# Todo list

# Contents

# 1 Einleitung

Struktur vom Dokument erläutern

## 1.1 Begriffe

**Definition:** Eigenes,Gesamtes
$M$ sei eine Menge von Elementen, die in zwei Teilmengen $M_A$ und $M_B$ zerfällt, sodass $M = M_A \cup M_B$ ist. Wir nehmen desweiteren an, dass Alice $M_A$ kennt, aber weder $M$ noch $M_B$ und dass foo Bob $M_B$ kennt, aber weder $M$ noch $M_A$. Dann bezeichnen wir:

- $M$ als **gesamtes** Wissen
- $M_A$ als das **eigene** Wissen von Alice
- $M_B$ als das **eigene** Wissen von Bob
- $M_B$ als das **andere** Wissen von Alice
- $M_A$ als das **andere** Wissen von Bob

**Definition:** Gemeinsam
Wenn beide Anwender das gleiche Wissen w haben, dann bezeichnen wir w als **gemeinsames Wissen** .
**Definition:** Vorwissen
Wenn Anwender verschiedene Phasen hintereinander ausführen, dann bezeichnen wir das Wissen aus den bereits ausgeführten Phasen als **Vorwissen** .

Definition: Entscheidungsbaum

**Definition:** Attribut
Wir definieren eine Menge von Wahrscheinlichkeiten $P = [0, 1] \subset \mathbb{R}$ und eine Menge von Buchstaben $\Sigma = \{a, b, \ldots, z, A, B, \ldots, Z\}$. Damit definieren wir ein **Attribut** als $\Sigma^+ \times P \times P$. Wenn ein Attribut $A = (w, l, h)$ gegeben ist, bezeichnen wir w als **Wort** , l als **unterer Schwellwert** und h als **oberer Schwellwert** . Es wird desweiteren von allen Attributen gefordert, dass $l \leq h$ ist.
**Definition:** Wahrheitstafel
Für eine boolsche Funktion ist eine **Wahrheitstafel** eine Tabelle, die für jede Kombination der Eingaben genau eine Ausgabe der beschriebenen Funktion definiert.
**Definition:** entstellte Wahrheitstafel
Gegeben sei eine Wahrheitstafel mit Eingabevariablen $\{i_1, i_2, \ldots, i_n\}$ und Ausgabevariable o und wir stellen für jede Variable v eine bijektive Verschleierung $f_v : 0, 1 \mapsto \mathbb{Z}$ auf. Dann entsteht eine entstellte Wahrheitstafel für W, wenn wir für jede Zeile in der Wahrheitstafel die verschleierte Ausgabe mittels einer symmetrischen Verschlüsselung mit den verschleierten Werte der Eingangsvariablen in einem Verschlüsselungsschritt pro Eingangsvariable verschlüsseln. Es

ist notwendig zu bemerken, dass eine Implementierung einen Weg anbieten muss, eine sinnlose Verschlüsselung zu erkennen.
Wenn also beispielsweise in einer Wahrheitstafel die Eingaben 1 und 1 auf 1 abgebildet werden und die erste Eingabevariable verschleiert 1 als 23, die zweite Eingabevariable verschleiert 1 als 49 und die Ausgabevariable verschleiert 1 als 12 und E ist die symmetrische Verschlueselung, dann ist das daraus entstehende Element der entstellten Wahrheitstafel $E_{f_{i_1}(1)}(E_{f_{i_2}(1)}(f_o(1))) = E_{49}(E_{23}(12))$
**Definition:** Schaltkreis
Wir definieren einen **Schaltkreis** als einen gerichteten azyklischen Graphen. Die Knoten dieses Graphen sind annotiert mit Wahrheitstafeln. Die Kanten sind eingeteilt in **Eingabekanten** , **innere Kanten** und **Ausgangskanten** . Der Wert einer Eingabekante wird vom Anwender zu Beginn festgelegt. Der Wert einer inneren Kante oder einer Ausgangskante ergibt sich durch anwenden der Wahrheitstafel auf die Eingangskanten. Da wir nur boolsche Schaltkreise mit den Knotenannotationen ,,and", ,,or", ,,xor" und ,,not" brauchen und all diese Funktionen entweder unär oder kommutativ sind, brauchen wir keine Ordnung der Eingabekanten festlegen.
**Definition:** Entstellter Schaltkreis
Ein **entstellter Schaltkreis** entsteht aus einem normalen Schaltkreis, indem jede an einen Knoten annotierte Wahrheitstafel entstellt wird, wenn dabei garantiert wird, dass für die Verbindung einer Ausgangsvariablen o mit einer Eingangsvariablen i auf beide Variablen die gleiche Verschleierung angewendet wird.
Desweiteren wird für jede Eingabekante die Verschleierungsfunktion gespeichert, um die Eingabe des Schaltkreises kodieren zu können und für jede Ausgangskante die Inversion der Verschleierungsfunktion gespeichert, um die Ausgabe dekodieren zu können.

## 1.2 Annahmen

Wir nehmen im Folgenden an, dass die Anwender ehrlich sind. Das bedeutet, dass die Anwender zwar versuchen, aus den Informationen, die sie im Protokoll erhalten, möglichst viel zu lernen, allerdings werden sie sich an das Protokoll halten. (d.h., eine Annahme der Form ,,Jetzt sendet Alice diesen Wert erneut" funktioniert).

# 2 Grundlagen der Anwendung

Vision der Anwendung

## 2.1 Form der Benutzereingabe

Festlegen, wie die E-Mails ins Programm kommen

## 2.2 Interaktion der verteilten Programme

Festlegen, wie das Programm verteilt wird und die Teile kommunizieren

Austauschen der Anzahl der E-Mails

## 2.3 Phasen der Anwendung

Kurz die Einzelnen phasen der Anwendung beschreiben

# 3 Finden der gemeinsamen Wortliste

In dieser Phase berechnen die beiden Parteien aus den gesamten E-Mails eine gemeinsame Wortliste, die mit grosser Wahrscheinlichkeit aussagekräftige Attribute für den Entscheidungsbaum liefert. Die Phase besteht aus zwei Schritten. Im ersten Schritt berechnen beide Parteien getrennt eine eigene Wortliste. Jedes Wort auf dieser Liste hat in den eigenen E-Mails hat eine starke Aussagekraft über die Klassifikation der E-Mails, die das Wort oft enthalten. Im zweiten Schritt vereinen beide Parteien ihre eigenen Wortlisten, um die gemeinsame Wortliste zu berechnen. Dieser Vorgang ist in Abbildung 2 illustriert.

## 3.1 Berechnung der Anteile an den Wortmengen

Wir verwenden eine Heuristik für den Informationsgehalt des Vorkommens eines Wortes in einer E-Mail, die auf dem Verhältnis der Vorkommnisse des Wortes zu der Gesamtanzahl Worte in einer Klasse basiert. Um diese zu berechnen werden konzeptionell alle Worte in E-Mails einer Klasse zu einer Multimenge hinzugefügt. Für jedes Wort in dieser Multimenge ist das Verhältnis der Vielfachheit des Wortes zur Mächtigkeit der Multimenge das gesuchte Verhältnis. Damit ergibt sich folgendes Akzeptanzkriterium:
**Requirement 1:** Wenn die Spam-Mails die Mails "Foo Bar", "Foo Bar", "Foo Foo" und "Foo" sind und die Nicht-Spam-Mails "Bar Bar", "Foo" und "Bar", dann muss diese Phase die folgende Tabelle berechnen:

| Wort | Spam-Anteil | Nicht-Spam-Anteil |
|---|---|---|
| Foo | $\frac{5}{7}$ | $\frac{1}{4}$ |
| Bar | $\frac{2}{7}$ | $\frac{3}{4}$ |

Eine Approximation der Werte durch Fliesskommazahlen ist ebenfalls akzeptabel.

