Samenvatting

Deze thesis handelt over privacy preserving data mining. Data mining is een tak van de wetenschap waarin men grote hoeveelheden data onderzoekt met de bedoeling er bepaalde patronen in te ontdekken. Enkele belangrijke data mining algoritmes zullen aangehaald en besproken worden.
Dankzij de groeiende populariteit van data mining blijft het uitvoeren van de algoritmes niet beperkt tot één partij. Er zijn dikwijls meerdere partijen betrokken bij het data mining proces. Toch is het niet mogelijk om al de data zomaar samen te brengen. In veel gevallen zullen de betrokken partijen dit niet willen. Denk bijvoorbeeld aan concurrerende bedrijven of concurrerende afdelingen binnen een bedrijf die gemeenschappelijke data mining resultaten nodig hebben. Verder moet er ook rekening gehouden worden met de bestaande privacy wetten. Kortom, het wordt al snel duidelijk dat het samenbrengen van de gegevens geen mogelijke oplossing is.
Een tweede voorstel, dat op het eerste zicht veel haalbaarder lijkt, is het verwijderen van de identificatie gegevens uit de data. Toch gaat ook dit volledig in strijd met de privacy van de gegevens. Het is namelijk zo dat de mogelijkheid bestaat om uit de algemene gegevens de identificatie gegevens in zekere mate af te leiden. Deze situatie is gekend als het *inferentie probleem*.
Er zijn verschillende methodes bekend die het probleem van data minen in verdeelde data aanpakken. Deze thesis handelt grotendeels over één aanpak. Deze methode is gebaseerd op de theorie van *secure multiparty computation* en maakt gebruik van *encryptie*. De basis van deze aanpak ligt in het feit dat alle data mining problemen bepaalde gemeenschappelijke componenten hebben. Zo zal er bijvoorbeeld in veel gevallen een som moeten berekend worden waarvan elke partij één term heeft. Dit resultaat moet dan op zo een manier bepaald worden dat alle partijen door de uitvoering van het protocol niets meer leren dan het resultaat.
Dit vertoont heel veel gelijkenissen met de theorie van secure multiparty computation. De bedoeling bij secure multiparty computation is het berekenen van een functie, die haar input krijgt van verschillende partijen, op zo een manier dat alle partijen niets meer leren dan de uiteindelijke output. Het is aangetoond dat het zowel voor twee partijen als voor meer dan twee partijen mogelijk is om elke functie f waarbij de input verdeeld is over de partijen op zo een manier te berekenen dat de partijen niets meer leren dan de output. De methodes die worden voorgesteld in de secure multiparty computation zijn echter niet efficiënt genoeg om gebruikt te worden indien er meer dan

twee partijen zijn. Aangezien het bij data mining gaat om grote hoeveelheden data is het belangrijk om praktische oplossingen te hebben. Daarom zijn er nieuwe protocollen ontwikkeld voor al de aparte componenten zoals de hierboven aangehaalde som. Deze protocollen zijn zo opgesteld dat ze voldoen aan de veiligheidseisen uit de secure multiparty computation en toch hun efficiëntie behouden. Met behulp van deze bouwstenen kunnen er privacy preserving data mining algoritmes geconstrueerd worden. Zowel de bouwstenen als mogelijke algoritmes zullen in deze thesis verder beschreven worden.
Naast de literatuurstudie van het domein bevat deze thesis ook nog enkele uitbreidingen. Er werd onderzocht welke problemen nog geen concrete oplossingen hadden. Door gebruik te maken van de bouwstenen werden nieuwe algoritmes geconstrueerd om deze open problemen op te lossen. Verder werd er ook een speciale situatie bekeken waarin de data op een bijzondere manier verdeeld is over de verschillende partijen. Aan de hand van het ID3 algoritme wordt aangetoond dat ook deze situatie kan opgelost worden met behulp van de bouwstenen.
Deze thesis is grotendeels als volgt opgedeeld. In Hoofdstuk 1 wordt een korte inleiding gegeven tot data mining en worden de belangrijkste algoritmes besproken. Hoofdstuk 2 beschrijft de problematiek rond privacy preserving data mining en bespreekt de mogelijke oplossingen. Hoofdstukken 3 en 4 bevatten de nodige achtergrondinformatie in verband met encryptie en secure multiparty computation. In Hoofdstuk 5 worden de privacy preserving protocollen besproken en in Hoofdstuk 6 de privacy preserving algoritmes. Hoofdstukken 7 en 8 bevatten de uitbreidingen van de literatuur. In Hoofdstuk 7 worden enkele ontbrekende algoritmes aangehaald. Hoofdstuk 8 bevat een studie van een speciale verdeling van de data. Als laatste worden in het besluit de hoofdzaken nog eens herhaald en enkele ideëen in verband met toekomstig werk vermeld.

Voorwoord

Het kiezen van een thesis onderwerp is geen makkelijke beslissing. Het is dan ook iets waar je een heel jaar lang mee bezig moet zijn. Vanzelfsprekend dus dat je best iets zoekt waar je echt in geïnteresseerd bent. Toen de thesis voorstellen online kwamen te staan op het einde van onze eerste licentie ging ik meteen op zoek naar een boeiend onderwerp. Ik had graag iets gedaan in verband met data mining en mijn keuze was al snel gemaakt: Privacy Preserving Data Mining.

Het tot een goed einde brengen van deze thesis was niet altijd even voor de hand liggend. Ik zou dan ook graag alle mensen bedanken die mij hier bij geholpen hebben. Als eerste natuurlijk mijn ouders (en grootouders) omdat ik zonder hen nog niet eens aan deze studies had kunnen beginnen. Verder wil ik ook mijn promotor Prof. Bart Kuijpers en mijn co-promotor Dr. Karl Tuyls bedanken voor de begeleiding.

Ook mijn vriend wil ik bedanken voor de onvoorwaardelijke steun tijdens alle moeilijke periodes. En dat niet alleen tijdens het schrijven van mijn thesis, maar op alle momenten tijdens de afgelopen drie jaar. Niet te vergeten natuurlijk ook mijn vrienden voor het zorgen voor de soms wel heel broodnodige momenten van ontspanning. Als laatste wil ik ook mijn klasgenoten bedanken voor alle hulp en aanmoediging tijdens de meest stressvolle periodes in onze opleiding.

Na het schrijven van deze thesis kan ik met zekerheid zeggen dat ik veel heb bijgeleerd. Niet alleen over Privacy Preserving Data Mining, maar ook over inzet, discipline en doorzettingsvermogen.

Inhoudsopgave

Lijst van tabellen

Lijst van figuren

Lijst van algoritmes

Hoofdstuk 1

Inleiding tot data mining

1.1 Wat is data mining?

Data mining is een proces waarbij grote hoeveelheden data worden geanalyseerd om bepaalde patronen te ontdekken. Deze patronen kunnen dan geïnterpreteerd worden, wat nieuwe kennis over de data zal opleveren.
Data mining is een zeer populaire wetenschap en wordt dan ook in verschillende domeinen gebruikt [1]. Enkele van deze gebieden zijn onder andere de medische wereld, het verzekeringsleven, de verkoopswereld en de marketingsector.
Er bestaan allerhande data mining methodes die verschillende soorten patronen opleveren. Grotendeels kunnen al deze algoritmes opgedeeld worden in twee soorten. In een eerste aanpak draait het om het analyseren en beschrijven van de data. Hierin horen onder andere associatie analyse, cluster analyse en outlier analyse thuis. De tweede aanpak handelt over het classificeren van nieuwe gegevens. Een voorbeeld hiervan zijn onder andere decision trees en Naïve Bayes. Bij deze methodes draait het om het classificeren van nieuwe data in een bepaalde klasse.
De volgende sectie zal enkele belangrijke data mining methodes bespreken. Meer informatie over deze en andere algoritmes kan gevonden worden in [2] en [3].

1.2 Enkele data mining algoritmes

De algoritmes maken allemaal gebruik van gegevens uit een database. Zo een database heeft een aantal attributen en een aantal transacties. Een transactie wordt in deze context gezien als een tuple uit de database. Elke transactie kent dan een waarde toe aan de attributen.

1.2.1 Associatie analyse

Een associatie regel is een implicatie van de vorm $X \Rightarrow Y$, waarbij X en Y verzamelingen van attributen zijn. In de context van een winkel kan de regel $X \Rightarrow Y$ dan net zoveel zeggen als: als een klant product X koopt, zal hij ook product Y kopen. Een associatie regel heeft ook een *support* en een *confidence*. Het support van de regel $X \Rightarrow Y$ is het percentage transacties die X en Y bevatten. Het confidence is het percentage transacties, van de transacties die X bevatten, die zowel X als Y bevatten. Dit geeft de volgende resultaten:

$$support(X \Rightarrow Y) = aantal\ transacties\ met\ X\ en\ Y$$

$$confidence(X \Rightarrow Y) = \frac{aantal\ transacties\ met\ X\ en\ Y}{aantal\ transacties\ met\ X}$$

1.2.1.1 Het Apriori algoritme

Het Apriori algoritme krijgt als input een database met transacties en een minimumwaarde voor support en confidence. Het algoritme bestaat grotendeels uit twee delen. Deze zijn het opsporen van itemsets met een groot genoeg support en het bepalen van de associatie regels waarvan het confidence boven de vooropgestelde waarde ligt.
Om de geschikte itemsets te vinden, worden er twee soorten verzamelingen gevormd, namelijk een verzameling met kandidaat k-itemsets en een verzameling met frequent k-itemsets. De verzameling C_k met kandidaat k-itemsets bevat itemsets van grootte k die mogelijk frequent kunnen zijn. Uit deze wordt dan de verzameling L_k met frequent k-itemsets berekend. Dit zijn degene waarvan het support groot genoeg is.
Uit de gevonden itemsets kunnen dan kandidaat regels worden opgesteld. Deze regels zullen getest worden op hun confidence. Degene die slagen voor deze test vormen de output van het algoritme.

1.2.1.2 Voorbeeld

Een voorbeeld zal de werking van het Apriori algoritme verder verduidelijken. Beschouw de database met transacties in Tabel 1.1 en veronderstel een support van 50% en een confidence van 75%. De volledige uitwerking van het algoritme kan gevolgd worden in Tabel 1.2 en 1.3.

Opstellen frequent itemsets Als eerste worden de kandidaat 1-itemsets opgesteld. Dit wil dus zeggen dat elk item een aparte itemset zal vormen. Voor elke itemset wordt dan het support berekend.

ID	Items
1	A, C, D
2	B, C, E
3	A, B, C, E
4	B, E

Tabel 1.1: Transactie database met itemsets.

Degene met een support hoger dan 50% vormen de frequent 1-itemsets. Op basis van deze worden de kandidaat 2-itemsets berekend. Zo gaat het algoritme verder tot er geen kandidaat itemsets meer kunnen gegenereerd worden. In dit geval is dat bij de vierde iteratie.
Merk op dat bij het opstellen van de kandidaat 3-itemsets niet alle mogelijke itemsets voorkomen. Zo zitten bijvoorbeeld $\{A, B, C\}$ en $\{A, C, E\}$ niet bij de kandidaat 3-itemsets. Dit komt omdat de respectievelijke subsets $\{A, B\}$ en $\{A, E\}$ niet bij de frequent 2-itemsets horen. Deze eigenschap staat bekend als het zogenaamde Apriori principe.

Opstellen associatie regels Hierna zullen met de bekomen frequent itemsets mogelijke associatie regels gevormd worden. Deze worden dan getest op hun confidence. Aangezien er een confidence van 75% gevraagd wordt, zal het algoritme alleen die vijf regels selecteren met een confidence van 100%.

1.2.2 Decision trees

Bij het opstellen van decision trees wordt een transactie database beschouwd met attributen en één classificatie of class attribuut. Een decision tree is een boom waarin elke knoop een attribuut voorstelt. Een knoop heeft dan net zoveel takken als het attribuut mogelijke waardes heeft. Elk blad van een decision tree kent een bepaalde classificatie toe. Een decision tree kan dan ook gezien worden als een verzameling regels. Elke regel stelt dan een pad voor van de wortel naar een blad.

1.2.2.1 Het ID3 algoritme

Het ID3 algoritme zal de boom top down construeren. Het algoritme begint dus bij de wortel en zal zo toewerken naar de bladeren. Bij elke knoop zal het attribuut gekozen worden dat de transacties het beste classificeert. Om dit te bepalen wordt gebruikt gemaakt van een maat, namelijk de *information gain*. Deze maakt gebruik van de *entropy*. De entropy drukt de wanorde uit in de transacties. Het is nu uiteraard de bedoeling om die wanorde zo klein

Kandidaat 1-Itemsets C_1	**Support**
$\{A\}$	50%
$\{B\}$	75%
$\{C\}$	75%
$\{D\}$	25%
$\{E\}$	75%

Frequent 1-Itemsets L_1
$\{A\}$
$\{B\}$
$\{C\}$
$\{E\}$

Kandidaat 2-Itemsets C_2	**Support**
$\{A, B\}$	25%
$\{A, C\}$	50%
$\{A, E\}$	25%
$\{B, C\}$	50%
$\{B, E\}$	75%
$\{C, E\}$	50%

Frequent 2-Itemsets L_2
$\{A, C\}$
$\{B, C\}$
$\{B, E\}$
$\{C, E\}$

Kandidaat 3-Itemsets C_3	**Support**
$\{B, C, E\}$	50%

Frequent 3-Itemsets L_3
$\{B, C, E\}$

Tabel 1.2: Bepalen van de frequent itemsets.

Associatie regels	**Confidence**
$A \Rightarrow C$	100%
$C \Rightarrow A$	66,6%
$B \Rightarrow C$	66,6%
$C \Rightarrow B$	66,6%
$B \Rightarrow E$	100%
$E \Rightarrow B$	100%
$C \Rightarrow E$	66,6%
$E \Rightarrow C$	66,6%

Associatie regels	**Confidence**
B *en* $C \Rightarrow E$	100%
B *en* $E \Rightarrow C$	66,6%
C *en* $E \Rightarrow B$	100%
$B \Rightarrow C$ *en* E	66,6%
$C \Rightarrow B$ *en* E	66,6%
$E \Rightarrow B$ *en* C	66,6%

Tabel 1.3: Bepalen van de associatie regels.

mogelijk te krijgen. De information gain is dan ook een maat die voor elk attribuut kan berekend worden en uitdrukt hoeveel de entropy zal verminderen bij een opdeling volgens dat attribuut. Er wordt telkens gekozen voor het attribuut met de grootste information gain. De volledige uitwerking van het ID3 algoritme kan gevonden worden in Algoritme 1.

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

$$Gain(A) = Entropy(T) - \sum_{j=1}^{m} \frac{|T(a_j)|}{|T|} Entropy(T(a_j))$$

Hierin stellen $c_1, ..., c_l$ de waardes van het class attribuut voor, $T(c_i)$ de transacties met c_i bij het class attribuut, $a_1, ..., a_m$ de waardes van het attribuut A en $T(a_j)$ de transacties met a_j bij het attribuut A.

Algoritme 1 Het ID3 algoritme

Require: R, de set van attributen.
Require: C, het class attribuut.
Require: T, de set van transacties.
if R is leeg **then**
    Return een blad met als class waarde, de waarde die het meeste voorkomt bij de transacties in T.
else if Alle transacties in T hebben dezelfde class waarde **then**
    Return een blad met als class waarde die waarde.
else
    Bepaal het attribuut A dat de transacties in T het beste classificeert.
    Verdeel T in m delen $T(a_1), ..., T(a_m)$ zodat $a_1, ..., a_m$ de verschillende waardes van attribuut A zijn.
    Return een boom met als wortel een knoop A en m takken $a_1, ..., a_m$ zodat tak i $ID3(R - \{A\}, C, T(a_i))$ bevat.
end if

1.2.2.2 Voorbeeld

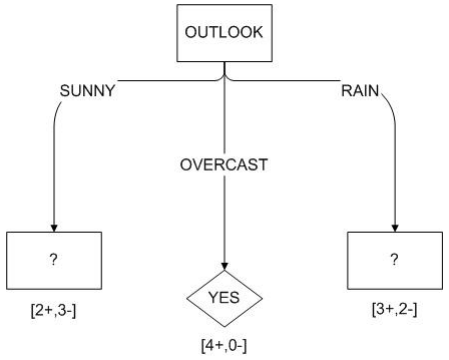
Een voorbeeld transactie database kan gevonden worden in Tabel 1.4. Dit voorbeeld komt uit [3]. Het gaat hier over gegevens in verband met het weer en hun invloed op het spelen van tennis. De attributen zijn Outlook, Temperature, Humidity en Wind. Het class attribuut is Play Tennis. Om te beginnen moet de wortel bepaald worden. Hiervoor zal de information gain berekend worden:

- Gain(Outlook) = 0.246,
- Gain(Humidity) = 0.151,
- Gain(Wind) = 0.048,
- Gain(Temperature) = 0.029.

Hieruit kan besloten worden dat Outlook de wortel van de boom wordt. De mogelijke waardes van het Outlook attribuut zijn Sunny, Overcast en Rain. De wortel zal dus drie overeenkomstige takken krijgen. Dit kan gezien worden in Figuur 1.1.
Vervolgens moet voor elke tak van de wortel een nieuwe knoop bepaald worden. Voor de eerste tak zal de information gain berekend moeten worden voor de drie overblijvende attributen: Humidity, Wind en Temperature. Om deze gain te berekenen moeten alleen de transacties die Sunny als Outlook hebben beschouwd worden. Dit zal dan de volgende resultaten geven:

- Gain(Humidity) = 0.970,
- Gain(Temperature) = 0.570,
- Gain(Wind) = 0.019.

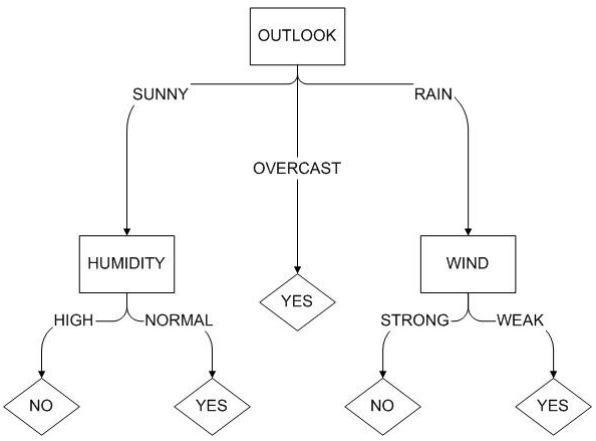
Humidity zal hier gekozen worden als attribuut voor die knoop. Voor de tweede tak zal volgens het ID3 algoritme geen information gain moeten bepaald worden omdat alle beschouwde transacties dezelfde class waarde hebben. Er zal dan ook een blad geconstrueerd worden met als waarde de bijhorende classificatie. De derde tak zal weer analoog verlopen aan de eerste. De uiteindelijke decision tree kan gevonden worden in Figuur 1.2.



Figuur 1.1: De wortel van de decision tree.

Day	Outlook	Temperature	Humidity	Wind	Play Tennis
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Weak	Yes
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
10	Rain	Mild	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No

Tabel 1.4: Transactie database met weer gegevens.



Figuur 1.2: De volledige decision tree.

De regels die uit deze decision tree kunnen gehaald worden zijn dan:

- If Outlook = Sunny and Humidity = High then Play Tennis = No,
- If Outlook = Sunny and Humidity = Normal then Play Tennis = Yes,
- If Outlook = Overcast then Play Tennis = Yes,
- If Outlook = Rain and Wind = Strong then Play Tennis = No,
- If Outlook = Rain and Wind = Weak then Play Tennis = Yes.

Aan de hand van de decision tree of de bijhorende regels kunnen dan nieuwe transacties geclassificeerd worden. Neem bijvoorbeeld de transactie (Outlook = Sunny, Temperature = Cool, Humidity = High, Wind = Strong). Om het class attribuut van deze transactie te bepalen moet er een pad gevonden worden van de wortel naar een blad, dat overeenkomt met de waardes in de transactie. De waarde van het blad zal dan de classificatie zijn. In dit geval kiest men in de wortel de tak Sunny en in de Humidity knoop de tak High. Dit komt dan uit in een blad met als waarde No.

1.2.3 Naïve Bayes

Er zijn verschillende classifier algoritmes gebaseerd op de stelling van Bayes. Deze stelling zegt het volgende:

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)}$$

Deze stelling drukt de kans uit op h, gegeven D. In deze notatie staat h voor een hypothese en D voor de data. Dit kan gezien worden als de kans op classificatie h, gegeven een transactie database D. Eén van de mogelijke classifiers die gebruik maken van deze stelling is Naïve Bayes.

1.2.3.1 De Naïve Bayes methode

Naïve Bayes is een zeer praktische classifier en heeft in bepaalde domeinen een zeer goede betrouwbaarheid. De classifier zal aan de hand van gegeven transacties een nieuwe transactie correct proberen te classificeren:

$$
\begin{aligned}
C_{MAP} &= argmax_{c_i} P(c_j|a_1, a_2, ..., a_n) \\
&= argmax_{c_i} \frac{P(a_1,a_2,...,a_n|c_j)P(c_j)}{P(a_1,a_2,...,a_n)} \\
&= argmax_{c_i} P(a_1, a_2, ..., a_n|c_j)P(c_j) \\
&= argmax_{c_i} P(c_j) \prod_i P(a_i|c_j)
\end{aligned}
$$

Hierin stellen c_i de class attribuut waardes voor en a_j de waardes van de attributen in de nieuwe transactie.

Zoals gemerkt kan worden in de afleiding van de formule is de Naïve Bayes methode gebaseerd op de veronderstelling dat de attributen conditioneel onafhankelijk zijn. Dit is dan ook de reden dat de classifier naïef genoemd wordt. Het bepalen van de kansen $P(X = x|c_i)$ verloopt verschillend voor nominale en numerieke attributen.

Nominale attributen $P(X = x|c_i) = \frac{n_j}{n}$ met n het aantal transacties die c_i als class attribuut hebben en n_j het aantal transacties die c_i als class attribuut hebben en de waarde x bij het attribuut X.

Numerieke attributen $P(X = x|c_i) = \frac{1}{\sqrt{2\pi}\sigma} e^{\frac{-(x-\mu)^2}{2\sigma^2}}$. Het gemiddelde μ en de variantie σ^2 kunnen voor elk numeriek attribuut in de transactie database bepaald worden.

1.2.3.2 Voorbeeld

Beschouw opnieuw de transactie database in Tabel 1.4. In de uitwerking van het decision tree voorbeeld werd eerst een decision tree opgebouwd. Met behulp van deze boom werd dan een classificatie toegekend aan een nieuwe transactie. Dit zou nu ook kunnen met de Naïve Bayes classifier. Beschouw hiervoor de transactie (Outlook = Sunny, Temperature = Cool, Humidity = High, Wind = Strong) die volgens de decision tree als No geclassificeerd wordt. De uitwerking volgens de Naïve Bayes classifier zal dan als volgt verlopen:

- P(No).P(Sunny|No).P(Cool|No).P(High|No).P(Strong|No)
  = 5/14 . 3/5 . 1/5 . 4/5 . 3/5 = 0.0206
- P(Yes).P(Sunny|Yes).P(Cool|Yes).P(High|Yes).P(Strong|Yes)
  = 9/14 . 2/9 . 3/9 . 3/9 . 3/9 = 0.0053

Het resultaat zal hier No zijn omdat $0.0206 > 0.0053$. De classificatie met Naïve Bayes geeft dus hetzelfde resultaat als met de decision tree.

1.2.4 Cluster analyse

Het doel bij cluster analyse is het verdelen van de transacties in clusters. Dit moet op zo een manier gebeuren dat elke transactie zo veel mogelijk gemeen heeft met de transacties in dezelfde cluster, en zo verschillend mogelijk is van de transacties in de andere clusters. Er zijn verschillende cluster methodes. Twee van deze methodes zijn enerzijds partition based clustering en anderzijds density based clustering.

