGain \leftarrow gain$
8:             $best \leftarrow attribute$
9:         end if
10:    end for
11:    return $best$
12: end procedure
```

$$H_C(T \mid A) = \sum_{j=1}^{m} -\frac{| T(a_j) |}{| T |} H_C(T(a_j)) \tag{6.7}$$

$$Gain(A) = H_C(T) - H_C(T \mid A) \tag{6.8}$$

Finally, the set T is then split by the selected attribute A (best-attribute algorithm 12) that has the maximum gain (equation 6.8 and algorithm 11) – or equivalently minimum $H_C(T \mid A)$ – to produce subsets of the data. The algorithm continues to recurse on each subset, considering only attributes never selected before.

The ALLEXAMPLESSAME (see algorithm 13) procedure is used to determine if all transactions are of the same class, by checking the value of $classAttribute$ for each transaction. It returns $true$ if they are, $false$ otherwise.

Algorithm 13 All Examples Same Textbook Algorithm

```
Global Vars: $classAttribute$
procedure $\text{ALLEXAMPLESSAME}(examples, attribute)$    ▷ Check if all examples are of the same class
    for $i \in \{0, \dots, \text{LENGTH}(examples) - 2\}$ do    ▷ Check all examples, two at a time
        $ex_1 \leftarrow examples[i]$
        $ex_2 \leftarrow examples[i+1]$
        if $(ex_1[classAttribute] \neq ex_2[classAttribute])$ then
            return $false$
        end if
    end for
    return $true$
end procedure
```

The MOSTCOMMONLABEL (algorithm 14) procedure returns the class label that is most common in all available transactions.

Algorithm 14 Most Common Label Textbook Algorithm

```
Global Vars: $classAttribute$
procedure $\text{MOSTCOMMONLABEL}(examples)$    ▷ Return the most common class in all $examples$
    $maxCount \leftarrow -\infty$
    $maxLabel \leftarrow -1$
    for each possible class $c_i$ of $classAttribute$ do
        $eq \leftarrow examples[classAttribute] = c_i$    ▷ SIMD; $eq$ vector gets $1$ at positions where the equality holds, $0$ otherwise
        $count \leftarrow \text{SUM}(eq)$    ▷ $count$ gets the number of examples with class $c_i$
        if $count > maxCount$ then
            $maxCount \leftarrow count$    ▷ Choose maximum count
            $maxLabel \leftarrow c_i$    ▷ Choose class with maximum count
        end if
    end for
    return $maxLabel$
end procedure
```

The recursive ID3 algorithm (see algorithm 15 shown below) has the following structure with three halting conditions.

1. If the set of remaining attributes is empty, then the algorithm returns the label that is most common in all transactions as a leaf (lines 2-3).
2. If all transactions are of the same class, then the algorithm returns this class as a leaf (lines 4-5).
3. If none of the above conditions hold, then the algorithm finds the best splitting attribute (using algorithm 12) and makes a branch for every possible value of that attribute (lines 7-19).

Algorithm 15 ID3 Textbook Algorithm

```
Global Vars: $classAttribute$
1: procedure ID3($examples, attributes$)
2:     if $attributes = \{\}$ then                      ▷ If number of predicting attributes is empty
3:         return MostCommonLabel($examples$)          ▷ Return the label with the most common
                                                        value of the target attribute in the examples
4:     else if AllExamplesSame($examples$) then
5:         return $examples[0][classAttribute]$        ▷ Return a node with the label that is common
                                                        to all examples
6:     end if
7:     $bestAttribute \leftarrow$ Best($examples, attributes$)   ▷ Best attribute is the one with maximum
                                                        information gain – or similarly, minimum entropy
8:     $branches \leftarrow \{\}$
9:     for each possible value $v_i$ of $bestAttribute$ do
10:        Let $branch$ be a new tree branch below root, corresponding to the test $bestAttribute = v_i$
11:        $subset \leftarrow \{example \in examples \mid example[bestAttribute] = v_i\}$
12:        if $subset = \{\}$ then
13:            $branch \leftarrow$ AddLeaf($branch$, MostCommonLabel($examples$)))   ▷ Add a leaf
14:        else
15:            $branch \leftarrow$ AddTree($branch$, ID3($subset$, $attributes - \{bestAttribute\}$))
                                                        ▷ Recurse and add the subtree
16:        end if
17:        $branches \leftarrow branches \cup branch$
18:    end for
19:    return $branches$
20: end procedure
```

6.7.2 Privacy Preserving ID3

In the previous subsection we described the ID3 algorithm which operates on public data. We should now describe the privacy-preserving version of the same algorithm, where all the transactions (*examples*) from which the algorithm builds the tree are private data.

First of all, a key difference from the textbook algorithm is the lack of ability to maintain subsets of the transactions based on private conditions. In particular, as you can see in line 11 of algorithm 15, we wish to keep a subset of all $examples$ that have a certain value in the $bestAttribute$ column.

As we cannot have conditional statements on private data, the only thing we can do is to apply the oblivious selection technique described in section 4.2. Thus, we are not able to keep only the subset of rows that satisfy our condition. A solution to this problem is to define $subset$ as a copy of the $examples$ array. We keep a vector eq (see line 17 of algorithm 22) of equal length with $examples$. The value of $eq[i]$ is $\widetilde{1}$ if $examples[bestAttribute]$ is equal to v_i, or $\widetilde{0}$ otherwise. Since the values of eq are encrypted, we perform a vector multiplication between eq and $examples$ so the result will be identical to $examples$ for ever row i that the condition holds ($\widetilde{eq[i]}$ is equal to $\widetilde{1}$), and $\widetilde{0}$ otherwise. Then we add a dummy row $[\widetilde{-1}, \widetilde{-1}, \ldots, \widetilde{-1}]$ (a row of all $\widetilde{-1}$, denoted as just $\widetilde{-1}$ for simplicity) to all rows that don't satisfy the condition (using multiplication with $\widetilde{1} - eq$). The resulting array has all rows from $examples$ that satisfy the condition, and rows full of $\widetilde{-1}$ wherever the condition does not hold.

For that reason, all the algorithms listed below operate on these kinds of “sets”. In algorithm 16 we describe the CountPositives procedure which returns the (encrypted) number of rows that are not the dummy $[\widetilde{-1}, \widetilde{-1}, \dots, \widetilde{-1}]$ row. The CountPositives procedure is equivalent to the call of $Length$ on sets with public data.

Algorithm 16 Privacy Preserving Count Positives Algorithm

```
Require: $examples, neq, count$
1: procedure CountPositives($examples$)
2:     $neq \leftarrow examples \hat{\neq} \widetilde{-1}$                ▷ SIMD comparison
3:     $count \leftarrow$ Sum($neq$)
4:     return $count$
5: end procedure
```

The AllExamplesSame procedure described in algorithm 17 returns the same result as that in algorithm 13. The way this is achieved is by counting the number of examples/transactions that are of class c_i, for each possible class c_i. The private variable res is initialized to $\widetilde{0}$, and gets increased if any of those counts is equal to the total number of examples, *i.e.* all examples are of that class. If res is greater than $\widetilde{0}$ (equal to $\widetilde{1}$) then all the examples have the same class.

Algorithm 17 Privacy Preserving All Examples Same Algorithm

```
Private Vars: $examples, res, eq, classCounts$
Global Vars: $classAttribute$
1: procedure AllExamplesSame($examples$)        ▷ Check if all examples have the same class
2:     $res \leftarrow \widetilde{0}$
3:     for each possible class $c_i$ of $classAttribute$ do
4:         $eq \leftarrow examples[classAttribute] \hat{=} c_i$      ▷ SIMD; $eq$ vector gets $\widetilde{1}$ at positions
                                                            where the equality holds, $\widetilde{0}$ otherwise
5:         $classCounts[c_i] \leftarrow$ Sum($eq$)      ▷ $classCounts[c_i]$ gets the number of examples with
                                                                                        class $c_i$
6:         $res \leftarrow res \hat{+} (classCounts[c_i] \hat{=}$ CountPositives($examples$))  ▷ $res$ will increase
                                                                   if all examples are of one class
7:     end for
8:     return $res \hat{>} \widetilde{0}$
9: end procedure
```

Similarly to the AllExamplesSame procedure, the MostCommonLabel procedure also maintains the transaction counts for each possible class c_i. Using the technique described in algorithm 2, we obliviously keep the maximum count and the class label corresponding to that count, which is returned.

Algorithm 18 Privacy Preserving Most Common Label Algorithm

```
Private Vars: $examples, maxCount, eq, gt, maxLabel, count$
Global Vars: $classAttribute$
 1: procedure MostCommonLabel($examples$)  ▷ Return the most common class in $examples$
 2:     $maxCount \leftarrow \widetilde{-\infty}$
 3:     $maxLabel \leftarrow \widetilde{-1}$
 4:     for each possible class $c_i$ of $classAttribute$ do
 5:         $eq \leftarrow examples[classAttribute] \mathrel{\hat{=}} c_i$   ▷ SIMD; $eq$ vector gets $\tilde{1}$ at positions
                                                   where the equality holds, $\tilde{0}$ otherwise
 6:         $count \leftarrow$ Sum($eq$)   ▷ $count$ gets the number of examples with class $c_i$
 7:         $gt \leftarrow count \mathrel{\hat{>}} maxCount$
 8:         $maxCount \leftarrow gt \star count \mathbin{\hat{+}} (\tilde{1} \mathbin{\hat{-}} gt) \star maxCount$   ▷ Obliviously choose
                                                   maximum count
 9:         $maxLabel \leftarrow gt \star c_i \mathbin{\hat{+}} (\tilde{1} \mathbin{\hat{-}} gt) \star maxLabel$   ▷ Obliviously choose class
                                                   with maximum count
10:     end for
11:     return $maxLabel$
12: end procedure
```

The Best procedure obliviously chooses the attribute that has the greatest information gain. This attribute is considered to be the best to split the dataset on an will be included in an output tree node. That is why this attribute can be handled as public data (note the usage of the Declassify operator in line 10) after it has been privately computed.

Algorithm 19 Privacy Preserving Best Algorithm

```
Require: $examples, maxGain, gain, gt, best$
 1: procedure Best($examples, attributes$)   ▷ Find the best attribute; the one with maximum
                                               information gain, as public data
 2:     $maxGain \leftarrow \widetilde{-\infty}$
 3:     $best \leftarrow \widetilde{-1}$
 4:     for $attribute \in attributes$ do
 5:         $gain \leftarrow$ InformationGain($examples, attribute$)
 6:         $gt \leftarrow gain \mathrel{\hat{>}} maxGain$
 7:         $maxGain \leftarrow gt \star gain \mathbin{\hat{+}} (\tilde{1} \mathbin{\hat{-}} gt) \star maxGain$   ▷ Obliviously choose maximum gain
 8:         $best \leftarrow gt \star attribute \mathbin{\hat{+}} (\tilde{1} \mathbin{\hat{-}} gt) \star best$   ▷ Obliviously choose best attribute
 9:     end for
10:     return Declassify($best$)
11: end procedure
```

As in algorithms 10 and 11, procedures Entropy and InformationGain (algorithms 20, 21) compute the entropy of a set of transactions, and an attribute's information gain respectively. The key differences with the previously described textbook algorithms include the usage of CountPositives procedure instead of Length, due to the difference in the sets involved in the computation, and the modified log_2 function that handles zero input. Also, note that the subsets (algorithm 21, line 5) are also constructed with the technique described in the beginning of this section.

Algorithm 20 Privacy Preserving Entropy Algorithm

```
Require: $examples, example, entropy, count, percentage, eq$
Global Vars: $classAttribute$
 1: procedure ENTROPY($examples$)    ▷ Entropy (H(S) equation 6.6), is a measure of the amount of uncertainty in the dataset
 2:     $entropy \leftarrow \widetilde{0}$
 3:     for each possible class $c_i$ of $classAttribute$ do
 4:         $eq \leftarrow examples[classAttribute] \hat{=} c_i$    ▷ SIMD; $eq$ vector gets $\widetilde{1}$ at positions where the equality holds, $\widetilde{0}$ otherwise
 5:         $count \leftarrow$ SUM($eq$)
 6:         $percentage \leftarrow count \div$ COUNTPOSITIVES($examples$)
 7:         $entropy \leftarrow entropy - (percentage * log_2(percentage))$    ▷ Here we use a modified $log_2$ function that returns $\widetilde{0}$ given $\widetilde{0}$ as input
 8:     end for
 9:     return $entropy$
10: end procedure
```

Algorithm 21 Privacy Preserving Information Gain Algorithm

```
Require: $examples, example, gain, subset, percentage$
 1: procedure INFORMATIONGAIN($examples, attribute$)    ▷ Information gain (equation 6.8) is the measure of the difference in entropy from before to after a dataset is split on an attribute
 2:     $gain \leftarrow$ ENTROPY($examples$)
 3:     for each possible value $v_i$ of $attribute$ do
 4:         $eq \leftarrow examples[attribute] \hat{=} v_i$    ▷ SIMD; $eq$ vector gets $\widetilde{1}$ at positions where the equality holds, $\widetilde{0}$ otherwise
 5:         $subset \leftarrow examples \star eq \hat{+} (\widetilde{1} \hat{-} eq) \star \widetilde{-1}$    ▷ SIMD; $subset$ gets a copy of $examples$ at positions where the equality holds, $\widetilde{-1}$ otherwise
 6:         $percentage \leftarrow$ COUNTPOSITIVES($subset$) $\hat{\div}$ COUNTPOSITIVES($examples$)
 7:         $gain \leftarrow gain \hat{-} percentage \star$ ENTROPY($subset$)
 8:     end for
 9:     return $gain$
10: end procedure
```

The complete $\mathrm{ID3}$ procedure is described in algorithm 22. In lines 6-11 of the algorithm we retrieve class label that is same to all examples ad return it. In order to do that, we cannot just take a random's example value in the $classAttribute$ column. We should make sure that this is not a dummy ($\widetilde{-1}$) example (line 8). The rest of the algorithm is quite similar to the textbook one. The differences mainly include the subsets handling. (creation, counting, etc.)

Algorithm 22 Privacy Preserving ID3 Algorithm