## 3.2 Auswahl der Worte nach Informationsheuristik

Wir wollen nun Worte auswählen, deren häufiges Vorkommen in einer E-Mail viel über die Klasse der E-Mail aussagen. Wir verwenden dabei eine deduktive Heuristik, die annimmt, dass die Wahrscheinlichkeit, dass eine E-Mail zu einer Klasse gehört, direkt mit den Wahrscheinlichkeiten zusammenhängt, dass die Wörter in dieser E-Mail zu dieser Klasse gehören. Das bedeutet, wir wollen Worte selektieren, bei denen genau eine Wahrscheinlichkeit, zu einer Klasse zu gehören, drastisch unterschiedlich ist.
Eine mögliche Quantifizierung dieser Unterschiedlichkeit ist im binären Fall $\Delta(x, y) = \|x - y\|$. Diese Quantifizierung hat die Eigenschaft, für x = 1 und y = 0 bzw x = 0 und y = 1 maximal 1 zu sein, und für identische x und y 0 zu sein. Da x und y für die Belegung (0, 1) bzw (1, 0) wirklich

maximal weit auseinander liegen und bei gleichen Werten wirklich die gerinstmögliche Aussage über die Klassezugehörigkeit eines Wortes getroffen wird, ist dies wirklich eine Quantifizierung der Aussagekraft, die wir anstreben. Damit wählen wir die aussagekräftigstenen Worte aus, indem wir alle Worte nach $\Delta(\text{Vorkommen in Spam} - \text{E} - \text{Mails}, \text{Vorkommen in Nicht} - \text{Spam} - \text{E} - \text{Mails})$ absteigend sortieren und die ersten N Worte wählen. Damit ergibt sich folgendes Akzeptanzkriterium:
**Requirement 2:** Wenn N = 2 ist, und als Worte mit Vorkomnissen gegeben sind:

| Wort | Vorkomnisse in Spam-E-Mails | Vorkommnisse in Nicht-Spam-Emails |
|---|---|---|
| A | 0.5 | 0.5 |
| B | 0.2 | 0.2 |
| C | 0.3 | 0.5 |
| D | 0.2 | 0.8 |
| E | 0.9 | 0.2 |

dann werden als Wortliste D und E gewählt.

## 3.3 Zusammenfassen der Wortlisten

Wir haben nun zwei separate, lokale Wortlisten berechnet. Diese müssen zusammengefasst werden zu einer gemeinsamen Wortliste. Da wir annehmen, dass die Benutzer ehrlich sind, können wir annehmen, dass die Benutzer als Eingabe dieses Protokolles keine beliebige Liste festlegen und auch ihre Daten nicht so manipulieren, dass eine bestimmte Wortliste versendet wird. Zudem werden die Attribute des Baumes öffentlich sein, sodass eine Geheimhaltung der Wortlisten keinen Sinn macht. Deswegen können wir die Wortlisten durch eine einfache Vereinigung der separaten Wortlisten zu einer gesamten Wortliste vereinigen. Deswegen wird das Zusammenfassen der Wortliste implementiert, indem beide Anwender ihre lokale Wortliste an den jeweils anderen Anwender versenden und beide Anwender lokal die Wortlisten vereinigen. Damit ergibt sich folgendes Akzeptanzkriterium:
**Requirement 3:** Wenn Alice die Wortliste A, B, C berechnet hat, und Bob die Wortliste C, D, E berechnet hat, dann muss die zusammengefasste Wortliste A, B, C, D, E sein.

# 4 Finden der gemeinsamen Schwellwerte

Einleitung, auf Figure für Schwellwerte verweisen

Für section-Titel besseren Begriff für "Vorkomnisse der Worte in eigenen Spam/Nicht Spam E-Mails" finden

## 4.1 Berechnung der Anteile

Content: Mittlwerte ausrechnen

## 4.2 Bestimmung der eigenen Schwellwerte

Es muss nun für ein Wort $w$ eine Schwelle gefunden werden, ab wann das Wort w in einer E-Mail oft vorkommt. Wir wählen hierzu den Mittelwert der Vorkommnisse des Wortes in allen eigene E-Mails eines Anwenders, da wir dann die Begriffe ,,überdurchschnittlich oft" bzw ,,unterdurchschnittlich oft"als eigenen Schwellwert quantifizieren. Damit ergibt sich folgendes Akzeptanzkriterium:
**Requirement 4:** Wenn die E-Mails "A A A", "A B B", "A C C" und "A A C" sind, dann muss die Software als eigenen Mittelwert für das Wort A $\frac{1}{4} \cdot (1 + \frac{1}{3} + \frac{1}{3} + \frac{2}{3}) = \frac{7}{12}$ bestimmen.

## 4.3 Syncronisierung der Schwellwerte

Content: Alice a, Bob b -¿ ¡a, b¿, unter dem minimum der beiden werte def. selten, ueber dem max der werte def. oft und dazischen keine ahnugn. Implementierung als simples herumschicken

# 5 Diskretisieren der eigenen E-Mails

Einleitung, Figure referenzieren

Content

# 6 Lernen der gesamten E-Mails

verteiltes ID3 beschreiben

## 6.1 Yaos Protokoll

Die Eingabe jedes Anwenders für Yaos Protokoll ist eine Eingabe, und nach Kerckhoffs Prinzip auch der auszuführende Schaltkreis in Reinform. Einer der beiden Anwender, ohne Beeintraechtigung der Allgemeinheit Alice, muss nun aus diesem Schaltkreis einen enstellten Schaltkreis konstruieren. Dies kann am einfachsten passieren, indem der Schaltkreis in einer Breitensuche traversiert wird und die Kanten verschleiert werden. Das bedeutet, dass in einem Schritt die zusammenhängenden Eingangs- und Ausgangsvariablen verschleiert werden. Dadurch werden die Anforderungen aus der Definition der entstellten Schaltkreise erfüllt. Dabei kann dann auch erkannt werden, dass eine Eingangsvariable bzw Ausgabevariable kein ,,anderes Ende" hat, d.h. diese Verschleierung in die Eingabekodierung bzw Ausgabekodierung geschrieben werden muss. Dieser Schaltkreis und die Ausgangskodierung wird dann an Bob übertragen, während die Eingangskodierung bei Alice verbleibt.
Bei der Verschlüsselung der Ausgaben verwenden wir ein einfaches XOR. Um zu garantieren, dass wir Unsinnige Verschlüsselungen erkennen können, erweitern wir jeden Tabelleneintrag um eine zweiten Tabelleneintrag, der aus einem analog zum Nutzeintrag verschlüsselten Nullvektor besteht. Wir bezeichnen diesen Eintrag als Markierungseintrag. Dann ist es möglich, diesen Eintrag zu entschlüsseln, zu überprüfen, ob dies der 0-Vektor ist und falls dies der Fall ist, den Nutzeintrag zu entschlüsseln. Der einzige Fall, wo dies Probleme bereitet ist, wenn die bei zwei Eingaben die Verschleierungen genau konträr sind, da dann $a \oplus b = 0 = b \oplus a$ gilt und somit die Eingabe (0,1) und (1,0) die gleiche Ausgabe liefert. Dies muss also beim Verschleiern verhindert werden, was jedoch kein Problem ist.
Bob muss nun noch die Belegung der Eingangskanten lernen. Die Eingabevariablen, die von Alice Eingabe belegt werden müssen werden belegt, indem Alice anhand der Eingangskodierung ihre Eingabe kodiert und diesen String an Bob versendet. Da Bob keine Informationen darüber hat, welcher der beiden Verschleierungswerte für 0 oder für 1 steht, erhält Bob damit keine Informationen über Alice Eingabe.
Für die Kodierung von Bob's Eingabe muss etwas mehr Aufwand getrieben werden, da weder Bob die Eingabekodierung lernen darf (weil er sonst Alice Eingabe lernt), noch Alice Bob's Eingabe lernen darf (indem z.B. Bob seine Eingabe an Alice schickt, damit Alice die Eingabe kodieren kann). Daher wird dies durch eine Anwendung des 1-2-Oblivious-Transfers gelöst: Alice bietet für jede von Bob zu belegende Eingabevariable die Werte für 0 bzw 1 an, und Bob wählt mithilfe seines Eingabebits den richtigen Wert aus. Aus den Eigenschaften von OT folgt, dass Alice nicht weiss, welchen der beiden Werte Bob gewählt hat, und dass Bob den nicht gewählten Wert nicht kennt. Somit ist die Sicherheit