1.2.4.1 Partition based clustering

Het k-means algoritme Bij het k-means algoritme wordt op zoek gegaan naar k cluster centers. Het algoritme begint met random waardes toe te kennen aan de k centers. Elke transactie in de database wordt dan toegekend aan één van de k clusters. Namelijk degene die de transactie het beste benadert. Na zo een toekenning moeten de k centers herberekend worden en kunnen de transacties opnieuw toegewezen worden. De volledige beschrijving van het algoritme kan gevonden worden in Algoritme 2.
Om te bepalen bij welke cluster een transactie hoort moet de afstand tussen de transactie en de cluster centers bepaald worden. Dit kan met behulp van verschillende afstandsmaten. Twee hiervan zijn de Euclidische en de Manhattan afstand. Voor twee punten p_1 en p_2 met respectievelijke coördinaten (x_1, y_1) en (x_2, y_2) kunnen deze maten als volgt gedefinieerd worden:

$$\textit{Euclidische afstand } d(p_1, p_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

$$\textit{Manhattan afstand } d(p_1, p_2) = |(x_1 - x_2)| + |(y_1 - y_2)|$$

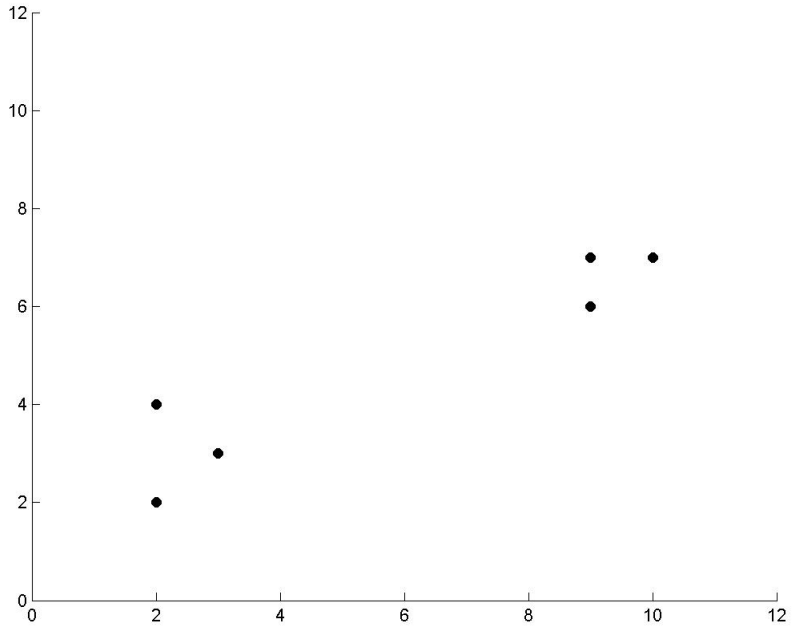
Algoritme 2 Het k-means algoritme

Require: T, de set van transacties.
Require: $C_1, ..., C_k$, de k clusters.
Require: $k_1, ..., k_k$, de k cluster centers.
1: Ken random waardes toe aan $k_1, ..., k_k$.
2: **while** De cluster centers $k_1, ..., k_k$ veranderen **do**
3: Ken elke transactie uit T toe aan een cluster $C_1, ..., C_k$.
4: Bereken nieuwe cluster centers $k_1, ..., k_k$ van $C_1, ..., C_k$.
5: **end while**

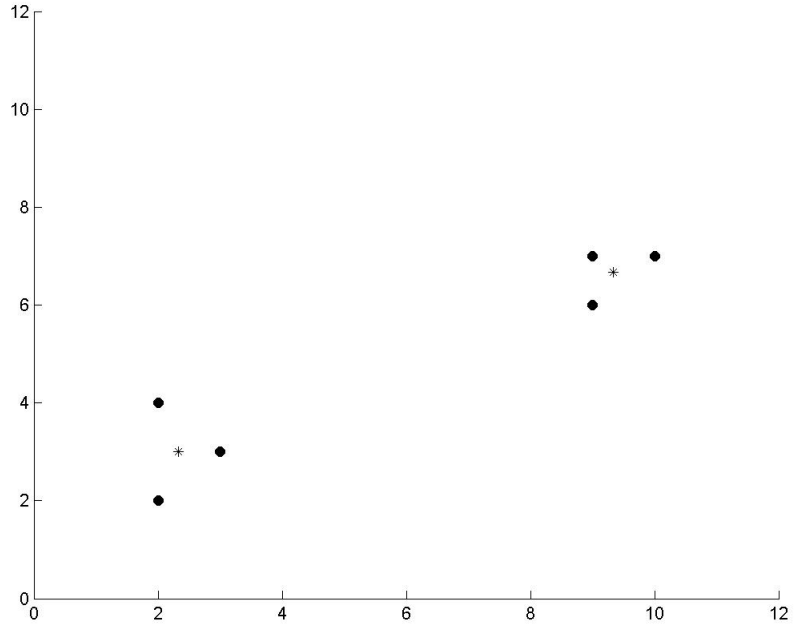
X coörd	Y coörd
2	2
2	4
3	3
9	6
9	7
10	7

Tabel 1.5: Transactie database met coördinaten.

Voorbeeld Beschouw de database weergegeven in Tabel 1.5. De grafische weergave van deze transacties kan gevonden worden in Figuur 1.3. Op deze grafiek is het duidelijk dat de punten opgedeeld zijn in twee clusters. Het uitvoeren van het k-means algoritme op twee clusters geeft dan ook het verwachte resultaat in Figuur 1.4. De cluster centers liggen op (2,3 ; 3) en (9,3 ; 6,6) en zijn aangegeven door een sterretje.



Figuur 1.3: De coördinaten grafisch weergegeven.



Figuur 1.4: Het resultaat van het k-means algoritme met $k = 2$.

1.2.4.2 Density based clustering

Het DBSCAN algoritme De clusters die opgespoord worden bij het DBSCAN algoritme zijn dense regio's. Om deze regio's te ontdekken maakt het algoritme gebruik van enkele parameters. Deze parameters zijn twee waardes, namelijk ϵ en $minPts$. Het algoritme begint met de punten te onderzoeken en te bepalen welke punten *core punten* zijn. Een punt is een core punt als het binnen een afstand ϵ minstens $minPts$ punten heeft. Alle punten q die binnen een afstand ϵ liggen van een core punt p zijn directly density reachable vanuit p.
Het algoritme vormt clusters door in de eerste stap voor elk core punt een cluster te vormen. Deze cluster bevat dan alle punten die directly density reachable zijn vanuit dat core punt. Wanneer een core punt directly density reachable is vanuit een ander core punt worden er clusters samengevoegd. Het kan voorkomen dat een punt net op de grens ligt van twee clusters. Zo een punt kan dan aan één van beide clusters worden toegewezen.

Voorbeeld Beschouw de transacties in Figuur 1.5. Het DBSCAN algoritme zal de hele dense regio als één cluster herkennen. Dit in tegenstelling tot het k-means algoritme, dat de cluster zou opdelen in k partities.

1.2.5 Outlier analyse

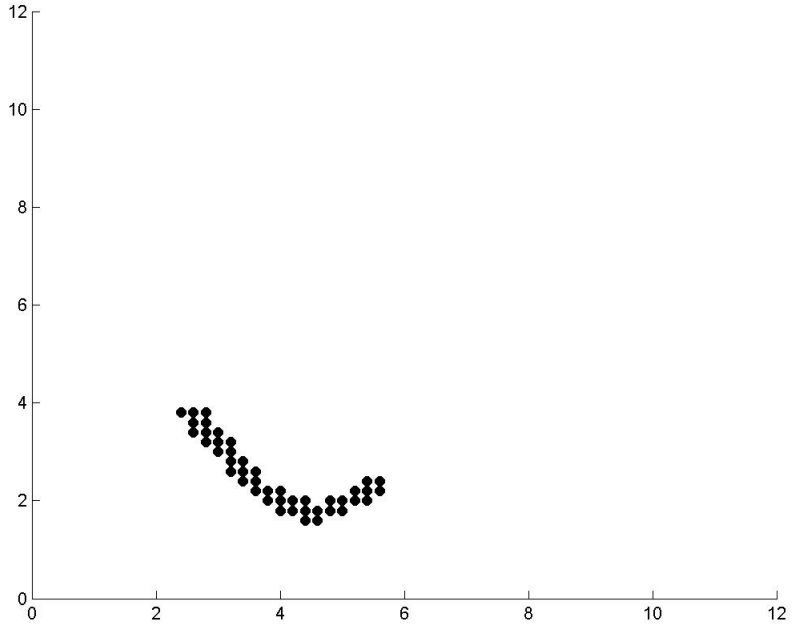
Een outlier is een bepaalde transactie die duidelijke verschillen vertoont van de rest van de data. Het doel van outlier analyse is dergelijke inconsistente transacties opsporen. Er zijn verschillende aanpakken om outliers op te sporen. Eén van deze methodes is outliers opsporen op basis van hun afstand tot de overige transacties.

1.2.5.1 De distance based outlier detection methode

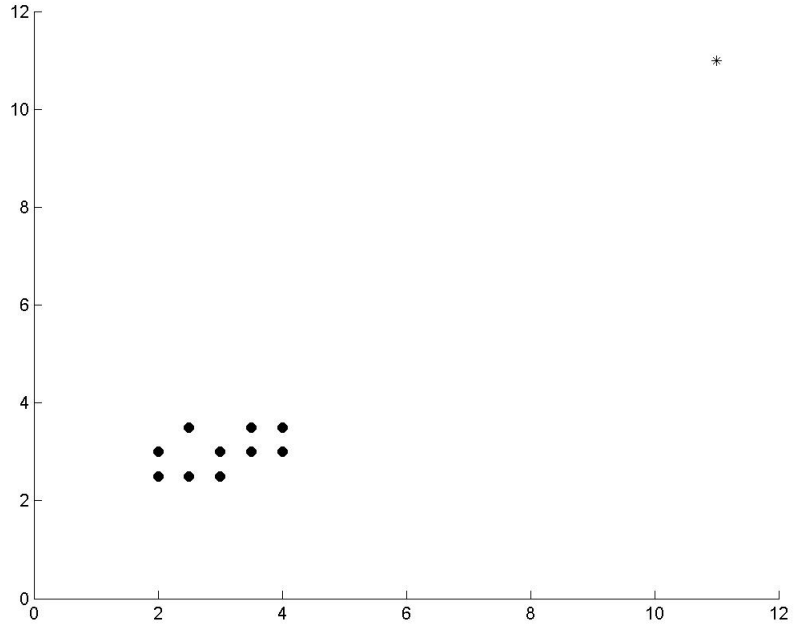
Een transactie is een *(p,d)-outlier* als er minstens p punten binnen een afstand d van dat punt liggen.

1.2.5.2 Voorbeeld

Beschouw de transacties in Figuur 1.6. Het is duidelijk dat het punt aangegeven door een sterretje een outlier is.



Figuur 1.5: Het resultaat van het DBSCAN algoritme.

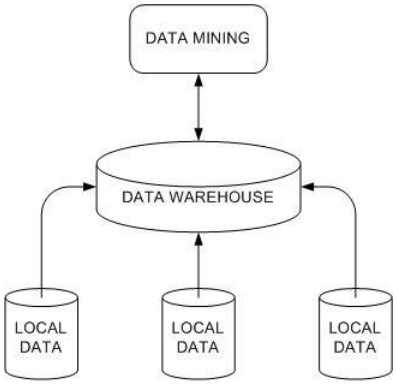


Figuur 1.6: Het resultaat van het distance based outlier detection algoritme.

Hoofdstuk 2

Van data mining naar privacy preserving data mining

Het uitvoeren van data mining algoritmes blijft niet beperkt tot gegevens uit één database. Er zijn dikwijls verschillende partijen betrokken bij het data mining proces. Al deze partijen willen hun gegevens samenbrengen om gemeenschappelijke data mining resultaten verkrijgen. Dankzij alle technologische vooruitgangen op het gebied van ICT is dit zeker geen ondenkbaar scenario. Denk bijvoorbeeld aan het combineren van ziekenhuis gegevens over patiënten met verkoopgegevens van klanten in een winkel. Misschien komen er zo verbanden aan het licht tussen bepaalde voedingswaren en gezondheidsproblemen. Of het samenbrengen van de data van verschillende filialen van een winkelketen om een grotere omzet te bereiken. De mogelijkheden van data mining zijn op zich al heel groot. Dit blijkt uit de vele domeinen waarin data mining gebruikt wordt. Wanneer nu deze domeinen hun gegevens samen kunnen gebruiken om aan data mining te doen, zullen de resultaten nog krachtiger worden.
Uiteraard zal dit niet zomaar gaan. Een naïeve oplossing zou zijn om, zoals weergegeven in Figuur 2.1, alle gegevens samen te brengen en de data mining algoritmes los te laten op de verzamelde data. Hierbij zullen vanzelfsprekend ontelbaar veel privacy problemen komen kijken. Ziekenhuizen en winkels kunnen niet zomaar vertrouwelijke gegevens over hun patiënten en klanten vrijgeven. Vaak zal ook de concurrentiegeest een belangrijke rol spelen. Bedrijven zullen hun gegevens uit de handen van de concurrentie willen houden omwille van voor de hand liggende problemen.
Er kan dus besloten worden dat het samenbrengen van alle data een onmogelijke opdracht is. Een betere oplossing voor deze problematiek zou kunnen zijn om de identificatie gegevens uit de data te halen. Zoals zal blijken uit de volgende sectie is ook dit geen veilige uitweg.



Figuur 2.1: Data verzamelen in een data warehouse.

2.1 Het inferentie probleem

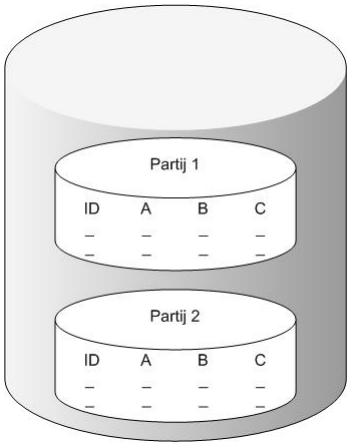
Het inferentie probleem is ontstaan in database onderzoek uit de jaren '70. In deze context had het inferentie probleem te maken met ongeautoriseerde toegang tot databases. Dezelfde term wordt nu gebruikt in het onderzoek naar privacy preserving data mining. Hier handelt het inferentie probleem over het afleiden van identificatie gegevens uit meer algemene data. In deze context worden de identificatie gegevens high data genoemd. Dit zijn de gegevens die geheim moeten blijven. Hieronder kan bijvoorbeeld de naam van een persoon thuishoren. De meer algemene gegevens zijn low data en in deze categorie horen onder andere postcodes en geboortedatums thuis. Van de low data wordt aangenomen dat deze wel gekend mag zijn [5].
Sweeney bestudeerde in [6] de mogelijkheid om uit low data, high data af te leiden. Er werd een studie gedaan met behulp van de stemlijst van Cambridge Massachusetts. Deze lijst bevatte gegevens over meer dan 54000 personen. De namen van de personen werden uit de lijst verwijderd en wat overbleef werd onderzocht. Eén van de opmerkelijke resultaten was dat 69% van de personen uniek konden geïdentificeerd worden op hun postcode en geboortedatum. Dit wil dus zeggen dat indien de postcode en de geboortedatum van een bepaalde persoon gekend zijn, hiermee ook al zijn andere gegevens worden vrijgegeven.
Dit is iets wat uiteraard niet kan getolereerd worden. Het inferentie probleem zorgt ervoor dat verborgen informatie toch nog kan afgeleid worden. Op deze manier gaat alle privacy natuurlijk verloren. Daarom kan ook het verwijderen van identificatie gegevens niet als een risicoloze oplossing gezien worden. Het is dus nodig om een andere en meer doeltreffende aanpak te bepalen.

2.2 Horizontaal en verticaal verdeelde data

Een bijkomende kwestie is de manier waarop de data verdeeld is. In de literatuur worden twee mogelijke verdelingen onderzocht: een horizontale en een verticale opsplitsing.

2.2.1 Horizontaal verdeelde data

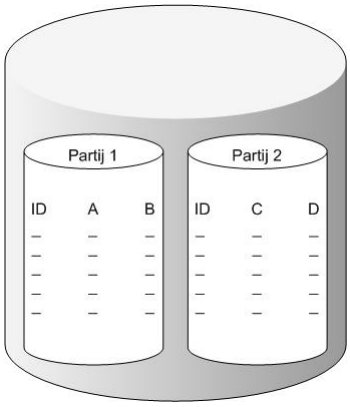
Bij een horizontale verdeling (of homogene verdeling) hebben alle partijen een database met dezelfde attributen. Een voorbeeld van zo een horizontale verdeling is een winkelketen met verschillende filialen. Deze opdeling kan gevonden worden in Figuur 2.2. Aangezien bij een horizontale verdeling alle partijen dezelfde attributen hebben, kan er niets vrijgegeven worden door de attributen te leren.



Figuur 2.2: Horizontaal verdeelde data.

2.2.2 Verticaal verdeelde data

Bij een verticale verdeling (of heterogene verdeling) hebben alle partijen een database met verschillende attributen en één gemeenschappelijk join attribuut. Een voorbeeld van een verticale verdeling is het combineren van de gegevens van een ziekenhuis en een winkel. Deze opdeling kan gevonden worden in Figuur 2.3. Bij een verticale verdeling kunnen er privacy problemen ontstaan wanneer partijen elkaars attributen leren. Dit is dan ook iets waar rekening mee gehouden moet worden bij het opstellen van privacy preserving data mining algoritmes.



Figuur 2.3: Verticaal verdeelde data.

2.2.3 Formele definitie

De hierboven beschreven verdelingen kunnen als volgt formeel worden weergegeven:

Neem T de set van transacties, die verdeeld is in delen T_{ij}. De i duidt hier op een verdeling van de attributen, dus een verticale verdeling. De j geeft een verdeling van de transacties weer, wat een horizontale verdeling voorstelt. Er zijn partijen P_{ij} die de transactiesets T_{ij} bezitten. Het aantal horizontale verdelingen is gelijk aan het aantal $j's$, noem dit s. Het aantal verticale verdelingen is gelijk aan het aantal $i's$, noem dit r.

Horizontaal De data is horizontaal verdeeld indien $s > 1$ en $r = 1$.

Verticaal De data is verticaal verdeeld indien $r > 1$ en $s = 1$.

2.3 Stand van zaken in privacy preserving data mining

In het onderzoek naar privacy preserving data mining zijn er twee strekkingen te vinden. In een eerste methode gaat het over het beschermen van de output. De data wordt aangepast zodat gevoelige resultaten verborgen blijven.
Een tweede methode handelt over het beschermen van de input. Alle resultaten mogen dus geleerd worden maar de data zelf moet geheim blijven.

2.3.1 Het beschermen van de output

Bij deze methode wil men de data op zo een manier aanpassen dat bepaalde gevoelige resultaten verborgen blijven. De optimale aanpassing vinden is een NP hard probleem. Daarom worden er heuristieken gebruikt die de optimale oplossing zo goed mogelijk benaderen. Om deze reden worden dergelijke methodes ook wel heuristiek gebaseerde methodes genoemd.
Een eerste manier om de data aan te passen is het veranderen van bepaalde waardes. Zo kan een nul-waarde bijvoorbeeld omgezet worden in een één-waarde of omgekeerd. Een toepassing van deze techniek kan gevonden worden in [9]. De methode die hier wordt voorgesteld verandert één-waardes in nul-waardes zodat het support van gevoelige regels daalt. Er moet wel op gelet worden dat deze aanpassingen zo weinig mogelijk invloed hebben op de andere regels. In [10] wordt verder gewerkt met deze methode. Hier wordt naast het support, ook geprobeerd om het confidence van de gevoelige regels te verminderen. Op deze manier kan dan voorkomen worden dat deze regels in het resultaat zitten.
Een tweede manier om de data aan te passen, is het weglaten van bepaalde waardes. Dit kan voor sommige toepassingen betere resultaten geven dan het veranderen van de waardes. Methodes die van deze techniek gebruik maken kunnen gevonden worden in [11] en [12].

2.3.2 Het beschermen van de input

Voor het beschermen van de input zijn er twee werkwijzes. Een eerste manier is het vermommen van de gegevens alvorens ze vrij te geven. De vermomde gegevens worden dan verzameld en aan de hand van bepaalde reconstructie methodes wordt de verdeling van de gegevens gereconstrueerd. Deze reconstructie zal dan gebruikt worden om data mining resultaten te verkrijgen.
Een tweede manier is het geheim houden van de gegevens. De gegevens worden niet vrijgegeven en alle partijen zullen samenwerken om data mining resultaten te leren. Deze methode maakt gebruik van encryptie.

2.3.2.1 Reconstructie gebaseerde methodes

Er zijn verschillende voorstellen die gebruik maken van reconstructie gebaseerde methodes. In [13] wordt een methode besproken waarbij met gereconstrueerde data een decision tree gebouwd wordt. De reconstructie is in dit geval gebaseerd op de stelling van Bayes. Er wordt hier eveneens een maat voorgesteld om de privacy van de vermomming te meten. In [14] wordt aangetoond dat deze maat geen rekening houdt met de kennis van de recon-

structie en dus een foutief beeld geeft. Er wordt ook een andere reconstructie methode geïntroduceerd gebaseerd op het Expectation Maximization algoritme. Verdere methodes voor het leren van associatie regels met behulp van reconstructie gebaseerde methodes kunnen gevonden worden in [15] en [16].

2.3.2.2 Cryptografie gebaseerde methodes

Over deze aanpak zal de rest van deze thesis handelen. Aan de basis van deze aanpak liggen enkele onderzoeken naar de aard van privacy preserving data mining problemen. Deze leverden namelijk de waarneming op dat elk probleem grotendeels afhangt van drie parameters:

- Welke data mining aanpak er gebruikt wordt.
- Welke privacy constraints er concreet zijn.
- Op welke manier de data verdeeld is, namelijk horizontaal of verticaal.

Verschillende combinaties van deze parameters leveren verschillende problemen op. Toch hebben deze problemen bepaalde gemeenschappelijke componenten. Zo zal er bijvoorbeeld dikwijls een som moeten berekend worden, waarbij de termen van de som verdeeld zijn over verschillende partijen. Bij het berekenen van zo een component is de input verspreid over meerdere partijen. De partijen mogen echter alleen het resultaat leren en niet de waardes van de andere partijen want dan zou de privacy geschonden worden. Dit vertoont heel veel gelijkenissen met de theorie van *secure multiparty computation*. De bedoeling bij secure multiparty computation is het berekenen van een functie, die haar input krijgt van verschillende partijen, op zo een manier dat alle partijen niets meer leren dan de uiteindelijke output.
Het idee is om een *trusted party* te simuleren. Zo een partij ontvangt alle input, berekent het resultaat en maakt dat bekend. De andere partijen leren hierdoor niets buiten de output. Bij secure multiparty computation wil men hetzelfde resultaat verkrijgen als met een trusted party zonder echt een extra partij te hebben. De partijen werken samen om de output te leren. Ze komen echter niets te weten over de input van de andere partijen. Deze theorie steunt op bepaalde concepten uit de cryptografie. Enkele belangrijke zaken in verband met cryptografie en secure multiparty computation worden meer in detail aangehaald in Hoofdstuk 3 en Hoofdstuk 4.
Het is aangetoond dat het zowel voor twee partijen als voor meer dan twee partijen mogelijk is om elke functie f waarbij de input verdeeld is over de partijen op zo een manier te berekenen dat de partijen niets meer leren dan de output. De methodes die worden voorgesteld in de secure multiparty

computation zijn echter niet efficiënt genoeg om gebruikt te worden indien er meer dan twee partijen zijn. Aangezien het bij data mining gaat om grote hoeveelheden data is het belangrijk om praktische oplossingen te hebben. Daarom zijn er nieuwe protocollen ontwikkeld voor al de aparte componenten zoals onder andere de som. Deze protocollen zijn zo opgesteld dat ze voldoen aan de veiligheidseisen uit de secure multiparty computation en toch hun efficiëntie behouden. Deze protocollen worden uitgelegd in Hoofdstuk 5. Met behulp van deze bouwstenen kunnen er privacy preserving data mining algoritmes geconstrueerd worden. Een beschrijving van deze algoritmes kan gevonden worden in Hoofdstuk 6 en Hoofdstuk 7.

2.4 Privacy preserving data mining in de wereld

Privacy preserving data mining is geen wetenschap die zich beperkt tot de onderzoekswereld. Het is een reëel probleem dat ook in de echte wereld voorkomt. Privacy gaat over meer dan de concurrentie geen gegevens gunnen. Ook in België is er een wetgeving voorzien in verband met privacy. Er zijn eveneens situaties die baat kunnen hebben aan een manier om op een privacy preserving manier aan data mining te kunnen doen. De belangrijkste wetten en een treffend voorbeeld zullen in deze sectie verder besproken worden.

2.4.1 Privacy wetgeving in België

Dankzij de groeiende ICT markt ontstaan er verschillende nieuwe mogelijkheden om gegevens te verspreiden. Deze gegevens zullen vaak betrekking hebben op personen. Overal worden er databases of bestanden met persoonsgegevens aangelegd. Door de nieuwe technologieën is het moeilijk om bij te houden wie gegevens heeft over wie en wat er met die gegevens gebeurt. Het gevaar voor misbruik is hierdoor dan ook steeds aanwezig.
Om dit probleem op te vangen is er in 1992 in België een nieuwe wet[1] ontstaan. Deze wet handelt over de bescherming van persoonsgegevens. Het bijhouden, gebruiken en verspreiden van persoonsgegevens moet op een transparante manier gebeuren. De personen over wie gegevens worden bijgehouden moeten daarvan op de hoogte gebracht worden. De personen die de gegevens

[1]Deze wet van 8/12/1992 verscheen in het Belgisch Staatsblad op 18/3/1993. Een Europese wet is goedgekeurd op 24/10/1995. Door deze wet is de wet van 8/12/1992 aangepast naar de Europese richtlijnen. Deze wet is gekend als de wet van 11/12/1998, en verscheen in het Belgisch Staatsblad op 3/2/2001. Meer informatie over deze normen kan gevonden worden op [17].

bijhouden moeten duidelijk maken wie ze zijn en waarom ze de gegevens nodig hebben. De wet bepaalt eveneens welke soort gegevens mogen verzameld worden en wat er met die gegevens mag gebeuren. De personen over wie er gegevens worden bijgehouden hebben bovendien ook het recht de gegevens op te vragen of te verbeteren. Ze kunnen zich indien nodig eveneens verzetten tegen het gebruik van hun gegevens.

2.4.2 Een treffend voorbeeld: Ford/Firestone

Een spijtig voorval dat het nut van privacy preserving data mining aantoont is de situatie met Ford Motor en Firestone Tire. Een model van Ford, namelijk de Ford Explorer, in combinatie met Firestone banden had bepaalde veiligheidsproblemen. De combinatie Ford/Firestone veroorzaakte in Amerika meer dan 150 doden en nog eens meer dan 500 gewonden.
Beide bedrijven hadden elk hun eigen data en baseerden hun onderzoeken dan ook alleen daarop. Wegens allerlei privacy conflicten was het niet mogelijk om de gegevens samen te analyseren. Indien dit wel mogelijk geweest was, waren de veiligheidsproblemen misschien veel vroeger aan het licht gekomen en waren er vele mensenlevens gespaard gebleven. De Ford/Firestone kwestie kan verder bestudeerd worden op [18] en [19].

Hoofdstuk 3

Cryptografische achtergrond

Encryptie is een belangrijk element in de wereld van privacy en beveiliging. In het onderzoek naar privacy preserving data mining zijn er belangrijke eigenschappen van encryptie systemen waar dankbaar gebruik van gemaakt wordt. Deze kenmerken zullen hier besproken worden. Meer informatie over encryptie kan gevonden worden in [31] en [32].

3.1 Inleidende begrippen

Encryptie is het coderen van een bepaalde waarde. Dit gebeurt met behulp van een encryptie key. De waarde wordt de *plaintext* genoemd. Wanneer de plaintext geëncodeerd is, geeft dit de *ciphertext*.
Het is mogelijk om de ciphertext opnieuw te decoderen via een decryptie key. Dit wordt decryptie genoemd. Decryptie van de ciphertext levert opnieuw de plaintext op.

3.2 One-way vs. two-way encryptie

One-way encryptie Een encryptie schema wordt one-way genoemd indien het niet mogelijk is om een geëncrypteerde waarde opnieuw te decrypteren. Dit kan handig zijn om wachtwoorden te controleren. Het wachtwoord wordt geëncrypteerd opgeslagen in de database. Wanneer er een controle moet gebeuren dan wordt het ingegeven wachtwoord geëncrypteerd en vergeleken met wat in de database staat. Op deze manier is het niet nodig om een decryptie key te hebben. Een voorbeeld van one-way encryptie is MD5 [32].