```
Private Vars: $examples, example, eq, subset$
Global Vars: $classAttribute$
procedure ID3($examples, attributes$)
    if $attributes = \{\}$ then    ▷ If number of predicting attributes is empty
        return DECLASSIFY(MOSTCOMMONLABEL($examples$))    ▷ Return the most common class in the dataset
    else if DECLASSIFY(ALLEXAMPLESSAME($examples$)) then
        $label \leftarrow \widetilde{-1}$
        for $example \in examples$ do
            $neq \leftarrow example[classAttribute] \mathrel{\hat{\neq}} \widetilde{-1}$
            $label \leftarrow neq \star example[classAttribute] \mathbin{\hat{+}} (\tilde{1} - neq) \star label$
        end for
        return DECLASSIFY($label$)
    end if
    $bestAttribute \leftarrow$ BEST($examples, attributes$)    ▷ Best attribute is the one with maximum information gain – or similarly, minimum entropy
    $branches \leftarrow \{\}$
    for each possible value $v_i$ of $bestAttribute$ do
        Let $branch$ be a new tree branch below root, corresponding to the test $bestAttribute = v_i$
        $eq \leftarrow examples[bestAttribute] \mathrel{\hat{=}} v_i$    ▷ SIMD; $eq$ vector gets $\tilde{1}$ at positions where the equality holds, $\tilde{0}$ otherwise
        $subset \leftarrow examples \star eq \mathbin{\hat{+}} (\tilde{1} \mathbin{\hat{-}} eq) \star \widetilde{-1}$    ▷ SIMD; $subset$ gets a copy of $examples$ at positions where the equality holds, $\widetilde{-1}$ otherwise
        if DECLASSIFY(COUNTPOSITIVES($subset$) $\hat{=}$ $\tilde{0}$) then
            $branch \leftarrow$ ADDLEAF($branch$, MOSTCOMMONLABEL($examples$))    ▷ Add leaf
        else
            $branch \leftarrow$ ADDTREE($branch$, ID3($subset, attributes - \{bestAttribute\}$))    ▷ Recurse and add the subtree
        end if
        $branches \leftarrow branches \cup branch$
    end for
    return $branches$
end procedure
```

6.7.2.1 Privacy Assessment

In privacy-preserving algorithms, all intermediate values should remain private. In many cases, intermediate values may reveal some patterns, and in general sensitive information about the private inputs.

However, in the case of ID3, some of these intermediate values are part of the output and eventually will be revealed. For instance, the attributes chosen in each node of the tree will be revealed in the final results. Thus, there is no reason to trying to protect them during the protocol execution. In the privacy-preserving ID3, as in all algorithms in this thesis, we explicitly define which values are private (and thus encrypted), and which are public.

As stated in [41], although the name of the attribute with the highest information gain is revealed, nothing is learned of the actual $H_C(T \mid A)$ values themselves. This observation holds true for all such cases, since neither the information gain nor any other intermediate values which are not explicitly denoted as public can be leaked.

For an algorithm to remain private, the requirement is that any information is learned by the algorithm, can also be learned directly from the public input and output [41].

As mentioned in section 4.2 the DECLASSIFY operator publishes a private value. Consecutively, we reason about the selection of public and private data in the described algorithms and also about the usage of the DECLASSIFY operator.

First of all, we consider the case that all involved parties have the same schema for their data. Thus, attribute names are known to all parties an should not be considered private information.

Lindell and Pinkas have constructed a protocol in [41] for privately computing ID3 that works in the two-party setting. This two-party protocol is proven to be private. However that algorithm does not consider the case in where the set of transactions T is empty. Algorithm 22 is a generalization to the multi-party setting from the 2-party protocol that is described in [41]. The algorithm also addresses the empty transaction set case.

The ID3 algorithm (see algorithm 22) can terminate (*i.e.* return a value) in three different cases. Each time, the algorithm's output is a node of the resulting tree. That is why, everything that the algorithm returns in any of the three cases should be considered as public data (see DECLASSIFY operator in lines 9, 11 and public $branches$ variable in line 14).

Empty attribute set: The first termination channel of the algorithm is when the set of remaining attributes is empty, *i.e.* when there are no attributes left for the algorithm to check.. This information is publicly known since the remaining attributes can be derived from the output tree. In this case, the algorithm returns the most common label of the remaining transactions. Here, we use the DECLASSIFY operator to publish that label and add it as a leaf to the tree.

All examples same: The second way that algorithm 22 can terminate, is when all transactions of the dataset are of the same class. We use the DECLASSIFY operator to determine that information (*i.e.* if all the dataset is of the same class).

After recursive calls: Finally, if none of the above cases terminate the execution of ID3, the algorithm will recursively add either a leaf or a sub-tree for each possible value of the selected attribute, and then return. In order to determine whether to add a leaf or a sub-tree, we publish the information that there are no transactions left in the subset of the dataset that corresponds to a particular value of the selected attribute.

By observing the output tree, as well as the public input, one can make the following observations. Wherever there is a leaf node with class c_i, one can tell if the set of remaining attributes is empty, simply by observing the path from the root of the tree. If all attributes have been used then the set is empty, otherwise it is not. If the attribute set is not empty, we can deduce that one of two things holds. Either all transactions that belong to the current path are of the same class (c_i), or that there are no transactions at all in that path. These exact information is also published by algorithm 22, with the only difference being that the information published by the algorithm distinguishes between when all transactions are of the same class and the corner case of having no transactions for a particular path of the tree.

6.7.3 Privacy Preserving C4.5

C4.5 is a decision tree generating algorithm developed by Ross Quinlan [52]. The C4.5 algorithm is an extension of the ID3 algorithm described in section 6.7.1 also developed by Quinlan [51]. It was ranked #1 in the *Top 10 Algorithms in Data Mining* pre-eminent paper published by Springer Knowledge and Information Systems in 2008 [62].

The main difference between C4.5 and ID3 is that the former has native support for both continuous (*i.e.* numerical) data and discrete (*i.e.* categorical) data. The idea behind C4.5 classification is to find the best splitting point for an attribute and split the dataset on that point. Transactions that are below that threshold belong to one branch and transaction above that threshold belong to the other branch. The splitting criterion is based on the information gain 6.8 (difference in entropy 6.6) given by the splitting point. The attribute that most efficiently splits the dataset is chosen by the algorithm at each level of the tree. After that the algorithm recurses on the two subsets created by that split.

In algorithm 23 we present the privacy-preserving $\mathrm{C45}$ procedure. It is quite similar to the $\mathrm{ID3}$ one in algorithm 22. The main differences include that the call of $\textsc{Best}$ procedure (see algorithm 24), now not only returns the "best" attribute, but also the "best" threshold that this attribute should split on, and the subsets that result from that split. In case that the chosen attribute is categorical, then the $bestSplitted$ variable will contain the subsets resulting from each possible value v_i of that attribute. ($\{example \in examples \mid example[bestAttribute] = v_i\}$ for each possible v_i), and the algorithm creates that many branches. In case that $bestAttribute$ is a numerical attribute, then the $bestSplitted$ variable will contain the two subsets – the set of transactions t having $t[bestAttribute] <= bestThreshold$ ($less$), and the set of transactions t having $t[bestAttribute] > bestThreshold$ ($greater$). Also, in that case creates two branches one for each aforementioned subsets. Since the algorithm creates two branches for every split for numerical attributes) the output tree is a binary tree (branching factor $= 2$), unlike ID3 where the branching factor of the tree is equal to the number of possible values for every attribute.

Algorithm 23 Privacy Preserving C4.5 Algorithm

```
Private Vars: $examples, allSame, subset, less, greater, bestSplitted$
Global Vars: $classAttribute, categoricalAttributes$
1:  procedure $\mathrm{C45}(examples, attributes)$
2:      if $attributes = \{\}$ then                    ▷ If number of predicting attributes is empty
3:          return $\textsc{MostCommonLabel}(examples)$  ▷ Return the label with the most common
                                                value of the target attribute in the examples
4:      else if $\textsc{Declassify}(\textsc{AllExamplesSame}(examples))$ then
5:          $label \leftarrow \widetilde{-1}$
6:          for $example \in examples$ do
7:              $neq \leftarrow example[classAttribute] \hat{\neq} \widetilde{-1}$
8:              $label \leftarrow neq \star example[classAttribute] \hat{+} (\tilde{1} - neq) \star label$
9:          end for
10:         return $\textsc{Declassify}(label)$
11:     end if
12:     $bestAttribute, bestThreshold, bestSplitted \leftarrow \textsc{Best}(examples, attributes)$    ▷ Best
            attribute is the one whose split provides maximum information gain. $bestSplitted$
                                              corresponds to the subsets generated by that split.
13:     $branches \leftarrow \{\}$
```

```
14:     if $bestAttribute \in categoricalAttributes$ then        ▷ If best attribute is categorical
15:         for each possible value $v_i$ of $bestAttribute$ do
16:             Let $branch$ be a new tree branch corresponding to $bestAttribute = v_i$
17:             $subset \leftarrow bestSplitted[v_i]$
18:             if $\textsc{Declassify}(\textsc{CountPositives}(subset) \hat{=} \widetilde{0})$ then
19:                 $branch \leftarrow \textsc{AddLeaf}(branch, \textsc{MostCommonLabel}(examples))$
20:             else
21:                 $branch \leftarrow \textsc{AddTree}(branch, \text{C45}(subset, attributes - \{bestAttribute\}))$
22:             end if
23:             $branches \leftarrow branches \cup branch$
24:         end for
25:     else
26:         Let $branch$ be a new tree branch corresponding to $bestAttribute <= bestThreshold$
27:         $less \leftarrow bestSplitted[0]$
28:         if $\textsc{Declassify}(\textsc{CountPositives}(less) \hat{=} \widetilde{0})$ then
29:             $branch \leftarrow \textsc{AddLeaf}(branch, \textsc{MostCommonLabel}(examples))$
30:         else
31:             $branch \leftarrow \textsc{AddTree}(branch, \text{C45}(less, attributes - \{bestAttribute\}))$
32:         end if
33:         $branches \leftarrow branches \cup branch$
34:         Let $branch$ be a new tree branch corresponding to $bestAttribute > bestThreshold$
35:         $greater \leftarrow bestSplitted[1]$
36:         if $\textsc{Declassify}(\textsc{CountPositives}(greater) \hat{=} \widetilde{0})$ then
37:             $branch \leftarrow \textsc{AddLeaf}(branch, \textsc{MostCommonLabel}(examples))$
38:         else
39:             $branch \leftarrow \textsc{AddTree}(branch, \text{C45}(greater, attributes - \{bestAttribute\}))$
40:         end if
41:         $branches \leftarrow branches \cup branch$
42:     end if
43:     return $branches$
44: end procedure
```

Here we present the privacy-preserving algorithms for C45 and BEST procedures. As in section 6.7.2 we face the same restrictions regarding the subset creation, for the same reasons as described before. We omit the rest procedures, as well as their textbook equivalents, as they are similar to the ones presented in sections 6.7.1 and 6.7.2. One difference for example is that the INFORMATIONGAIN procedure takes the disjoint subsets that occur from a split as an argument instead of taking the attribute and computing the subsets inside. Also, another argument is the current entropy of the dataset that gets computed once in the BEST procedure for optimization purposes.

Algorithm 24 Privacy Preserving C4.5 Best Algorithm