von Bobs Eingabe gegeben und sichergestellt, dass Bob wirklich seine Eingabe verwendet.
Wir implementieren den 1-2 Oblivious Transfer mittels RSA. Die Eingabe von Alice seien $m_1$, $m_2$ und die Eingabe von Bob sei $b$. Alice ein Schlüsselpaar, den privaten Exponenten e, den öffentlichen Exponenten d und sendet den Modulus N, d und zwei zufällige Nachrichten $x_1$ und $x_2$ an Bob. Bob wählt eine zufällige Nachricht k, verschlüsselt k zu $k'$ und sendet $v = x_b + k' mod N$ an Alice. Alice berechnet nun $k_0$ als Entschlüsselung von $v - x_0$ und $k_1$ als Entschlüsselung von $v - x_1$ und sendet $m_0 + k_0$ und $m_1 + k_1$ an Bob. Wenn Bob b = 0 gewählt hatte, kann Bob $m_0$ durch Subtraktion von $k'$ von Alice erster Nachricht berechnen, und wenn Bob b = 1 gewählt hatte, kann Bob $m_1$ durch Subtraktion von $k'$ von Alice zweiter Nachricht berechnen. Das folgt (oBdA für b = 0) aus $m_0 + k_0 - k' = m_0 + D(v - x_0) - k' = m_0 + k' - k' = m_0$. Falls b nicht 0 war, heben sich in der Entschlüsselung die Faktoren $x_0$ und $x_1$ nicht auf und ein undefinierter Wert wird berechnet.
Danach kann Bob den Schaltkreis auswerten, indem er immer für eine Entscheidungstabelle, deren Eingabevariablen vollständig belegt sind, denjenigen Eintrag sucht, indem für jeden Eintrag der Markierungseintrag entschlüsselt wird und falls das Ergebnis 0 ist, der Nutzeintrag entschlüsselt wird und der Wert entsprechend weiterverwendet wird. Sobald alle Ausgangsvariablen des gesamten Schaltkreises bekannt sind, kann anhand der Ausgabedekodierung die Ausgabe in Klartext berechnet werden und Alice zugeschickt werden. (Hier wird wieder die Annahme des ehrlichen Anwenders getroffen).

## 6.2 Feststellen der dominierenden Ausgabe

Da die Menge der Attribute öffentlich ist und die Menge der bereits verwendeten Attribute ebenfalls öffentlich ist, können beide Parteien erkennen, dass sie nun einen Blattknoten mit der dominierenden Ausgabe konstruieren müssen. Dazu ist es notwendig, zu erkennen, ob in den verbleibenden E-Mails mehr Spam-E-Mails oder mehr Nicht-Spam-E-Mails vorhanden sind. Dazu muss ein Schaltkreis designed werden, welcher als Eingabe zwei Paare von Zahlen erhählt und als Ausgabe den Index der maximalen Summe liefert. Da hierzu die Summe gebildet werden muss, muss hierzu die verwendete Bitbreite bestimmt werden. Da jedoch beide Anwender die Anzahlen der von ihnen verwendeten E-Mail-Mengen preisgegeben haben, kann jedoch die Summe der verbleibenden Teilmengen abgeschätzt werden durch die Summe der Anzahlen der gesamten E-Mails und somit kann die geasmte Bitbreite der zu addierenden Zahlen abgeschätzt werden als Logarithmus dieser Gesamtanzahl.
Dadurch verbleibt es dann, einen Schaltkreis zu designen, welcher zwei Zahlen bekannter Bitbreite addiert und 0 bzw 1 ausgibt, wenn die erste bzw zweite Zahl grösser ist. Da Effizienz erst einmal sekundär ist, kann die Addition der Zahlen durch einen einfachen Riple-Carry-Adder vollzogen werden.
Die Maximumsbestimmung von zwei binären Zahlen ist dann die Frage, welche der beiden Zahlen die hoechstwertige 1 hat. Wenn wir zudem definieren, dass die Ausgabe 0 bedeutet, dass die erste Zahl in den Paaren die kleinere ist und die

Ausgabe 1 die zweite, dann vereinfacht sich die Maximumsbestimmung noch weiter zum finden der höchstwertigen Stelle, die sich unterscheidet und dem zurückgeben des Bits in der ersten Zahl. Dies ist in Abbildung 5 angedeutet. Formal gilt also: $lt(a,b) = b_i \Leftrightarrow a_i \neq b_i \wedge \forall k = i+1, \ldots, n : a_k = b_k$ Der Allquantor ist als Kaskade von Und-Gattern implementierbar und die Gleichheit bzw Ungleichheit als Negation bzw direkte Verwendung des XOR-Gatters implementierbar, damit kann diese Formel einfach in einen Schaltkreis überführt werden.

## 6.3 Feststellen ob Ausgabe eindeutig

Es muss ein Schaltkreis designed werden, der Feststellt, ob alle Elemente einer Menge eindeutig sind und dann entweder das Klassenlabel oder ein Fehlersymbol ausgibt. Die Eingabe ist somit für jede E-Mail eine Kodierung für Spam, Nicht Spam oder Abwesenheit, die Anzahl der Eingaben ist bekannt und durch die Gesamtanzahl der E-Mails beschränkt.
Wir implementieren diesen Schaltkreis dann als Zustandsmaschine, die die internen Zustände ,,Alles bis hier Spam", ,,Alles bis hier kein Spam" und ,,Bereits uneindeutig" hat. Wenn dann $T$: State $\times$ Mailtype ist, muss folgende Zustandsübergangstabelle implementiert werden:

| | | | |
|---|---|---|---|
| T | Alles bis hier Spam | Alles bis hier kein Spam | Bereits uneindeutig |
| Spam | Alles bis hier Spam | Uneindeutig | Bereits Uneindeutig |
| Nicht Spam | Uneindeutig | Alles bis hier kein Spam | Bereits Uneindeutig |
| Abwesenheit | Alles bis hier Spam | Alles bis hier kein Spam | Bereits uneindeutig |

Zudem wäre es sehr angenehm, wenn die Kodierung von Spam und ,,Alles bis hier Spam" und von ,,Nicht Spam" und ,,Alles bis hier kein Spam" gleich sind, da dies den Anfangsfall vereinfacht.
Wenn wir nun ,,Bereits uneindeutig" durch 11 kodieren und ,,Alles bis hier Spam" mit 01 kodieren und ,,Alles bis hier kein Spam" mit 00 kodieren und davon ausgehend ,,Spam" als 01 kodieren und ,,Nicht Spam" als 00 und ,,Abwesenheit" als 11, dann muss folgende Zustandsübergangstabelle gelten:

| | | | |
|---|---|---|---|
| T | 01 | 00 | 11 |
| 01 | 01 | 11 | 11 |
| 00 | 11 | 00 | 11 |
| 11 | 01 | 00 | 11 |

Durch Anwenden von KV-Diagrammen ergeben sich als Zustandsübergangsgleichungen somit, mit $s_0, s_1$ als Bits des vorherigen Zustandes, $e_0, e_1$ als Bits der E-Mail Kodierung und $s'_0, s'_1$ als Bits des neuen Zustandes:

$$\begin{aligned} s'_0 &= s_0 \vee \overline{s_1 e_0} e_1 \vee s_1 \overline{e_0 e_1} \\ s'_1 &= s_1 \vee \overline{e_0} e_1 \end{aligned}$$

Der Anfangszustand des Automatens ist die Kodierung der Klasse der ersten E-Mail. Es muss dann nur noch für jede E-Mail das beschriebene Zustandsübergangsnetz von den vorherigen Zuständen und der Kodierung der Klasse der aktuellen E-Mail implementiert werden. Das Fehlersymbol, das ausgegeben wird, ist 11, andernfalls ist die Klasse im zweiten Ausgabebit Kodiert, wobei Spam als 1 kodiert ist und Nicht Spam als 0.