Two-way encryptie Een encryptie schema wordt two-way genoemd indien er zowel een encryptie als een decryptie key bestaat. Het is dus mogelijk

om een waarde te encrypteren en dan opnieuw te decrypteren. Een voorbeeld van two-way encryptie is RSA [32].

3.3 Deterministische vs. probabilistische encryptie

Deterministische encryptie Een encryptie schema wordt deterministisch genoemd indien het encrypteren van een plaintext met een bepaalde encryptie key altijd dezelfde ciphertext oplevert. Het is dus niet mogelijk om met één encryptie key twee verschillende plaintexts te encrypteren naar dezelfde ciphertext. Voorbeelden van deterministische encryptie zijn RSA [32] en Pohlig-Hellman [33].

Probabilistische encryptie Een encryptie schema wordt probabilistisch genoemd indien het encrypteren van een plaintext met een bepaalde encryptie key verschillende ciphertexts kan opleveren. Dit kan handig zijn wanneer er waardes uit een klein domein moeten geëncrypteerd worden. Door probabilistische encryptie te gebruiken zal het niet mogelijk zijn om de encryptie te breken door alle waardes uit het domein één voor één te encrypteren en de ciphertexts te vergelijken. Er is namelijk geen enkele garantie dat deze hetzelfde zullen zijn voor overeenkomstige waardes. Voorbeelden van probabilistische encryptie zijn Paillier [34] en Naccache-Stern [35].

3.4 Commutatieve vs. homomorfe encryptie

Commutatieve encryptie Een encryptie schema wordt commutatief genoemd indien er aan de volgende vereiste voldaan is. Stel dat er twee encryptie keys E_1 en E_2 zijn met de commutatieve eigenschap. Wanneer de plaintext x achtereenvolgens door E_1 en E_2 geëncrypteerd wordt dan zal dit dezelfde ciphertext opleveren als wanneer x eerst door E_2 en dan door E_1 geëncrypteerd wordt.

$$\textit{Commutatieve encryptie: } E_2(E_1(x)) = E_1(E_2(x))$$

Voorbeelden van commutatieve encryptie zijn RSA [32] en Pohlig-Hellman [33].

Homomorfe encryptie Een encryptie schema is homomorf indien het mogelijk is om de som te berekenen van twee waardes die geëncrypteerd zijn, zonder te decrypteren. Neem E een encryptie key die voldoet aan de eisen

van homomorfe encryptie. Dan kan de som van de twee waardes x_1 en x_2 berekend worden door de geëncrypteerde waardes te vermenigvuldigen. Dit product zal dan gelijk zijn aan de geëncrypteerde som van x_1 en x_2.

$$\textit{Homomorfe encryptie: } E(x_1).E(x_2) = E(x_1 + x_2)$$

Voorbeelden van homomorfe encryptie zijn Paillier [34] en Naccache-Stern [35].

Hoofdstuk 4

Secure multiparty computation

De kern van secure multiparty computation werd reeds aangehaald. Er zijn twee pioniers die een belangrijke rol gespeeld hebben in deze theorieën, namelijk Andrew Yao (in 1986) en Oded Goldreich (in 1987). Hun bijdragen zullen hier besproken worden.

4.1 Secure two party computation - Yao

Yao introduceerde in [24] als eerste het begrip *secure two party computation*. Hij toonde hier aan dat elke functie $f(x, y)$, waarbij x de input is van partij 1 en y de input van partij 2, op een veilige manier kan geëvalueerd worden.

4.1.1 Formele definitie

Om het begrip *veilig* op een formele manier te noteren werd een definitie vastgelegd. Deze definitie handelt over een functie f, zodat $f(x, y) = (f_1(x, y), f_2(x, y))$. De functie f krijgt als input een deel van partij 1 (namelijk x) en een deel van partij 2 (namelijk y). De output van de functie f bestaat uit twee delen (namelijk $f_1(x, y)$ en $f_2(x, y)$). Partij 1 wil $f_1(x, y)$ leren en partij 2 wil $f_2(x, y)$ leren. Veronderstel nu dat Π het protocol is om de functie f te berekenen. $View_i^{\Pi}$ is wat partij i leert bij het uitvoeren van het protocol en $Output_i^{\Pi}$ is de output van partij i. Als laatste wordt S_i gedefinieerd als een algoritme dat in polynomiale tijd kan uitgevoerd worden. Deze S_i wordt gezien als een algoritme dat de partijen kunnen gebruiken om extra informatie uit de gegevens te halen. De definitie gaat dan als volgt:

$$\{(S_1(x, f_1(x, y)), f_2(x, y))\} = \left\{(View_1^{\Pi}(x, y), Output_2^{\Pi}(x, y))\right\}$$
$$\{(f_1(x, y), S_2(y, f_2(x, y)))\} = \left\{(Output_1^{\Pi}(x, y), View_2^{\Pi}(x, y))\right\}$$

Dit komt er op neer dat wat een partij leert door de functie f uit te voeren aan de hand van het protocol Π gelijk moet zijn aan alles wat de partij in polynomiale tijd kan leren uit enerzijds zijn input en anderzijds zijn output. Het uitvoeren van het protocol zelf levert dus geen extra informatie op.

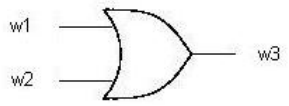
4.1.2 Circuitvoorstelling van een functie

Yao maakt gebruik van een booleaans circuit C om een functie f voor te stellen. Zo een circuit bestaat uit poorten die verbonden zijn door draden. Er bestaan 4 soorten draden, namelijk circuit inputdraden, circuit outputdraden, poort inputdraden en poort outputdraden. Het circuit C zal x en y als input krijgen op de circuit inputdraden. Deze input reist door het circuit via de poorten en draden. Na uitvoering van het circuit zullen de outputdraden de output bevatten en dit zijn dan ook de enige waardes die geleerd mogen worden.
Beschouw nu Alice en Bob als de twee partijen die een functie f willen berekenen. Eerst en vooral zal Alice een geëncrypteerde vorm van het circuit opstellen, haar input erin verwerken en dat naar Bob sturen. Bob moet dan zijn eigen input nog in het circuit krijgen en kan daarna het circuit uitvoeren en de output bekomen. Hoe dit exact gebeurt zal nu besproken worden.

4.1.2.1 Wat Alice opstelt

Voor elke draad van het circuit Alice zal voor elke draad i van het circuit C twee random waardes vastleggen, namelijk een waarde k_i^0 en k_i^1. De waarde k_i^0 staat voor een 0 op draad i, de waarde k_i^1 staat voor een 1 op draad i. Deze waardes moeten groot genoeg zijn omdat ze verder nog zullen gebruikt worden als encryptie keys.

Voor elke poort van het circuit Alice stelt voor elke poort van het circuit een *garbled computation table* op. Dit is een tabel waarmee met twee inputwaardes, alleen de bijhorende outputwaarde kan geleerd worden. Dit gebeurt door de outputwaarde te encrypteren met de inputwaardes.
In Figuur 4.1 kan een voorbeeld gevonden worden van een OR gate. Tabel 4.1 bevat de bijhorende garbled gate. Met behulp van twee inputwaardes kan de bijhorende outputwaarde gedecrypteerd worden. Zo kan met behulp van k_1^0 en k_2^0, $E_{k_1^0}(E_{k_2^0}(k_3^0))$ gedecrypteerd worden en k_3^0 verkregen worden. Over de andere outputwaardes kan niets geleerd worden aangezien daar de geschikte inputwaardes gekend voor moeten zijn.



Figuur 4.1: OR poort.

w_1	w_2	**GCT**
k_1^0	k_2^0	$E_{k_1^0}(E_{k_2^0}(k_3^0))$
k_1^0	k_2^1	$E_{k_1^0}(E_{k_2^1}(k_3^1))$
k_1^1	k_2^0	$E_{k_1^1}(E_{k_2^0}(k_3^1))$
k_1^1	k_2^1	$E_{k_1^1}(E_{k_2^1}(k_3^1))$

Tabel 4.1: Garbled computation table van een OR poort.

Voor elke output draad Alice stelt voor elke output draad een *output decryption table* op. Met behulp van deze tabel kan een waarde k_i^α terug omgezet worden naar de oorspronkelijke 0 of 1 waarde. Tabel 4.2 bevat een voorbeeld van zo een tabel.

Voor haar eigen input bits Alice verwerkt haar eigen input in het circuit. Dit doet ze door k_i^0 of k_i^1 op de overeenkomstige inputdraden te plaatsen.

4.1.2.2 Wat Bob ontvangt

Bob ontvangt dan van Alice het circuit met de garbled gates, de output decryption tables en haar input. Het enige wat er nu nog ontbreekt is de input van Bob in het circuit. Dit wil zeggen dat Bob van Alice de random waardes moet ontvangen die overeenkomen met zijn input. Dit moet echter op zo een manier gebeuren dat Alice de input van Bob niet kan leren en dat Bob alleen de random waardes leert voor zijn eigen input. Een techniek om dit resultaat te verkrijgen is een *one out of two oblivious transfer*.

Random output	**Echte output**
k_i^0	0
k_i^1	1

Tabel 4.2: Output decryption table van een output draad.

Dit is een protocol waarbij Alice twee waardes heeft, in dit geval zijn dat k_i^0 en k_i^1. Bob heeft één waarde σ en die is gelijk aan 0 of 1. Het protocol is nu zo dat Bob k_i^σ leert zonder de andere waarde van Alice te leren en dat Alice niets leert over de waarde van Bob. De volledige uitwerking van dit protocol kan gevonden worden in sectie 5.1.1. Het zal hier kort vermeld worden ter volledigheid. Bob genereert twee public keys, namelijk E_σ en $E_{1-\sigma}$. Van E_σ kent hij ook de decryptie key, van $E_{1-\sigma}$ niet. Bob stuurt deze keys naar Alice. Zij encrypteert k_i^0 met E_0 en k_i^1 met E_1. Alice stuurt deze geëncrypteerde waardes terug naar Bob. Bob kan dan k_i^σ decrypteren en leert niets over de andere waarde.
Bob heeft nu alles wat hij nodig heeft om het circuit uit te voeren. De output van het circuit kan hij, na uitvoering, naar Alice sturen. Op deze manier weten zowel Alice als Bob de output zonder iets geleerd te hebben over de andere partij zijn input.

4.2 Secure multiparty computation - Goldreich

Goldreich heeft deze redenering in [25] uitgebreid naar meerdere partijen en bijgevolg dus aangeduid dat elke functie f die haar input van meerdere partijen moet krijgen op een veilige manier kan berekend worden. Het werk van Goldreich bevat ook nog een belangrijke stelling, namelijk de compositiestelling.

Compositiestelling Wanneer de functie g op een veilige manier reduceerbaar is tot de functie f, en voor deze functie bestaat een protocol dat f veilig berekent, dan bestaat er eveneens een protocol om de functie g veilig te berekenen.

4.3 Modellen in secure multiparty computation

In de secure multiparty computation theorieën worden verschillende modellen beschouwd betreffende de eerlijkheid van de partijen.

4.3.1 Semi-honest model

In het semi-honest model volgen de partijen het protocol volledig. Het is wel toegelaten om alles te onthouden wat ze leren tijdens de uitvoering van het protocol. Deze informatie kunnen ze dan later gebruiken om meer te weten te komen over de andere partijen.
Bij het evalueren van privacy preserving data mining algoritmes wordt meestal ook uitgegaan van een semi-honest model. Dit is om de efficiëntie van de algoritmes te bewaren en omdat er aangenomen wordt dat de partijen de protocollen zullen volgen omdat ze correcte en betrouwbare resultaten willen verkrijgen.

4.3.2 Malicious model

In het malicous model wordt aangenomen dat de partijen mogen valsspelen. Ze kunnen dus onder andere hun input vervalsen om meer te leren over de input van de andere partij. In het hierboven beschreven protocol voor een one out of two oblivious transfer zou Bob bijvoorbeeld twee public keys kunnen genereren waarvoor hij de decryptie keys kent. Op deze manier kan hij beide waardes van Alice leren.

Hoofdstuk 5

Privacy preserving protocollen: Een overzicht

Om de data mining algoritmes beschreven in Hoofdstuk 1 te transformeren naar algoritmes die niet alleen de privacy van de gegevens respecteren maar die ook voldoende efficiënt blijven, wordt gebruik gemaakt van talrijke bouwstenen.
Deze protocollen zijn in veel gevallen speciaal ontwikkeld voor het gebruik in privacy preserving data mining algoritmes. Er wordt echter ook vaak een beroep gedaan op enkele primitieven uit de secure multiparty computation wereld en reeds bestaande protocollen uit de privacy preserving statistiek. Deze protocollen zullen in dit hoofdstuk verder besproken worden.

5.1 Primitieven uit de secure multiparty computation

Er bestaan verschillende protocollen voor het berekenen van de onderstaande primitieven. Ter volledigheid wordt er telkens één mogelijkheid besproken. Andere voorstellen kunnen gevonden worden in [37] en [38].

5.1.1 Oblivious transfer

Doelstelling Er zijn twee partijen, namelijk Alice en Bob. Alice heeft twee waardes $\{x_0, x_1\}$. Bob heeft één waarde $\sigma \in \{0, 1\}$. Bob moet x_σ verkrijgen zonder dat hij iets leert over de andere waarde van Alice en zonder dat Alice iets leert over de waarde van Bob.

Werking (zie Algoritme 3) Het oblivious transfer protocol is reeds aan bod gekomen in sectie 4.1.2.2. Het beschreven protocol is een one out of two oblivious transfer maar kan uiteraard uitgebreid worden naar meerdere waardes.

Algoritme 3 Oblivious transfer protocol

Require: Bob en Alice, de twee partijen.
Require: Alice heeft twee waardes, nl x_0 en x_1.
Require: Bob heeft $\sigma \in \{0, 1\}$.

1: Bob genereert twee commutatieve, deterministische encryptie keys, namelijk E_σ en $E_{1-\sigma}$. Van E_σ kent hij de decryptie key D_σ, van $E_{1-\sigma}$ niet.
2: Bob stuurt de twee keys naar Alice.
3: Alice encrypteert x_0 met E_0 en x_1 met E_1.
4: Alice stuurt de twee geëncrypteerde waardes naar Bob.
5: Bob decrypteert x_σ met de decryptie key D_σ.

5.1.2 Oblivious polynomial evaluation

Doelstelling Er zijn twee partijen, namelijk Alice en Bob. Alice heeft een polynoom P met een graad d over een domein $\mathcal{F}$. Bob heeft een element $x_* \in \mathcal{F}$. Bob moet $P(x_*)$ verkrijgen, zonder dat hij iets leert over de polynoom van Alice en zonder dat Alice iets leert over de waarde van Bob.

Werking (zie Algoritme 4) De polynoom P kan bekeken worden als $P(x) = \sum_{i=0}^{d} a_i x^i$. Elke a_i uit de sommatie kan geschreven worden als $a_i = \sum_{j=1}^{m} a_{ij} 2^{j-1}$ met $a_{ij} \in \{0, 1\}$ en $m = \lceil log_2 |\mathcal{F}| \rceil$.
Er kunnen nu elementen v_{ij} geconstrueerd worden zodat $v_{ij} = 2^{j-1} x_*^i$ met $j \in \{1, ..., m\}$ en $i \in \{1, ..., d\}$. Met $v_{ij} = 2^{j-1} x_*^i$ zal ook $x_*^i = \frac{v_{ij}}{2^{j-1}}$ gelden. Aangezien $a_i = \sum_{j=1}^{m} a_{ij} 2^{j-1}$ zal $a_i x_*^i = \sum_{j=1}^{m} a_{ij} v_{ij}$.
Bob zal paren $(r_{ij}, v_{ij} + r_{ij})$ construeren met (r_{ij}) een random waarde. Alice en Bob starten dan oblivious transfer protocollen. Deze worden zo uitgevoerd dat Alice (r_{ij}) leert indien $a_{ij} = 0$, en $(v_{ij} + r_{ij})$ indien $a_{ij} = 1$.
Alice kan na deze oblivious tranfers $a_0 + \sum_{i=1}^{d} \sum_{j=1}^{m} (a_{ij} v_{ij} + r_{ij}) = P(x_*) + \sum_{i,j} r_{ij}$ opstellen en zal dit naar Bob sturen. Bob zal dit resultaat met $\sum_{i,j} r_{ij}$ verminderen en $P(x_*)$ leren. Bob leert op deze manier niets over de polynoom van Alice en Alice leert niets over de waarde van Bob.

Voorbeeld Een voorbeeld zal de werking van het protocol verder verduidelijken. Neem $|\mathcal{F}| = 80$, $d = 3$ en de polynoom $P(x) = 1 + x + 3x^2 + x^3$. De waarde x_* is gelijk aan 2.
De polynoom kan dan geschreven worden als $P(x) = a_0 + a_1x + a_2x^2 + a_3x^3$. Hierin is $a_0 = 1$, $a_1 = 1$, $a_2 = 3$ en $a_3 = 1$. Elke a_i kan bekeken worden als $a_i = \sum_{j=1}^{m} a_{ij}2^{j-1}$ met $m = \lceil log_2|\mathcal{F}| \rceil = \lceil log_2 80 \rceil = 7$ en $a_{ij} \in \{0, 1\}$. Zo zal $a_2 = 1.2^0 + 1.2^1 + 0.2^2 + 0.2^3 + 0.2^4 + 0.2^5 + 0.2^6$. Op deze manier kan Alice bepalen of ze (r_{ij}) of $(v_{ij} + r_{ij})$ van Bob wil ontvangen.
Alice haar gegevens kunnen gevonden worden in Tabel 5.1. Aangezien $a_2 = 1.2^0 + 1.2^1$ zal Alice voor deze a_i de v_{ij} waardes nodig hebben met $i = 2$, $j = 1$ en $j = 2$. In dit geval zijn dat v_{21} en v_{22}.
Bob stelt op zijn beurt de nodige $(r_{ij}, v_{ij}+r_{ij})$ paren op, zoals weergegeven in Tabel 5.2. De (r_{ij}) waardes zijn hier achterwege gelaten om de tabel leesbaar te houden.
De waardes die Alice nodig heeft zijn dus 1.2, 1.4, 2.4 en 1.8. De som die Alice dan construeert, is de som van deze waardes met a_0. Dit is gelijk aan $1 + 2 + 4 + 8 + 8 = 23$ en dat komt inderdaad overeen met $P(2)$.

i	a_i	**j waardes waarvoor** $a_{ij} = 1$	**nodige** v_{ij} **waardes**
1	1	1	v_{11}
2	2	1, 2	v_{21}, v_{22}
3	1	1	v_{31}

Tabel 5.1: Gegevens van Alice: De (v_{ij}) waardes die ze nodig heeft.

	j=1	**j=2**	**j=3**	**j=4**	**j=5**	**j=6**	**j=7**
i=1	$2^0.2^1$	$2^1.2^1$	$2^2.2^1$	$2^3.2^1$	$2^4.2^1$	$2^5.2^1$	$2^6.2^1$
i=2	$2^0.2^2$	$2^1.2^2$	$2^2.2^2$	$2^3.2^2$	$2^4.2^2$	$2^5.2^2$	$2^6.2^2$
i=3	$2^0.2^3$	$2^1.2^3$	$2^2.2^3$	$2^3.2^3$	$2^4.2^3$	$2^5.2^3$	$2^6.2^3$

Tabel 5.2: Gegevens van Bob: Alle (v_{ij}) waardes.

5.2 Reeds ontwikkelde protocollen uit privacy preserving statistiek

De privacy preserving data mining algoritmes maken ook gebruik van enkele protocollen die al bekend waren. Deze protocollen zijn te vinden in [41] en

Algoritme 4 Oblivious polynomial evaluation protocol

Require: Bob en Alice, de twee partijen.
Require: Alice heeft een polynoom P met graad d over een domein $\mathcal{F}$.
Require: Bob heeft een element $x_* \in \mathcal{F}$ en kent d.

1: Alice schrijft haar polynoom als $P(x) = \sum_{i=0}^{d} a_i x^i$ met $a_i = \sum_{j=1}^{m} a_{ij} 2^{j-1}$.
2: Bob construeert paren $(r_{ij}, v_{ij} + r_{ij})$ $\forall i, j$ met (r_{ij}) een random waarde.
3: Bob en Alice nemen deel aan een oblivious tranfer $\forall i, j$. Alice leert (r_{ij}) indien $a_{ij} = 0$ en $(v_{ij} + r_{ij})$ indien $a_{ij} = 1$.
4: Alice construeert $a_0 + \sum_{i=1}^{d} \sum_{j=1}^{m} (a_{ij} v_{ij} + r_{ij}) = P(x_*) + \sum_{i,j} r_{ij}$.
5: Alice stuurt dit resultaat naar Bob.
6: Bob vermindert deze waarde met $\sum_{i,j} r_{ij}$ en leert $P(x_*)$.

komen uit het onderzoek naar privacy preserving statistiek. Het verband met data mining is dan ook voor de hand liggend.

5.2.1 Secure scalair product protocol

Doelstelling Er zijn twee partijen, namelijk Alice en Bob. Alice heeft een vector $\vec{X}$. Bob heeft een vector $\vec{Y}$. Alice en Bob moeten allebei een share van het scalair product $s_a + s_b = \vec{X}.\vec{Y}$ verkrijgen, zonder iets te leren over de vector van de andere partij. Alice leert dan s_a en Bob leert s_b.

Werking (zie Algoritme 5) Alice en Bob komen twee getallen overeen, namelijk p en m. Alice zal haar vector opsplitsen in m verschillende delen. Elk deel wordt beveiligd door $p-1$ random vectoren en zo naar Bob gestuurd. Bob krijgt dus m keer een verzameling van p vectoren aan. Hij zal voor elke vector uit zo een verzameling het scalair product berekenen met zijn vector en dit resultaat verbergen door er een random waarde bij op te tellen. Alice moet van al deze verzamelingen telkens één element terugkrijgen. Met behulp van m one out of p oblivious transfers kan Alice de delen verkrijgen die opgeteld haar share s_a van het scalair product vormen. Bob zijn share s_b bestaat uit de som van alle random waardes die hij gebruikt heeft. Beide partijen leren een share van het scalair product, zonder iets te leren over de vector van de andere partij.

5.2.2 Secure permutation protocol

Doelstelling Er zijn twee partijen, namelijk Alice en Bob. Alice heeft een vector $\vec{X}$. Bob heeft een vector $\vec{Y}$ en een permutatie Π. Alice moet $\Pi(\vec{X} + \vec{Y})$

Algoritme 5 Secure scalair product protocol

Require: Bob en Alice, de twee partijen.
Require: Alice heeft een n-dimensionale vector $\vec{X}$.
Require: Bob heeft een n-dimensionale vector $\vec{Y}$.

1: Alice en Bob komen twee getallen overeen, namelijk p en m.
2: Alice genereert m random vectoren $\vec{V}_1, ..., \vec{V}_m$ zodat $\vec{X} = \sum_{i=1}^{m} \vec{V}_i$.
3: Bob genereert m random getallen $r_1, ..., r_m$ zodat $s_b = \sum_{i=1}^{m} r_i$.
4: **for** $i = 1...m$ **do**
5: Alice genereert een getal k zodat $1 \leq k \leq p$.
6: Alice vormt ($\vec{H}_1$,...,$\vec{H}_p$) met $\vec{H}_k = \vec{V}_i$ en de andere $\vec{H}_j$'s random vectoren.
7: **for** $j = 1...p$ **do**
8: Bob berekent $Z_{i,j} = \vec{H}_j.\vec{Y} + r_i$.
9: **end for**
10: Bob en Alice nemen deel aan een oblivious transfer. Alice leert de $Z_{i,j}$ waarde die ze nodig heeft, namelijk $Z_{i,k}$.
11: **end for**
12: Alice construeert $s_a = \sum_{i=1}^{m} Z_{i,k} = \vec{X}.\vec{Y} + s_b$.

verkrijgen, zonder dat zij iets leert over de vector $\vec{Y}$ of de permutatie Π van Bob en zonder dat Bob iets leert over haar vector.

Werking (zie Algoritme 6) Het protocol maakt gebruik van homomorfe, probabilistische encryptie. Alice heeft in dit geval de encryptie en de decryptie key. Zij encrypteert haar vector en stuurt die samen met de encryptie key naar Bob. Bob encrypteert zijn vector en berekent de vector som. Dit is mogelijk omdat er gebruik gemaakt wordt van homomorfe encryptie. Dankzij het probabilistische kenmerk van de encryptie heeft Bob er niets aan dat hij de geëncrypteerde vector van Alice samen met de encryptie key heeft. Bob kan op de vector som zijn permutatie toepassen en dit resultaat naar Alice sturen. Alice is de enige met de decryptie key en kan dit decrypteren. Alice leert $\Pi(\vec{X} + \vec{Y})$, zonder iets te leren over Bob zijn vector of de permutatie en zonder dat Bob iets leert over haar vector.

5.2.3 Secure square computation protocol

Doelstelling Er zijn twee partijen, namelijk Alice en Bob. Alice heeft een waarde x_a. Bob heeft een waarde x_b. Alice en Bob moeten allebei een share van $s_a + s_b = (x_a + x_b)^2$ verkrijgen, zonder iets te leren over de waarde van de andere partij. Alice leert dan s_a en Bob leert s_b.

Algoritme 6 Secure permutation protocol

Require: Bob en Alice, de twee partijen.
Require: Alice heeft een n-dimensionale vector $\vec{X}$.
Require: Bob heeft een n-dimensionale vector $\vec{Y}$ en een permutatie Π voor de n elementen.

1: Alice genereert een homomorfe, probabilistische encryptie key E en een decryptie key D.
2: Alice encrypteert haar vector $\vec{X}$ zodat $\vec{X'} = (E(x_1), ..., E(x_n))$.
3: Alice stuurt de encryptie key E en haar geëncrypteerde vector $\vec{X'}$ naar Bob.
4: Bob encrypteert zijn vector $\vec{Y}$ zodat $\vec{Y'} = (E(y_1), ..., E(y_n))$.
5: Bob berekent $\vec{T'} = E(t_1, ..., t_n)$ met $t_i = E(x_i).E(y_i) = E(x_i + y_i)$.
6: Bob past de permutatie Π toe op de vector $\vec{T'}$ en verkrijgt zo $\vec{T'_p} = \Pi(\vec{T'})$.
7: Bob stuurt de vector $\vec{T'_p}$ naar Alice.
8: Alice decrypteert de componenten van $\vec{T'_p}$ met de decryptie key D en bekomt het resultaat.