```
Global Vars: $classAttribute, categoricalAttributes$
 1: procedure BEST($examples, attributes$)
 2:     $bestSplitted \leftarrow \widetilde{\{\}}$, $maxGain \leftarrow \widetilde{-\infty}$, $bestThreshold \leftarrow \widetilde{-1}$, $bestAttribute \leftarrow \widetilde{-1}$
 3:     $ent \leftarrow$ ENTROPY($examples$)
 4:     for $attribute \in attributes$ do
 5:         if $attribute \in categoricalAttributes$ then            ▷ If attribute is categorical
 6:             $splitted \leftarrow \{\}$
 7:             for each possible value $v_i$ of $attribute$ do
 8:                 $eq \leftarrow examples[attribute] \hat{=} v_i$       ▷ SIMD; $eq$ vector gets $\tilde{1}$ at positions
                                                    where the equality holds, $\tilde{0}$ otherwise
 9:                 $subset \leftarrow examples \star eq \hat{+} (\tilde{1} \hat{-} eq) \star \widetilde{-1}$     ▷ SIMD; $subset$ gets a copy of
                                  $examples$ at positions where the equality holds, $\widetilde{-1}$ otherwise
10:                 $splitted \leftarrow splitted \cup subset$
11:             end for
12:             $gain \leftarrow$ INFORMATIONGAIN($ent, examples, splitted$)
13:             $gt \leftarrow gain \hat{>} maxGain$
14:             $maxGain \leftarrow gt \star gain \hat{+} (\tilde{1} \hat{-} gt) \star maxGain$    ▷ Obliviously choose max gain
15:             $bestAttribute \leftarrow gt \star attribute \hat{+} (\tilde{1} \hat{-} gt) \star bestAttribute$
16:             $bestSplitted \leftarrow gt \star splitted \hat{+} (\tilde{1} \hat{-} gt) \star bestSplitted$
17:         else                                                      ▷ If attribute is numerical
18:             Sort $examples$ by $attribute$ column
19:             for $i \in \{0, \ldots,$ LENGTH($examples$) $- 2\}$ do   ▷ Check all examples, two at a time
20:                 Let $ex_1$ be the first valid example
21:                 Let $ex_2$ be the second valid example
22:                 $neq \leftarrow ex_1 \hat{\neq} ex_1$
23:                 $threshold \leftarrow (ex_1[attribute] + ex_1[attribute])/2$
24:                 $lt \leftarrow examples[attribute] \hat{\leq} threshold$      ▷ SIMD; $lt$ vector gets $\tilde{1}$ at positions
                                                  where the inequality holds, $\tilde{0}$ otherwise
25:                 $less \leftarrow examples \star lt \hat{+} (\tilde{1} \hat{-} lt) \star \widetilde{-1}$            ▷ SIMD; $less$ gets a copy of
                                $examples$ at positions where the inequality holds, $\widetilde{-1}$ otherwise
26:                 $gt \leftarrow examples[attribute] \hat{>} threshold$
27:                 $greater \leftarrow examples \star gt \hat{+} (\tilde{1} \hat{-} gt) \star \widetilde{-1}$
28:                 $splitted \leftarrow \{less, greater\}$
29:                 $gain \leftarrow$ INFORMATIONGAIN($ent, examples, splitted$)
30:                 $gt \leftarrow gain \hat{>} maxGain$
31:                 $maxGain \leftarrow gt \star gain \hat{+} (\tilde{1} \hat{-} gt) \star maxGain$
32:                 $bestAttribute \leftarrow gt \star attribute \hat{+} (\tilde{1} \hat{-} gt) \star bestAttribute$
33:                 $bestThreshold \leftarrow gt \star threshold \hat{+} (\tilde{1} \hat{-} gt) \star bestThreshold$
34:                 $bestSplitted \leftarrow gt \star splitted \hat{+} (\tilde{1} \hat{-} gt) \star bestSplitted$
35:             end for
36:         end if
37:     end for
38:     return DECLASSIFY($bestAttribute$), DECLASSIFY($bestThreshold$), $bestSplitted$
39: end procedure
```

7. IMPLEMENTATION DETAILS

Our application consists of four concrete parts, the *coordinator*, the data providers, the SMPC cluster and the user interface.

The code for the end-to-end architecture, which is deployed in [7], as well as the privacy-preserving algorithms is open-source and hosted in GitHub[8].

7.1 Coordinator

The *coordinator* handles all private computation requests, communicates with the data providers – requesting them to securely import their data to the computing cluster, initiates the secure computation in the SMPC cluster, and finally returns to the user the requested analytics results.

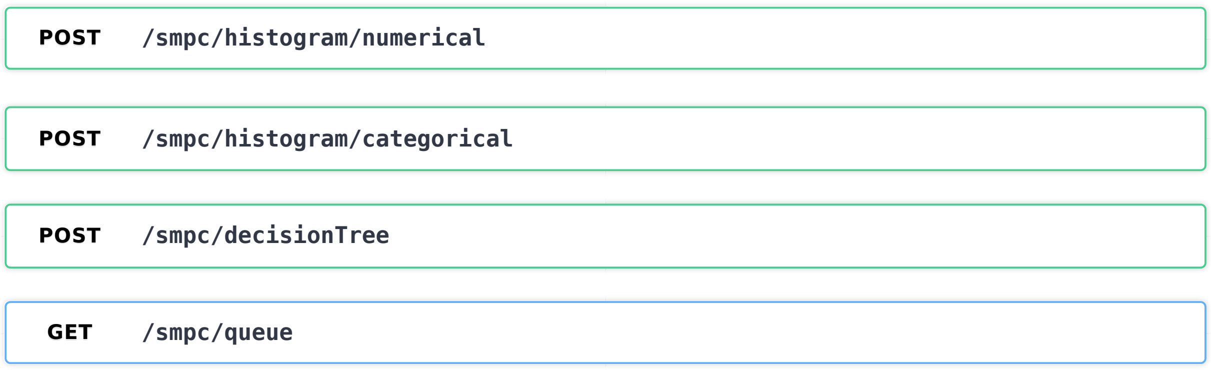
The coordinator server has been designed as a Representational State Transfer (REST) web service. This design helps decoupling server and client applications; the API that provides is cleaner and easier to understand/discover. It's standard format/usage makes it easy for new clients to use the application even if the application was not designed with these clients' use cases in mind.

We have implemented this web service using `Node.js` and more specifically the `Express server` minimal web framework.

The RESTful API provided by the coordinator server can be found in the following section.

7.1.1 RESTful API

Table 1: Coordinator's RESTful API



POST	`/smpc/histogram/numerical`
POST	`/smpc/histogram/categorical`
POST	`/smpc/decisionTree`
GET	`/smpc/queue`

The RESTful API of the *coordinator* is summarized in table 1. The two first POST requests initiate a secure histogram computation for numerical and categorical values respectively. With the third POST request a user can request a secure decision tree computation for either numerical or categorical values, specifying in the request body one of ID3 and C4.5 classifiers. The `/smpc/queue` GET request is for checking the status and/or result of a

[7]https://mhmd.madgik.di.uoa.gr/

[8]https://github.com/Athena-MHMD/smpc-analytics

specified computation request (the aforementioned POST requests). In appendix A.1 we explain in more detail each individual POST and GET request.

7.1.2 Sequence of Actions

In section 5.4.1 we have presented a brief overview of the end-to-end query execution. In this section, we elaborate on this brief overview, including many implementation details that have been omitted.

Step 1: A user makes a privacy-preserving analytics request to the *coordinator*.

This request is either made from the UI (presented in section 7.4) or from another service (*e.g.* Postman tool[9], or even another custom UI that accords with our API).

Step 2: The *coordinator* server is responsible for orchestrating all the involved parties: it communicates with the data-providers requesting secure data-import to the SMPC cluster.

- **2.a:** The *coordinator* maintains a database with the IP addresses of all hospitals involved in the privacy-preserving computation so it can send import requests.
- **2.b:** The user has the option to select which of the data providers to involve in the secure computation. The *coordinator* sends import requests to the selected data providers.
- **2.c:** After the import procedure has finished, the *coordinator* is responsible to return the results to the user. If the initial request has been made from the UI, the results are visualized and returned through the UI, otherwise a JSON file is returned to the user.

Step 3: The data-providers extract the requested data from their datasets and securely import them to the SMPC cluster applying secret-sharing.

A database is maintained in each hospital with all its private data. However, in each request, the user selects which attributes take part in the privacy-preserving analytics.

- **3.a:** Each hospital's server receives a secure data import request from the *coordinator* for the user selected attributes.
- **3.b:** A data extraction and a secret sharing procedure takes place in each hospital's server in order to securely import the data to the SMPC cluster.

The data-import procedure is described in more detail in section 7.2.2.

Step 4: The SMPC cluster computes the privacy preserving analytics on the requested data and returns the results to the *coordinator*.

Depending on the user's request, the SMPC cluster execute either a secure aggregation or a secure decision tree algorithm. After the computation has finished, the SMPC cluster returns the results to the *coordinator*.

[9]https://www.getpostman.com

Step 5: Finally, the results are returned to the user through the *coordinator*.

If the initial request has been made from the UI, the results are visualized and returned through the UI, otherwise a JSON file is returned to the user as described in section 7.1.1.

7.1.3 Result Caching

Privacy-preserving algorithms can sometimes be highly time-demanding. Execution times vary depending on the number of patients, attributes, and also the analytics algorithm that is used. For instance, by definition, decision trees require way more computational power that aggregators do. Similarly, their privacy-preserving equivalents are computationally intensive tasks.

Furthermore, since our end-to-end system is designed to accept hundreds of requests per day, it is computationally infeasible to meet them all. Not to mention, that many of them could possible be the same request – or even repeated.

For all those reasons, we have implemented a caching mechanism that stores secure computation results so future requests for that data can be served faster. If a cache hit occurs, the secure computation is omitted and the results are immediately returned to the user. Otherwise, if a cache miss occurs, the secure computation will normally continue as described in sections 5.4 and 7.1.2.

However, a proficient user may want to omit the already computed results, in case of a recent dataset modification. In this case, he/she can explicitly specify that the cache should not be used by adding a “cache”: “no” option in the JSON request body. For the average user, that would not be necessary, since our system invalidate cache results that are older than 1 month.

7.2 Data Providers

When a private computation request arises, the *coordinator* server communicates with the data providers and requests them to securely import their data to the computing cluster. Finally the *coordinator* returns the results of the computation to the researcher.

Data providers run their own web server that listen to import requests from the *coordinator*.

That server is implemented using `Node.js Express Server`, and provides the following RESTful API.

7.2.1 RESTful API

Table 2: Data Providers’ RESTful API

POST	`/smpc/import/numerical`
POST	`/smpc/import/categorical`