## 6.4 Das Entropien-Protokoll

Es muss in diesem Teilschritt im Kern $x \cdot \ln x$ berechnet werden und danach das Maximum dieser Werte bestimmen. Um den ersten Schritt mit Yaos Algorithmus zu berechnen, müssen wir einige Teilprobleme lösen. Es muss zum einen ein Weg gefunden werden, durch den man mit Yaos Protokoll für beide Anwender verschiedene Ausgaben produziert werden können und zum zweiten muss ein Weg gefunden werden, eine Zahl in zwei Shares zu zerlegen, sodass die Summe der Shares wieder die Zahl ist.
Betrachten wir das erste Teilproblem, separate Ausgaben für die Anwender zu erzeugen. Dies ist ein Problem, da Yaos Protokoll fuer die private zwei Parteien berechnung einer Funktion eben eine Funktion berechnet, d.h. genau einen Wert, der dann beiden Parteien bekannt gegeben wird. Um dies zu lösen, nehmen wir an, dass i und j die Eingaben von Alice und Bob sind und der evaluierte Schaltkreis $f(i,j) = A\&B$ berechnet, wobei & die Konkatenation von Binärsequenzen bezeichne. Falls die Ausgabe nicht derartig sortiert ist, muss dies in einem separaten Schaltkreis vor der Ausgabe passieren. Wir erweitern dann die Eingabe i von Alice um $\|A\|$ Zufallsbits zu $i\&C_A$ und analog die Eingabe j von Bob um $\|B\|$ Zufallsbits zu $j\&C_B$ und erweitern den Schaltkreis am Ende, sodass er $f'(i\&C_A, j\&C_B) = A \oplus C_A\&B \oplus C_B$ berechnet, wobei $\oplus$ das bitweise XOR von zwei Binärsequenzen beschreibe. Da $C_A$ und $C_B$ im eigentlichen Schaltkreis zur Berechnung von A und B nicht verwendet werden und zufällig sind und dem jeweils anderen Anwender unbekannt sind, ist dies also äquivalent zu einer One-Time-Pad-Verschlüsselung der Ergebnisse mit einem Schüssel, den nur der Anwender kennt, der diese Ausgabe wisssen soll.
Zum zweiten muss ein binärkodierter Integer $S$ in zwei zufällige Shares $S_A, S_B$ zerlegt werden, sodass $S_A + S_B = S$ ist. Da unsere Schaltkreise deterministisch sind, bedeutet das, dass wir Zufallsbits als Eingabe hinzugeben müssen, um eine zufällige Aufteilung zu erreichen. Wir lösen dieses Problem dann so, dass wir einen Verteilungsvektor $V$ berechnen mit $\|V\| = \|S\|$ und kopieren das Bit i in Alice' Share $S_A$, falls $V^i = 1$ ist, andernfalls kopieren wir das Bit i in Bobs Share $S_B$. Das Kopieren kann implementiert werden, indem $S_A^i = S^i \wedge V^i$ und $S_B^i = S^i \wedge \overline{V^i}$ implementiert wird.
Die Summe der beiden Shares ist dann wieder S, weil die Shares den Wert von S als zwei disjunkte Mengen von Zweierpotenzen darstellen, sodass die Addition keinen Carry erfordert, sondern einfach eine Veroderung der beiden Shares ist. Somit wird S wieder entstehen, wenn jedes Bit in $S_A$ oder in $S_B$ zu finden ist. Das ist nach Konstruktion aequivalent dazu, dass jedes Bit in V entweder 0 oder 1 ist. Bleibt die Frage, wieviele Informationen aus dem Share gewonnen

werden kann. Es gilt, dass eine 0 eine Unsicherheitsquelle ist, da die 0 entweder eine 0 sein kann, die enstanden ist, weil der Verteilungsvektor an dieser Position ungünstig war oder weil wirklich im Share eine 0 war. Wenn wir eine 1 sehen, wissen wir auf jeden Fall, dass in $S$ an dieser Position 1 war. Da wir jedoch als Menge aufzuteilender Werte zum einen den Wert $2^N \cdot n \cdot \ln 2$ haben für eine Konstante $N$ und zum anderen $\epsilon \cdot 2^N$ für ein beiden Parteien unbekanntes Epsilon, sind die vorkommenden Bitmuster so überlappend, dass nur aus der Position einiger 1en nicht geschlossen werden kann, welcher Wert aufgetreten ist. Wenn zudem der Verteilungsvektor unbekannt ist, ist somit aus dem Share auf jeden Fall nicht ausreichend Informationen abzuleiten, um das Geheimnis zu lernen.
Es bleibt nun V zu berechnen. Da die Bitbreite von $S$ abgeschätzt werden kann, definieren wir, dass Alice und Bob zwei weitere Vektoren $V_A$ und $V_B$ eingeben und definieren $V = V_A \oplus V_B$. Dadurch kann keiner der beiden Anwender alleine den Wert von V zu seinem Vorteil beeinflussen.
Im $x \cdot \ln x$ Entropienprotokoll wird nun eine Taylorreihe aufgestellt, anhand derer festgestellt werden kann, das zwei Werte berechnet werden müssen: $2^N \cdot n \cdot \ln 2$ und $\epsilon \cdot 2^N$, wobei x die Summe der Anwendereingaben ist und $x = 2^n \cdot (1+\epsilon)$ mit $-\frac{1}{2} \leq \epsilon \leq \frac{1}{2}$ und N eine obere Schranke für n ist. Der erste Wert ist kein Integer, kann jeoch auf einen Integer gerundet werden, ohne dass die Genauigkeit einbricht, da er nur als Anfangsannäherung dient, der zweite Wert ist ein Integer. Der erste Wert wird in einem Schaltkreis berechnet, indem das n berechnet wird und die möglichen Werte des Ausdruckes in einer hardkodierten Tabelle nachgesehen werden. Das n wird berechnet, indem der Index des signifikantesten Bits von x berechnet wird. Der zweite Ausdruck wird berechnet, indem $\epsilon \cdot 2^N = 2^(N-n) \cdot \epsilon \cdot 2^n = 2^(N-n) \cdot (x - 2^n)$ festgestellt wird. Das bedeutet, man muss das signifikanteste Bit von x löschen und das Ergebnis davon um N-n bit nach links shiften. Der Wert von N wird vorher berechnet und hardkodiert.Die Lookuptable und der parametrierte Shift werden wieder über vollständige Suchen implementiert, da so nur der (relativ einfache) Vergleich implementiert werden muss. Danach sind diese beiden Werte berechnet und ausgehend von der Vorüberlegung muss noch die Aufteilung beider Werte in Shares sowie die Verschleierung der Ausgaben für die beiden Anwender orthogonal hinzugefügt werden.

Verbesserung der Approximation

Private Multiplikation zm Berechnen von x * ln x

Schaltkreis für x * ln x -Protokoll aus dem Paper zusammenfassen

## 6.5 Attribut mit maximalem Informationsgewinn finden

Entropien berechnen

Index der maximalen Summe der Shares finden, verwenden vom Schaltkris aus dominierender Ausgabe

# 7 Verwenden des Klassifikators

Es muss zusätzlich zum Lern-Programm ein Programm geschrieben werden, welches den Klassifikator auf eine Menge von E-Mails anwendet. Wir bieten hierzu ein Programm an, welches den Klassifikator in einem einfachen Textformat einliest und diesen auf eine Menge von E-Mails anwendet. Diese Menge von E-Mails ist als Dateien in einem Verzeichnis gegeben und für jeden Dateinamen wird eine Klassifikation als Spam oder Not Spam ausgegeben.

## 7.1 Eingabe des Klassifikators

Ausgehend von den Definitionen eines Attributes und eines Entscheidungsbaumes kann leicht eine Grammatik erzeugt werden, welche die Form des eingegebenen Klassifikators eindeutig bestimmt. Wir notieren diese Grammatik in BNF. Diese ist so gewählt, dass sie eine LL(1)-Grammatik ist, d.h., sie ist besonders einfach zu parsen.

```
tree -> ‘Decide’ ‘(’ attribute (‘,’ tree)+ ‘)’
      | ‘Output’ ‘(’ class ‘)’.
class -> ‘Spam’ | ‘Not Spam’.
attribute -> ‘(’ word ‘,’ probability ‘,’ probabilty ‘)’.
word ->   (‘a’ | ‘b’ | ... | ‘z’ | ‘A’ | ... | ‘Z’)+
probability -> number ‘.’ number .
number -> (‘0’ | ‘1’ | ... | ‘9’)+.
```

Somit wäre ein kodierter Klassifikationsbaum beispielsweise:

```
Decide((Foo, 0.3, 0.6),
        Output(Spam),
        Decide((Bar, 0.5, 0.6),
                Output(Spam),
                Output(Not Spam)
              )
      )
```