Werking (zie Algoritme 7) Het protocol maakt gebruik van een oblivious polynomial evaluation. Alice stelt de polynoom $P(x) = x^2 + (2x_a)x + (x_a^2 - s_a)$ op. Bob wil $P(x_b)$ verkrijgen. Dit is gelijk aan $P(x_b) = x_b^2 + (2x_a)x_b + (x_a^2 - s_a)$. Als Bob $P(x_b)$ als zijn share s_b neemt en Alice heeft s_a als haar share dan zal $s_a + s_b = x_b^2 + (2x_a)x_b + x_a^2 = (x_a + x_b)^2$. Beide partijen leren een share van $(x_a + x_b)^2$, zonder iets te leren over de waarde van de andere partij.

Algoritme 7 Secure square computation protocol

Require: Bob en Alice, de twee partijen.
Require: Alice heeft een waarde x_a.
Require: Bob heeft een waarde x_b.

1: Alice genereert een random waarde s_a en stelt de polynoom $P(x) = x^2 + (2x_a x) + (x_a^2 - s_a)$ op.
2: Alice en Bob nemen deel aan een oblivious polynomial evaluation met $P(x)$ en x_b.
3: Bob heeft $s_b = P(x_b)$.

5.2.4 Secure division computation protocol

Doelstelling Er zijn twee partijen, namelijk Alice en Bob. Alice heeft waardes a_1 en a_2. Bob heeft waardes b_1 en b_2. Alice en Bob moeten allebei een share van $s_a + s_b = \frac{a_1+b_1}{a_2+b_2}$ verkrijgen, zonder iets te leren over de waardes van de andere partij. Alice leert dan s_a en Bob leert s_b.

Werking (zie Algoritme 8) Alice en Bob vormen respectievelijk de vectoren $\vec{A}$ en $\vec{B}$ en berekenen het scalair product zoals beschreven op regels (1-4).

Bob verkrijgt dan $x_1 = \vec{A}.\vec{B}$
$= r_1(a_1 - s_a.a_2) + r_1.b_1 - r_1.s_a.b_2$
$= r_1(a_1 + b_1 - s_a(a_2 + b_2))$.

Opnieuw vormen Alice en Bob een vector, in dit geval zijn dat $\vec{A'}$ en $\vec{B'}$, en nemen deel aan een scalair product protocol.

Bob verkrijgt $x_2 = \vec{A'}.\vec{B'}$
$= r_2.a_2 + r_2.b_2$
$= r_2(a_2 + b_2)$.

Dit staat in het algoritme op regels (5-9). Als laatste zal Alice $x_3 = r_2/r_1$ naar Bob sturen. Bob kan dan zijn share s_b berekenen.

Namelijk $s_b = \frac{x_1}{x_2}.x_3$
$= \frac{a_1+b_1-s_a(a_2+b_2)}{a_2+b_2}$
$= \frac{a_1+b_1}{a_2+b_2} - s_a$.

Zoals vermeld staat op regels (10-11). Beide partijen leren een share van $\frac{a_1+b_1}{a_2+b_2}$, zonder iets te leren over de waardes van de andere partij.

5.3 Nieuw ontwikkelde protocollen uit privacy preserving data mining

Uiteraard volstonden de bestaande protocollen niet om privacy preserving data mining algoritmes op te bouwen. Daarom werden er ook nieuwe protocollen ontwikkeld. Al deze protocollen zijn zo opgesteld dat ze veilig mogen genoemd worden volgens de normen van de secure multiparty computation theorie in een semi-honest model. Dit komt er op neer dat alle partijen niets meer leren dan de output van het protocol. Om te bewijzen dat dit wel degelijk het geval is wordt een *polynomiale tijd simulator* geschreven.

Algoritme 8 Secure division computation protocol

Require: Bob en Alice, de twee partijen.
Require: Alice heeft twee waardes a_1 en a_2.
Require: Bob heeft twee waardes b_1 en b_2.

1: Alice genereert drie random waardes, namelijk s_a, r_1 en r_2.
2: Alice vormt de vector $\vec{A} = (r_1(a_1 - s_a.a_2), r_1, -r_1.s_a)$.
3: Bob vormt de vector $\vec{B} = (1, b_1, b_2)$.
4: Alice en Bob nemen deel aan een scalair product protocol met $\vec{A}$ en $\vec{B}$.
5: Bob heeft dan $x_1 = r_1(a_1 + b_1 - s_a(a_2 + b_2))$.
6: Alice vormt de vector $\vec{A'} = (r_2.a_2, r_2)$.
7: Bob vormt de vector $\vec{B'} = (1, b_2)$.
8: Alice en Bob nemen deel aan een scalair product protocol met $\vec{A'}$ en $\vec{B'}$.
9: Bob heeft dan $x_2 = r_2(a_2 + b_2)$.
10: Alice stuurt $x_3 = \frac{r_2}{r_1}$ naar Bob.
11: Bob berekent $s_b = \frac{x_1}{x_2}.x_3 = \frac{a_1+b_1}{a_2+b_2} - s_a$.

Deze simulator moet de view van de partijen tijdens de stappen van het protocol simuleren. Dit moet mogelijk zijn met de input van een partij en de output van het protocol. Volgens de secure multiparty computation is een protocol veilig als dit lukt. De simulator toont dan aan dat de partijen niets meer leren door het protocol uit te voeren dan de output. Aangezien er alleen informatie kan worden vrijgegeven in de stappen waarbij de partijen communiceren, moeten alleen deze stappen worden opgenomen in de simulator. De andere stappen bevatten alleen lokale berekeningen en hierdoor kunnen de partijen niets leren wat niet mag geleerd worden.
Merk op dat het bij sommige protocollen is toegestaan om extra gegevens vrij te geven. Deze gegevens zijn dan kleine dingen waarvan aangenomen wordt dat het niet erg is om ze vrij te geven. Het is uiteraard belangrijk om deze gegevens in de simulator te verwerken aangezien dit dingen zijn die de partijen leren en dus kunnen gebruiken.
Het opstellen van een simulator bevat enkele veelgebruikte technieken. Om niet steeds weer in herhaling te vallen zullen niet alle simulators worden opgesteld. De ontbrekende simulators kunnen op een analoge manier worden geconstrueerd.

5.3.1 Secure sum protocollen

Een som kan nooit op een veilige manier berekend worden in het geval van twee partijen. Wanneer een partij de som kent van twee waardes en hij kent

uiteraard ook zijn eigen waarde, dan kan die partij altijd de waarde van de tweede partij leren. Geen enkel protocol kan dit vermijden. Daarom zal in de volgende twee protocollen uitgegaan worden van k partijen met $k > 2$.

Doelstelling Er zijn $k > 2$ partijen. Elke partij heeft een waarde x_i. De partijen moeten allemaal de som van de waardes verkrijgen, zonder iets te leren over de waardes van de andere partijen.

5.3.1.1 Secure sum protocol

In deze sectie wordt een secure sum protocol besproken uit [42].

Werking (zie Algoritme 9) Het protocol beschermt de waardes door gebruik te maken van een random waarde. Partij 0 telt een random getal op bij zijn waarde en stuurt dit zo door naar partij 1. Partij 1 kan dus niets leren over de waarde van partij 0 dankzij het random getal.
Elke partij telt zijn waarde op bij het resultaat dat hij ontvangt. Partij 0 kan het random getal aftrekken van wat hij ontvangt en op die manier de eigenlijke som leren. Merk op dat alle bewerkingen modulo een waarde gedaan worden. Dit is om er voor te zorgen dat alle waardes die worden doorgestuurd binnen een domein blijven. Dit is nodig om de veiligheid van het protocol aan te tonen. Het zal dan ook gebruikt worden bij het opstellen van de simulator. Partij 0 stuurt het resultaat naar de andere partijen. Alle partijen leren de som, zonder iets te leren over de waardes van de andere partijen.

Voorbeeld Een voorbeeld kan gevonden worden in Figuur 5.1. Partij 0 genereert het random getal 17 en gebruikt dit om zijn eigen waarde 5 te vermommen. Partij 0 stuurt 22 naar partij 1. Elke partij telt zijn deel op bij wat hij ontvangt. Op het einde van het protocol ontvangt partij 0 de waarde 53. Om de som te bekomen moet hier de random waarde van afgetrokken worden. Dit geeft dan het resultaat, namelijk 36.

Hoe veilig is het protocol? Er kan aangetoond worden dat het protocol veilig is door een polynomiale tijd simulator te beschrijven:

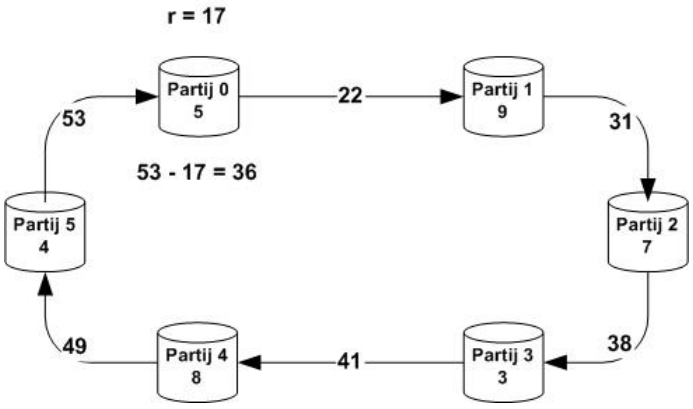
Input Alle partijen hebben een waarde x_i als input en kennen de grootte m van het domein waarin het resultaat zal liggen. Deze waarde moet gekend zijn om de veiligheid aan te tonen. Partij 0 kent hiernaast ook nog een random getal $r \in [0...m]$.

Algoritme 9 Secure sum protocol

Require: Partijen $0, ..., k-1$ met $k > 2$ en elk een waarde x_i.

Require: Een domein $[0...m]$ waarin het resultaat zal liggen.

1: Partij 0 genereert een random getal $r \in [0...m]$ en telt dit op bij x_0.
2: Partij 0 stuurt $(x_0 + r) mod\ m$ naar partij 1.
3: **for** $i = 1...k-1$ **do**
4: Partij i ontvangt $s_i = (r + \sum_{j=0}^{i-1} x_j) mod\ m$.
5: Partij i stuurt $(s_i + x_i) mod\ m$ door naar partij $(i+1) mod\ k$.
6: **end for**
7: Partij 0 ontvangt $(r + \sum_{i=0}^{k-1} x_i) mod\ m$.
8: Partij 0 vermindert de ontvangen waarde met r en bekomt zo de som $\sum_{i=0}^{k-1} x_i$.
9: Partij 0 stuurt het resultaat naar de andere partijen.



Figuur 5.1: Voorbeeld: Secure sum (1).

Output Alle partijen leren de som van hun waardes, namelijk $\sum_{i=0}^{k-1} x_i$.

Elke partij i krijgt $(r + \sum_{j=0}^{i-1} x_j) mod\ m$ aan van partij $i-1$. Deze waarde ligt telkens in het vastgelegde domein en kan dus gesimuleerd worden door een random waarde uit dat domein te kiezen. Op deze manier wordt de view van de partijen gesimuleerd.
Op het einde van het protocol krijgt partij 0 $(r + \sum_{i=0}^{k-1} x_i) mod\ m$ aan. Deze waarde kan eveneens gesimuleerd worden omdat partij 0 zowel r als $\sum_{i=0}^{k-1} x_i$ kent. De random waarde r behoort tot de input van partij 0 en de som is de output van het protocol, en dus ook gekend door partij 0.

Opmerking Het protocol kan gebroken worden door partijen die valsspelen. Wanneer er wordt uitgegaan van een malicious model is dit protocol dus niet veilig. Wanneer partij $i-1$ en $i+1$ samenwerken, kan de waarde van partij i geleerd worden. Dit is zo omdat de waarde van partij i gelijk is aan de waarde die partij $i+1$ ontvangt, vermindert met de waarde die partij $i-1$ doorstuurt.
Dit gebrek kan opgelost worden door het protocol aan te passen. Elke partij deelt zijn waarde x_i op in n delen. Van al deze delen wordt dan de som berekend. Om te voorkomen dat partij $i-1$ en partij $i+1$ kunnen samenwerken worden de n sommen telkens via een ander pad berekend. Op deze manier moeten er veel meer partijen samenwerken eer ze iets kunnen leren.

5.3.1.2 Secure sum protocol

In deze sectie wordt een nieuw secure sum protocol besproken.

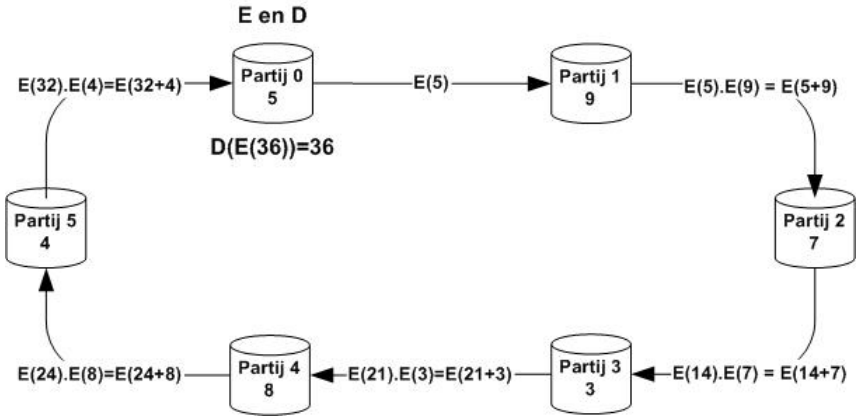
Werking (zie Algoritme 10) Het protocol beschermt de waardes door gebruik te maken van homomorfe, probabilistische encryptie. Partij 0 heeft de encryptie en de decryptie key, en stuurt de encryptie key naar de andere partijen. Partij 0 zal zijn waarde encrypteren en dit naar partij 1 sturen. Dankzij de homomorfe, probabilistische encryptie kan partij 1 niets leren over de waarde van partij 0.
Elke partij encrypteert zijn waarde met de encryptie key en vermenigvuldigt deze waarde met de waarde die hij ontvangt van de vorige partij. Partij 0 is de enige met de decryptie key en kan het resultaat decrypteren. Op deze manier leert partij 0 de som en kan deze naar de andere partijen sturen. Alle partijen leren de som, zonder iets te leren over de waardes van de andere partijen.

Voorbeeld Een voorbeeld kan gevonden worden in Figuur 5.2. Partij 0 genereert een homomorfe, probabilistische encryptie en decryptie key. De encryptie key wordt naar de andere partijen gestuurd. Partij 0 gebruikt de encryptie key om zijn waarde te vermommen. Partij 0 stuurt $E(5)$ naar partij 1. Elke partij vermenigvuldigt de waarde die hij ontvangt met zijn eigen geëncrypteerde waarde. Dit komt dan overeen met het optellen van de gedecrypteerde waardes. Partij 0 ontvangt na het uitvoeren van het protocol $E(36)$. Hij kan dit decrypteren, wat 36 oplevert.

Algoritme 10 Secure sum protocol

Require: Partijen $0, ..., k-1$ met $k > 2$ en elk een waarde x_i.

1: Partij 0 genereert een homomorfe, probabilistische encryptie key E en een decryptie key D.
2: Partij 0 stuurt E naar de andere partijen.
3: Partij 0 stuurt $E(x_0)$ naar partij 1.
4: **for** $i = 1...k-1$ **do**
5: Partij i ontvangt $s_i = E(\sum_{j=0}^{i-1} x_j)$.
6: Partij i stuurt $s_i.E(x_i)$ door naar partij $(i+1) mod\ k$.
7: **end for**
8: Partij 0 ontvangt $E(\sum_{i=0}^{k-1} x_i)$.
9: Partij 0 decrypteert de ontvangen waarde en bekomt zo de som $\sum_{i=0}^{k-1} x_i$.
10: Partij 0 stuurt het resultaat naar de andere partijen.



Figuur 5.2: Voorbeeld: Secure sum (2).

Hoe veilig is het protocol? Er kan aangetoond worden dat het protocol veilig is door een polynomiale tijd simulator te beschrijven:

Input Alle partijen hebben een waarde x_i als input en kennen de encryptie key E. Partij 0 kent hiernaast ook nog de decryptie key D.

Output Alle partijen leren de som van hun waardes, namelijk $\sum_{i=0}^{k-1} x_i$.

Elke partij i krijgt $E(\sum_{j=0}^{i-1} x_j)$ aan van partij $i-1$. Dankzij de encryptie kan hieruit niets geleerd worden en is er geen onderscheid te maken tussen de geëncrypteerde waarde en een random getal. De view van de partijen kan dus gesimuleerd worden door een random getal te genereren.
Op het einde van het protocol krijgt partij 0 $E(\sum_{i=0}^{k-1} x_i)$ aan. Deze waarde kan eveneens gesimuleerd worden omdat partij 0 zowel E als $\sum_{i=0}^{k-1} x_i$ kent. De encryptie key E behoort tot de input van partij 0 en de som is de output van het protocol, en dus ook gekend door partij 0.

Opmerking Het protocol is beter beveiligd tegen valsspelende partijen dan het protocol beschreven in de vorige sectie. Wanneer partij $i-1$ en $i+1$ willen samenwerken om de waarde van partij i te weten te komen, moeten ze altijd over de medewerking van partij 0 beschikken. Partij 0 is namelijk de enige partij met de decryptie key, en zonder deze key kan er nooit iets geleerd worden.

5.3.2 Secure scalair product protocollen

Er werd reeds een scalair product protocol besproken in sectie 5.2.1. Naast dit reeds bestaande protocol zijn er in het recente onderzoek naar privacy preserving data mining ook nieuwe scalair product protocollen voorgesteld.

5.3.2.1 Secure scalair product protocol

In deze sectie wordt een secure scalair product protocol besproken uit [43].

Doelstelling Er zijn twee partijen, namelijk Alice en Bob. Alice heeft een vector $\vec{X}$. Bob heeft een vector $\vec{Y}$. Alice en Bob moeten allebei het scalair product $\vec{X}.\vec{Y}$ verkrijgen, zonder iets te leren over de vector van de andere partij.

Werking (zie Algoritme 11) Alice en Bob starten het protocol door een $n \times n$-matrix C overeen te komen. Deze matrix stelt de coëfficiënten voor van n lineair onafhankelijke vergelijkingen. Alice zal met deze matrix en n

random getallen haar vector vermommen en het resultaat hiervan naar Bob sturen. Deze handelingen staan in het algoritme beschreven op regels (1-5). Bob berekent het scalair product S van de vector die hij van Alice krijgt en zijn eigen vector. Hij zal ook n waardes $n_1, ..., n_n$ construeren die bestaan uit de elementen van zijn vector, de coëfficiënten van de matrix C en r random waardes. Deze n waardes zullen er dan als volgt uitzien:

$(a_{11}.y_1 + a_{21}.y_2 + ... + a_{n1}.y_n + r'_1)$
...
$(a_{1(n/r)}.y_1 + a_{2(n/r)}.y_2 + ... + a_{n(n/r)}.y_n + r'_1)$
$(a_{1((n/r)+1)}.y_1 + a_{2((n/r)+1)}.y_2 + ... + a_{n((n/r)+1)}.y_n + r'_2)$
...
$(a_{1(2n/r)}.y_1 + a_{2(2n/r)}.y_2 + ... + a_{n(2n/r)}.y_n + r'_2)$
...
$(a_{1(((r-1)n/r)+1)}.y_1 + a_{2(((r-1)n/r)+1)}.y_2 + ... + a_{n(((r-1)n/r)+1)}.y_n + r'_r)$
...
$(a_{1n}.y_1 + a_{2n}.y_2 + ... + a_{nn}.y_n + r'_r)$

Bob stuurt het berekende scalair product samen met de n waardes naar Alice. Dit kan gevonden worden in het algoritme op regels (6-10). Het scalair product S dat Alice van Bob ontvangt is dan van de volgende vorm:

$S = (x_1 + a_{11}.r_1 + a_{12}.r_2 + ... + a_{1n}.r_n).y_1$
$+ (x_2 + a_{21}.r_1 + a_{22}.r_2 + ... + a_{2n}.r_n).y_2$
...
$+ (x_n + a_{n1}.r_1 + a_{n2}.r_2 + ... + a_{nn}.r_n).y_n$

Door de x en y termen samen te brengen wordt dit:

$S = (x_1.y_1 + x_2.y_2 + ... + x_n.y_n)$
$+ (y_1.a_{11}.r_1 + y_1.a_{12}.r_2 + ... + y_1.a_{1n}.r_n)$
$+ (y_2.a_{21}.r_1 + y_2.a_{22}.r_2 + ... + y_2.a_{2n}.r_n)$
...
$+ (y_n.a_{n1}.r_1 + y_n.a_{n2}.r_2 + ... + y_n.a_{nn}.r_n)$

Dit kan herschreven worden als:

$S = \sum_{i=1}^{n} x_i.y_i$
$+ r_1(a_{11}.y_1 + a_{21}.y_2 + ... + a_{n1}.y_n)$
$+ r_2(a_{12}.y_1 + a_{22}.y_2 + ... + a_{n2}.y_n)$
...

$+ r_n(a_{1n}.y_1 + a_{2n}.y_2 + ... + a_{nn}.y_n)$

Door bij elk deel r_i, $r_i.r'_j - r_i.r'_j$ op te tellen verandert het resultaat niet:

$S = \sum_{i=1}^{n} x_i.y_i$
$+ r_1(a_{11}.y_1 + ... + a_{n1}.y_n)$
$+ r_1.r'_1 - r_1.r'_1$
$+ ...$
$+ r_{(n/r)}(a_{1(n/r)}.y_1 + ... + a_{n(n/r)}.y_n)$
$+ r_{(n/r)}.r'_1 - r_{(n/r)}.r'_1$
$+ r_{((n/r)+1)}(a_{1((n/r)+1)}.y_1 + ... + a_{n((n/r)+1)}.y_n)$
$+ r_{((n/r)+1)}.r'_2 - r_{((n/r)+1)}.r'_2$
$+ ...$
$+ r_{(2n/r)}(a_{1(2n/r)}.y_1 + ... + a_{n(2n/r)}.y_n)$
$+ r_{(2n/r)}.r'_2 - r_{(2n/r)}.r'_2$
$+ ...$
$+ r_{(((r-1)n/r)+1)}(a_{1(((r-1)n/r)+1)}.y_1 + ... + a_{n(((r-1)n/r)+1)}.y_n)$
$+ r_{(((r-1)n/r)+1)}.r'_r - r_{(((r-1)n/r)+1)}.r'_r$
$+ ...$
$+ r_{(n)}(a_{1n}.y_1 + ... + a_{nn}.y_n)$
$+ r_n.r'_r - r_n.r'_r$

Dit kan verder vereenvoudigt worden door de r_i termen samen te brengen:

$S = \sum_{i=1}^{n} x_i.y_i$
$+ r_1(a_{11}.y_1 + ... + a_{n1}.y_n + r'_1)$
$+ ...$
$+ r_{(n/r)}(a_{1(n/r)}.y_1 + ... + a_{n(n/r)}.y_n + r'_1)$
$+ r_{((n/r)+1)}(a_{1((n/r)+1)}.y_1 + ... + a_{n((n/r)+1)}.y_n + r'_2)$
$+ ...$
$+ r_{(2n/r)}(a_{1(2n/r)}.y_1 + ... + a_{n(2n/r)}.y_n + r'_2)$
$+ ...$
$+ r_{(((r-1)n/r)+1)}(a_{1(((r-1)n/r)+1)}.y_1 + ... + a_{n(((r-1)n/r)+1)}.y_n + r'_r)$
$+ ...$
$+ r_{(n)}(a_{1n}.y_1 + ... + a_{nn}.y_n + r'_r)$
$- r_1.r'_1 - ... - r_{(n/r)}.r'_1$
$- r_{((n/r)+1)}.r'_2 - ... - r_{(2n/r)}.r'_2$
$- ...$
$- r_{(((r-1)n/r)+1)}.r'_r - ... - r_n.r'_r$

Hierin komen de n termen die Alice van Bob gekregen heeft voor. Alice

kan die termen met de overeenkomstige r_i waardes vermenigvuldigen en S met deze waardes verminderen. In het overblijvende deel kunnen de r'_i termen samengebracht worden:

$S' = \sum_{i=1}^{n} x_i.y_i$
- $(r_1 + r_2 + ... + r_{(n/r)}).r'_1$
- $(r_{((n/r)+1)} + r_{((n/r)+2)} + ... + r_{(2n/r)}).r'_2$
- ...
- $(r_{(((r-1)n/r)+1)} + r_{(((r-1)n/r)+2)} + ... + r_n).r'_r$

Alice kan de nodige sommen over haar random waardes r_i berekenen. Deze stuurt ze dan samen met S' naar Bob. Bob heeft de ontbrekende r'_j waardes en kan het uiteindelijke scalair product berekenen. In het algoritme staat dit uitgelegd op regels (11-16). Beide partijen leren het scalair product, zonder iets te leren over de vector van de andere partij.

5.3.2.2 Secure scalair product protocol

In deze sectie wordt een secure scalair product protocol besproken uit [49].

Doelstelling Er zijn k partijen. Elke partij heeft een booleaanse vector $\vec{X_i}$, dit wil zeggen dat alle elementen van de vector 0 of 1 zijn. De partijen moeten allemaal het scalair product $\vec{X}.\vec{Y}$ verkrijgen, zonder iets te leren over de vectoren van de andere partijen.