The RESTful API of the data providers is summarized in table 2. The above two POST requests initiates a secure import for numerical and categorical datasets respectively. In appendix A.2 we explain in more detail the two types of POST requests that the *coordinator* sends to the data providers.

7.2.2 Data Importer

The secure data import is one of the most essential procedures in the secure multi-party computation scenario. In this step, the data are secret-shared to the three computing parties. The shares are cryptographic part of the secret but reveal nothing about it. In section 2.6, we examined some secret sharing techniques and the homomorphic properties that some of them have. In simple words, the secret sharing homomorphic property enables meaningful computation on the shares of the initial data.

We use a standard tabular structure for the data in order to import them in the SMPC cluster's encrypted database provided by Sharemind. The datasets to be imported should be in comma-separated values (CSV) format. In case that a dataset has some other non-tabular arrangement (see section 8), we transform them in the standard CSV format using software that we developed for that particular task.

The aforementioned procedure is described in more detail in section 8.

7.2.3 Containerization

The number of data providers can start from just one (trivial case), and grow unrestrictedly. The case with only one data provider is when that provider just wants to outsource the computation to a cloud, without having to trust an individual server. The case of multiple data providers is the most common case, where the providers wish to perform computation over the union of all datasets without anyone being able to learn anything more than the computation's output. Thus, we wanted the deployment of each data provider to be easy and fast; veiling all the individual parts, such as the required packages, the SMPC importer installation, any software for data pre-processing etc.

For that reason we decided that the best choice is to offer a containerized solution. We used Docker[10] for the building and shipping of an *image* with everything installed and ready to run. That image also includes the web server accepting requests for data import (according to the API specified in section 7.2.1), already up and running.

Each individual data provider, can use this image to create a lightweight, portable, self-sufficient container, which can run virtually anywhere. The container will run locally in the provider's premises. The only requirement that a data provider has is to upload its datasets to the container in order to be able to securely import them into the SMPC cluster. In this way, non-expert users can easily provide their data to be a part of the secure computation.

7.3 SMPC Cluster

For the purpose of this dissertation, we have deployed the Sharemind secure computing platform on three 64-bit virtual machines (VM), each running Ubuntu 18.04. The three

[10]https://www.docker.com/

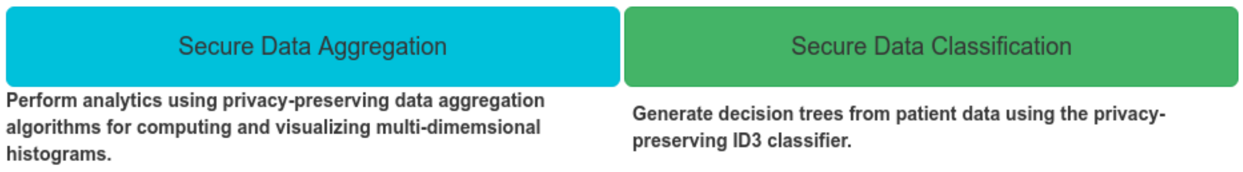
VMs share an Intel Xeon E5-2670 (v2) at 2.50 GHz. Each VM utilizes 6 cores and 8 GB of memory.

As described in section 5.4.2.1, ideally, the three computing nodes should be deployed in premises with conflicting interests, preventing any adversary from controlling all of them simultaneously. This should not be confused with trusted servers.

7.4 User Interface

Apart from the RESTful API, a user/researcher can initiate a secure computation through our user interface (UI), deployed in https://mhmd.madgik.di.uoa.gr/. Our the goal was to design a simple and elegant user interface which is easy to use and in the same time enjoyable (user-friendly).

In the landing page, except from reading and understanding useful information about the end-to-end infrastructure, the operator can choose between two options: secure data aggregation and secure data classification, as depicted bellow[11].

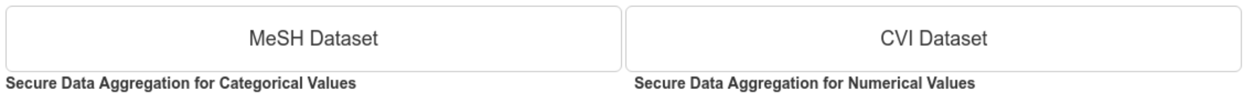


Perform analytics using privacy-preserving data aggregation algorithms for computing and visualizing multi-dimemsional histograms.

Generate decision trees from patient data using the privacy-preserving ID3 classifier.

Figure 13: Landing page: choosing between secure data aggregation & classification

Consecutively, the user can select the desired dataset on which the algorithms will execute, as in figure 14[12].



Secure Data Aggregation for Categorical Values

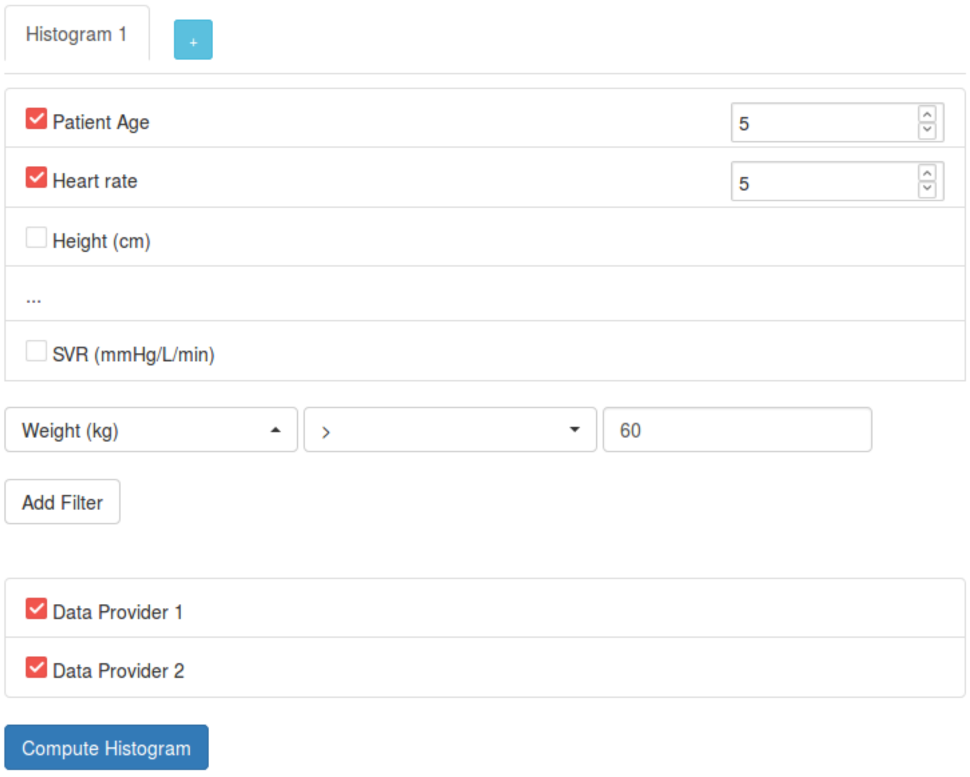
Secure Data Aggregation for Numerical Values

Figure 14: Selecting dataset for secure aggregation

Secure Aggregation for the CVI dataset: After selecting the secure aggregation for the CVI (numerical) dataset, the researcher has to choose attributes and their corresponding histogram cells (β factor), and optionally filters and data-providers for the privacy-preserving computation, as in figure 15. The results of this computation are shown in figure 16.

[11]In appendix B we have included the whole landing page as a screen-shot, as well as many other components of the UI.

[12]The option for selecting dataset for secure classification is identical.



☑ Patient Age 5
☑ Heart rate 5
☐ Height (cm)
...
☐ SVR (mmHg/L/min)

Weight (kg) > 60

Add Filter

☑ Data Provider 1
☑ Data Provider 2

Figure 15: Choosing attributes, filters and data-providers for secure aggregation for numerical dataset

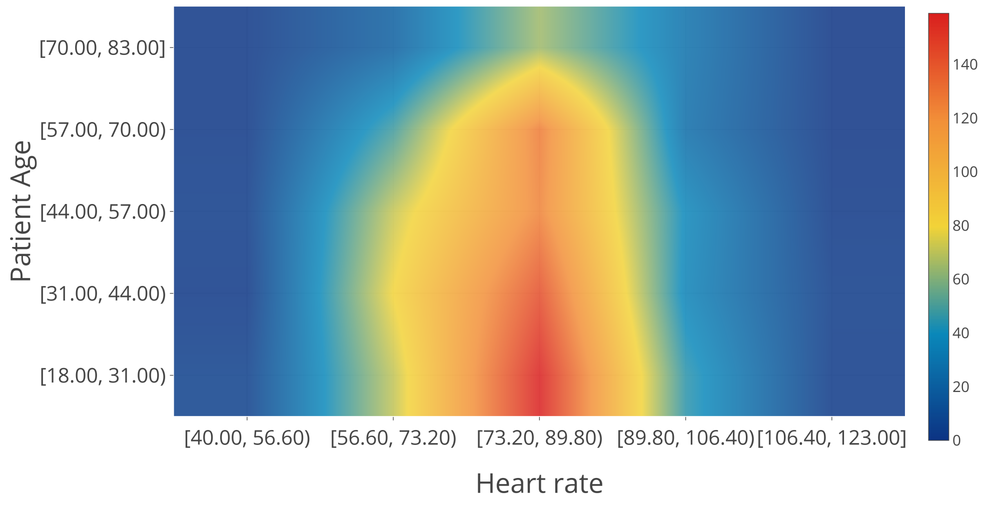
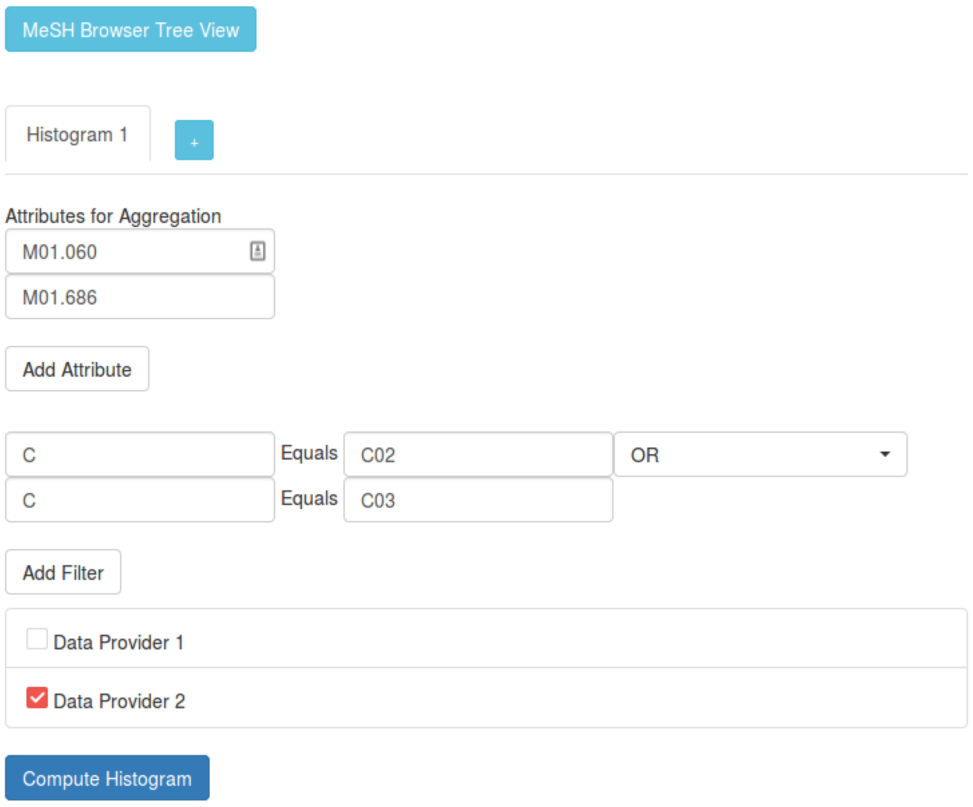


Figure 16: 2D Histogram for “Patient Age” & “Heart Rate” for $\beta = 5$ for each dimension

Secure Aggregation for the MeSH dataset: After selecting the secure aggregation for the MeSH (categorical) dataset, the researcher has to choose attributes, filters and data-providers for the privacy computation. For instance, in figure 17 the user has specified a 2-dimensional histogram for attributes “Age Groups [M01.060]” and “Population Groups [M01.686]”, filtering the “Diseases [C]” attribute in “Virus Diseases [C02]” or “Parasitic Diseases [C03]”.



Attributes for Aggregation

M01.060

M01.686

C	Equals	C02	OR
C	Equals	C03	

☐ Data Provider 1

☑ Data Provider 2

Figure 17: Creating a histogram on the MeSH dataset

Secure Classification for the CVI dataset: The UI for secure classification on the CVI dataset is similar to the secure aggregation, however it has two extra entries, one for the *class-attribute* and one for the classification algorithm (currently ID3 or C4.5).

Secure Classification for the MeSH dataset: Finally, the UI for secure classification on the MeSH dataset is similar to the secure aggregation one, having two extra entries as well.

More screenshots of the User Interface, such as those for secure decision tree creation, are included in appendix B.

7.5 Communication

We employ encryption for data in transit using Transport Layer Security (TLS). All communication through the *coordinator* uses Hypertext Transfer Protocol Secure (HTTPS), with a certificate issued from Let's Encrypt[13]. Thus, all secure computation requests as well as the private-computed results are transferred encrypted. Also, since HTTPS provides authentication of the accessed website, the users know that they communicate with the correct server.

[13]https://letsencrypt.org/

8. DATASETS

As we have stated in section 6.5, the input data can have many different types, since our system can serve a wide variety of applications. We have separated the data in two broad categories – categorical and continuous, therefore our algorithms are also logically separated for those two different kinds of data.

The second reason that we separated our algorithms in those two types, was that we experimented with datasets both types. In medical research it is common to have standard datasets with continuous exact values corresponding to a set of attributes, but it is also common for the datasets to be semantically annotated. For instance a dataset of the first form could have a column corresponding to attribute *Height (cm)* with values including $146.84, 139.35, 189.00, 182.68, 160.19, 138.66, 173.06$ etc. On the other hand, a dataset of the second time would have normalized values including $Tall, Average, Short$ etc. The synthetic datasets we had available for experimentation were the following.

CVI Dataset: Cardiovascular disease is a class of diseases that involve the heart or blood vessels. Cardiovascular disease includes coronary artery diseases (CAD) such as angina and myocardial infarction (commonly known as a heart attack). This dataset contains CardioVascular Imaging (CVI) information, which are represented as numerical values – not normalized. The format of this dataset is the standard tabular format, we find in CSV files or in tables of relational databases. An (incomplete) example can be found in table 3.

Table 3: CVI dataset

Heart rate	Height (cm)	Weight (kg)	LVEDV (ml)	...
90	146.84	61.94	118.36	...
82	139.35	41.51	133.39	...
⋮	⋮	⋮	⋮	⋮

MeSH Dataset: MeSH[14] provides a hierarchically-organized[15] terminology for indexing and cataloging of biomedical information such as MEDLINE/PUBmed and other United States National Library of Medicine (NLM) databases. Created and updated by the NLM, it is used by articles databases and by NLM's catalog of book holdings. This dataset is based on the MeSH tree structure. MeSH terms are represented as normalized values; this means that for instance attributes like Age, are separated into groups (for instance Child, Adult, etc). This dataset contains semantically annotated patient data. The layout of this dataset is quite different from the ordinary tabular data format, it follows the DATS format. DATS is the underlying model powering metadata ingestion, indexing and searches in DataMed, a NIH (National Institute of Health - US) funded project that aims to represent for biomedical datasets what PubMed (https://www.ncbi.nlm.nih.gov/pubmed) is for the biomedical literature. The DATS model is designed around the *Dataset* element as shown in figure 18.

[14]https://meshb.nlm.nih.gov/

[15]https://meshb.nlm.nih.gov/treeView

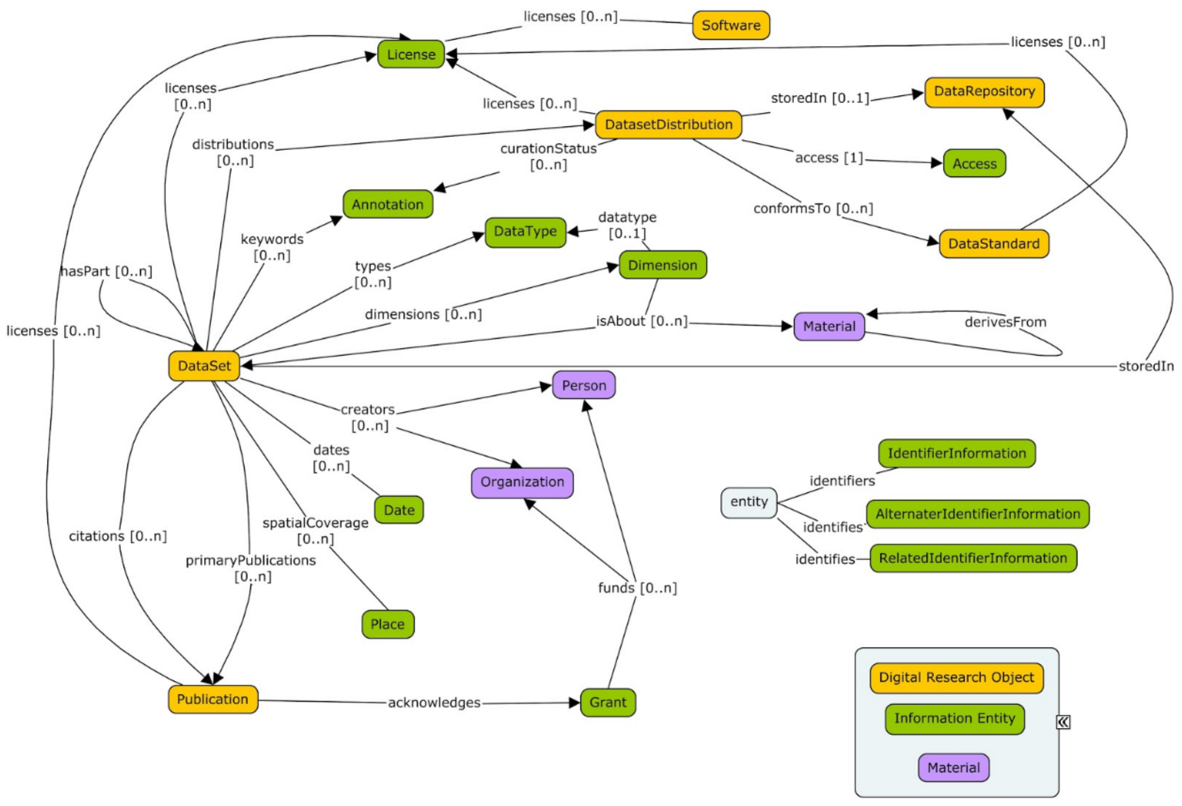


Figure 18: DATS model

Among other metadata, the DATS model contains the *keywords* element which is a set consisting of (in our case) MeSH codes. As a result, our dataset is comprised of a JSON object for each patient. This JSON object contains a *keywords* element with a list of MeSH codes describing that patient. An example can be seen in Code 5.