Die Produktion ,,probability" beschreibt eine Zahl, die sich als eine Folge von Ziffern vor einem Dezimalpunkt und einer Folge von Ziffern nach dem Dezimalpunkt beschreiben lassen. Dies sind alle reellen Zahlen, und da wir keine komplexen Zahlen betrachten, sondern nur Verhältnisse von natürlichen Zahlen zueinander ist diese Darstellung ausreichend, um alle möglichen Zahlenwerte darzustellen. Da die Buchstabenmenge als die lateinischen Klein- und Grossbuchstaben definiert ist und ein Wort definiert ist als Sequenz dieser Zeichen, ist die Produktion ,,word" ausreichend, um alle möglichen Worte darzustellen. Damit ergibt sich, dass die Produktion ,,attribute" in der Lage ist, alle Attribute, die in dieser Anwendung auftreten können, darzustellen.
Es gibt desweiteren bei uns nur die Ausgaben ,,Spam" und ,,Nicht Spam". Damit ist die Produktion ,,class" ausreichend, um alle möglichen Ausgaben

des Baumes darzustellen. Daraus folgt, dass die zweite Produktion von tree in der Lage ist, alle möglichen Blätter darzustellen. Desweiteren folgt induktiv aus der Vollständigkeit der Produktion „attribute" und der Darstellbarkeit der Blätter als Induktionsanfang, dass die erste Produktion von tree in der Lage ist, alle möglichen auszugebenden Entscheidungsbäume darzustellen. Damit ist die Grammatik mächtig genug für unsere Zwecke.
Es ist weiterhin zu bemerken, dass eine semantische Validierung notwendig ist, da in der Grammatik weder gefordert ist, dass der untere Schwellwert eines Attributes wirklich kleiner ist als der obere Schwellwert eines Attributes noch dass beide Schwellwerte zwischen 0 und 1 liegen, noch dass die Anzahl der Teilbäume richtig ist. Damit ergeben sich die folgenden Akzeptanzkriterien:
**Requirement 5:** Der Klassifizierer weist die Eingabe Decide((Foo, 0.5,2), Output(Spam), Output(Spam)) wegen eines zu grossen Schwellwertes zurück
**Requirement 6:** Der Klassifizierer weist die Eingabe Decide((Foo, 2, 3), Output(Spam), Output(Spam)) wegen eines zu grossen Schwellwertes zurück
**Requirement 7:** Der Klassifizierer weist die Eingabe Decide((Foo, 1, 0), Output(Spam), Output(Spam)) wegen falsch sortierter Schwellwerte zurück
**Requirement 8:** Der Klassifizierer weist die Eingabe Decide((Foo, 0.2, 0.3), Output(Spam), Output(Spam)) wegen zuweniger Teilbäume zurück
**Requirement 9:** Der Klassifizierer weist die Eingabe Decide((Foo, 0.2, 0.3), Output(Spam), Output(Spam), Output(Spam), Output(Spam)) wegen zuvieler Teilbäume zurück
**Requirement 10:** Der Klassifizierer akzeptiert die Eingabe Decide((Foo, 0.2, 0.3), Output(Spam), Decide((Bar, 0.3, 0.4) Output(Spam), Output(Not Spam), Output(Spam)), Output(Spam))
**Requirement 11:** Der Klassifizierer akzeptiert die Eingabe Decide((Foo, 0, 0.5), Output(Spam), Output(Spam))
**Requirement 12:** Der Klassifizierer akzeptiert die Eingabe Decide((Foo, 0.5, 1), Output(Spam), Output(Spam))
**Requirement 13:** Der Klassifizierer akzeptiert die Eingabe Decide((Foo, 0.5, 0.5)), Output(Spam), Output(Spam))
**Requirement 14:** Der Klassifizierer akzeptiert die Eingabe Decide((Foo, 0, 0), Output(Spam))
**Requirement 15:** Der Klassifizierer akzeptiert die Eingabe Decide((Foo, 1, 1), Output(Spam))

## 7.2 Arbeitsweise des Klassifikators

Der Klassifikator bekommt als Eingabe ein Verzeichnis mit E-Mails und eine Datei meinem einem Klassifikator. Den Klassifikator liest er ein und speichert ihn intern. Danach traversiert der Klassifikator das gegebene Verzeichnis rekursiv und behandelt jede Datei, die er in diesem Verzeichnis findet als E-Mail-Inhalt. (Dadurch ist es möglich, die Trainingsdaten auch als Versuchsdaten zu benutzen, ohne sie zu bewegen).
Für jede E-Mail wird dann der Inhalt der E-Mail eingelesen und die Klassi-

fikation durch den eingegebenen Klassifikator durchgeführt, d.h., für jeden Ast werden die Vorkommnisse des Wortes im Attribut festgestellt und rekursiv im entsprechenden Teilbaum weiterklassifiziert und in einem Blatt wird die Klasse festgestellt. Diese festgestellte Klasse wird dann zusammen mit dem relativen Pfad vom eingegebenen Verzeichnis ausgegeben.
Wenn also beispielsweise folgende Verzeichnisstruktur gegeben ist:

```
mails/hank/mail1
mails/hank/mail2
mails/bob/mail1
```

und wir annehmen, dass der eingegebene Klassifikator E-Mails mit dem Index 1 als Spam erkennt und E-Mails mit dem Index 2 als Nicht Spam, dann wäre die Ausgabe:

```
hank/mail1 Spam
hank/mail2 Not Spam
bob/mail1 Spam
```

Damit ergeben sich folgende Akzeptanzkriterien:
**Requirement 16:** Gegeben der Klassifikator Decision((Bar, 0.3, 0.6), Output(Spam), Output(Not Spam), Output(Spam)), dann wird die E-Mail "Bar Foo Foo Foo" als Spam klassifiziert
**Requirement 17:** Gegeben der Klassifikator Decision((Bar, 0.3, 0.6), Output(Spam), Output(Not Spam), Output(Spam)), dann wird die E-Mail "Bar, Bar, Foo, Foo" als Not Spam klassifiziert
**Requirement 18:** Gegeben der Klassifikator Decision((Bar, 0.3, 0.6), Output(Spam), Output(Not Spam), Output(Spam)), dann wird die E-Mail "Bar, Bar, Bar, Foo" als Spam klassifiziert
**Requirement 19:** Gegen der Klassifikator Decision((Bar, 0.3, 0.6), Output(Spam), Output(Not Spam), Output(Spam)) und eine Verzeichnisstruktur wie oben skizziert, wobei hank/mail1 "Bar Foo Foo Foo", hank/mail2 "Bar Bar Foo Foo" und bob/mail1 "Bar Bar Bar Foo" enthält, dann wird die oben als Beispiel genannte Ausgabe produziert (oder in einer anderen Reihenfolge