Werking (zie Algoritme 12) Het protocol maakt gebruik van homomorfe, probabilistische encryptie. Partij 0 heeft de encryptie en de decryptie key. Voor elk van de n elementen in de vector stuurt partij 0 een geëncrypteerde versie naar partij 1.
De volgende partijen zullen telkens naar hun eigen waarde kijken. Indien de waarde 0 is, dan encrypteren ze die waarde en sturen de geëncrypteerde 0 naar de volgende partij. Indien de waarde 1 is, dan sturen ze het bericht dat ze zelf ontvangen hebben naar de volgende partij. Het idee hierachter is dat het product van een aantal waardes alleen maar 1 kan zijn als alle termen van dat product 1 zijn. Wanneer één van de termen 0 is dan maakt het niet meer uit wat de andere termen zijn. De laatste partij zal op die manier n producten verzamelen.
Om het scalair product te bekomen moeten deze waardes opgeteld worden. Aangezien er gebruik gemaakt wordt van homomorfe encryptie is het mogelijk deze optelling te maken met de geëncrypteerde waardes. Het resultaat ervan wordt naar partij 0 gestuurd. Deze partij heeft de decryptie key en

Algoritme 11 Secure scalair product protocol

Require: Bob en Alice, de twee partijen.
Require: Alice heeft een n-dimensionale vector $\vec{X}$.
Require: Bob heeft een n-dimensionale vector $\vec{Y}$.

1: Alice en Bob komen een $n \times n$-matrix C overeen.
2: Alice genereert n random getallen $r_1, ..., r_n$.
3: Alice gebruikt C en de n random getallen om haar vector te beschermen.
4: Alice construeert $\vec{X'}$:

$$\vec{X'} = (x_1 + a_{11}.r_1 + a_{12}.r_2 + ... + a_{1n}.r_n,$$
$$... ,$$
$$x_n + a_{n1}.r_1 + a_{n2}.r_2 + ... + a_{nn}.r_n)$$

5: Alice stuurt $\vec{X'}$ naar Bob.
6: Bob berekent $S = \vec{X'}.\vec{Y}$.
7: Bob berekent de volgende n waardes $n_1, ..., n_n$:

$(a_{11}.y_1 + a_{21}.y_2 + ... + a_{n1}.y_n)$
...
$(a_{1n}.y_1 + a_{2n}.y_2 + ... + a_{nn}.y_n)$

8: Bob genereert r random getallen $r'_1, ..., r'_r$ en gebruikt deze om de n gegenereerde waardes te beschermen.
9: Bob verdeelt de n waardes in r verzamelingen en telt dan bij de elementen uit verzameling i r'_i op:

$(a_{11}.y_1 + a_{21}.y_2 + ... + a_{n1}.y_n + r'_1)$
...
$(a_{1n}.y_1 + a_{2n}.y_2 + ... + a_{nn}.y_n + r'_r)$

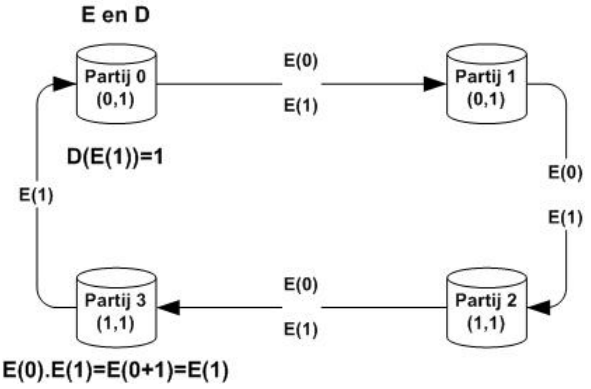
10: Bob stuurt deze n waardes samen met S naar Alice.
11: Alice kan S herschrijven:

$$S = \sum_{i=1}^{n} x_i.y_i$$
$$+r_1.(a_{11}.y_1 + a_{21}.y_2 + ... + a_{n1}.y_n + r'_1)$$
$$+...$$
$$+r_n.(a_{1n}.y_1 + a_{2n}.y_2 + ... + a_{nn}.y_n + r'_n)$$
$$-(r_1 + ...).r'_1$$
$$-...$$
$$-(... + r_n).r'_r$$

12: Alice maakt gebruikt van de n waardes die ze van Bob gekregen heeft en berekent zo $n_i.r_i$ en trekt deze waardes van S af.
13: Alice berekent de nodige sommen over haar random r_i waardes.
14: Alice stuurt dit alles door naar Bob.
15: Bob heeft de ontbrekende r'_i waardes en kan $\sum_{i=1}^{n} x_i.y_i$ berekenen.
16: Bob stuurt het resultaat naar Alice.

kan het scalair product decrypteren. Wanneer dit gebeurt is, zal partij 0 het resultaat naar de andere partijen sturen. Alle partijen leren het scalair product, zonder iets te leren over de vectoren van de andere partijen.

Voorbeeld Een voorbeeld kan gevonden worden in Figuur 5.3. Partij 0 genereert een homomorfe, probabilistische encryptie en decryptie key. De encryptie key wordt naar de andere partijen gestuurd. Alle partijen hebben een binaire vector met twee elementen. Partij 0 begint met zijn eerste element te encrypteren. Hij stuurt dan $E(0)$ naar partij 1. Partij 1 krijgt dit aan en bekijkt zijn eigen element. Dit is een nul en dus encrypteert hij zijn element en stuurt dat door naar partij 2. Partij 2 zijn eerste element is een één. Daarom stuurt hij het bericht door dat hij zelf ontvangen heeft en dat is in dit geval $E(0)$.
Hetzelfde gebeurt voor het tweede element van de vector. Partij 3 heeft dan twee geëncrypteerde waardes in zijn bezit namelijk $E(0)$ en $E(1)$. Hij kan de som berekenen van de waardes door de geëncrypteerde waardes te vermenigvuldigen. Dit geeft dan $E(1)$. Partij 3 stuurt dit door naar partij 0. Hij kan dit decrypteren, wat 1 oplevert.



Figuur 5.3: Voorbeeld: Secure scalair product.

5.3.2.3 Secure scalair product protocol

In deze sectie wordt een secure scalair product protocol besproken uit [50]. De scalair product protocollen uit sectie 5.2.1 en 5.3.2.1 worden betwist in [50]. Er wordt beweerd dat indien de protocollen worden gebruikt om het scalair product te berekenen van binaire vectoren met een laag support (dit wil zeggen weinig enen) de kans groot is dat er informatie wordt vrijgegeven. Er wordt eveneens een nieuw scalair product protocol voorgesteld.

Algoritme 12 Secure scalair product protocol

Require: Partijen $0, ..., k-1$ met $k \geq 2$.

Require: Elke partij heeft een n-dimensionale vector $\vec{X_i}$ met elementen $x_{i1}, ..., x_{in}$.

Partij 0 genereert een homomorfe, probabilistische encryptie key E en een decryptie key D.
Partij 0 stuurt E naar de andere partijen.
for $i = 1...n$ **do**
  for $j = 0...k-1$ **do**
    if $j == 0$ **then**
      Partij 0 stuurt $E(x_{0i})$ naar partij 1.
    else if $j == 1...k-2$ **then**
      Partij j ontvangt een bericht $M_{(j-1)i}$ van partij $j-1$.
      if $x_{ji} == 0$ **then**
        Partij j stuurt $E(x_{ji}) = E(0)$ naar partij $j+1$.
      else
        Partij j stuurt $M_{(j-1)i}$ naar partij $j+1$.
      end if
    else if $j == k-1$ **then**
      Partij $k-1$ ontvangt een bericht $M_{(k-2)i}$ van partij $k-2$.
      if $x_{(k-1)i} == 0$ **then**
        Partij $k-1$ stelt $M_{(k-1)i}$ gelijk aan $E(x_{(k-1)i}) = E(0)$.
      else
        Partij $k-1$ stelt $M_{(k-1)i}$ gelijk aan $M_{(k-2)i}$.
      end if
    end if
  end for
end for
Partij $k-1$ berekent $\prod_{i=1}^{n} M_{(k-1)i}$.
Partij $k-1$ stuurt dit resultaat naar partij 0.
Partij 0 decrypteert de som en stuurt het resultaat naar de andere partijen.

Doelstelling Er zijn twee partijen, namelijk Alice en Bob. Alice heeft een vector $\vec{X}$. Bob heeft een vector $\vec{Y}$. Alice en Bob moeten allebei een share van het scalair product $s_a + s_b = \vec{X}.\vec{Y}$ verkrijgen, zonder iets te leren over de vector van de andere partij. Alice leert dan s_a en Bob leert s_b.

Werking (zie Algoritme 13) Het protocol maakt gebruik van homomorfe, probabilistische encryptie. Het idee achter dit protocol is het berekenen van elke $x_i.y_i$ van het scalair product door de geëncrypteerde waarde van x_i, y_i aantal keren met zichzelf te vermenigvuldigen. Het resultaat van deze machtsverheffing is dan een geëncrypteerde waarde die, wanneer ze gedecrypteerd wordt, $x_i.y_i$ voorstelt.
Al deze geëncrypteerde tussenresultaten moeten vermenigvuldigd worden om het geëncrypteerde scalair product te bekomen. Aangezien vermenigvuldigingen tussen geëncrypteerde waardes gelijk zijn aan sommen tussen de gedecrypteerde waardes geeft deze werkwijze daadwerkelijk het gevraagde scalair product. Om beide partijen een share te geven zal Bob zijn share s_b random genereren. Om het scalair product met dit share te verminderen moet Bob het scalair product vermenigvuldigen met zijn geëncrypteerde share vermenigvuldigt met -1. Dankzij de homomorfe encryptie zorgt dit voor een aftrekking, wat dan Alice haar share s_a geeft.
Het is mogelijk om dit protocol uit te breiden zodat er meerdere partijen kunnen deelnemen. Alle partijen leren een share van het scalair product, zonder iets te leren over de vectoren van de andere partijen.

Algoritme 13 Secure scalair product protocol

Require: Bob en Alice, de twee partijen.
Require: Alice heeft een n-dimensionale vector $\vec{X}$.
Require: Bob heeft een n-dimensionale vector $\vec{Y}$.

1: Alice genereert een homomorfe, probabilistische encryptie key E en een decryptie key D.
2: Alice stuurt E naar Bob.
3: **for** $i = 1...n$ **do**
4: Alice stuurt $E(x_i)$ naar Bob.
5: **end for**
6: Bob berekent $\prod_{i=1}^{n} E(x_i)^{y_i}$.
7: Bob genereert s_b en berekent $s_a = \prod_{i=1}^{n} E(x_i)^{y_i}.E(-s_b)$.
8: Bob stuurt s_a naar Alice.
9: Alice decrypteert deze waarde en dat geeft haar haar share van het scalair product.

5.3.3 Secure union protocol

In deze sectie wordt een secure union protocol besproken uit [42]. Het berekenen van een unie kan voor problemen zorgen in het geval van twee partijen. Wanneer een partij de unie kent van twee sets en hij kent uiteraard ook zijn eigen set, dan kan die partij hier volledig of gedeeltelijk uit afleiden welke items in de set van de andere partij zitten. Het kan dus gewenst zijn om bij dit protocol uit te gaan van k partijen met $k > 2$.

Doelstelling Er zijn k partijen. Elke partij heeft een itemset. De partijen moeten allemaal de unie van de itemsets verkrijgen, zonder iets te leren over de itemsets van de andere partijen.

Werking (zie Algoritme 14) Het protocol maakt gebruik van commutatieve, deterministische encryptie. Er zijn grotendeels vier fases te vinden in het protocol:

Fase 1: De eerste stap staat in het algoritme beschreven op regels (1-5). Alle partijen genereren een commutatieve, deterministische encryptie key E_i en een decryptie key D_i. Elke itemset wordt aangevuld met valse items. Dit wordt gedaan om te verbergen hoeveel items er in een itemset zitten. De items worden geëncrypteerd doorgestuurd maar dat zal uiteraard niet verbergen hoe groot een itemset is.
Op het einde van fase 1 heeft elke partij een itemset van dezelfde grootte. De grootte van deze set is vooraf bepaald en is afhankelijk van de toepassing waarin het protocol gebruikt wordt.

Fase 2: De tweede stap staat in het algoritme beschreven op regels (6-18). Alle itemsets worden geëncrypteerd door alle partijen. Elke partij begint met zijn eigen items te encrypteren en de geëncrypteerde set naar de volgende partij te sturen. Die partij encrypteert wat hij ontvangt en stuurt de set weer verder. Dit gaat door tot elke partij i de volledig geëncrypteerde itemset van partij $(i + 1) mod\ k$ in zijn bezit heeft. Het is belangrijk dat er niet nog één stap verder wordt gegaan. Indien een partij zijn eigen volledig geëncrypteerde itemset in handen krijgt, kan hij die vergelijken met de volledig geëncrypteerde itemset van partij $(i + 1) mod\ k$. Veronderstel dat partij i 7 items heeft en partij $(i + 1) mod\ k$ 8 items. Indien partij i dan zowel zijn volledig geëncrypteerde itemset kent als die van partij $(i + 1) mod\ k$ en hij merkt dat ze 7 items gemeenschappelijk hebben, dan weet hij dat dat zijn 7 items moeten zijn. Op deze manier leert hij 7 van de 8 items van partij $(i + 1) mod\ k$. Dit is uiteraard iets wat vermeden moet worden.

Fase 3: De derde stap staat in het algoritme beschreven op regels (19-31). Alle partijen sturen de geëncrypteerde itemsets die ze op dat moment in hun bezit hebben naar partij 0 of 1. De even partijen sturen naar partij 0 en de oneven partijen sturen naar partij 1. De laatste partij echter, moet altijd naar partij 1 sturen. Dit wordt weer gedaan om te vermijden dat partijen hun eigen volledig geëncrypteerde itemset te zien krijgen.
Partijen 0 en 1 nemen de unie van de itemsets die ze ontvangen en halen de dubbels eruit. Partij 1 stuurt op zijn beurt alles wat hij in zijn bezit heeft naar partij 0. Partij 0 haalt opnieuw de dubbels eruit en bekomt zo de unie van de itemsets, die weliswaar nog volledig geëncrypteerd is.

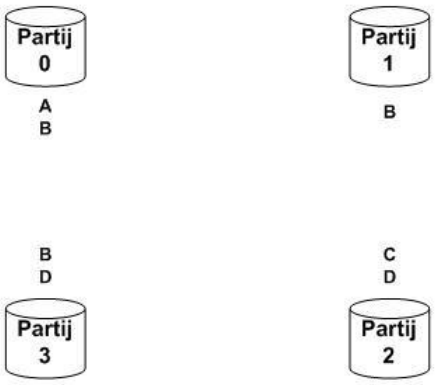
Fase 4: De laatste stap staat in het algoritme beschreven op regels (32-36). De geëncrypteerde unie wordt langs alle partijen gestuurd om gedecrypteerd te worden. Als laatste moeten de valse items nog uit de unie gehaald worden. Hierna wordt het resultaat bekend gemaakt aan de andere partijen. Alle partijen leren de unie, zonder iets te leren over de itemsets van de andere partijen.

Voorbeeld Een voorbeeld kan gevonden worden in Figuren 5.4 tot 5.17. Er zijn 4 partijen. Elke partij heeft een itemset die uit enkele items bestaat. Partij 0 heeft A en B, partij 1 heeft B, partij 2 heeft C en D en partij 3 heeft B en D.
Al de itemsets zullen langs de verschillende partijen gestuurd worden tot alle itemsets door alle partijen geëncrypteerd zijn. Dit stopt dus wanneer partij i de items van partij $(i+1) mod\ 4$ geëncrypteerd heeft. Deze stappen kunnen bekeken worden in Figuren 5.4 tot 5.8.
Hierna zal partij 2 wat hij op dat moment in zijn bezit heeft naar partij 0 sturen. Partij 3 stuurt wat hij op dat moment in zijn bezit heeft naar partij 1. Partij 0 en partij 1 houden de items in hun bezit. Dit gebeurt in Figuur 5.9. Partijen 0 en 1 maken de unie van alle items en halen de dubbels eruit. Partij 1 stuurt zijn unie naar partij 0 en partij 0 maakt opnieuw de unie en haalt de dubbels eruit. Dit kan gevonden worden in Figuren 5.10 tot 5.12.
Partij 0 zal de geëncrypteerde unie langs alle partijen sturen zodat de items kunnen gedecrypteerd worden. Partij 0 krijgt dan de volledig gedecrypteerde unie aan en kan deze bekend maken aan de andere partijen, zoals getoond wordt in Figuren 5.13 tot 5.17.

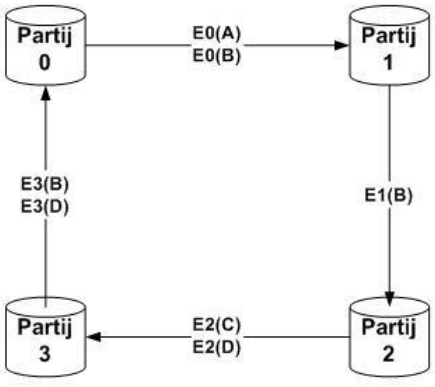
Hoe veilig is het protocol? Het protocol beschermt de itemsets van de partijen, maar geeft toch wat extra informatie vrij:

Algoritme 14 Secure union protocol

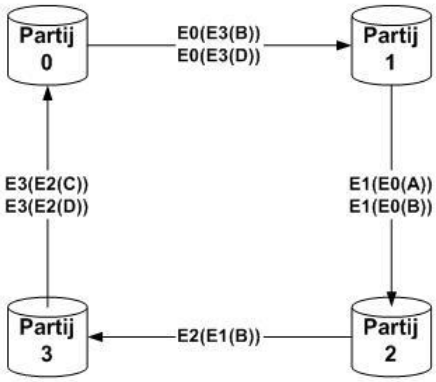
Require: Partijen $0, ..., k-1$ met $k \geq 2$ en elk een set items S_i.
Require: Een set F die bestaat uit valse items.
Fase 1: Genereren keys en aanvullen set i.
for $i = 0...k-1$ **do**
  Partij i genereert een commutatieve, deterministische encryptie key E_i en een decryptie key D_i.
  Partij i vult zijn set aan met valse items tot de maximum grootte.
end for
Fase 2: Encrypteren van de sets door alle partijen.
for $i = 0...k-2$ **do**
  if $i == 0$ **then**
    for $j = 0...k-1$ **do**
      Partij j encrypteert zijn eigen set met E_j en permuteert de items.
      Partij j stuurt de geëncrypteerde set door naar partij $(j+1) mod\ k$.
    end for
  else
    for $j = 0...k-1$ **do**
      Partij j ontvangt een set, encrypteert deze met E_j, permuteert de items en stuurt de set door naar partij $(j+1) mod\ k$.
    end for
  end if
end for
Fase 3: Samenvoegen van de set en de dubbels eruit halen.
for $i = 0...k-2$ **do**
  Partij i encrypteert de set op dat moment in zijn bezit met E_i.
  if $i == even$ **then**
    Partij i stuurt de set op dat moment in zijn bezit naar partij 0.
  else
    Partij i stuurt de set op dat moment in zijn bezit naar partij 1.
  end if
end for
Partij $k-1$ encrypteert de set op dat moment in zijn bezit met E_{k-1} en stuurt de set naar partij 1.
Partij 0 en partij 1 nemen de unie en halen de dubbels eruit.
Partij 1 permuteert de elementen van de set en stuurt die naar partij 0.
Partij 0 neemt de unie van de twee sets en haalt de dubbels eruit.
Fase 4: Decrypteren van de set.
for $i = 0...k-1$ **do**
  Partij i decrypteert de items en stuurt ze door naar partij $(i+1) mod\ k$.
end for
Partij 0 haalt de valse itemsets eruit en stuurt de unie door.



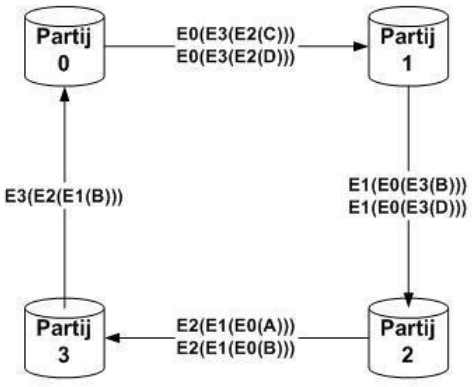
Figuur 5.4: Secure union (1).



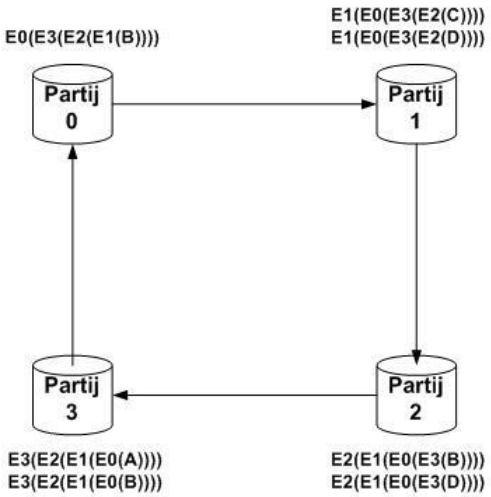
Figuur 5.5: Secure union (2).



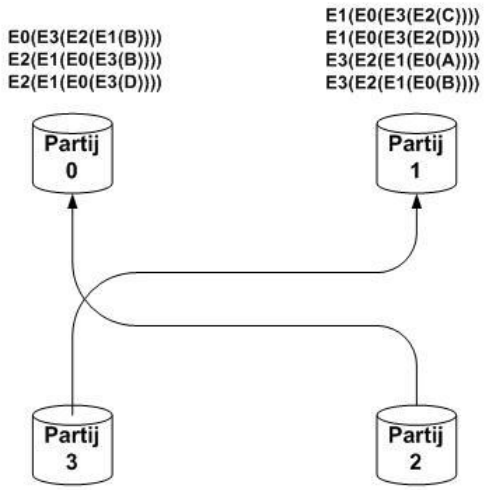
Figuur 5.6: Secure union (3).



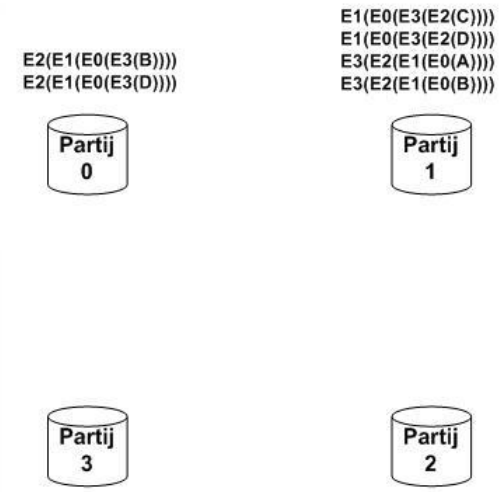
Figuur 5.7: Secure union (4).



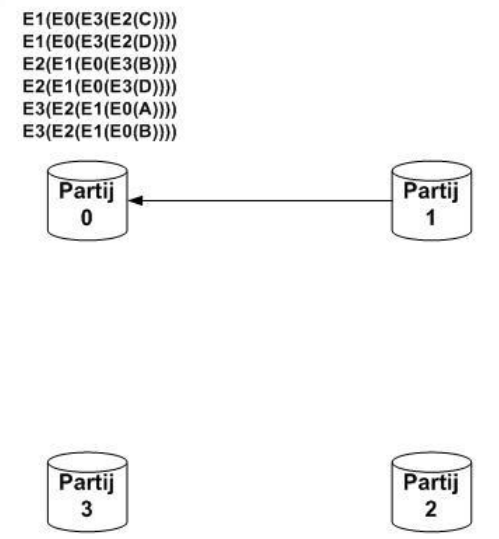
Figuur 5.8: Secure union (5).



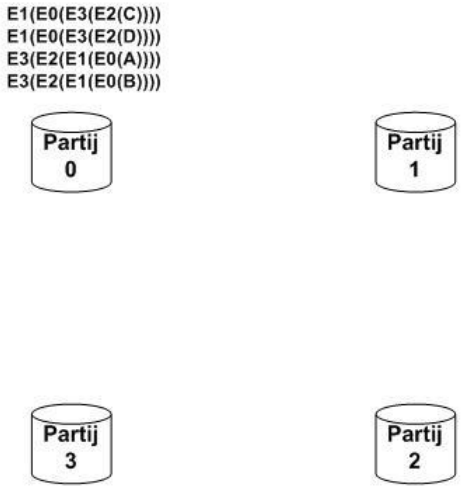
Figuur 5.9: Secure union (6).



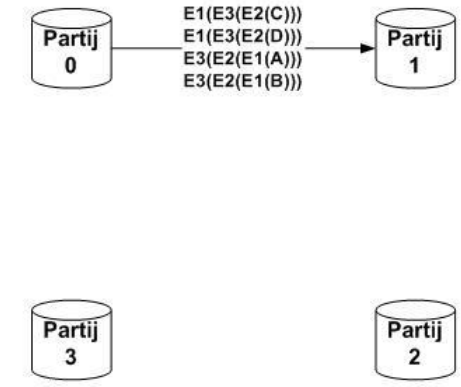
Figuur 5.10: Secure union (7).



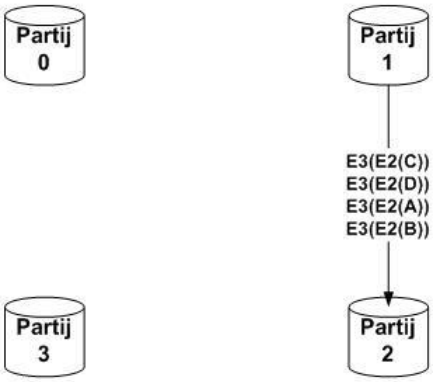
Figuur 5.11: Secure union (8).



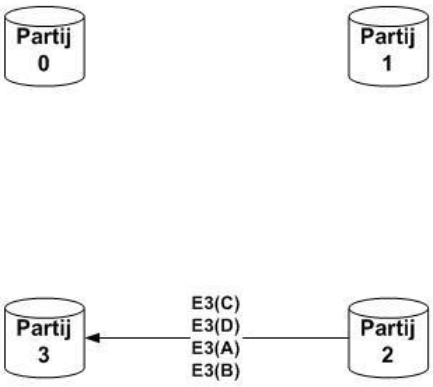
Figuur 5.12: Secure union (9).



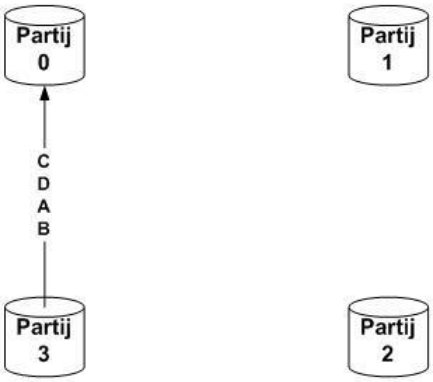
Figuur 5.13: Secure union (10).