```
{
    "keywords": [
        {
            "value": "Women",
            "valueIRI": "https://meshb.nlm.nih.gov/record/ui?ui=D014930"
        },
        {
            "value": "Adult",
            "valueIRI": "https://meshb.nlm.nih.gov/record/ui?ui=D000328"
        },
        {
            "value": "African Americans",
            "valueIRI": "https://meshb.nlm.nih.gov/record/ui?ui=D001741"
        },
        .
        .
        .
    ],
    .
    .
    .
}
```

Code 5: Example of a patient JSON object with MeSH codes

The problem with this approach is that the information about a patient is “flat” as it does not follow the *attribute-value* model, it is just a set of MeSH terms. In order to import it in our private secret-shared database that resides in the SMPC cluster, we need to transform this kind of data to a tabular structure. For that reason we developed a procedure that performs this conversion at the time of secure importing. This procedure considers the importing attributes as the “column”, and its direct (1 level down) child that is a parent of the found keyword as the “value”.

To demonstrate how this conversion works, considered the following (incomplete) hierarchy of MeSH terms.

```
Anatomy [A]
├── Organisms [B]
├── Diseases [C]
│   ├── Bacterial Infections and Mycoses [C01]
│   ├── Virus Diseases [C02]
│   ├── Parasitic Diseases [C03]
│   ├── Neoplasms [C04]
│   ├── Musculoskeletal Diseases [C05]
│   ├── Digestive System Diseases [C06]
│   ├── Stomatognathic Diseases [C07]
│   │   ├── Ankyloglossia [C07.160]
│   │   ├── Jaw Diseases [C07.320]
│   │   ├── Mouth Diseases [C07.465]
│   │   ├── Pharyngeal Diseases [C07.550]
│   │   ├── Stomatognathic System Abnormalities [C07.650]
│   │   ├── Temporomandibular Joint Disorders [C07.678]
│   │   │   └── Temporomandibular Joint Dysfunction Syndrome [C07.678.949]
│   │   └── Tooth Diseases [C07.793]
│   └── ...
└── ...
```

Let’s also assume that the JSON object of a patient contains the keyword *Temporomandibular Joint Dysfunction Syndrome [C07.678.949]* and that the chosen attribute for importing is *Diseases [C]*. In this case we will take *Diseases [C]* as the “column” and *Stomatognathic Diseases [C07]* as the “value”, since it is the direct “child” of the chosen attribute, which is a “parent” of *Temporomandibular Joint Dysfunction Syndrome [C07.678.949]* found in the patient JSON object. The resulting (partial) table can be seen in table 4. If the chosen attribute for importing was *Stomatognathic Diseases [C07]* instead of *Diseases [C]*, then the “column” would be *Stomatognathic Diseases [C07]*, and the “value” would be *Temporomandibular Joint Disorders [C07.678]*. The resulting (partial) table can be seen in table 5.

Table 4: Importing of Diseases [C]

Diseases
Stomatognathic Diseases
⋮

Table 5: Importing of Stomatognathic Diseases [C07]

Stomatognathic Diseases
Temporomandibular Joint Disorders
⋮

9. EXPERIMENTAL EVALUATION

We measured the runtime performance of our privacy-preserving algorithms using the Sharemind secure computing platform. We instantiated all private aggregators and decision tree classifiers using the SecreC programming language [33] that Sharemind provides. In Sharemind, the number of the computing parties are restricted by the SMPC protocol that is been used. Our experiments use the *pd_shared3p* security protocol, which utilizes three nodes in the private domain.

Below, we present the experimental evaluation timings of the developed privacy-preserving histograms and decision trees. Our goal is to provide a concise and holistic view of our experiments[16], by carefully select the most descriptive diagrams.

Experimental Setup: All experiments were performed on three 64-bit virtual machines (VM), each running Ubuntu 18.04. The three VMs share an Intel Xeon E5-2670 (v2) at 2.50 GHz. Each VM utilizes 6 cores and 8 GB of memory.

9.1 Histograms

The first experimental timing results are about the privacy-preserving histogram computations. In figure 19 we show the scaling in execution time of histograms on numerical data. In figure 20 we exhibit the respective measurements for histograms on categorical data.

Note: both x and y axes are log-scaled.

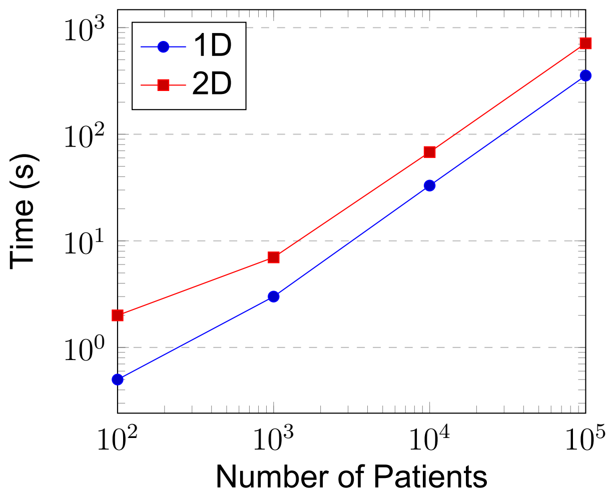


Figure 19: Numerical histograms timings

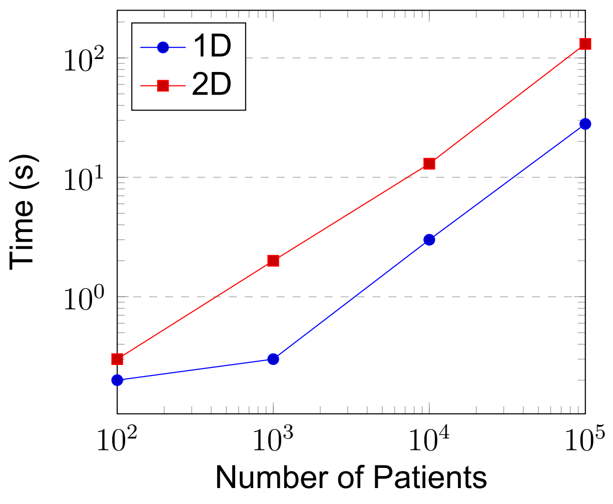


Figure 20: Categorical histograms timings

One can observe that the histograms on categorical data perform better than their numerical data equivalents, especially when applied over bigger datasets. This is due to the extra computation that needs to be done in the case of numerical data, in order to determine the corresponding histogram cell for each data tuple. Moreover, the privacy-preserving histograms scale linearly with the dataset size, regardless the type of it.

[16]All time measurements presented in this section correspond to the sole privacy computation. The importing timings were roughly 1 second, even for the largest dataset, thus we did not take it into account in the graphs.

The chart below (figure 21) depicts the timings of histograms on numerical data (such as figure 19), but also with the application of some filters/constraints. We can clearly see that as the dataset size grows, the number of filters does not make much of a difference in the total algorithm's execution time.

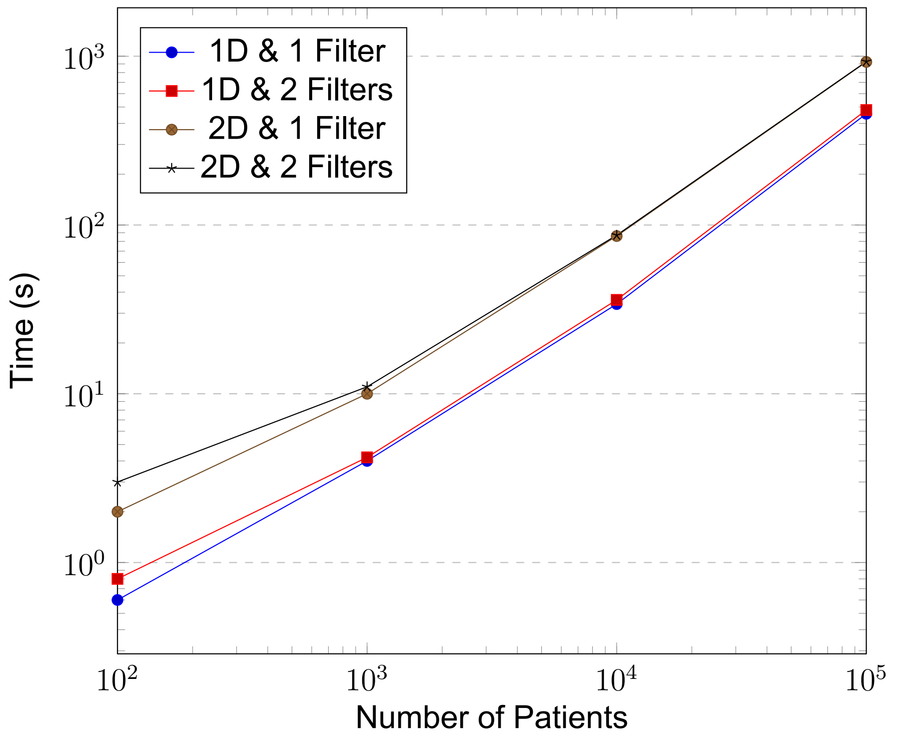


Figure 21: Numerical histograms with filters timings

9.2 Decision Trees

Consecutively, we measured the privacy-preserving classification algorithms. Figure 22 portrays how the ID3 algorithm scales as the dataset size grows, for a constant number (3) of classification attributes ("Heart rate", "Height (cm)" and "Patient Age"). C4.5 algorithm is more time consuming than ID3. For reference, the training of the classification algorithm for 50 patients took about $13 * 10^3$ seconds, while for 100 patients the algorithm executed for $41 * 10^3$ seconds[17].

Note: both x and y axes are log-scaled.

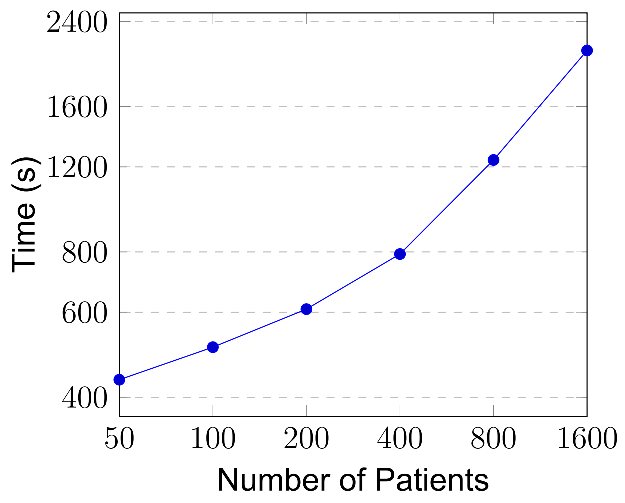


Figure 22: ID3 decision tree classifier timings with variable patients for 3 attributes

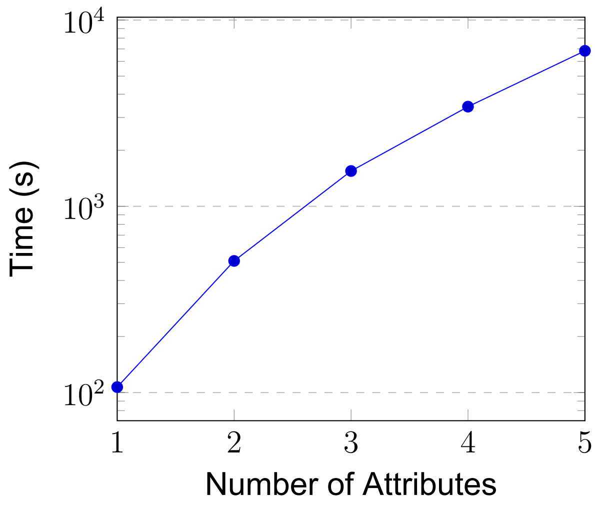


Figure 23: ID3 decision tree classifier timings with variable number of attributes

Finally, an overview of the runtime performance of the ID3 algorithm for a constant number of patients and variable number of classification attributes is presented in figure 23.
Note: y axis is log-scaled.

As we observe from our experiments, the number of attributes used for classification is a dominating factor in the algorithm's performance.

The execution times for the privacy-preserving creation of the decision trees is orders of magnitude greater than those of equivalent algorithms working on unencrypted data. However, this is the training part of the classification algorithm which is performed offline. For this reason, it does not affect the user experience, since training can happen non-interactively at predefined periods of time (*e.g.* every night or once a week).

[17]For many classification attributes C4.5 has an unbearable runtime overhead, and for that reason we do not provide a corresponding chart as in the ID3 algorithm

10. CONCLUSIONS & FUTURE WORK

In this thesis we have developed some fundamental privacy-preserving algorithms as well as an end-to-end infrastructure for computing privacy-preserving analytics. There are numerous use-cases that require from different parties to jointly compute some function – such as medical research from different hospitals, but due to data-privacy limitations are avoided. Our end-to-end system utilizes encrypted computation and can constitute the building blocks on top of which meaningful privacy computation can take place, as we have demonstrated with secure aggregators and secure decision tree classifiers. In our use-case, we focused on medical research, however, in our point of view, this system can serve a wide range of case-studies.

Every coin has two sides, so the same applies to encrypted computation as well. Utilizing the homomorphic properties of secret-shares enables a whole new dimension on computing that respects data privacy, but comes at a high cost, as we observed in the decision tree classifiers. Although such sophisticated algorithms seem impractical at first sight due to their high execution timings, one should understand that there is no other way respecting data-privacy to obtain such results. Moreover, simpler algorithms – suchlike aggregators – scale linearly to the size of dataset, which casts them practical for everyday use.

Future work will explore more elaborate algorithms, such as stochastic gradient descent [56] and/or deep neural networks, that will provide useful analytics results in order to enhance our framework. Privacy-preserving clustering algorithms [32, 34, 61] are also a topic worth exploring.

In addition, we aim to refine the ID3 and C4.5 algorithms, taking more advantage of SIMD instructions, with the purpose of optimizing their execution times. Although the classification training is performed in an offline phase, more efficient tree-classifiers would improve the overall platform.

Privacy-preserving algorithms are inherently more time consuming than their textbook equivalents. Much work has to be done in the underlying encrypted architectures in order to keep up with conventional computers. Thus, our goal is to explore other SMPC frameworks than Sharemind, such as FRESCO (a FRamework for Efficient and Secure COmputation) [3,17], SPDZ [21,22], Obliv-C [65] and ObliVM [44]. Trying different MPC frameworks opens many possibilities, since they support different security protocols, different number of computing nodes, and in general different security configurations. Moreover, from the programming side of view, each framework provides its own toolkit and API.

Another interesting research topic is to incorporate a blockchain technology such as Hyperledger [16] for enhancing the transparency, traceability, non-repudation, data provenance, and auditability of our system's computations [68], since all actions are immutable. In more detail, a distributed ledger using smart contracts can act as the controller of the system, orchestrating all actions with the use of a Zero Knowledge Verifiable Computation scheme [11] where data processors are enforced to produce a proof of correctness of computation without revealing the dataset itself. This records the fact that correct processing has taken place without disclosing any information about the data. Finally, every private computation request and result could be logged for transparency and future use.

ABBREVIATIONS - ACRONYMS

2PC	Two-party Computation (Millionaire’s Problem)
BDVA	Big Data Value Association
BGV	Brakerski – Gentry – Vaikuntanathan
BMI	Body Mass Index
CAD	Coronary Artery Diseases
CSV	Comma-separated Values
CVI	CardioVascular Imaging
DATS	DAta Tag Suite
Dec	Decryption Algorithm
DH	Diffie Hellman
DSL	Domain Specific Language
Enc	Encryption Algorithm
EU	European Union
FHE	Fully Homomorphic Encryption
FRESCO	FRamework for Efficient and Secure COmputation
GDPR	General Data Protection Regulation
GPS	Global Positioning System
HE	Homomorphic Encryption
HElib	Homomorphic Encryption Library, implementing the BGV scheme
HTTPS	Hypertext Transfer Protocol Secure
IoT	Internet of Things
JSON	JavaScript Object Notation
MHMD	My Health My Data
MPC	Secure Multi-party Computation
NIH	National Institute of Health
NLM	National Library of Medicine
OT	Oblivious Transfer
PHE	Partially Homomorphic Encryption
PKE	Public Key Encryption

$PrivKey$	Private Key
$PubKey$	Public Key
PPDM	Privacy Preserving Data Mining
REST	Representational State Transfer
RSA	Rivest – Shamir – Adleman
SFE	Secure Function Evaluation
SIMD	Single Instruction Multiple Data
SMPC	Secure Multi-party Computation
SODA	Scalable Oblivious Data Analytics
SPDZ	Smart, Pastro, Damgård and Zakarias. SMPC protocol
SVM	Support Vector Machines
TFHE	Fast Fully Homomorphic Encryption over the Torus
TLS	Transport Layer Security
UID	Unique Identifier
VM	Virtual Machine

APPENDIX A. RESTFUL APIS

A.1 Coordinator's RESTful API

A.1.1