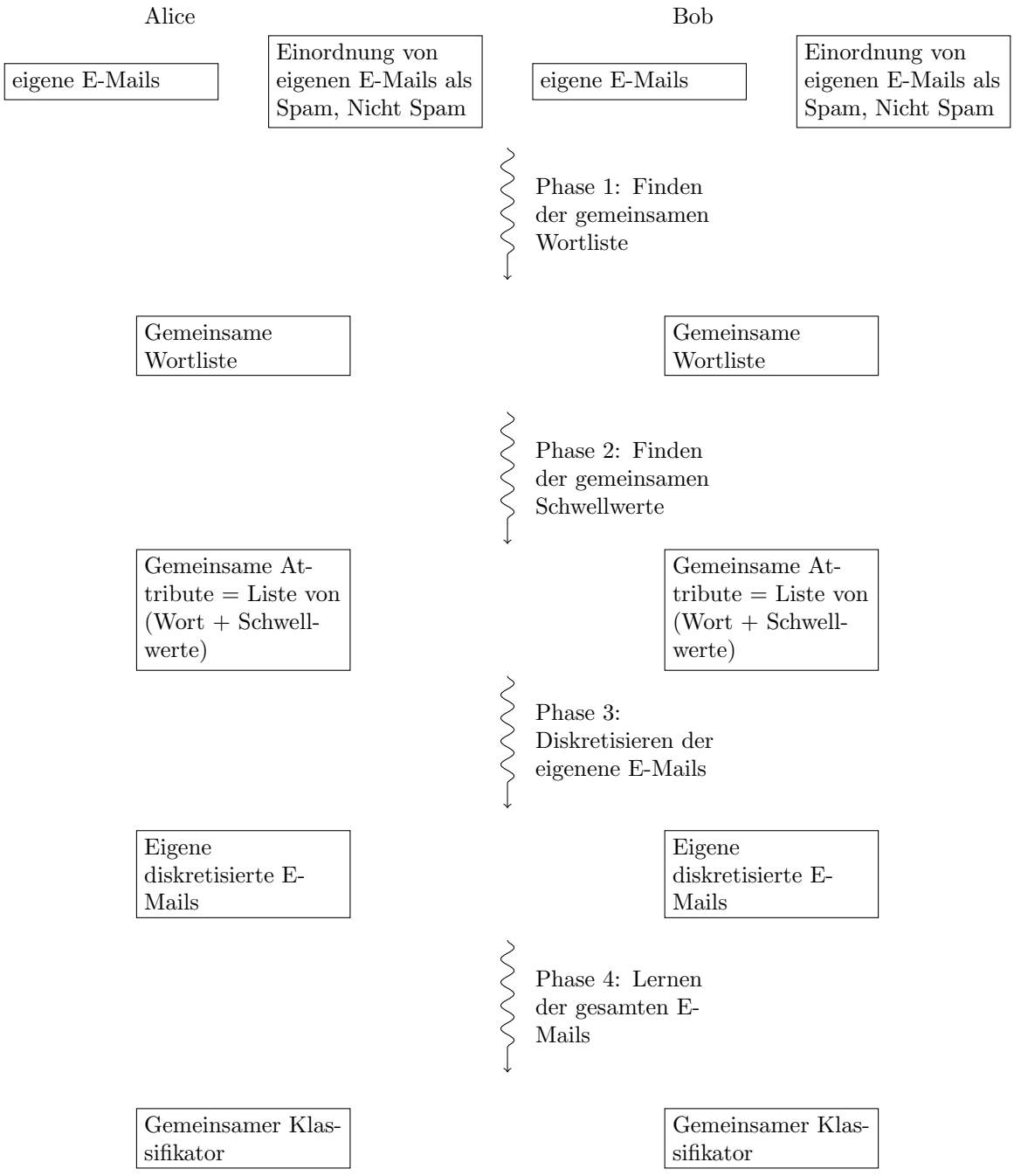



Figure 1: Phasen der Anwendung

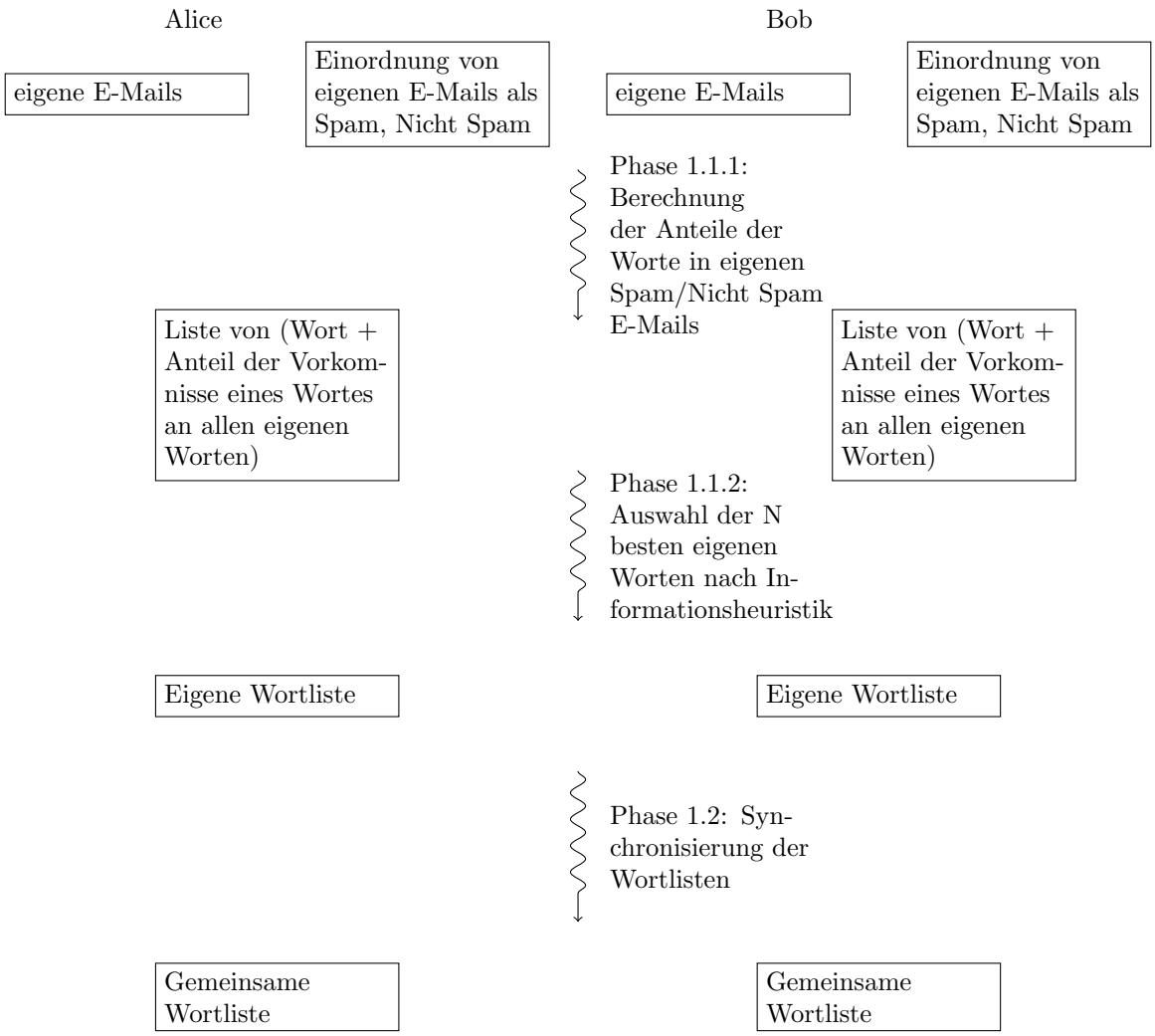
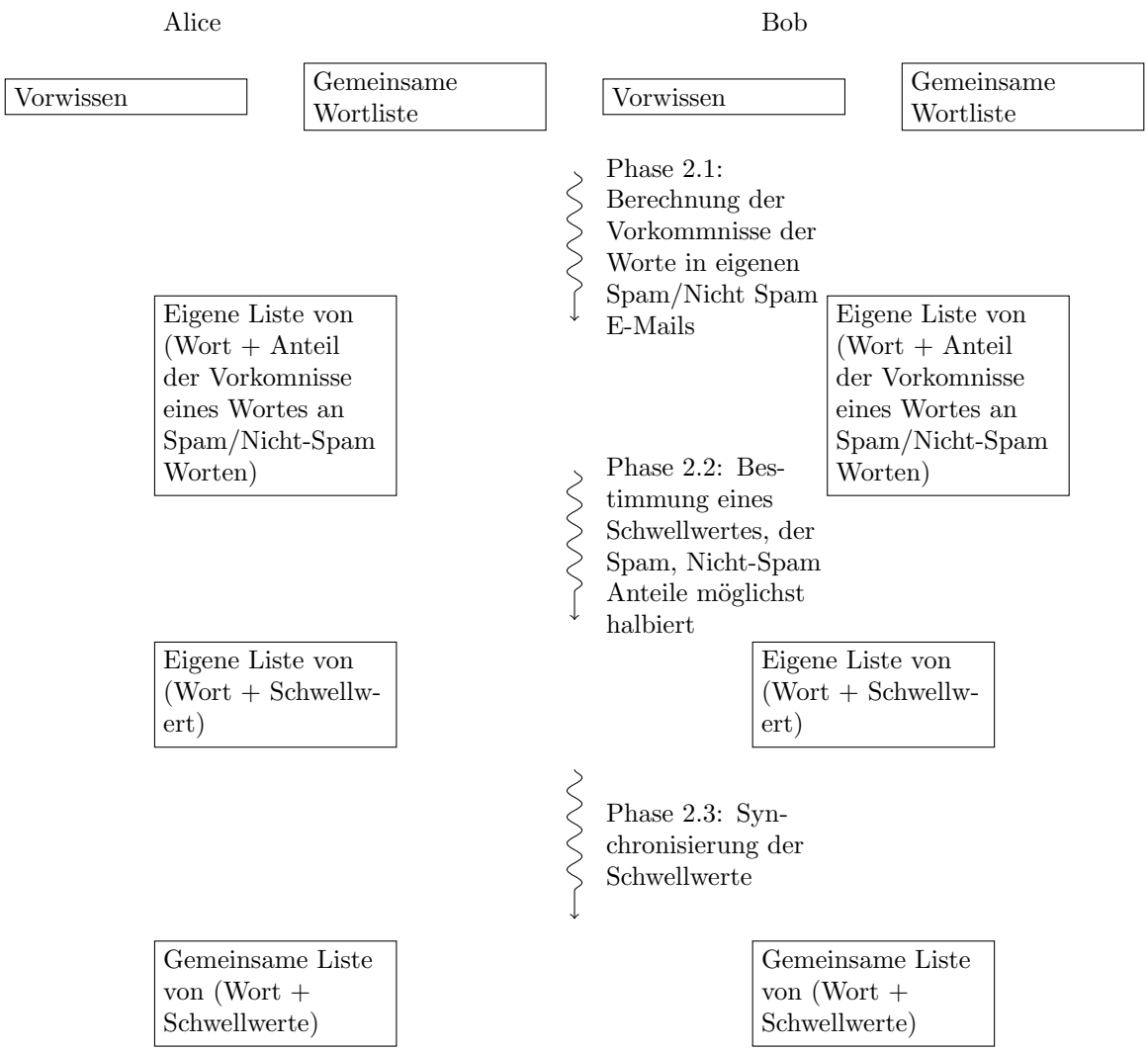
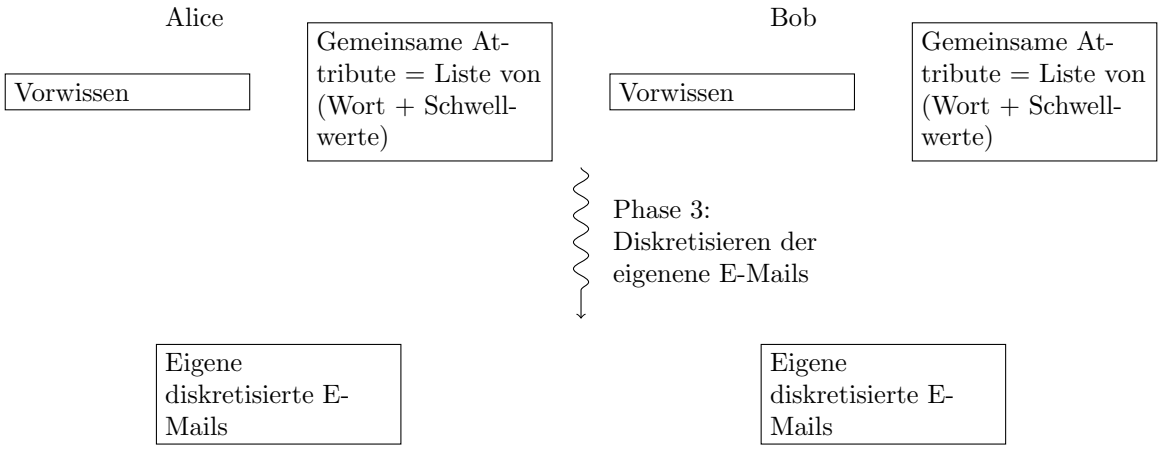