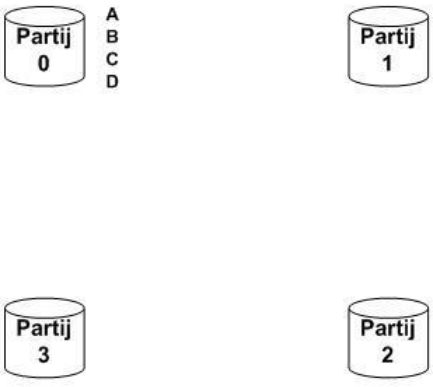
Figuur 5.14: Secure union (11).



Figuur 5.15: Secure union (12).



Figuur 5.16: Secure union (13).



Figuur 5.17: Secure union (14).

1. Partij 0 leert in fase 3 hoeveel items de itemsets van de oneven partijen gemeenschappelijk hebben.

2. Partij 1 leert in fase 3 hoeveel items de itemsets van de even partijen gemeenschappelijk hebben.

3. Partij 0 leert in fase 3 hoeveel items de itemsets van enerzijds de even partijen en anderzijds de oneven partijen gemeenschappelijk hebben.

4. De partijen leren in fase 3 en fase 4 hoe groot de unie is en ook hoe groot de unie aangevuld met de valse itemsets is.

Met de aanname dat deze gegevens de privacy niet schenden, kan aangetoond worden dat het protocol veilig is. Dit gebeurt door een polynomiale tijd simulator te beschrijven die de views van de partijen doorheen de uitvoering van het protocol simuleert aan de hand van de input, de output en de extra gegevens.

Input Elke partij i heeft een itemset S_i, een encryptie key E_i en een decryptie key D_i. Elke partij kent ook de grootte die de itemsets maximaal kunnen aannemen.

Output Elke partij leert de unie van de itemsets.

Fase 1: Al de handelingen die gebeuren in fase 1 zijn lokaal. Elke partij kan zijn aangevulde itemset lokaal construeren aan de hand van zijn eigen input. De view van elke partij bestaat dan uit wat hij zelf geconstrueerd heeft.

Fase 2: Elke partij krijgt een aantal keer een geëncrypteerde itemset aan. Dankzij de encryptie zullen de items in deze set niet te onderscheiden zijn van een random getal. Aangezien de grootte van de itemsets gekend is door alle partijen, kan de geëncrypteerde itemset gesimuleerd worden door een set van dezelfde grootte. Deze set bevat dan random getallen.

Fase 3: Partij 0 en partij 1 krijgen een aantal volledig geëncrypteerde itemsets aan. Voor partij 0 zijn dit de sets van de oneven partijen, voor partij 1 de sets van de even partijen. Omdat de itemsets volledig geëncrypteerd zijn en de encryptie commutatief is, is het niet mogelijk om allemaal random getallen te gebruiken. Wegens de commutativiteit van de encryptie zullen gemeenschappelijke items tussen twee partijen ook gemeenschappelijk moeten zijn in de simulaties. Voor partij 0 is dit mogelijk dankzij de extra informatie (1^e gegeven) die het protocol vrijgeeft. Voor partij 1 kan dit met behulp

van het 2^e gegeven. De overblijvende items kunnen gesimuleerd worden door random getallen. Partij 0 krijgt dan de unie aan die partij 1 geconstrueerd heeft. Door gebruik te maken van de extra informatie (4^e gegeven) kan dit gesimuleerd worden. Als partij 0 weet hoe groot de unie met de valse itemsets is, kan hij afleiden hoe groot de set is die hij van partij 1 aankrijgt. Door dan verder gebruik te maken van het 3^e gegeven, weet hij hoeveel items de twee sets gemeenschappelijk moeten hebben. De overige items kunnen voorgesteld worden door een random getal.

Fase 4: Elke partij krijgt een unie aan van partij $(i-1)$. Partij 0 kent de unie al uit de vorige stap en gebruikt die simulatie hier opnieuw. De andere partijen hebben in fase 2 ook al een deel van de unie gezien. Ze hebben namelijk al een volledig geëncrypteerde itemset in hun bezit gehad. Aangezien ze weten hoe groot de set is die ze aankrijgen dankzij de extra informatie (4^e gegeven) is het mogelijk om de rest aan te vullen met random getallen. De unie die de laatste partij aankrijgt, bestaat uit het resultaat van het protocol nog steeds geëncrypteerd door zijn encryptie key aangevuld met random getallen voor de valse itemsets. Omdat de eigenlijke unie zeker gekend is en verder ook de grootte van de unie met de valse itemsets bekend is, is het mogelijk om deze stap te simuleren.

5.3.4 Secure size of set intersection protocol

In deze sectie wordt een secure size of set intersection protocol besproken uit [49]. Het berekenen van de grootte van de doorsnede kan voor problemen zorgen in het geval van twee partijen. Wanneer een partij de grootte van de doorsnede kent van twee sets en hij kent uiteraard ook zijn eigen set, dan kan die partij hier volledig of gedeeltelijk uit afleiden welke items in de set van de andere partij zitten. Het kan dus gewenst zijn om bij dit protocol uit te gaan van k partijen met $k > 2$.

Doelstelling Er zijn k partijen. Elke partij heeft een itemset. De partijen moeten allemaal de grootte van de doorsnede van de itemsets verkrijgen, zonder iets te leren over de itemsets van de andere partijen.

Werking (zie Algoritme 15) Het protocol maakt gebruik van commutatieve, deterministische encryptie. Er zijn grotendeels vier fases te vinden in het protocol:

Fase 1: De eerste stap staat in het algoritme beschreven op regels (1-5).

De eerste fase van het protocol verloopt analoog aan de eerste fase van het secure union protocol. Het enige verschil is dat er een one way commutatieve, deterministische encryptie key E_i gegenereerd wordt.

Fase 2: De tweede stap staat in het algoritme beschreven op regels (6-18). De tweede fase van het protocol verloopt volledig analoog aan de tweede fase van het secure union protocol.

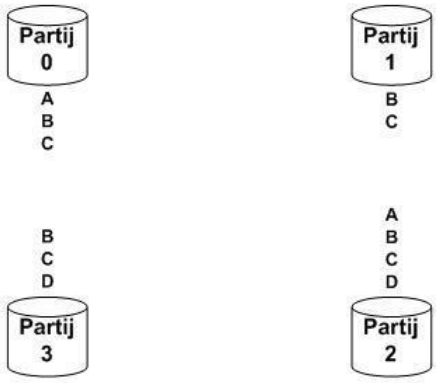
Fase 3: De derde stap staat in het algoritme beschreven op regels (19-25). Alle partijen sturen de set die ze op dat moment in hun bezit hebben naar alle andere partijen behalve naar partij $(i+1) mod\ k$. Dit wordt gedaan om te vermijden dat partijen hun eigen volledig geëncrypteerde itemset te zien krijgen.

Fase 4: De laatste stap staat in het algoritme beschreven op regels (26-32). Elke partij heeft dan een aantal itemsets en kan van deze sets de doorsnede maken. Alle partijen sturen dan deze gemaakte doorsnede naar partij $(i+1) mod\ k$. Wanneer de partijen vervolgens de doorsnede nemen van de doorsnede die ze al hadden en degene die ze net ontvangen hebben, kennen ze het resultaat. Er moet hier dus niet gedecrypteerd worden omdat het alleen belangrijk is om de grootte van de doorsnede te leren en niet de doorsnede zelf. Alle partijen leren de grootte van de doorsnede, zonder iets te leren over de itemsets van de andere partijen.

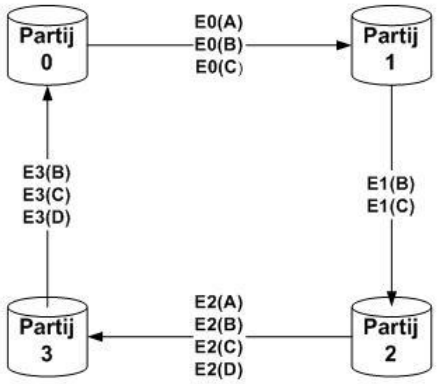
Voorbeeld Een voorbeeld kan gevonden worden in Figuren 5.18 tot 5.25. Er zijn 4 partijen. Elke partij heeft een itemset die uit enkele items bestaat. Partij 0 heeft A, B en C, partij 1 heeft B en C, partij 2 heeft A, B, C en D en partij 3 heeft B, C en D. Al de itemsets zullen langs de verschillende partijen gestuurd worden tot alle itemsets door alle partijen geëncrypteerd zijn. Dit stopt dus wanneer partij i de items van partij $(i+1) mod\ 4$ geëncrypteerd heeft. Deze stappen kunnen bekeken worden in Figuren 5.18 tot 5.22.
Hierna zullen alle partijen wat ze op dat moment in hun bezit hebben naar alle partijen behalve partij $(i+1) mod\ 4$ sturen. Partij 0 stuurt dus naar partijen 2 en 3, partij 1 naar 0 en 3, partij 2 naar 0 en 1, en partij 3 naar 1 en 2. De partijen nemen dan de doorsnede van wat ze al hadden en wat ze nu ontvangen. Deze doorsnede wordt naar partij $(i+1) mod\ 4$ gestuurd. Partij 0 stuurt dus naar partij 1, partij 1 naar 2, partij 2 naar 3 en partij 3 naar 0. De uiteindelijke doorsnede kan nu gemaakt worden. Alle partijen hebben de geëncrypteerde doorsnede en kunnen bijgevolg dus leren hoe groot die doorsnede is. In dit geval is dat 2. Dit kan gevonden worden in Figuren 5.23 tot 5.25.

Algoritme 15 Secure size of set intersection protocol

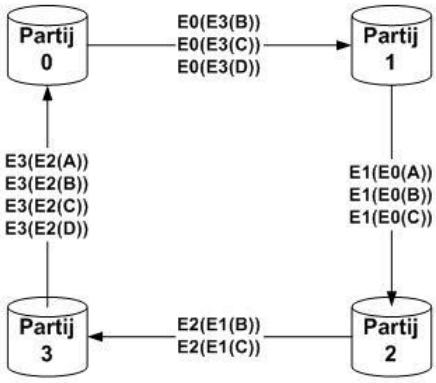
Require: Partijen $0, ..., k-1$ met $k \geq 2$ en elk een set items S_i.
Require: Een set F die bestaat uit valse items.
Fase 1: Genereren keys en aanvullen set i.
for $i = 0...k-1$ **do**
    Partij i genereert een one-way commutatieve, deterministische encryptie key E_i.
    Partij i vult zijn set aan met valse items tot de maximum grootte.
end for
Fase 2: Encrypteren van de sets door alle partijen.
for $i = 0...k-2$ **do**
    if $i == 0$ **then**
        for $j = 0...k-1$ **do**
            Partij j encrypteert zijn eigen set met E_j en permuteert de items.
            Partij j stuurt de geëncrypteerde set door naar partij $(j+1) mod\ k$.
        end for
    else
        for $j = 0...k-1$ **do**
            Partij j ontvangt een set, encrypteert deze met E_j, permuteert de items en stuurt de set door naar partij $(j+1) mod\ k$.
        end for
    end if
end for
Fase 3: Maken van eerste intersectie.
for $i = 0...k-1$ **do**
    Partij i stuurt de set op dat moment in zijn bezit naar alle partijen behalve naar partij $(i+1) mod\ k$.
end for
for $i = 0...k-1$ **do**
    Partij i berekent de doorsnede van de sets die hij op dat moment in zijn bezit heeft.
end for
Fase 4: Maken van tweede intersectie.
for $i = 0...k-1$ **do**
    Partij i stuurt de gemaakte doorsnede door naar partij $(i+1) mod\ k$.
end for
for $i = 0...k-1$ **do**
    Partij i berekent de doorsnede tussen de set die hij al had en de set die hij ontvangt.
end for



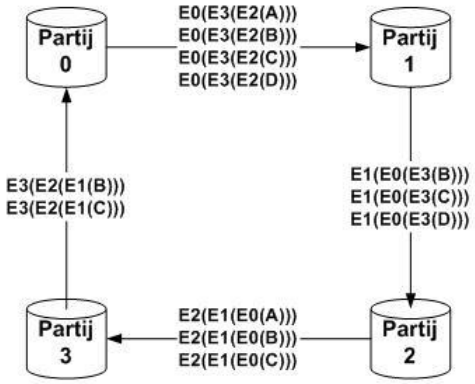
Figuur 5.18: Secure size of set intersection (1).



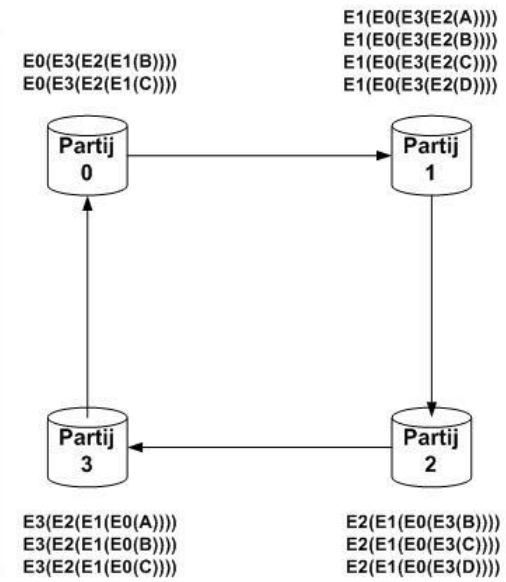
Figuur 5.19: Secure size of set intersection (2).



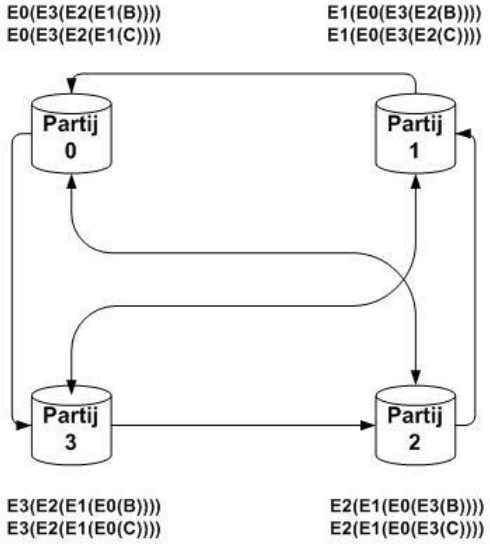
Figuur 5.20: Secure size of set intersection (3).



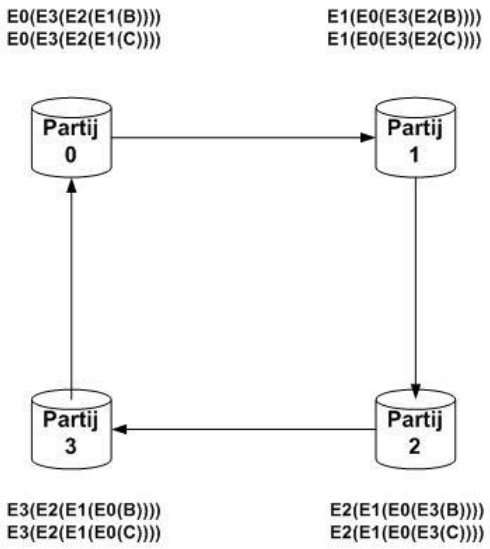
Figuur 5.21: Secure size of set intersection (4).



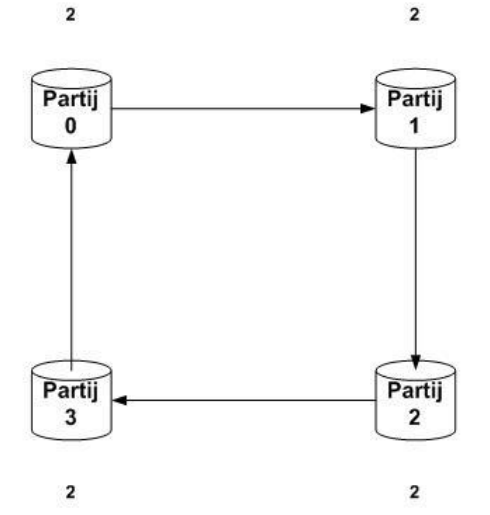
Figuur 5.22: Secure size of set intersection (5).



Figuur 5.23: Secure size of set intersection (6).



Figuur 5.24: Secure size of set intersection (7).



Figuur 5.25: Secure size of set intersection (8).

Hoe veilig is het protocol? Het protocol beschermt de itemsets van de partijen, maar geeft toch wat extra informatie vrij:

- Partij i leert in fase 3 hoeveel items alle partijen $\{1, ..., i-1, i+1, ..., k\}$ gemeenschappelijk hebben. Ze kunnen de volledig geëncrypteerde itemsets van alle partijen behalve zichzelf onderzoeken, en voor elke subset van de partijen bepalen hoeveel items er gemeenschappelijk zijn.
- Partij i leert in fase 4 hoeveel items alle partijen $\{1, ..., i-2, i, ..., k\}$ gemeenschappelijk hebben. De partijen ontvangen immers een doorsnede gemaakt door partij $i-1$. Deze doorsnede is de doorsnede van alle itemsets behalve de itemset van partij $i-1$ zelf.

Met de aanname dat deze gegevens de privacy niet schenden, kan aangetoond worden dat het protocol veilig is. Dit gebeurt door een polynomiale tijd simulator te beschrijven die de views van de partijen doorheen de uitvoering van het protocol simuleert aan de hand van de input, de output en de extra gegevens.

Input Elke partij i heeft een itemset S_i en een encryptie key E_i. Elke partij kent ook de grootte die de itemsets maximaal kunnen aannemen.

Output Elke partij leert de grootte van de doorsnede van de itemsets.

Fase 1: Al de handelingen die gebeuren in fase 1 zijn lokaal. Elke partij kan zijn aangevulde itemset lokaal construeren aan de hand van zijn eigen input. De view van elke partij bestaat dan uit wat hij zelf geconstrueerd heeft.

Fase 2: Elke partij krijgt een aantal keer een geëncrypteerde itemset aan. Dankzij de encryptie zullen de items in deze set niet te onderscheiden zijn van een random getal. Aangezien de grootte van de itemsets gekend is door alle partijen, kan de geëncrypteerde itemset gesimuleerd worden door een set van dezelfde grootte. Deze set bevat dan random getallen.

Fase 3: Elke partij krijgt een aantal volledig geëncrypteerde itemsets aan. Dit zijn de sets van alle partijen behalve zichzelf. Eén van die itemsets is de set die de partij op het einde van fase 2 in zijn bezit had. Deze itemset kan dan ook gesimuleerd worden met de simulatieset uit de simulatie van fase 2. Naast deze set moeten de andere itemsets uiteraard ook gesimuleerd worden. Omdat de itemsets volledig geëncrypteerd zijn en de encryptie commutatief is, is het niet mogelijk om allemaal random getallen te gebruiken. Wegens de commutativiteit van de encryptie zullen gemeenschappelijke items tussen

twee partijen ook gemeenschappelijk moeten zijn in de simulaties. Dit is mogelijk dankzij de extra informatie (1^e gegeven) die het protocol vrijgeeft. De simulatiesets kunnen zo opgebouwd worden dat ze gemeenschappelijke items correct voorstellen. Om de view van de partijen volledig te maken worden de overige items gesimuleerd door random getallen.

Fase 4: Elke partij krijgt een doorsnede aan van partij $(i-1)$. Deze doorsnede kan gesimuleerd worden met behulp van de output van het protocol en de doorsnede die de partij al in zijn bezit heeft. De doorsnede van de set al in zijn bezit met de ontvangen set moet even groot zijn als het resultaat van het protocol. De set kan dus gesimuleerd worden door evenveel items te kiezen uit de doorsnede die de partij al had, als er items in de output zitten. Dankzij de extra informatie (2^e gegeven) die het protocol vrijgeeft kan de view volledig gesimuleerd worden. De partijen weten hierdoor hoe groot de doorsnede die ze ontvangen zal zijn. Ze kunnen deze dan nog aanvullen met random getallen voor zover dat nodig is.

5.3.5 Secure xln(x) protocol

In deze sectie wordt een secure $xln(x)$ protocol besproken uit [44]. Het ontwikkelde $xln(x)$ protocol is van een heel andere aard dan de voorgaande protocollen. Het protocol maakt gebruik van een Yao circuit beschreven in sectie 4.1.2. Het volgt de constructie echter niet volledig. Het probleem wordt opgedeeld in deelproblemen en bepaalde onderdelen hiervan worden met een circuit opgelost. Dit maakt het protocol efficiënter dan wanneer een volledige circuit oplossing zou gebruikt worden. Omdat de circuits alleen praktisch bruikbaar zijn in het geval van twee partijen, is dit protocol dan ook niet geschikt voor meer dan twee partijen.

Doelstelling Er zijn twee partijen, namelijk Alice en Bob. Alice heeft een waarde x_a. Bob heeft een waarde x_b. Alice en Bob moeten allebei een share van $s_a + s_b = (x_a + x_b)ln(x_a + x_b)$ verkrijgen, zonder iets te leren over de waarde van de andere partij. Alice leert dan s_a en Bob leert s_b.

Werking (zie Algoritme 16) Het protocol om $xln(x)$ te berekenen maakt gebruik van twee subprotocollen. Een eerste protocol, het *ln(x) protocol*, krijgt twee waardes x_a en x_b als input en geeft twee random shares van $ln(x_a + x_b)$ als output. Het tweede protocol, krijgt twee waardes u_a en u_b als input en geeft twee random shares van $u_a.u_b$ als output. Dit protocol wordt het *multiplication protocol* genoemd.

Alice en Bob runnen het $ln(x)$ protocol en bekomen zo de shares u_a en u_b. Hierna wordt het multiplication protocol twee keer uitgevoerd. De eerste keer met u_a en x_b als input waardoor Alice en Bob respectievelijk de shares v_a en v_b verkijgen. De tweede keer met x_a en u_b waardoor Alice en Bob respectievelijk de shares w_a en w_b leren.
Alice heeft nu x_a, u_a, v_a en w_a. Ze kan hiermee $s_a = x_a.u_a + v_a + w_a$ construeren. Aangezien $x_a.u_a + x_b.u_b + x_a.u_b + x_b.u_a = (x_a + x_b)(u_a + u_b) = (x_a + x_b)ln(x_a + x_b)$ vormt dit Alice haar share s_a van $(x_a + x_b)ln(x_a + x_b)$. Bob kan dan op dezelfde manier $s_b = x_b.u_b + v_b + w_b$ construeren. Beide partijen leren een share van $xln(x)$, zonder iets te leren over de waarde van de andere partij.

Het ln(x) protocol Het $ln(x)$ protocol berekent een benadering van $ln(x)$ aan de hand van een Taylorreeks:

$$ln(1+\epsilon) = \sum_{i=1}^{k} \frac{(-1)^{i-1}\epsilon^i}{i} = \epsilon - \frac{\epsilon^2}{2} + \frac{\epsilon^3}{3} - ...\frac{\epsilon^k}{k}.$$

Hoe goed de benadering is, hangt af van de waarde van k. Om $ln(x)$ te berekenen zal x eerst geschreven worden als $x = 2^n(1+\epsilon)$ met $-\frac{1}{2} \leq \epsilon \leq \frac{1}{2}$. Hierbij is 2^n de n^e macht van 2 die het kortste bij x ligt. De Taylorreeks van $ln(x)$ zal er dan als volgt uitzien:

$$ln(x) = ln(2^n(1+\epsilon)) = nln2 + \epsilon - \frac{\epsilon^2}{2} + \frac{\epsilon^3}{3} - ...\frac{\epsilon^k}{k}.$$

Het protocol bestaat uit twee delen. Het eerste deel is de uitvoering van een Yao circuit. Het tweede deel bestaat uit een oblivious polynomial evaluation.

- **Yao circuit** Het circuit krijgt x_a als input van Alice en x_b als input van Bob. De som van deze waardes is gelijk aan x. Het circuit zal random shares van $\epsilon.2^N$ en $2^N.nln2$ als output geven. Hierbij is N een vastgelegde waarde zodat $n < N$.

  Shares van $\epsilon.2^N$ Om shares van $\epsilon.2^N$ te bekomen moet eerst de waarde n gekend zijn. Deze kan bekomen worden door naar de twee meest significante bits van x te kijken. Veronderstel dat $x = 9$, x ligt dan tussen 2^3 en 2^4. De binaire voorstelling van x is gelijk aan 1001. Aangezien $n = 3$ of $n = 4$ en 10 de twee meest significante bits zijn, zal er gekozen worden voor $n = 3$. Indien dat 11 geweest was, zou er gekozen zijn voor $n = 4$.
  Omdat $x = 2^n(1+\epsilon)$ zal ook $\epsilon.2^n = x - 2^n$ gelden. Wanneer de waarde van n gekend is, is het mogelijk om $x - 2^n$ te berekenen. Door deze waarde $N - n$ bits naar links te schuiven kan $\epsilon.2^N$ bepaald worden.

Veronderstel opnieuw $x = 9$, $n = 3$ en $N = 7$. De waarde van $x - 2^n$ is dan gelijk aan $9 - 8 = 1$. Deze waarde heeft 1 als binaire voorstelling. Wanneer deze bit $7 - 3 = 4$ plaatsen naar links verschoven wordt, geeft dit 10000, wat gelijk is aan 16.
Om aan te tonen dat dit inderdaad gelijk is aan $\epsilon.2^N$, zal ϵ geleerd worden uit de vergelijking $\epsilon.2^n = x - 2^n$. Door de variabelen x en n in te vullen, kan ϵ berekend worden: $\epsilon = \frac{1}{8}$. Deze ϵ kan dan ingevuld worden in $\epsilon.2^N$, wat inderdaad 16 oplevert.

Shares van $2^N.nln2$ De mogelijke waardes van $2^N.nln2$ worden gehardwired in het circuit. Aangezien $n < N$, kan $2^N.nln2$ maar een vast aantal waardes aannemen.

Het circuit moet verder nog met één mogelijkheid rekening houden en dat is het geval waarin $x = 0$. Als dit zo is, kan x niet geschreven worden als $x = 2^n + \epsilon$. Het circuit zal in dit geval random shares van 0 als output geven.
Na uitvoering van dit protocol heeft Alice de shares α_a en β_a en Bob de shares α_b en β_b zodat $\alpha_a + \alpha_b = \epsilon.2^N$ en $\beta_a + \beta_b = 2^N.nln2$.