```
POST   /smpc/histogram/numerical
```

This POST request initiates a secure computation for numerical/continuous values dataset.

Request: Through the request's body, one can specify all the desired properties of the histograms to be computed. These include the attributes on which the histogram will be built (names and cells – β factor), the datasources from which data will be used, and a set of filters/conditions that should hold for each data tuple taken into consideration when building the histogram.

The request's body is a `JSON` object with the following properties:

- `attributes`: *required* A list of JSON objects, corresponding to the attributes on which this histogram will be built. These JSON objects have the following keys:
  - `name`: *required* The name of the attribute (*string*).
  - `cells`: *required* The number of histogram cells (β factor) to be created for this attribute (*positive integer*).
- `datasources`: *optional* A list of the datasources (*strings*) from which the histogram will be computed. If this key is left empty or not specified, all available datasources will be used.
- `filters`: *optional* A JSON object containing a boolean operator and a list of filters/conditions that should be met for each data tuple considered in the secure histogram computation. If this field is left blank or not specified, all data tuples will be used for the computation. The object has the following keys:
  - `operator`: *required* The boolean operator (*string*) that will be applied between all the specified conditions that follow. One of `[AND, OR, XOR]`. In the case of multiple conditions, the operator is left-associative.
  - `conditions`: *required* The list of conditions that should be met by each data tuple in the computation. Each condition is represented as a JSON object with the following keys:
    - `attribute`: *required* The name of the attribute (*string*).
    - `operator`: *required* The condition's operator (*string*). One of `[>, <, =]`.
    - `value`: *required* The attribute's value (*string*).

Example (without `datasources` or `filters`):

```
{
   "attributes": [
       {
           "name": "Patient Age",
           "cells": "5"
       },
       {
           "name": "Height (cm)",
           "cells": "3"
       }
   ]
}
```

Code 6: Example 1 of /smpc/histogram/numerical POST request body

Example (with `datasources` and `filters`):

```
{
   "attributes": [
       {
           "name": "Patient Age",
           "cells": "5"
       },
       {
           "name": "Height (cm)",
           "cells": "3"
       }
   ],
   "datasources": [
       "HospitalA",
       "HospitalB"
   ],
   "filters": {
       "operator": "AND",
       "conditions": [
           {
               "attribute": "Patient Age",
               "operator": ">",
               "value": "45"
           },
           {
               "attribute": "Weight (kg)",
               "operator": "<",
               "value": "90"
           }
       ]
   }
}
```

Code 7: Example 2 of /smpc/histogram/numerical POST request body

Response: The secure histogram computation is a potentially long running operation. For that reason the server's response to such a request is always `HTTP/1.1 202 Accepted` in case of correct request, along with a location in which one should periodically poll for the computation's status and/or result. An example response can be found below.

`Status: 202 Accepted`

```
{
    "location": "/smpc/queue?request=b171-3f4ade123374"
}
```

Code 8: Example 1 of /smpc/histogram/numerical response

In case of a malformed request the server responds with `HTTP/1.1 400 Bad Request` as seen below.

`Status: 400 Bad Request`

```
Bad Request
```

Code 9: Example 2 of /smpc/histogram/numerical response

A.1.2 `POST /smpc/histogram/categorical`

This POST request initiates a secure computation for categorical values dataset (MeSH). Through the request's body, one can specify the desired MeSH terms. The values the count of which will be computed are the children of the specified MeSH term from the MeSH ontology.

For example, if a user specifies that he/she wants the counts for the MeSH term "Age Groups" [M01.060], he/she will get four counts back corresponding to the four children of "Age Groups", namely "Adolescent" [M01.060.057], "Adult" [M01.060.116], "Child" [M01. 060.406] and "Infant" [M01.060.703]. If the specified MeSH terms are two, then the resulting counts will correspond to tuples of MeSH labels. If the specified MeSH terms are three, the result will be triples, etc.

Request: The request's body is a `JSON` object with the following properties:

`attributes`: *required* A list of MeSH terms, corresponding to the attributes on which this histogram will be built.

`datasources`: *optional* A list of the datasources (*strings*) from which the histogram will be computed. If this key is left empty or not specified, all available datasources will be used.

`filters`: *optional* A JSON object containing a boolean operator and a list of filters/conditions that should be met for each data tuple considered in the secure histogram computation. If this field is left blank or not specified, all data tuples will be used for the computation. The object has the following keys:

`operator`: *required* The boolean operator (*string*) that will be applied between all the specified conditions that follow. One of `[AND, OR, XOR]`. In the case of multiple conditions, the operator is left-associative.

`conditions`: *required* The list of conditions that should be met by each data tuple in the computation. Note that the only filter that can be applied in categorical values is the equality checking. Each condition is represented as a JSON object with the following keys:

- **`attribute`:** *required* The name of the attribute (*string*).
- **`value`:** *required* The value that the selected attribute should be equal with (*string*).

Example (without `datasources` or `filters`):

```
{
    "attributes": [
        "M01.060",
        "M01.686.508"
    ]
}
```

Code 10: Example 1 of /smpc/histogram/categorical POST request body

Example (with `datasources` and `filters`):

```
{
    "attributes": [
        "M01.060",
        "M01.686.508"
    ],
    "datasources": [
        "HospitalA",
        "HospitalB"
    ],
    "filters": {
        "operator": "OR",
        "conditions": [
            {
                "attribute": "C14",
                "operator": "=",
                "value": "C14.280"
            },
            {
                "attribute": "C12",
                "operator": "=",
                "value": "C12.294"
            }
        ]
    }
}
```

Code 11: Example 2 of /smpc/histogram/categorical POST request body

Response: The server's response for the `/smpc/histogram/categorical` POST request is identical to that of `/smpc/histogram/numerical`. The server returns `HTTP / 1.1 202 Accepted` or `HTTP/1.1 400 Bad Request`. Examples for both cases can be seen in code snippets 8 and 9.

A.1.3 **POST** `/smpc/decisionTree`

Finally, this POST request is for computing a decision tree from either dataset. The implemented classifiers are ID3 and C4.5, and one should specify which one to use in the request body along with the desired attributes. Moreover, he/she should specify the class-attribute, according to which the decision tree will occur.

Request: The request's body is a `JSON` object with the following properties:

`attributes`: *required* A list of MeSH terms, corresponding to the attributes on which this histogram will be built. These JSON objects have the following keys:

- **`name`:** *required* The name of the attribute (*string*).
- **`cells`:** *optional* The number of histogram cells (β factor) to be created for this attribute (*positive integer*).[18]

[18]In case, where C4.5 is selected as `classifier`, or the attribute is categorical, the `cells` object is omitted.

`classifier:`	*optional* The name of the decision tree classifier to be used. Possible values are {`ID3, C4.5`}. If this key is left empty or not specified, the `ID3` classifier will be used by default.
`class_attribute:`	*required* The attribute on which the classification will occur.
	`name:` *required* The name of the class attribute (*string*).
	`cells:` *optional* The number of histogram cells (β factor) to be created for the class attribute (*positive integer*).[19]
`datasources:`	*optional* A list of the datasources (*strings*) from which the histogram will be computed. If this key is left empty or not specified, all available datasources will be used.

Example (using ID3 as the classifier for categorical values):

```
{
    "attributes": [
        {
            "name": "C14"
        },
        {
            "name": "C12"
        },
        {
            "name": "M01.686"
        }
    ],
    "classifier": "ID3",
    "class_attribute": {
        "name": "M01.060"
    },
    "datasources": [
        "HospitalA",
        "HospitalB"
    ]
}
```

Code 12: Example 1 of /smpc/decisionTree POST request body for categorical values

[19]In case, where the `class_attribute` is categorical, the `cells` object is omitted.

Example (using ID3 as the classifier for numerical values):

```json
{
    "attributes": [
        {
            "name": "Height (cm)",
            "cells": "4"
        },
        {
            "name": "Weight (kg)",
            "cells": "5"
        },
        {
            "name": "Heart rate",
            "cells": "4"
        }
    ],
    "classifier": "ID3",
    "class_attribute": {
        "name": "Patient Age",
        "cells": "4"
    },
    "datasources": [
        "HospitalA",
        "HospitalB"
    ]
}
```

Code 13: Example 2 of /smpc/decisionTree POST request body for numerical values

Example (using C4.5 as the classifier for numerical values):

```json
{
    "attributes": [
        {
            "name": "Height (cm)"
        },
        {
            "name": "Weight (kg)"
        },
        {
            "name": "Heart rate"
        }
    ],
    "classifier": "C4.5",
    "class_attribute": {
        "name": "Patient Age",
        "cells": "4"
    },
    "datasources": [
        "HospitalA",
        "HospitalB"
    ]
}
```

Code 14: Example 3 of /smpc/decisionTree POST request body for categorical values

Response: The server's response for the `/smpc/decisionTree` POST request is the same as the ones described above. The server returns `HTTP/1.1 202 Accepted` or `HTTP/1.1 400 Bad Request`. Examples for both cases can be seen in code snippets 8 and 9.

A.1.4 `GET /smpc/queue`

The `/smpc/queue` GET request is for checking the status and/or result of a specified computation request (the aforementioned POST requests). The status of an ongoing computation request can be accessed through the /smpc/queue GET request by specifying its id.

Request: The only parameter this GET request accepts is the id of the desired computation request, as shown below.

```
/smpc/queue?request=81c2-b9c0e5589ec0
```

Code 15: Example /smpc/queue GET request

Response: In case of a correct request, the server responds with `HTTP/1.1 200 OK` and the response body is a JSON object containing the specified computation's status, and possibly its current step or result which is a JSON object too. The server's response body has the following structure.

`status`: *required* A string indicating the computation's status. One of [`running`, `succeeded`, `failed`, `notstarted`]

`step`: *optional* A string indicating the current step of the computation. This is present in case that the computation is in the running state.

`result`: *optional* A JSON object with the computation's result in case its status is succeeded. The JSON object contains a single key namely data which contains computation result.

In case of private histogram computations the result is in the form of tuples (label, value) tuples, which are described here.

`label`: A string, the value name corresponding to that count. Can be a tuple, triple etc. depending on the number of queried Mesh terms.

`value`: An integer, the actual count for that value.

In case of private decision tree computations the result is in the form of a JSON object which has a key for each tree branch, and a value for the corresponding subtree that resides under that branch.

In case of a malformed request the server responds with `HTTP/1.1 400 Bad Request` as in 9

Below you can find example responses for correct `/smpc/queue` GET requests.

Example (successfully computed private categorical histogram):
`Status: 200 OK`

```
{
    "status": "succeeded",
    "result": {
        "data": [
            {
                "value": 78,
                "label": "Infant"
            },
            {
                "value": 63,
                "label": "Adolescent"
            },
            {
                "value": 9936,
                "label": "Adult"
            },
            {
                "value": 154,
                "label": "Child"
            }
        ]
    }
}
```

Code 16: Example /smpc/queue GET response body for a successful Age Groups (M01.060) histogram

Example (successfully computed private numerical histogram):
`Status: 200 OK`

```
{
    "status": "succeeded",
    "result": {
        "data": [
            {
                "value": 84,
                "label": "[130.0, 144.0)"
            },
            {
                "value": 314,
                "label": "[144.0, 158.0)"
            },
            {
                "value": 428,
                "label": "[158.0, 172.0)"
            },
            {
                "value": 800,
                "label": "[172.0, 186.0)"
            },
            {
                "value": 374,
                "label": "[186.0, 200.0]"
            }
        ]
    }
}
```

Code 17: Example /smpc/queue GET response body for a successful Height(cm) histogram with $\beta = 5$

Example (successfully computed private decision tree using C4.5):
`Status: 200 OK`

```
{
    "status": "succeeded",
    "result": {
        "data": {
            "Height (cm) <= 173.01": {
                "Weight (kg) > 68.56": {
                    "Heart rate > 81.50": "[18.00, 31.00)",
                    "Heart rate <= 81.50": "[70.00, 83.00)"
                },
                "Weight (kg) <= 68.56": {
                    "Heart rate > 70.50": "[31.00, 44.00)",
                    "Heart rate <= 70.50": "[18.00, 31.00)"
                }
            },
            "Height (cm) > 173.01": {
                "Weight (kg) > 87.70": {
                    "Heart rate > 94.00": "[18.00, 31.00)",
                    "Heart rate <= 94.00": "[31.00, 44.00)"
                },
                "Weight (kg) <= 87.70": {
                    "Heart rate <= 65.50": "[31.00, 44.00)",
                    "Heart rate > 65.50": "[57.00, 70.00)"
                }
            }
        }
    }
}
```

Code 18: Example /smpc/queue GET response body for a successfully computed private decision tree using C4.5 on Patient Age from categorical values

Example (successfully computed private decision tree using ID3):
`Status: 200 OK`