Figure 2: Schritte zum Berechnen der gemeinsamen Wortliste


Alice
Bob
Vorwissen
Gemeinsame Wortliste
Vorwissen
Gemeinsame Wortliste
Phase 2.1: Berechnung der Vorkommnisse der Worte in eigenen Spam/Nicht Spam E-Mails
Eigene Liste von (Wort + Anteil der Vorkomnisse eines Wortes an Spam/Nicht-Spam Worten)
Eigene Liste von (Wort + Anteil der Vorkomnisse eines Wortes an Spam/Nicht-Spam Worten)
Phase 2.2: Bestimmung eines Schwellwertes, der Spam, Nicht-Spam Anteile möglichst halbiert
Eigene Liste von (Wort + Schwellwert)
Eigene Liste von (Wort + Schwellwert)
Phase 2.3: Synchronisierung der Schwellwerte
Gemeinsame Liste von (Wort + Schwellwerte)
Gemeinsame Liste von (Wort + Schwellwerte)


Alice
Bob
Vorwissen
Gemeinsame Attribute = Liste von (Wort + Schwellwerte)
Vorwissen
Gemeinsame Attribute = Liste von (Wort + Schwellwerte)
Phase 3:
Diskretisieren der eigenene E-Mails
Eigene diskretisierte E-Mails
Eigene diskretisierte E-Mails