- **Oblivious polynomial evaluation** Alice construeert een polynoom $Q(z) = kgv(2, ..., k) \sum_{i=1}^{k} \frac{(-1)^{i-1}(\alpha_a + z)^i}{2^{N(i-1)}i} - z_a$ met $lcm(2, ..., k)$ het kleinste gemeen veelvoud van $\{2, ..., k\}$. Er wordt vermenigvuldigd met deze waarde om breuken weg te werken. Alice en Bob nemen dan deel aan een oblivious polynomial transfer met $Q(z)$ en α_b. Bob leert hierdoor $z_b = Q(\alpha_b) = lcm(2, ..., k)2^N \sum_{i=1}^{k} \frac{(-1)^{i-1}(\epsilon)^i}{i} - z_a$. Alice kent z_a.
Alice construeert $u_a = z_a + lcm(2, ..., k)\beta_a$ en Bob construeert $u_b = z_b + lcm(2, ..., k)\beta_b$. Er kan nu aangetoond worden dat u_a en u_b shares zijn van $ln(x)$:

$$\begin{aligned} u_a + u_b &= z_a + z_b + lcm(2, ..., k).2^N nln2 \\ &= lcm(2, ..., k).2^N \sum_{i=1}^{k} \frac{(-1)^{i-1}\epsilon^i}{i} + lcm(2, ..., k).2^N nln2 \\ &\approx lcm(2, ..., k)2^N lnx \end{aligned}$$

Beide partijen hebben na de uitvoering van het $ln(x)$ protocol een share van $ln(x)$ geleerd, met $x = x_a + x_b$.

Het multiplication protocol Het multiplication protocol krijgt twee waardes u_a van Alice en u_b van Bob als input en geeft twee random shares van $u_a.u_b$ als output. Deze shares worden berekend met behulp van een oblivious

polynomial evaluation. Alice construeert de polynoom $Q(v) = u_a.v - v_a$. Alice en Bob nemen dan deel aan een oblivious polynomial transfer met $Q(v)$ en u_b. Bob leert hierdoor $v_b = Q(u_b) = u_a.u_b - v_a$. Aangezien Alice haar share v_a al kent, hebben beide partijen na uitvoering van het protocol een share van $u_a.u_b$ geleerd.

Algoritme 16 Secure xln(x) protocol

Require: Bob en Alice, de twee partijen.
Require: Alice heeft een waarde x_a.
Require: Bob heeft een waarde x_b.

1: Alice en Bob nemen deel aan het $ln(x)$ protocol met x_a en x_b en leren zo de shares u_a en u_b. (zie Algoritme 17)
2: Alice en Bob nemen deel aan het multiplication protocol met u_a en x_b en leren zo de shares v_a en v_b. (zie Algoritme 18)
3: Alice en Bob nemen deel aan het multiplication protocol met x_a en u_b en leren zo de shares w_a en w_b. (zie Algoritme 18)
4: Alice construeert $s_a = x_a.u_a + v_a + w_a$.
5: Bob construeert $s_b = x_b.u_b + v_b + w_b$.

Algoritme 17 Secure ln(x) protocol

Require: Bob en Alice, de twee partijen.
Require: Alice heeft een waarde x_a.
Require: Bob heeft een waarde x_b.

1: Alice en Bob voeren het Yao circuit uit en verkrijgen shares van $\epsilon.2^N$, namelijk α_a en α_b en shares van $2^N.nln2$, namelijk β_a en β_b.
2: Alice construeert $Q(z) = lcm(2, ..., k) \sum_{i=1}^{k} \frac{(-1)^{i-1}(\alpha_a+z)^i}{2^{N(i-1)}i} - z_a$.
3: Alice en Bob nemen deel aan een oblivious polynomial transfer met $Q(z)$ en α_b, waardoor Bob $z_b = Q(\alpha_b)$ leert.
4: Alice construeert $u_a = z_a + lcm(2, ..., k)\beta_a$ en Bob construeert $u_b = z_b + lcm(2, ..., k)\beta_b$, hun shares van $ln(x)$, met $x = x_a + x_b$.

Algoritme 18 Secure multiplication protocol

Require: Bob en Alice, de twee partijen.
Require: Alice heeft een waarde u_a.
Require: Bob heeft een waarde u_b.

1: Alice construeert $Q(v) = u_a.v - v_a$.
2: Alice en Bob nemen deel aan een oblivious polynomial transfer met $Q(v)$ en u_b, waardoor Bob $v_b = Q(u_b)$ leert. Dit geeft beide partijen hun share van $u_a.u_b$.

Hoofdstuk 6

Privacy preserving data mining algoritmes: Een overzicht

Met behulp van de bouwstenen beschreven in Hoofdstuk 5 is het mogelijk om een heel aantal data mining problemen op een veilige manier op te lossen. De algoritmes die in Hoofdstuk 1 werden besproken zullen hier worden omgevormd naar privacy preserving data mining algoritmes.
Merk op dat er bepaalde gevallen zijn waarvoor nog geen oplossing bestaat. Deze zullen verder aangehaald worden in Hoofdstuk 7.

6.1 Associatie regels in horizontaal verdeelde data: twee of meer partijen

Het algoritme om privacy preserving associatie regels te minen is gebaseerd op een bestaand algoritme voor het minen van associatie regels over horizontaal verdeelde data. Dit algoritme kan gevonden worden in [40] en zal in de volgende sectie beschreven worden. Met behulp van het secure sum en het secure union protocol kan dit algoritme getransformeerd worden naar een veilig algoritme. Dit wordt beschreven in [42].

6.1.1 Het Apriori algoritme voor horizontaal verdeelde data

In [40] wordt het Fast Distributed Mining algoritme (FDM), voorgesteld dat associatie regels zoekt in horizontaal verdeelde data. Het algoritme is gebaseerd op het Apriori algoritme en bestaat grotendeels uit vier delen. Deze delen zullen achtereenvolgens besproken worden, en geïllustreerd worden aan

Notatie	Betekenis
$L_{(n)}$	Globally large n-itemsets.
$CA_{(n)}$	n-Kandidaatsets gegenereerd uit $L_{(n-1)}$.
$LL_{i(n)}$	Locally large n-itemsets bij partij i.
$GL_{i(n)}$	Globally large en locally large n-itemsets bij partij i.
$CG_{i(n)}$	n-Kandidaatsets gegenereerd uit $GL_{i(n-1)}$ bij partij i.
$X.sup_i$	Lokaal support van itemset X bij partij i.
$X.sup$	Globaal support van itemset X.

Tabel 6.1: Notaties van het algoritme.

de hand van een voorbeeld. De gebruikte notaties kunnen gevonden worden in Tabel 6.1.

6.1.1.1 Lokaal kandidaten genereren

Een naïeve manier om de kandidaatsets te genereren is alle mogelijkheden beschouwen. Volgens de notatie zou dan in elke iteratie de set $CA_{(n)}$ gegenereerd worden. Deze set bevat alle kandidaat n-itemsets. Er is echter een efficiëntere methode. Deze methode maakt gebruik van twee notaties, namelijk locally large itemsets en globally large itemsets.
Een itemset X is locally large bij partij i indien het support van X bij die partij groot genoeg is. Dus X is een locally large itemset bij partij i indien $X.sup_i > s.|D_i|$ met s het gewenste support en $|D_i|$ de grootte van de database bij partij i. Wanneer het globaal support van een itemset X groot genoeg is, is X ook globally large. Met globaal support wordt hier het support bedoeld van de itemset X over alle partijen. Itemset X is dus globally large indien $X.sup > s.|D|$. Wanneer een itemset X zowel globally large als locally large is bij een partij i dan is X een gl-large itemset bij die partij.
Met behulp van deze begrippen kunnen enkele opmerkelijke resultaten aangetoond worden die zullen bijdragen tot een methode om op een efficiënte manier kandidaat itemsets te genereren.

Lemma Als een itemset X globally large is, dan bestaat er een partij i, zodat X en al zijn subsets gl-large zijn bij die partij.

Bewijs: Stel dat X een globally large itemset is, die niet locally large is bij een partij:
$\Rightarrow X.sup_i < s.D_i$ voor alle partijen $i = 0, ..., k-1$.
$\Rightarrow X.sup < s.D$.
$\Rightarrow X$ is niet globally large.

Dit geeft een contradictie, dus de veronderstelling is niet juist: X is locally large bij een partij i.

Stelling Voor elke n > 1 geldt dat de set van alle globally large itemsets $L_{(n)}$ een subset is van $CG_{(n)} = \bigcup_{i=0}^{k-1} CG_{i(n)}$, met $CG_{i(n)} = Apriori(GL_{i(n-1)})$.

Bewijs: $X \in L_{(n)}$, dit wil zeggen dat er een partij i bestaat zodat alle (n-1)-subsets van X gl-large zijn bij partij i:
$\Rightarrow X \in CG_{i(n)}$.
$\Rightarrow L_{(n)} \subseteq CG_{(n)} = \bigcup_{i=0}^{k-1} CG_{i(n)} = \bigcup_{i=0}^{k-1} Apriori(GL_{i(n-1)})$.

Voorbeeld De sterkte van deze stelling zal duidelijk worden met behulp van een voorbeeld. Veronderstel een database die verdeeld is over drie partijen: D_0, D_1 en D_2. Neem $L_{(1)} = \{A, B, C, D, E, F, G, H\}$ met A, B en C locally large bij partij 0, B, C en D locally large bij partij 1 en E, F, G en H locally large bij partij 2:

$\rightarrow GL_{0(1)} = \{A, B, C\}$
$\rightarrow GL_{1(1)} = \{B, C, D\}$
$\rightarrow GL_{2(1)}) = \{E, F, G, H\}$

Alle partijen kunnen nu lokaal kandidaat itemsets genereren:

$CG_{0(2)} = Apriori(GL_{0(1)}) = \{AB, AC, BC\}$
$CG_{1(2)} = Apriori(GL_{1(1)}) = \{BC, CD, BD\}$
$CG_{2(2)} = Apriori(GL_{2(1)}) = \{EF, EG, EH, FG, FH, GH\}$

Het effect van de snellere methode is nu meteen duidelijk: $CG_{(2)}$ heeft 11 elementen, terwijl $CA_{(2)}$ er 28 zou hebben.

6.1.1.2 Lokaal prunen

Wanneer de partijen hun kandidaat itemsets gegenereerd hebben, kunnen ze die set prunen. Dit gebeurt door uit $CG_{i(n)}$ de itemsets te verwijderen die niet locally large zijn en zo $LL_{i(n)}$ te bekomen.

Voorbeeld Veronderstel $|D_0| = 50$ en $s = 10\%$. De supports van de kandidaat itemsets van partij 0 zien er als volgt uit: $AB.sup_0 = 5, BC.sup_0 = 10$ en $AC.sup_0 = 2$.

Aangezien $AC.sup_0 = 2 < 5 = s.|D_0|$ zal bij partij 0 AC niet locally large zijn. Hierdoor zal AC uit $CG_{0(2)}$ gehaald worden. Dit zal als gevolg hebben dat $LL_{0(2)}$ alleen nog de itemsets AB en BC bevat. Voor de andere partijen kan dit analoog berekend worden. Dit zal dan bijvoorbeeld het volgende resultaat geven:

$LL_{0(2)} = \{AB, BC\}$
$LL_{1(2)} = \{BC, CD\}$
$LL_{2(2)} = \{EF, GH\}$

Hieruit blijkt dat meer dan de helft van de kandidaat itemsets werd weggepruned.

6.1.1.3 Support uitwisselen

Alle partijen sturen nu berichten naar de andere partijen met daarin hun locally large itemsets. Elke partij antwoordt op zo een bericht door voor elke itemset het lokaal support terug te sturen. Hierdoor kunnen alle partijen voor hun kandidaatsets het globaal support berekenen. Op deze manier kan $GL_{i(n)}$ berekend worden uit $LL_{i(n)}$.

Voorbeeld Veronderstel $D = 150$. De lokale supports van AB hebben de volgende waardes: $AB.sup_0 = 5, AB.sup_1 = 4$ en $AB.sup_2 = 4$.
Aangezien $AB.sup = 13 < 15 = s.|D|$ zal AB niet globally large zijn. De itemset AB zal dus niet voorkomen in $GL_{0(2)}$. Dit kan op dezelfde manier berekend worden voor de andere itemsets en partijen. Dit kan bijvoorbeeld het volgende resultaat geven:

$GL_{0(2)} = \{BC\}$
$GL_{1(2)} = \{BC, CD\}$
$GL_{2(2)} = \{EF\}$

6.1.1.4 Resultaten uitwisselen

Alle partijen sturen de set met hun globally large itemsets naar de andere partijen. Alle partijen nemen dan de unie van de sets die ze aankrijgen en bekomen zo L_n.

Voorbeeld Neem $GL_{i(2)}$ zoals in het vorige voorbeeld. Na het rondsturen van de berichten bekomen de partijen dan $L_{(2)} = \{BC, CD, EF\}$.

6.1.2 Het privacy preserving Apriori algoritme

Het FDM algoritme bevat enkele stappen die de privacy van de gegevens schenden. Deze stappen kunnen vervangen worden door privacy preserving varianten. Het algoritme bevat drie stappen die niet veilig zijn:

- De kandidaatsets moeten uitgewisseld worden tussen de partijen.
- Voor de kandidaatsets moet het support geleerd worden.
- Voor de mogelijke associatie regels moet het confidence bepaald worden.

De eerste stap kan vervangen worden door het secure union protocol. Op deze manier wordt een unie van itemsets berekend. De partijen weten dat al deze itemsets locally large zijn bij minstens één partij. De privacy wordt echter wel beschermd omdat ze niet weten bij welke partij zo een itemset hoort. Het secure union protocol maakt gebruikt van een vastgelegde grootte voor de verzameling die de partijen doorsturen. In dit geval zal dat de grootte van $CA_{(n)}$ zijn.
Het berekenen van het support van de itemsets in de unie kan gebeuren met het secure sum protocol. Het berekenen van het globaal support kan omgevormd worden naar de volgende vergelijking:

$$\begin{array}{rcl} X.sup \geq s.\sum_{i=0}^{k-1}|D_i| & \leftrightarrow & \sum_{i=0}^{k-1} X.sup_i \geq \sum_{i=0}^{k-1} s.|D_i| \\ & \leftrightarrow & \sum_{i=0}^{k-1}(X.sup_i - s.|D_i|) \geq 0 \end{array}$$

Dit komt inderdaad overeen met het berekenen van een som. Wanneer het secure sum protocol wordt uitgevoerd voor een bepaalde itemset en partij 0 berekent op het einde van het protocol de eigenlijke som kan hij beslissen of het globaal support van de itemset groot genoeg is. Indien de som groter is dan nul, is de itemset globally large.
Het bepalen van het confidence van een associatie regel kan eveneens met behulp van het secure sum protocol. Dit gebeurt volledig analoog aan het berekenen van het support van de itemsets:

$$\begin{array}{rcl} \frac{\{X \cup Y\}.sup}{X.sup} \geq c & \leftrightarrow & \frac{\sum_{i=0}^{k-1} XY.sup_i}{\sum_{i=0}^{k-1} X.sup_i} \geq c \\ & \leftrightarrow & \sum_{i=0}^{k-1} XY.sup_i \geq \sum_{i=0}^{k-1} c.(X.sup_i) \\ & \leftrightarrow & \sum_{i=0}^{k-1}(XY.sup_i - c.X.sup_i) \geq 0 \end{array}$$

Indien het niet gewenst is dat de partijen de eigenlijke waarde van het support en het confidence leren kan de laatste stap vervangen worden door een Yao circuit te construeren. De eerste en de laatste partij geven dan elk een

waarde aan het circuit en dit geeft dan true indien de waarde van de laatste partij groter is dan de waarde van de eerste partij. Partij 0 geeft het random getal als input. De laatste partij geeft de som vermeerdert met de random waarde als input aan het circuit. Het uitvoeren van het circuit is dan een veilige manier om de twee waardes te vergelijken. Geen enkele partij leert zo het support of het confidence, ze leren alleen de itemsets met een support dat groot genoeg is en de associatie regels met een confidence dat groot genoeg is. De volledige beschrijving van dit algoritme kan gevonden worden in Algoritme 19.

Algoritme 19 Het privacy preserving Apriori algoritme

1: Fase 1: De frequent itemsets bepalen.
2: Alle partijen genereren $GL_{i(1)}$.
3: **for** $(n = 2; GL_{i(n-1)} \neq \emptyset; n++)$ **do**
4: Alle partijen genereren $CG_{i(n)} = Apriori(GL_{i(n-1)})$.
5: Alle partijen prunen lokaal en verkrijgen zo $LL_{i(n)}$.
6: Het secure union protocol wordt uitgevoerd om $\cup_{i=0}^{k-1} LL_{i(n)}$ te bepalen.
7: Het secure sum protocol wordt gebruikt om uit $LL_{i(n)}$, $GL_{i(n)}$ te bepalen.
8: Het secure union protocol wordt uitgevoerd om $\cup_{i=0}^{k-1} GL_{i(n)}$ te bepalen.
9: **end for**
10: Fase 2: De associatie regels bepalen.
11: Voor elke regel uit het resultaat wordt getest of het confidence groot genoeg is. Hiervoor wordt het secure sum protocol gebruikt.
12: De regels die voldoen aan de voorwaarde zijn geldige associatie regels.

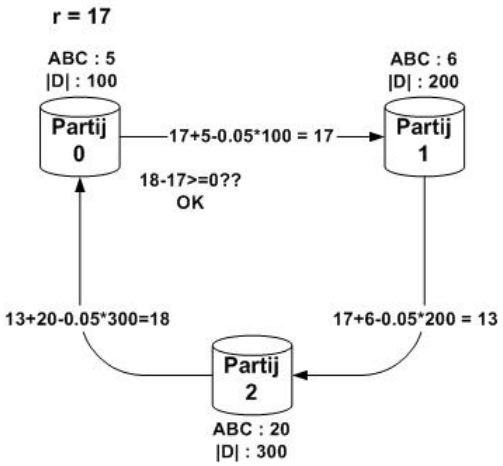
6.1.3 Moeilijkheden in het geval van twee partijen

Zowel het secure union protocol als het secure sum protocol hebben privacy problemen in het geval van twee partijen. De fase van het algoritme waarin het secure union protocol gebruikt wordt, kan extra beveiligd worden door het lokaal prunen weg te laten. Op deze manier worden alle itemsets getest op hun support en niet alleen de itemsets die locally large zijn. De partijen kunnen hierdoor niet afleiden welke itemsets locally large zijn bij de andere partij.
Het secure sum protocol kan beveiligd worden door de laatste stap te vervangen door een Yao circuit. Dit vangt het probleem op dat de partijen de echte waarde, en dus ook de waarde van de andere partij, leren.

Toch blijft deze fase van het algoritme een probleem. Als een itemset globally large is en een partij weet dat die itemset niet locally large is bij zichzelf dan moet die itemset wel locally large zijn bij de andere partij. Dit heeft niets te maken met de manier waarop het protocol werkt. Het probleem zit hier in het resultaat en kan dus niet vermeden worden zonder het resultaat minder correct te maken.

6.1.4 Voorbeeld

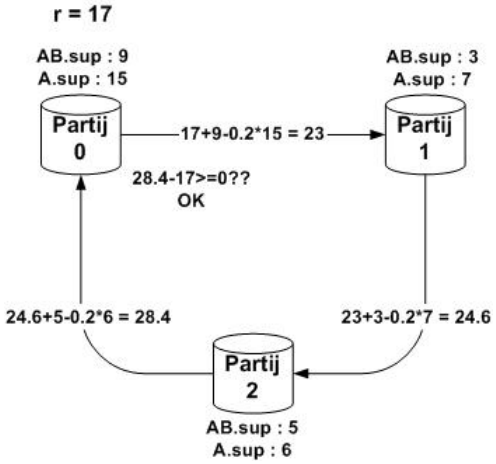
Een voorbeeld voor het berekenen van het globaal support van een itemset ABC kan gevonden worden in Figuur 6.1. Er wordt een support van 5% aangenomen. Aangezien $18 > 17$ is het globaal support van ABC ook groter dan 0. Hieruit volgt dan dat ABC globally large is. Figuur 6.2 illustreert het berekenen van het confidence van de associatie regel $A \Rightarrow B$. Volgens de uitwerking van het secure sum protocol voldoet de associatie regel aan het vooropgestelde confidence van 20%.



Figuur 6.1: Berekenen van het globaal support van de itemset ABC.

6.2 Associatie regels in verticaal verdeelde data

Het minen van associatie regels in verticaal verdeelde data op een privacy preserving manier kan met behulp van het secure scalair product protocol of het secure size of set intersection protocol.



Figuur 6.2: Berekenen van het confidence van de associatie regel $A \Rightarrow B$.

ID	A	B	C	D	E
1	1	0	1	1	0
2	0	1	1	0	1
3	1	1	1	0	1
4	0	1	0	0	1

Tabel 6.2: Voorstelling voor het secure scalair product protocol.

6.2.1 De voorstelling van de transactie database

Om de protocollen makkelijk te kunnen gebruiken, wordt de transactie database op een andere manier voorgesteld. Twee alternatieve methodes voor de voorstelling in Tabel 1.1 in Hoofdstuk 1 kunnen gevonden worden in Tabel 6.2 en Tabel 6.3. De eerste alternatieve voorstelling wordt gebruikt bij het secure scalair product protocol, de tweede voorstelling bij het secure size of set intersection protocol. Bij het secure scalair product protocol zijn er vectoren nodig, bij het secure size of set intersection protocol verzamelingen.

6.2.2 Het private minen van associatie regels in verticaal verdeelde data: twee partijen

In [43] wordt een eerste algoritme voorgesteld om op een veilige manier associatie regels te genereren over verticaal verdeelde data. Dit algoritme maakt gebruik van het secure scalair product protocol. Het algoritme wordt voorgesteld als een algoritme voor twee partijen. Het is gebaseerd op het Apriori algoritme.

Itemset	**ID's**
A	1,3
B	2,3,4
C	1,2,3
D	1
E	2,3,4
AB	3
AC	1,3
AD	1
AE	3
BC	2,3
BD	-
BE	2,3,4
CD	1
CE	2,3
DE	-
ABC	3
ABD	-
ABE	3
ACD	1
ACE	3
ADE	-
BCD	-
BCE	2,3
BDE	-
CDE	-
BCDE	-
ACDE	-
ABDE	-
ABCE	3
ABCD	-
ABCDE	-

Tabel 6.3: Voorstelling voor het secure size of set intersection protocol.

6.2.2.1 Veronderstellingen in verband met de attributen

Bij een verticale verdeling is het belangrijk om te weten welke veronderstellingen er gemaakt worden over de attributen. In dit geval wordt er aangenomen dat alle attributen gekend zijn door de partijen. Beide partijen kennen dus uiteraard hun eigen attributen maar weten ook welke attributen zich bij de andere partij bevinden.

6.2.2.2 Het privacy preserving Apriori algoritme

Het grote probleem bij het vinden van associatie regels in verticaal verdeelde data is het berekenen van het support van itemsets. Indien een itemset bestaat uit attributen die zich bij dezelfde partij bevinden dan is er geen enkel probleem. Die partij kan het support berekenen zonder informatie nodig te hebben van andere partijen. Indien de attributen van de itemset over de twee partijen verdeeld zijn moet er samengewerkt worden. Wanneer de transactie database voorgesteld is zoals in Tabel 6.2 kan het support berekend worden met behulp van het secure scalair product protocol.
Het secure scalair product protocol dat in [43] (sectie 5.3.2.1) wordt voorgesteld, heeft volgens [50] veiligheidsproblemen in het geval van binaire vectoren. Het is uiteraard mogelijk om andere secure scalair product protocollen in het algoritme te verwerken, zoals de protocollen uit [49] (sectie 5.3.2.2) en [50] (sectie 5.3.2.3). Deze twee protocollen kunnen ook gebruikt worden met meer dan twee partijen. Door deze protocollen te beschouwen als secure scalair product protocollen voor het voorgestelde algoritme kan het ook gebruikt worden voor meer dan twee partijen. De volledige beschrijving van dit algoritme kan gevonden worden in Algoritme 20.

6.2.2.3 Voorbeeld

Veronderstel de transactie database uit Tabel 6.2, een support van 75% en een confidence van 50%. Neem aan dat de attributen A en B zich bij partij 0 bevinden en C, D en E bij partij 1. De uitvoering van het Apriori algoritme begint met de frequent 1-itemsets te berekenen. Dit kan elke partij doen.
Partij 0 berekent het support van A en B. De itemset A heeft een te laag support en valt dus weg. Partij 1 berekent het support van C, D en E. Hier valt D weg. Dit geeft dan $L_1 = \{B, C, E\}$.
Volgens het Apriori algoritme geeft dit de kandidaat 2-itemsets BC, BE en CE. Hierdoor is $C_2 = \{BC, BE, CE\}$. Partij 0 kan het support van CE berekenen. Hij bezit namelijk beide attributen. Het support van CE is te laag en valt dus weg. Voor de twee andere itemsets moet het secure scalair product protocol gebruikt worden.

De nodige vectoren kunnen gevonden worden in Tabel 6.2:

$BC : (0, 1, 1, 1).(1, 1, 1, 0) = 0 + 1 + 1 + 0 = 2$
$BE : (0, 1, 1, 1).(0, 1, 1, 1) = 0 + 1 + 1 + 1 = 3$

Hieruit kan besloten worden dat alleen de itemset BE overblijft. De associatie regels $B \Rightarrow E$ en $E \Rightarrow B$ kunnen dan geconstrueerd worden. Partij 0 kan beslissen of $B \Rightarrow E$ een geldige regel is. Hij kent het support van BE dankzij het uitvoeren van het secure scalair product protocol en uiteraard ook het support van B aangezien dit zijn attribuut is. Hieruit kan afgeleid worden dat het confidence van $B \Rightarrow E$ gelijk is aan 100%, dus het is een geldige regel. Partij 1 kan hetzelfde doen voor $E \Rightarrow B$. Dit geeft eveneens een confidence van 100%, dus ook dit is een geldige regel.

6.2.3 Het private minen van associatie regels in verticaal verdeelde data: meer dan twee partijen

Een tweede algoritme voor het private minen van associatie regels wordt voorgesteld in [49]. Er wordt beweerd dat dit algoritme alleen kan gebruikt worden in het geval dat er meer dan twee partijen zijn. Ook dit algoritme is gebaseerd op het Apriori algoritme en maakt gebruik van het secure size of set intersection protocol.