```
{
    "status": "succeeded",
    "result": {
        "data": {
            "Height (cm) == [153.34, 176.67)": {
                "Heart rate == [95.33, 123.00)": "[34.25, 50.50)",
                "Heart rate == [67.67, 95.33)": "[18.00, 34.25)",
                "Heart rate == [40.00, 67.67)": "[18.00, 34.25)"
            },
            "Height (cm) == [130.01, 153.34)": {
                "Heart rate == [95.33, 123.00)": "[34.25, 50.50)",
                "Heart rate == [67.67, 95.33)": "[34.25, 50.50)",
                "Heart rate == [40.00, 67.67)": "[18.00, 34.25)"
            },
            "Height (cm) == [176.67, 200.00)": {
                "Heart rate == [95.33, 123.00)": "[18.00, 34.25)",
                "Heart rate == [67.67, 95.33)": "[34.25, 50.50)",
                "Heart rate == [40.00, 67.67)": "[34.25, 50.50)"
            }
        }
    }
}
```

Code 19: Example /smpc/queue GET response body for a successfully computed private decision tree using ID3 on Patient Age from categorical values

Example (ongoing computation):
`Status: 200 OK`

```
{
    "status": "running",
    "step": "Securely importing data"
}
```

Code 20: Example /smpc/queue GET response body for an ongoing computation during secure import

Example (failed computation):
`Status: 200 OK`

```
{
    "status": "failed"
}
```

Code 21: Example /smpc/queue GET response body for a failed computation

A.2 Data Providers' RESTful API

A.2.1

```
POST   /smpc/import/numerical
```

This POST request initiates a secure import for numerical/continuous values dataset.

Request: Through the request's body, one can specify the desired attribute names (*i.e.* columns of the dataset) to be imported. For instance, an example for the CVI dataset (see section 8) is shown in Code 22, in which a user specifies that he/she wants to compute private analytics for attributes "Patient Age" and "Heart rate".

The request's body is a JSON object of the following form.

`attributes`: *required* A list of attributes (*strings*) on which this histogram will be built.

`datasource`: *required* The name for the table of the SPMC cluster's encrypted database in which the data will be imported.

Example (importing of some attribute from the CVI dataset):

```
{
    "attributes": [
        "Patient Age",
        "Heart rate"
    ],
    "datasource": "HospitalA"
}
```

Code 22: Example /smpc/import/numerical POST request body

Response: The server responds with `HTTP/1.1 200 OK` upon successful importing or with `HTTP/1.1 400 Bad Request` upon failure.

A.2.2

```
POST   /smpc/import/categorical
```

This POST request initiates a secure import from datasets with categorical values dataset (MeSH).

Request: Through the request's body, one can specify the desired attribute names of the dataset to import. The request for importing attributes from th MeSH dataset is similar to the the numerical one, in Code 22, however in attributes should be specified by their MeSH codes, due to name ambiguity.

The request's body is a JSON object of the following form.

`attributes`: *required* A list of attributes (MeSH codes) on which this histogram will be built.

`datasource`: *required* The name for the table of the SPMC cluster's encrypted database in which the data will be imported.

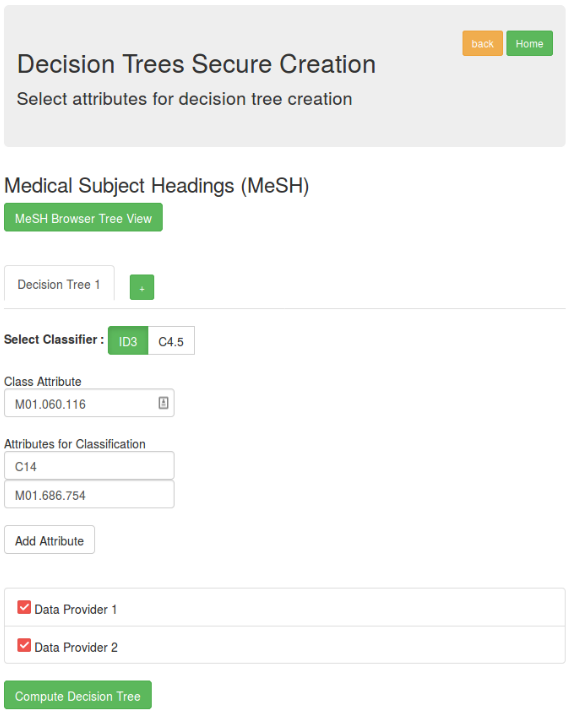
Example (importing of some attribute from the CVI dataset):