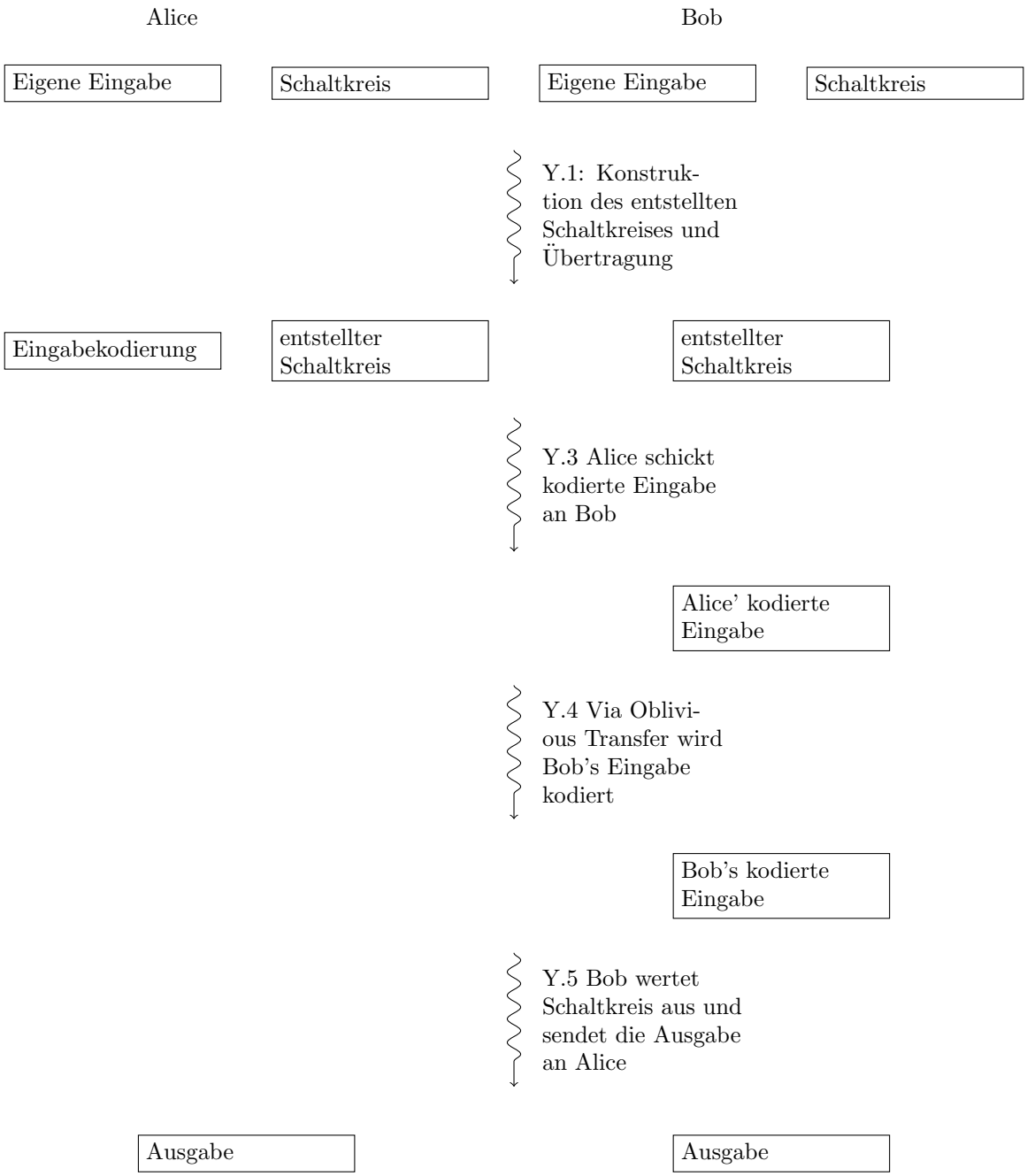



Figure 3: Yaos Protokoll

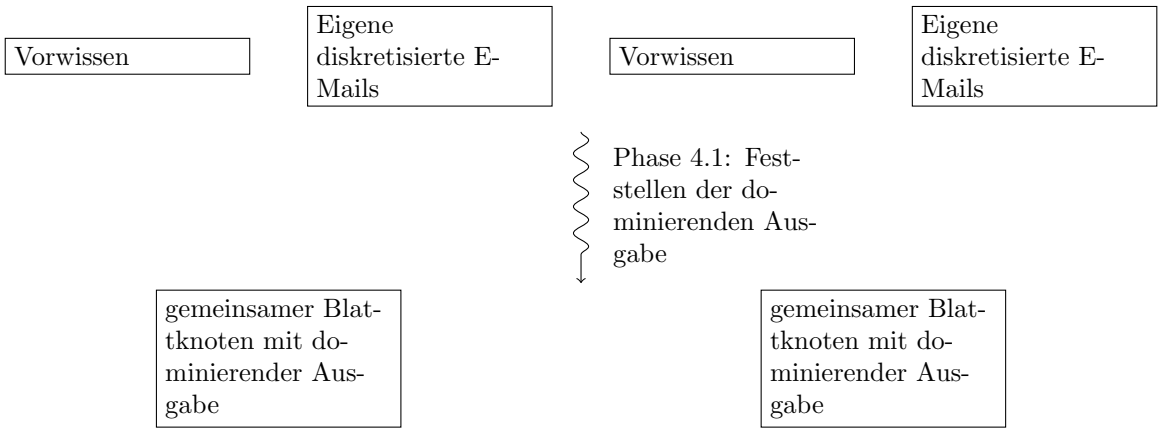



Figure 4: ID3-Algorithmus, Fall 1: Keine Attribute mehr vorhanden

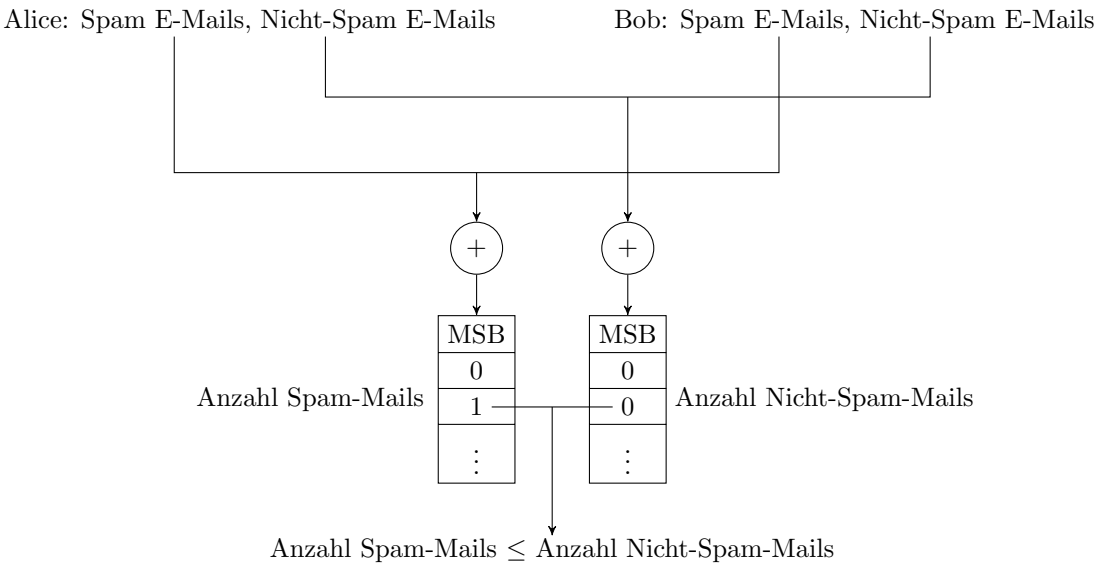



Figure 5: High-Level Struktur des Dominierende-Ausgabe-Schaltkreises

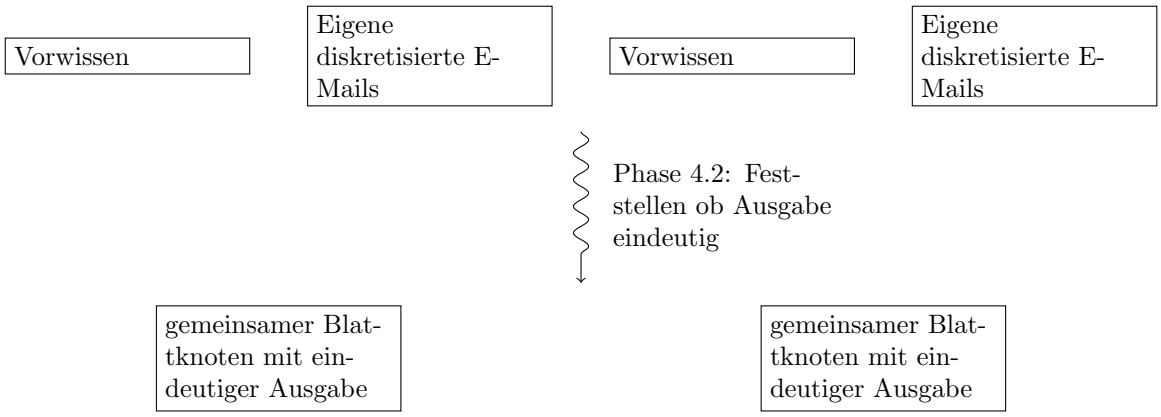



Figure 6: ID3-Algorithmus, Fall 2: Ausgabe eindeutig

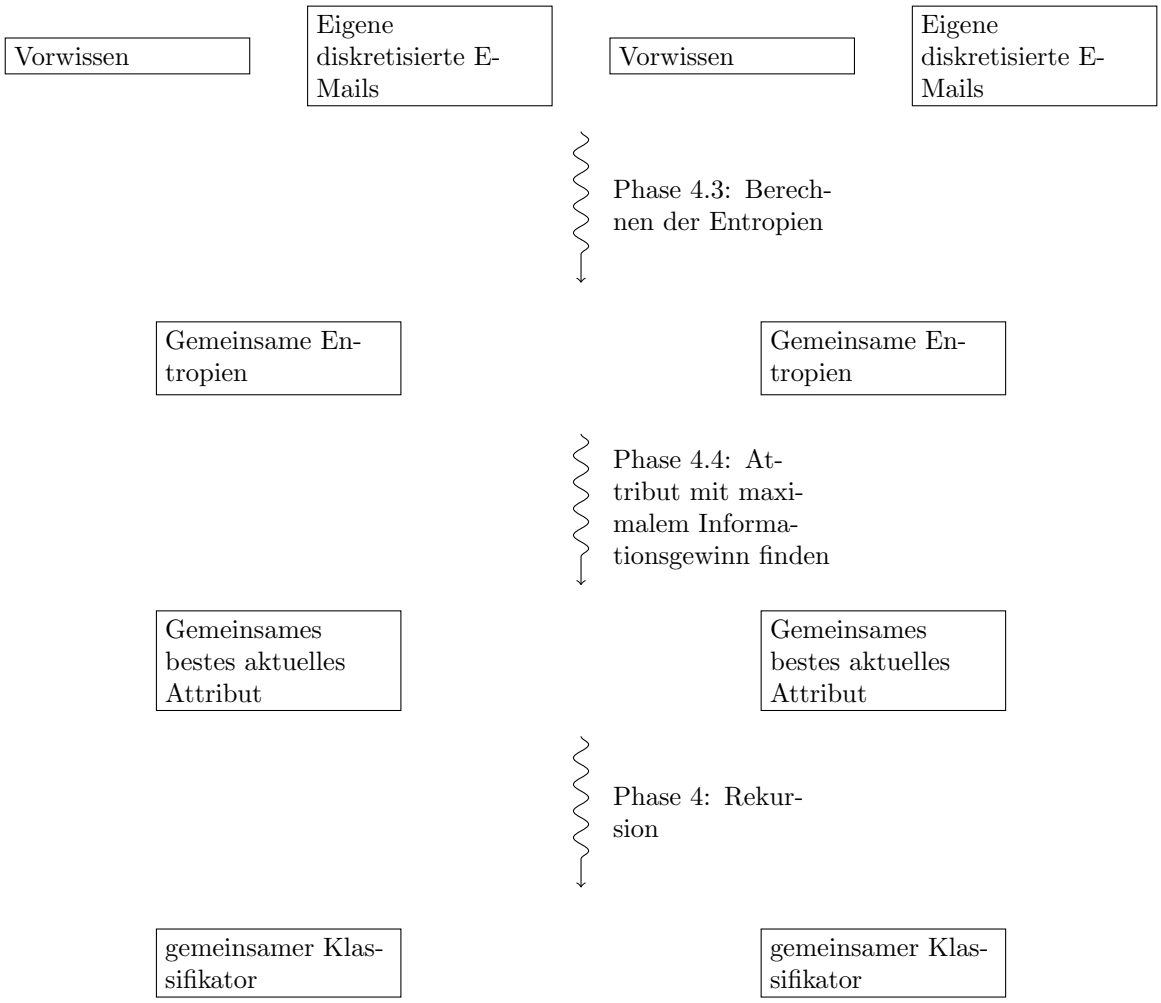



Figure 7: ID3-Algorithmus, Fall 3: Erzeugung eines Astes