```
{
    "attributes": [
        "M01.060",
        "C14.280",
        "C14.240"
    ],
    "datasource": "HospitalA"
}
```

Code 23: Example /smpc/import/categorical POST request body

Response: The server responds with `HTTP/1.1 200 OK` upon successful importing or with `HTTP/1.1 400 Bad Request` upon failure.

APPENDIX B. USER INTERFACE SCREENSHOTS


Decision Trees Secure Creation

Select attributes for decision tree creation

Medical Subject Headings (MeSH)

Decision Tree 1

Select Classifier : ID3 C4.5

Class Attribute: M01.060.116

Attributes for Classification: C14, M01.686.754

☑ Data Provider 1

☑ Data Provider 2

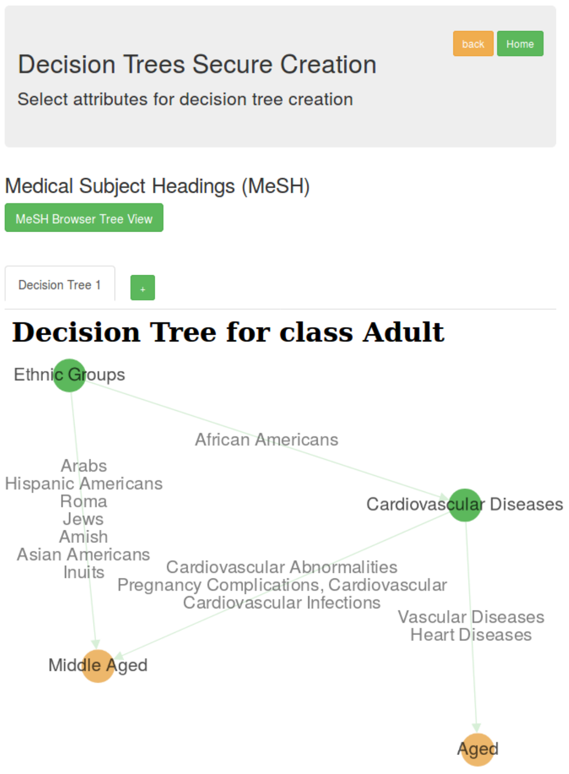
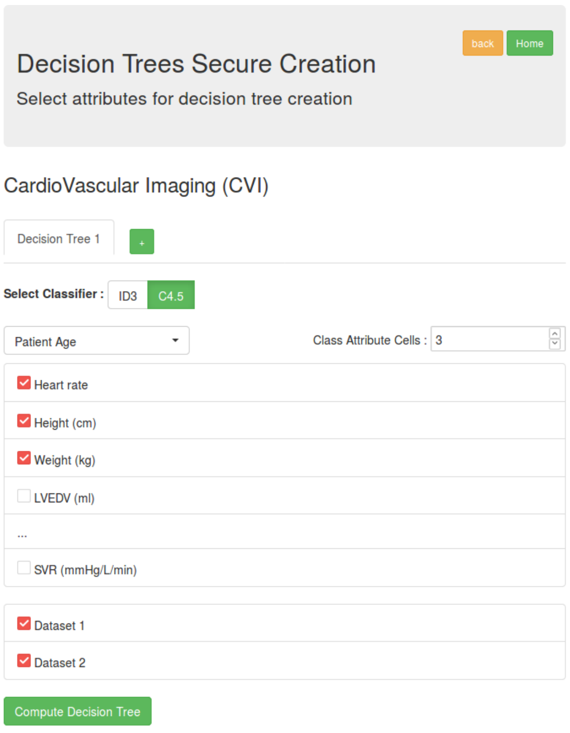

back
Home
Decision Trees Secure Creation
Select attributes for decision tree creation
Medical Subject Headings (MeSH)
MeSH Browser Tree View
Decision Tree 1
+
Decision Tree for class Adult
Ethnic Groups
African Americans
Arabs
Hispanic Americans
Roma
Jews
Amish
Asian Americans
Inuits
Cardiovascular Diseases
Cardiovascular Abnormalities
Pregnancy Complications, Cardiovascular
Cardiovascular Infections
Vascular Diseases
Heart Diseases
Middle Aged
Aged

Figure 24: Creating a decision tree on the MeSH dataset

Figure 25: ID3 decision tree for the request shown in Figure 24


Decision Trees Secure Creation

Select attributes for decision tree creation

CardioVascular Imaging (CVI)

Decision Tree 1

Select Classifier : ID3 C4.5

Patient Age   Class Attribute Cells : 3

☑ Heart rate

☑ Height (cm)

☑ Weight (kg)

☐ LVEDV (ml)

...

☐ SVR (mmHg/L/min)

☑ Dataset 1

☑ Dataset 2

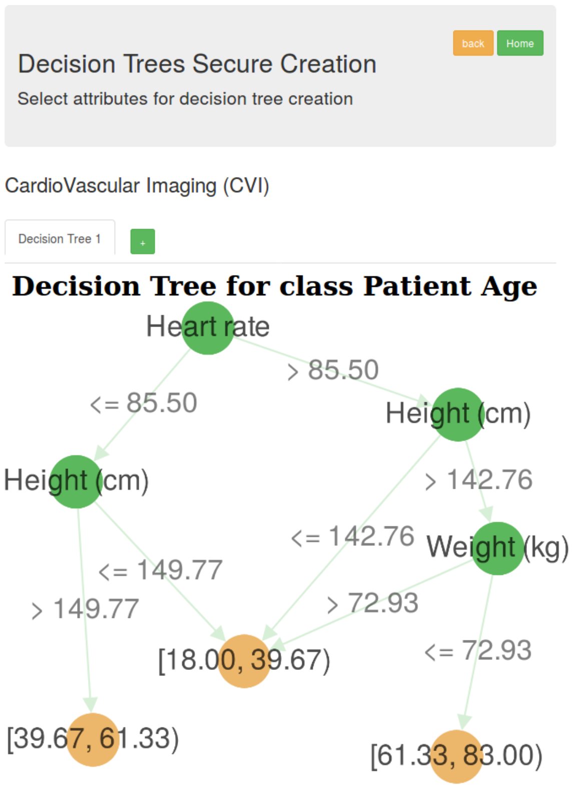


Figure 26: Creating a decision tree on the CVI dataset

Figure 27: C4.5 decision tree for the request shown in Figure 26

APPENDIX C. SOURCE CODE UNDER MIT LICENCE

APPENDIX D. CREATIVE COMMONS ATTRIBUTION 4.0 INTERNATIONAL PUBLIC LICENSE

By exercising the Licensed Rights (defined below), You accept and agree to be bound by the terms and conditions of this Creative Commons Attribution 4.0 International Public License (“Public License”). To the extent this Public License may be interpreted as a contract, You are granted the Licensed Rights in consideration of Your acceptance of these terms and conditions, and the Licensor grants You such rights in consideration of benefits the Licensor receives from making the Licensed Material available under these terms and conditions.

D.1 Section 1 – Definitions.

a. **Adapted Material** means material subject to Copyright and Similar Rights that is derived from or based upon the Licensed Material and in which the Licensed Material is translated, altered, arranged, transformed, or otherwise modified in a manner requiring permission under the Copyright and Similar Rights held by the Licensor. For purposes of this Public License, where the Licensed Material is a musical work, performance, or sound recording, Adapted Material is always produced where the Licensed Material is synched in timed relation with a moving image.

b. **Adapter’s License** means the license You apply to Your Copyright and Similar Rights in Your contributions to Adapted Material in accordance with the terms and conditions of this Public License.

c. **Copyright and Similar Rights** means copyright and/or similar rights closely related to copyright including, without limitation, performance, broadcast, sound recording, and Sui Generis Database Rights, without regard to how the rights are labeled or categorized. For purposes of this Public License, the rights specified in Section 2(b)(1)-(2) are not Copyright and Similar Rights.

d. **Effective Technological Measures** means those measures that, in the absence of proper authority, may not be circumvented under laws fulfilling obligations under Article 11 of the WIPO Copyright Treaty adopted on December 20, 1996, and/or similar international agreements.

e. **Exceptions and Limitations** means fair use, fair dealing, and/or any other exception or limitation to Copyright and Similar Rights that applies to Your use of the Licensed Material.

f. **Licensed Material** means the artistic or literary work, database, or other material to which the Licensor applied this Public License.

g. **Licensed Rights** means the rights granted to You subject to the terms and conditions of this Public License, which are limited to all Copyright and Similar Rights that apply to Your use of the Licensed Material and that the Licensor has authority to license.

h. **Licensor** means the individual(s) or entity(ies) granting rights under this Public License.

D.2 Section 2 – Scope.

b. **Other rights.**

1. Moral rights, such as the right of integrity, are not licensed under this Public License, nor are publicity, privacy, and/or other similar personality rights; however, to the extent possible, the Licensor waives and/or agrees not to assert any such rights held by the Licensor to the limited extent necessary to allow You to exercise the Licensed Rights, but not otherwise.
2. Patent and trademark rights are not licensed under this Public License.
3. To the extent possible, the Licensor waives any right to collect royalties from You for the exercise of the Licensed Rights, whether directly or through a collecting society under any voluntary or waivable statutory or compulsory licensing scheme. In all other cases the Licensor expressly reserves any right to collect such royalties.

D.3 Section 3 – License Conditions.

Your exercise of the Licensed Rights is expressly made subject to the following conditions.

a. **Attribution.**

1. If You Share the Licensed Material (including in modified form), You must:
   A. retain the following if it is supplied by the Licensor with the Licensed Material:
      i. identification of the creator(s) of the Licensed Material and any others designated to receive attribution, in any reasonable manner requested by the Licensor (including by pseudonym if designated); a copyright notice;
      ii. a notice that refers to this Public License;
      iii. a notice that refers to the disclaimer of warranties;
      iv. a URI or hyperlink to the Licensed Material to the extent reasonably practicable;
   B. indicate if You modified the Licensed Material and retain an indication of any previous modifications; and
   C. indicate the Licensed Material is licensed under this Public License, and include the text of, or the URI or hyperlink to, this Public License.
2. You may satisfy the conditions in Section 3(a)(1) in any reasonable manner based on the medium, means, and context in which You Share the Licensed Material. For example, it may be reasonable to satisfy the conditions by providing a URI or hyperlink to a resource that includes the required information.
3. If requested by the Licensor, You must remove any of the information required by Section 3(a)(1)(A) to the extent reasonably practicable.
4. If You Share Adapted Material You produce, the Adapter's License You apply must not prevent recipients of the Adapted Material from complying with this Public License.

D.4 Section 4 – Sui Generis Database Rights.

Where the Licensed Rights include Sui Generis Database Rights that apply to Your use of the Licensed Material:

a. for the avoidance of doubt, Section 2(a)(1) grants You the right to extract, reuse, reproduce, and Share all or a substantial portion of the contents of the database;

b. if You include all or a substantial portion of the database contents in a database in which You have Sui Generis Database Rights, then the database in which You have Sui Generis Database Rights (but not its individual contents) is Adapted Material; and

c. You must comply with the conditions in Section 3(a) if You Share all or a substantial portion of the contents of the database.

For the avoidance of doubt, this Section 4 supplements and does not replace Your obligations under this Public License where the Licensed Rights include other Copyright and Similar Rights.

D.5 Section 5 – Disclaimer of Warranties and Limitation of Liability.

a. Unless otherwise separately undertaken by the Licensor, to the extent possible, the Licensor offers the Licensed Material as-is and as-available, and makes no representations or warranties of any kind concerning the Licensed Material, whether express, implied, statutory, or other. This includes, without limitation, warranties of title, merchantability, fitness for a particular purpose, non-infringement, absence of latent or other defects, accuracy, or the presence or absence of errors, whether or not known or discoverable. Where disclaimers of warranties are not allowed in full or in part, this disclaimer may not apply to You.

b. To the extent possible, in no event will the Licensor be liable to You on any legal theory (including, without limitation, negligence) or otherwise for any direct, special, indirect, incidental, consequential, punitive, exemplary, or other losses, costs, expenses, or damages arising out of this Public License or use of the Licensed Material, even if the Licensor has been advised of the possibility of such losses, costs, expenses, or damages. Where a limitation of liability is not allowed in full or in part, this limitation may not apply to You.

c. The disclaimer of warranties and limitation of liability provided above shall be interpreted in a manner that, to the extent possible, most closely approximates an absolute disclaimer and waiver of all liability.

D.6 Section 6 – Term and Termination.

a. This Public License applies for the term of the Copyright and Similar Rights licensed here. However, if You fail to comply with this Public License, then Your rights under this Public License terminate automatically.

D.7 Section 7 – Other Terms and Conditions.

D.8 Section 8 – Interpretation.

REFERENCES

[1] Ben Adida. Helios: Web-based open-audit voting. In *USENIX security symposium*, volume 17, pages 335–348, 2008.

[2] Rakesh Agrawal and Ramakrishnan Srikant. Privacy-preserving data mining. In *ACM Sigmod Record*, volume 29, pages 439–450. ACM, 2000.

[3] Peter S Nordholt ALX, Nikolaj Volgushev ALX, Prastudy Fauzi AU, Claudio Orlandi AU, Peter Scholl AU, Mark Simkin AU, Meilof Veeningen PHI, Niek Bouman TUE, and Berry Schoenmakers TUE. D1. 1 State of the Art Analysis of MPC Techniques and Frameworks.

[4] BDVA. Big Data Value Association (BDVA), 2018. [Online; accessed 2018].

[5] Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 503–513. ACM, 1990.

[6] Georg T Becker, Francesco Regazzoni, Christof Paar, and Wayne P Burleson. Stealthy dopant-level hardware trojans. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 197–214. Springer, 2013.

[7] Mihir Bellare, Viet Tung Hoang, Sriram Keelveedhi, and Phillip Rogaway. Efficient garbling from a fixed-key blockcipher. In *Security and Privacy (SP), 2013 IEEE Symposium on*, pages 478–492. IEEE, 2013.

[8] Josh Cohen Benaloh. Secret sharing homomorphisms: Keeping shares of a secret secret. In *Conference on the Theory and Application of Cryptographic Techniques*, pages 251–260. Springer, 1986.

[9] Elisa Bertino, Beng Chin Ooi, Yanjiang Yang, and Robert H Deng. Privacy and ownership preserving of outsourced medical data. In *null*, pages 521–532. IEEE, 2005.

[10] GR Blakley and GA Kabatianskii. Linear algebra approach to secret sharing schemes. In *Error Control, Cryptology, and Speech Compression*, pages 33–40. Springer, 1994.

[11] Manuel Blum, Alfredo De Santis, Silvio Micali, and Giuseppe Persiano. Noninteractive zero-knowledge. *SIAM Journal on Computing*, 20(6):1084–1118, 1991.

[12] Dan Bogdanov. *Sharemind: programmable secure computations with practical applications*. PhD thesis, 2013.

[13] Dan Bogdanov, Sven Laur, and Jan Willemson. Sharemind: A framework for fast privacy-preserving computations. In *European Symposium on Research in Computer Security*, pages 192–206. Springer, 2008.

[14] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory (TOCT)*, 6(3):13, 2014.

[15] Michael Brenner, Jan Wiebelitz, Gabriele Von Voigt, and Matthew Smith. Secret program execution in the cloud applying homomorphic encryption. In *Digital Ecosystems and Technologies Conference (DEST), 2011 Proceedings of the 5th IEEE International Conference on*, pages 114–119. IEEE, 2011.

[16] Christian Cachin. Architecture of the hyperledger blockchain fabric. In *Workshop on Distributed Cryptocurrencies and Consensus Ledgers*, volume 310, 2016.

[17] Fook Mun Chan, Quanqing Xu, Hao Jian Seah, Zhaohui Tang, Sye Loong Keoh, and Khin Mi Mi Aung. Privacy Preserving Computation in Home Loans using the FRESCO Framework. 2017.

[18] Vikas Chaurasia and Saurabh Pal. Data mining techniques: To predict and resolve breast cancer survivability. 2017.

[19] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachene. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 3–33. Springer, 2016.

[20] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private information retrieval. In *Foundations of Computer Science, 1995. Proceedings., 36th Annual Symposium on*, pages 41–50. IEEE, 1995.

[21] Ivan Damgård, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel P Smart. Practical covertly secure MPC for dishonest majority–or: breaking the SPDZ limits. In *European Symposium on Research in Computer Security*, pages 1–18. Springer, 2013.

[22] Ivan Damgård, Valerio Pastro, Nigel Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *Advances in Cryptology–CRYPTO 2012*, pages 643–662. Springer, 2012.

[23] Whitfield Diffie and Martin Hellman. New directions in cryptography. *IEEE transactions on Information Theory*, 22(6):644–654, 1976.

[24] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE transactions on information theory*, 31(4):469–472, 1985.

[25] Bradley J Erickson, Panagiotis Korfiatis, Zeynettin Akkus, and Timothy L Kline. Machine learning for medical imaging. *Radiographics*, 37(2):505–515, 2017.

[26] Shimon Even, Oded Goldreich, and Abraham Lempel. A randomized protocol for signing contracts. *Communications of the ACM*, 28(6):637–647, 1985.

[27] Craig Gentry. *A fully homomorphic encryption scheme*. Stanford University, 2009.

[28] Oded Goldreich. Secure multi-party computation. *Manuscript. Preliminary version*, pages 86–97, 1998.

[29] Shai Halevi and Victor Shoup. Algorithms in helib. In *International cryptology conference*, pages 554–571. Springer, 2014.

[30] Ioannis Ioannidis and Ananth Grama. An efficient protocol for Yao's millionaires' problem. In *System Sciences, 2003. Proceedings of the 36th Annual Hawaii International Conference on*, pages 6–pp. IEEE, 2003.

[31] Yannis Ioannidis. The History of Histograms (abridged). In *Proceedings 2003 VLDB Conference*, pages 19–30. Elsevier, 2003.

[32] Geetha Jagannathan, Krishnan Pillaipakkamnatt, and Rebecca N Wright. A new privacy-preserving distributed k-clustering algorithm. In *Proceedings of the 2006 SIAM International Conference on Data Mining*, pages 494–498. SIAM, 2006.

[33] Roman Jagomägis. Secrec: a privacy-aware programming language with applications in data mining. *Master's thesis, University of Tartu*, 2010.

[34] Somesh Jha, Luis Kruger, and Patrick McDaniel. Privacy preserving clustering. In *European Symposium on Research in Computer Security*, pages 397–417. Springer, 2005.

[35] Burt Kaliski. *Asymmetric Cryptosystem*, pages 49–50. Springer US, Boston, MA, 2011.

[36] Liina Kamm and Jan Willemson. Secure floating point arithmetic and private satellite collision analysis. *International Journal of Information Security*, 14(6):531–548, 2015.

[37] Gang-Hoon Kim, Silvana Trimi, and Ji-Hyong Chung. Big-data applications in the government sector. *Communications of the ACM*, 57(3):78–85, 2014.

[38] Jon-Lark Kim and Vera Pless. Designs in additive codes over gf (4). *Designs, Codes and Cryptography*, 30(2):187–199, 2003.

[39] Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free xor gates and applications. In *International Colloquium on Automata, Languages, and Programming*, pages 486–498. Springer, 2008.

[40] Hsiao-Ying Lin and Wen-Guey Tzeng. An efficient solution to the millionaires' problem based on homomorphic encryption. In *International Conference on Applied Cryptography and Network Security*, pages 456–466. Springer, 2005.

[41] Yehuda Lindell and Benny Pinkas. Privacy preserving data mining. In *Annual International Cryptology Conference*, pages 36–54. Springer, 2000.

[42] Yehuda Lindell and Benny Pinkas. A proof of security of yao's protocol for two-party computation. *Journal of Cryptology*, 22(2):161–188, 2009.

[43] Yehuda Lindell and Benny Pinkas. Secure multiparty computation for privacy-preserving data mining. *Journal of Privacy and Confidentiality*, 1(1):5, 2009.

[44] Chang Liu, Xiao Shaun Wang, Kartik Nayak, Yan Huang, and Elaine Shi. Oblivm: A programming framework for secure computation. In *Security and Privacy (SP), 2015 IEEE Symposium on*, pages 359–376. IEEE, 2015.

[45] Oleg Mazonka, Nektarios Georgios Tsoutsos, and Michail Maniatakos. Cryptoleq: A heterogeneous abstract machine for encrypted and unencrypted computation. *IEEE Transactions on Information Forensics and Security*, 11(9):2123–2138, 2016.

[46] MHMD. My Health My Data (MHMD), 2018. [Online; accessed 2018].

[47] Dimitris Mouris, Nektarios Georgios Tsoutsos, and Michail Maniatakos. TERMinator Suite: Benchmarking Privacy-Preserving Architectures. *IEEE Computer Architecture Letters*, 2018.

[48] Moni Naor, Benny Pinkas, and Reuban Sumner. Privacy preserving auctions and mechanism design. In *Proceedings of the 1st ACM conference on Electronic commerce*, pages 129–139. ACM, 1999.

[49] Official Journal of the European Union (4 May 2016). Regulation (eu) 2016/679 of the european parliament and of the council of 27 april 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, 2016.

[50] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 223–238. Springer, 1999.

[51] J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

[52] J Ross Quinlan. C4. 5: Programming for machine learning. *Morgan Kauffmann*, 38:48, 1993.

[53] Michael O Rabin. How To Exchange Secrets with Oblivious Transfer. *IACR Cryptology ePrint Archive*, 2005:187, 2005.

[54] Ronald L Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[55] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[56] Reza Shokri and Vitaly Shmatikov. Privacy-preserving deep learning. In *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*, pages 1310–1321. ACM, 2015.

[57] Edward Snowden. Just days left to kill mass surveillance under Section 215 of the Patriot Act. We are Edward Snowden and the ACLU's Jameel Jaffer. AUA. `https://www.reddit.com/r/IAmA/comments/36ru89/just_days_left_to_kill_mass_surveillance_under/crglgh2/`. Accessed: 2018-07-04.

[58] SODA. Scalable Oblivious Data Analytics (SODA), 2018. [Online; accessed 2018].

[59] Nektarios Georgios Tsoutsos, Charalambos Konstantinou, and Michail Maniatakos. Advanced techniques for designing stealthy hardware trojans. In *Proceedings of the 51st Annual Design Automation Conference*, pages 1–4. ACM, 2014.

[60] Tiina Turban. A Secure Multi-Party Computation Protocol Suite Inspired by Shamir's Secret Sharing Scheme. Master's thesis, Institutt for telematikk, 2014.

[61] Maneesh Upmanyu, Anoop M Namboodiri, Kannan Srinathan, and CV Jawahar. Efficient privacy preserving k-means clustering. In *Pacific-Asia Workshop on Intelligence and Security Informatics*, pages 154–166. Springer, 2010.

[62] Xindong Wu, Vipin Kumar, J Ross Quinlan, Joydeep Ghosh, Qiang Yang, Hiroshi Motoda, Geoffrey J McLachlan, Angus Ng, Bing Liu, S Yu Philip, et al. Top 10 algorithms in data mining. *Knowledge and Information Systems*, 14(1):1–37, 2008.

[63] Andrew C Yao. Protocols for secure computations. In *Foundations of Computer Science, 1982. SFCS'08. 23rd Annual Symposium on*, pages 160–164. IEEE, 1982.

[64] Andrew Chi-Chih Yao. How to generate and exchange secrets. In *Foundations of Computer Science, 1986., 27th Annual Symposium on*, pages 162–167. IEEE, 1986.

[65] Samee Zahur and David Evans. Obliv-C: A Language for Extensible Data-Oblivious Computation. *IACR Cryptology ePrint Archive*, 2015:1153, 2015.

[66] Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 220–250. Springer, 2015.

[67] Yinqian Zhang, Ari Juels, Michael K Reiter, and Thomas Ristenpart. Cross-vm side channels and their use to extract private keys. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 305–316. ACM, 2012.

[68] Guy Zyskind, Oz Nathan, et al. Decentralizing privacy: Using blockchain to protect personal data. In *Security and Privacy Workshops (SPW), 2015 IEEE*, pages 180–184. IEEE, 2015.