6.2.3.1 Veronderstellingen in verband met de attributen

Net zoals bij het vorige algoritme wordt hier weer verondersteld dat alle partijen alle attributen kennen, en dus niet alleen hun eigen attributen. Ze weten ook welk attribuut zich bij welke partij bevind.

6.2.3.2 Het privacy preserving Apriori algoritme

Het probleem van het vinden van het support van een itemset wordt hier opgelost met behulp van het secure size of set intersection protocol. Het algoritme wordt voorgesteld als een algoritme voor meer dan twee partijen. Het is echter wel mogelijk om dit algoritme te gebruiken in het geval van twee partijen. Het secure size of set intersection protocol kan immers gebruikt worden voor twee partijen. Dit in tegenstelling tot wat er beweerd wordt.
Het algoritme gaat analoog aan het algoritme dat gebruik maakt van het secure scalair product protocol. Het support berekenen van een itemset waarvan alle attributen zich bij dezelfde partij bevinden, levert geen problemen

op. Die partij kan dat perfect berekenen zonder hulp van de andere partijen. Wanneer de attributen verdeeld zijn over meerdere partijen moet er wel samengewerkt worden. Met een transactie database zoals in Tabel 6.3 kan het secure size of set intersection protocol gebruikt worden om het support te berekenen. Net zoals het secure union protocol maakt het secure size of set intersection gebruik van een vastgelegde grootte voor de verzamelingen die de partijen rondsturen. In dit geval zal die grootte gelijk zijn aan het aantal transacties in de transactie database. De volledige beschrijving van dit algoritme kan gevonden worden in Algoritme 20.

Algoritme 20 Het privacy preserving Apriori algoritme

1: Fase 1: De frequent itemsets bepalen.
2: C_1 = Alle 1-itemsets.
3: L_1 = Alle large 1-itemsets.
4: **for** $(n = 2; L_{n-1} \neq \emptyset; n++)$ **do**
5: $C_n = Apriori(L_{n-1})$.
6: **for** Alle kandidaten in C_n **do**
7: **if** Alle items zich bij dezelfde partij bevinden **then**
8: Die partij berekent het support.
9: **else**
10: Support wordt berekend met het secure scalair product protocol of het secure size of set intersection protocol.
11: **end if**
12: De kandidaten waarvan het support groot genoeg is worden toegevoegd aan $L_{(n)}$.
13: **end for**
14: **end for**
15: Fase 2: De associatie regels bepalen.
16: Voor elke regel uit het resultaat wordt getest of het confidence groot genoeg is. De support waardes die hiervoor nodig zijn worden berekend met het secure scalair product protocol of het secure size of set intersection protocol.
17: De regels die voldoen aan de voorwaarde zijn geldige associatie regels.

6.2.3.3 Voorbeeld

Veronderstel de transactie database uit Tabel 6.3, een support van 75% en een confidence van 50%. Neem aan dat de attributen A en B zich bij partij 0 bevinden, C en D bij partij 1 en E bij partij 2. De uitvoering van het Apriori

algoritme begint met de frequent 1-itemsets te berekenen. Dit kan elke partij voor zich doen. Partij 0 berekent het support van A en B. De itemset A heeft een te laag support en valt dus weg. Partij 1 berekent het support van C en D. Hier valt D weg. Partij 2 berekent het support van E. Dit geeft dan $L_1 = \{B, C, E\}$. Volgens het Apriori algoritme geeft dit de kandidaat 2-itemsets $C_2 = \{BC, BE, CE\}$. Voor het berekenen van het support van deze itemsets moet het secure size of set intersection protocol uitgevoerd worden. De inputs van de verzamelingen kunnen gevonden worden in Tabel 6.3 achter de itemsets. Voor de itemset BC geeft dit $\{2, 3, 4\}$ voor B en $\{1, 2, 3\}$ voor C:

$BC : |\{2, 3, 4\} \cap \{1, 2, 3\}| = 2$
$BE : |\{2, 3, 4\} \cap \{2, 3, 4\}| = 3$
$CE : |\{1, 2, 3\} \cap \{2, 3, 4\}| = 2$

Hieruit kan besloten worden dat alleen de itemset BE overblijft. De rest van het algoritme verloopt analoog aan het voorbeeld in het geval van twee partijen.

6.2.4 Het gevaar van de verticale verdeling

De aard van dit probleem, namelijk de verticale verdeling van de data, zorgt voor bijkomende gevaren. Het volgende voorbeeld geeft weer wat zou kunnen voorkomen.

Voorbeeld Veronderstel de transactie database uit Tabel 6.4, een support van 75% en een confidence van 90%. Neem aan dat het attribuut A zich bij partij 0 bevindt en het attribuut B bij partij 1. De uitvoering van het algoritme zal er als volgt uitzien:

$L_1 = \{A, B\} \rightarrow C_2 = \{AB\} \rightarrow L_2 = \{AB\}$

Partij 0 weet nu dat de itemset AB een support heeft van 75%. Aangezien partij 0 een 1 heeft bij de transacties 2, 3 en 4 zal partij 1 bij dezelfde transacties ook een 1 moeten hebben. Anders zou AB nooit een support van 75% kunnen hebben. Partij 1 kan dan het confidence van de associatie regel $B \Rightarrow A$ bepalen. Deze partij bezit immers het attribuut B en kan dus het support van B berekenen. Het support van AB werd al berekend. Partij 1 kan hier uit afleiden dat het confidence van $B \Rightarrow A$ 3/4 bedraagt. Dit komt overeen met 75%, wat als gevolg heeft dat $B \Rightarrow A$ geen geldige regel is.
Partij 0 kan met dit gegeven weer iets afleiden. De partij weet namelijk al dat partij 1 een 1 heeft bij de transacties 2, 3 en 4. Bij transactie 1 kan

ID	A	B
1	0	1
2	1	1
3	1	1
4	1	1

Tabel 6.4: Gemanipuleerde transactie database.

dus nog een 0 of een 1 voorkomen. Indien dit een 0 zou zijn dan zou het confidence van $B \Rightarrow A$ 3/3, of 100% bedragen. In dit geval zou het dus wel een geldige regel geweest zijn. Aangezien dit volgens partij 1 niet zo is, kan partij 0 hieruit afleiden dat partij 1 een 1 heeft bij transactie 1. Op deze manier heeft partij 0 alle waardes van partij 1 kunnen leren.
Dit is uiteraard een gemanipuleerd voorbeeldje. Het is niet waarschijnlijk dat dergelijke voorbeelden in de realiteit voorkomen omdat data mining altijd handelt over grote hoeveelheden data.

6.2.5 Het gevaar van probing attacks

Wanneer data verticaal verdeeld is, is het gevaar van een probing attack zeer reëel. Een probing attack is een aanval waarbij een partij de data van een andere partij gaat onderzoeken. Dit doet hij door foutieve inputs te geven aan de protocollen. Veronderstel het voorbeeld waarin het secure scalair product protocol gebruikt werd om het support te berekenen. Voor de itemset BC gaat partij 0 $(0, 1, 1, 1)$ als input geven en partij 1 $(1, 1, 1, 0)$.
Wanneer partij 1 het protocol niet volgt, kan hij door middel van een probing attack waardes van partij 0 te weten komen. Om bijvoorbeeld te testen of partij 0 een 0 of een 1 heeft bij transactie 1 kan partij 1 als input aan het protocol $(1, 0, 0, 0)$ geven. Het secure scalair product zal dan 1 zijn als partij 0 een 1 heeft bij transactie 1, en 0 in het andere geval.
Bij het secure size of set intersection protocol kan hetzelfde gedaan worden. Voor de itemset BC zal partij 0 $\{2, 3, 4\}$ als input geven en partij 1 $\{1, 2, 3\}$. Partij 1 kan nu op dezelfde manier valsspelen door $\{1\}$ als input te geven. Het resultaat van het protocol zal dan 1 zijn als partij 0 een 1 heeft bij transactie 1, en 0 in het andere geval.

6.2.5.1 Beveiligen van het secure scalair product

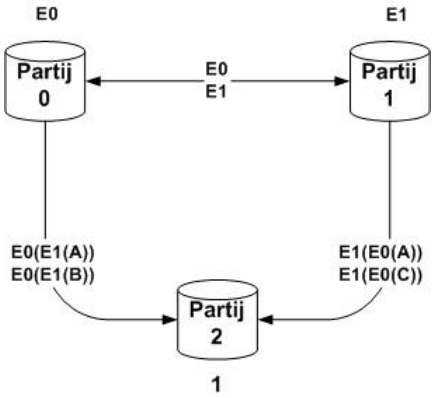
Er is nog geen methode gekend waarbij een secure scalair product beveiligd wordt tegen een probing attack.

6.2.5.2 Beveiligen van het secure size of set intersection protocol

Het is mogelijk om het secure size of set intersection protocol uit te breiden zodat het beschermd is tegen dergelijke probing attacks. Dit kan gedaan worden door fase drie, de eerste intersectie, aan te passen. Wanneer de partijen de eerste intersectie maken kunnen ze testen of de intersectie groter is dan een bepaalde threshold. Indien dit niet het geval is, wordt het protocol gestopt. Als de intersectie klein is, dan is de kans groot dat één van de partijen een verkeerde input gebruikt heeft. De threshold kan gelijk gesteld worden aan het support. Op deze manier leren de partijen het support van een itemset alleen als het support groot genoeg is. Probing attacks worden zo opgevangen.
Deze methode geeft een probleem wanneer het gaat over itemsets waarbij de items verdeeld zijn over twee partijen. Als dit het geval is dan zijn er slechts twee partijen die het protocol uitvoeren. Er kan dan niet gewerkt worden met de threshold omdat er maar één intersectie zal gebeuren en dat is die in fase drie. Het heeft geen zin om het protocol op dat moment te stoppen omdat beide partijen het resultaat al kennen.
Indien er meer dan twee partijen zijn, kan een derde partij gebruikt worden om te testen op een probing attack. Dit kan gebeuren zoals wordt weergegeven in Figuur 6.3. De twee partijen wisselen hun encryptie keys uit en encrypteren hun items met zowel hun eigen key als met de key van de andere partij. De geëncrypteerde waardes sturen ze dan naar een derde partij. Deze partij berekent de intersectie en kijkt of ze groter is dan de threshold. Indien dit het geval is, stuurt de partij de bekomen waarde naar de twee andere partijen. Als dit niet zo is, stuurt de partij niets door en leren de twee partijen de waarde niet.
Wanneer er maar twee partijen zijn kan het secure size of set intersection protocol geen probing attack voorkomen.

6.3 Decision trees in horizontaal verdeelde data: twee partijen

Voor het minen van decision trees in horizontaal verdeelde data is een algoritme voorgesteld dat zich beperkt tot het geval waarin er twee partijen zijn. Dit algoritme staat beschreven in [44] en maakt gebruik van het secure $xln(x)$ protocol. Het algoritme berekent een approximatie van het ID3 algoritme.



Figuur 6.3: Voorkomen van een probing attack.

6.3.1 De ID3$_\delta$ approximatie

Het ID3 algoritme kiest in elke stap het attribuut dat de beste classificatie geeft. Veronderstel nu dat een attribuut A de beste information gain heeft. Bij het ID3 algoritme zou A gekozen worden als attribuut voor die knoop. Bij de ID3$_\delta$ approximatie wordt er ook nog gekeken naar de attributen met een information gain die niet meer dan δ verschilt van de gain van A. Indien het attribuut B voldoet aan deze eis zullen er twee decision trees worden opgesteld. Namelijk één met A in de knoop en één met B in de knoop.
Dit heeft als gevolg dat een δ-approximatie niet één decision tree als resultaat geeft maar een verzameling van decision trees. In deze toepassing is de approximatie het gevolg van het gebruik van het $xln(x)$ protocol. Dit protocol maakt gebruik van een Taylorreeks die wordt uitgewerkt tot een bepaalde term. Hoever de Taylorreeks wordt uitgewerkt bepaalt hoe goed de approximatie is.

6.3.2 Het privacy preserving ID3$_\delta$ algoritme

Het ID3 algoritme bestaat uit drie verschillende stappen. Elk van deze onderdelen kan vervangen worden door een privacy preserving variant. Hiervoor wordt gebruik gemaakt van Yao circuits en het secure $xln(x)$ protocol.

R is leeg Aangezien het gaat om een horizontale verdeling kennen beide partijen alle attributen. Ze kennen ook het tussenresultaat van de boom en weten dus of R leeg is of niet. Er moet nu berekend worden welke class waarde het meeste voorkomt in de transacties in T.

Dit gebeurt met behulp van een Yao circuit. Partij 0 geeft $(|T_0(c_1)|, ..., |T_0(c_l)|)$ als input en partij 1 geeft $(|T_1(c_1)|, ..., |T_1(c_l)|)$ als input. Het circuit output c_i met de grootste $|T_0(c_i) + T_1(c_i)|$.

Alle transacties in T hebben dezelfde class waarde Om te testen of alle transacties in T dezelfde class waarde hebben wordt een Yao circuit gebruikt. Elke partij kijkt naar zijn eigen transacties en bepaalt of er maar één class waarde voorkomt of niet. Als dit het geval is dan wordt die class waarde c_i als input gegeven aan het circuit. Zijn er nog meerdere waardes dan wordt een vast symbool $\perp$ als input gegeven. Het circuit vergelijkt de beide input waardes en output de class waarde c_i als ze gelijk zijn, en $\perp$ in het andere geval.

Anders In het andere geval moet het attribuut bepaald worden dat de transacties in T het beste classificeert. Hiervoor moeten de partijen samenwerken aangezien ze beide een deel van de transacties in hun bezit hebben. De information gain moet berekend worden voor elk attribuut A:

$$
\begin{aligned}
&\textstyle\sum_{j=1}^{m} \frac{|T(a_j)|}{|T|} Entropy(T(a_j)) \\
&\qquad = \ \textstyle\frac{1}{|T|} \sum_{j=1}^{m} |T(a_j)| \sum_{i=1}^{l} \frac{-|T(a_j,c_i)|}{|T(a_j)|} log \frac{|T(a_j,c_i)|}{|T(a_j)|} \\
&\qquad = \ \textstyle\frac{1}{|T|} \sum_{j=1}^{m} |T(a_j)| \sum_{i=1}^{l} \frac{-|T(a_j,c_i)|}{|T(a_j)|} log|T(a_j,c_i)| \\
&\qquad\qquad\qquad\qquad - \textstyle\frac{-|T(a_j,c_i)|}{|T(a_j)|} log|T(a_j)| \\
&\qquad = \ \textstyle\frac{1}{|T|} - \sum_{j=1}^{m} \sum_{i=1}^{l} |T(a_j)| \frac{|T(a_j,c_i)|}{|T(a_j)|} log|T(a_j,c_i)| \\
&\qquad\qquad\qquad\qquad + \textstyle\sum_{j=1}^{m} \sum_{i=1}^{l} |T(a_j)| \frac{|T(a_j,c_i)|}{|T(a_j)|} log|T(a_j)| \\
&\qquad = \ \textstyle\frac{1}{|T|} - \sum_{j=1}^{m} \sum_{i=1}^{l} |T(a_j,c_i)| log|T(a_j,c_i)| \\
&\qquad\qquad\qquad\qquad + \textstyle\sum_{j=1}^{m} |T(a_j)| log|T(a_j)|
\end{aligned}
$$

Beide partijen moeten dus een aantal keer $xlog(x)$ berekenen, met x een waarde waarvan ze beide een share hebben. Dit kan omgezet worden naar het secure $xln(x)$ protocol:

$$
\begin{aligned}
Gain' &= \textstyle\sum_{j=1}^{m} \frac{|T(a_j)|}{|T|} Entropy(T(a_j)).|T|.ln2 \\
&= - \textstyle\sum_{j=1}^{m} \sum_{i=1}^{l} |T(a_j,c_i)| ln|T(a_j,c_i)| + \sum_{j=1}^{m} |T(a_j)| ln|T(a_j)|
\end{aligned}
$$

Met behulp van het secure $xln(x)$ protocol zullen beide partijen een share verkrijgen van het resultaat. Deze waardes worden als input gegeven aan een circuit dat de overeenkomstige waardes optelt en output welk attribuut

de transacties het beste classificeert. De volledige werking van het algoritme kan gevonden worden in Algoritme 21.

Algoritme 21 Het privacy preserving ID3 algoritme

Require: R, de set van attributen.
Require: C, het class attribuut.
Require: T, de set van transacties.
1: **if** De partijen testen of R leeg is **then**
2: Yao circuit: $(|T_0(c_1)|, ..., |T_0(c_l)|$ en $(|T_1(c_1)|, ..., |T_1(c_l)|$ als input en als output c_i met de grootste $|T_0(c_i) + T_1(c_i)|$.
3: Return een blad met als class waarde c_i.
4: **else if** De partijen gebruiken een Yao circuit om te testen of alle transacties in T dezelfde class waarde c_i hebben **then**
5: Return een blad met als class waarde c_i.
6: **else**
7: Bepaal het attribuut A dat de transacties in T het beste classificeert: Run het secure $xln(x)$ protocol voor de waardes $|T(a_j)|$ en $|T(a_j, c_i)|$ en gebruik een Yao circuit om de waardes te vergelijken en het beste attribuut te leren.
8: Verdeel T in m delen $T(a_1), ..., T(a_m)$ zodat $a_1, ..., a_m$ de verschillende waardes van attribuut A zijn.
9: Return een boom met als wortel een knoop A en m takken $a_1, ..., a_m$ zodat tak i $ID3(R - \{A\}, C, T(a_i))$ bevat.
10: **end if**

6.3.3 Evaluatie van een nieuwe transactie

Aangezien beide partijen de volledige boom leren, is het niet nodig om samen te werken als er een nieuwe transactie moet geclassificeerd worden. De partijen kunnen dit individueel doen.

6.4 Decision trees in verticaal verdeelde data

Het probleem van het minen van decision trees in verticaal verdeelde data is van dezelfde aard als het berekenen van associatie regels in verticaal verdeelde data. Ook in dit geval wordt weer gebruik gemaakt van het secure scalair product protocol of het secure size of set intersection protocol.

6.4.1 Het private minen van decision trees in verticaal verdeelde data: twee partijen

Een eerste oplossing wordt voorgesteld in [45]. Het algoritme is gebaseerd op het ID3 algoritme en behoudt de privacy van beide partijen door het secure scalair product protocol te gebruiken. Dit algoritme wordt beschreven als een algoritme voor twee partijen.
Zoals reeds vermeld zijn er verscheidene secure scalair product protocollen die wel bruikbaar zijn in het geval van meerdere partijen. Er kan hier dus net dezelfde opmerking gemaakt worden als bij het minen van associatie regels in verticaal verdeelde data. Het algoritme kan gebruikt worden in het geval van meerdere partijen indien de secure scalair product protocollen gebruikt worden die wel meer dan twee partijen aankunnen.

6.4.1.1 Veronderstellingen in verband met de attributen

Bij verticaal verdeelde data is het belangrijk om te weten wat er mag geleerd worden over de attributen. Bij dit algoritme wordt aangenomen dat beide partijen alle attributen kennen.
Verder wordt er ook verondersteld dat beide partijen het class attribuut bezitten. Het resultaat van het algoritme mag ook door beide partijen gekend zijn. Dit wil zeggen dat beide partijen de volledige decision tree mogen leren.

6.4.1.2 Het privacy preserving ID3 algoritme

Het ID3 algoritme bestaat uit drie verschillende stappen. Deze delen kunnen beveiligd worden met het secure scalair product protocol.

R is leeg Beide partijen kennen de attributen dus ze weten ook of R leeg is of niet. Als dit het geval is moet er berekend worden welke class waarde het meeste voorkomt in de transacties in T. De partijen kunnen dit berekenen met behulp van het secure scalair product protocol.
Elke partij stelt een vector op met daarin een element per transactie. Wanneer de transactie nog beschouwd wordt in de huidige knoop dan krijgt het element in de vector de waarde 1. Indien dit niet het geval is krijgt de transactie de waarde 0.
Zo kan elke partij per class waarde een vector opstellen en met deze vectoren het secure scalair product protocol uitvoeren. Het opstellen van de vectoren is mogelijk omdat beide partijen over het class attribuut beschikken. De class waarde met het hoogste scalair product is dan de waarde die het meeste voorkomt.

Alle transacties in T hebben dezelfde class waarde Om te bepalen of alle transacties in T dezelfde class waarde hebben moeten beide partijen net zoals in de vorige stap per class waarde een vector opstellen en het secure scalair product protocol uitvoeren. Indien er maar één scalair product niet gelijk is aan 0, is dat de enige class waarde.

Anders In het andere geval moet het attribuut bepaald worden dat de transacties in T het beste classificeert. Hiervoor moet de information gain berekend worden voor elk attribuut. Om dit te berekenen moeten er verschillende transacties geteld worden:

- Het aantal transacties die tot in de huidige knoop geraken.
- Het aantal transacties die tot in de huidige knoop geraken voor elke mogelijke waarde van het class attribuut.
- Het aantal transacties die tot in de huidige knoop geraken voor elke mogelijke waarde van het attribuut waarvoor de gain berekend wordt.
- Het aantal transacties die tot in de huidige knoop geraken voor elke mogelijke waarde van het attribuut waarvoor de gain berekend wordt en voor elke mogelijke waarde van het class attribuut.

Dit komt neer op het opstellen van een aantal vectoren en het uitvoeren van secure scalair product protocollen. De volledige werking van het algoritme kan gevonden worden in Algoritme 22.

6.4.1.3 Evaluatie van een nieuwe transactie

Aangezien beide partijen de volledige boom leren, is het niet nodig om samen te werken als er een nieuwe transactie moet geclassificeerd worden. De partijen kunnen dit individueel doen.

6.4.1.4 Voorbeeld

Beschouw de data in Tabel 1.4 uit Hoofdstuk 1. Veronderstel dat de attributen Outlook en Temperature zich bij partij 0 bevinden en de attributen Humidity en Wind bij partij 1. Elke partij heeft ook het class attribuut Play Tennis.
De partijen moeten beginnen met de wortel van de boom te bepalen. Ze kunnen elk de information gain bepalen van de attributen die ze in hun bezit hebben:

Algoritme 22 Het privacy preserving ID3 algoritme

Require: R, de set van attributen.
Require: C, het class attribuut.
Require: T, de set van transacties.

if De partijen testen of R leeg is **then**
    Met behulp van het secure scalair product protocol wordt berekend welke class waarde c_i het meeste voorkomt.
    Return een blad met als class waarde c_i.
else if De partijen gebruiken het secure scalair product protocol om te testen of alle transacties in T dezelfde class waarde c_i hebben **then**
    Return een blad met als class waarde c_i.
else
    Bepaal het attribuut A dat de transacties in T het beste classificeert: Met behulp van het secure scalair product protocol wordt het beste attribuut bepaald.
    Verdeel T in m delen $T(a_1), ..., T(a_m)$ zodat $a_1, ..., a_m$ de verschillende waardes van attribuut A zijn.
    Return een boom met als wortel een knoop A en m takken $a_1, ..., a_m$ zodat tak i $ID3(R - \{A\}, C, T(a_i))$ bevat.
end if

- Partij 0:
  - Gain(Outlook) = 0.246
  - Gain(Temperature) = 0.029
- Partij 1:
  - Gain(Humidity) = 0.151
  - Gain(Wind) = 0.048

De partijen wisselen deze waardes uit en bepalen de wortel van de boom. In dit geval zal dat Outlook zijn. De knoop Outlook krijgt dan drie takken. Eén tak voor elke waarde die het attribuut Outlook kan aannemen. Om de eerste tak verder uit te werken moeten alleen die transacties beschouwd worden die als Outlook Sunny hebben. Voor de attributen Temperature, Humidity en Wind moet nu de information gain berekend worden. Voor Temperature kan partij 0 dit alleen doen. Voor Humidity en Wind moeten de partijen samenwerken. Zo zal de information gain van Humidity er als volgt uitzien:

$$Entropy(T(Sunny)) - \frac{|(T(Sunny,High))|}{|T(Sunny)|}.Entropy(T(Sunny,High)) \\ -\frac{|(T(Sunny,Normal))|}{|T(Sunny)|}.Entropy(T(Sunny,Normal))$$

Hierin kunnen twee delen gevonden worden. Voor elke waarde van het attribuut Humidity moeten twee dingen berekend worden, namelijk $\frac{|(T(Sunny,High))|}{|T(Sunny)|}$ en $Entropy(T(Sunny,High))$. $|(T(Sunny,High))|$ kan berekend worden met behulp van een secure scalair product protocol. In dit voorbeeld zal partij 0 de vector (1,1,0,0,0,0,0,1,1,0,1,0,0,0) opstellen. Dit zijn de transacties die als Outlook Sunny hebben. Partij 1 construeert de vector (1,1,1,1,0,0,0,1,0,0, 0,1,0,1), de transacties met als Humidity High. $|T(Sunny)|$ kan door partij 0 individueel berekend worden. De waarde van $Entropy(T(Sunny,High))$ is gelijk aan:

$$-\frac{|(T(Sunny,High,No))|}{T(Sunny,High)}log\frac{(|T(Sunny,High,No))|}{T(Sunny,High)} \\ -\frac{|T((Sunny,Normal,Yes))|}{T(Sunny,Normal)}log\frac{|(T(Sunny,Normal,Yes))|}{T(Sunny,Normal)}$$

Al deze delen kunnen ook weer met behulp van een secure scalair product berekend worden. Na het berekenen van de information gain van Temperature, Humidity en Wind wordt het beste attribuut gekozen. De rest van de decision tree kan op een analoge manier bekomen worden.

Opmerking Net zoals bij associatie regels zorgt de verticale verdeling van de data er weer voor dat de partijen iets kunnen leren. Het uitvoeren van een aantal scalair product protocollen kan iets vrijgeven over de data. De kans dat dit in de realiteit gebeurd, is zeer klein omdat de transactie databases zowel veel attributen als veel transacties bevatten. Verder is het gevaar van probing attacks ook hier nog steeds aanwezig.

6.4.2 Het private minen van decision trees in verticaal verdeelde data: twee of meer partijen

In [49] wordt een tweede algoritme voorgesteld. Dit algoritme is eveneens gebaseerd op het ID3 algoritme en maakt gebruik van het secure sum protocol en het secure size of set intersection protocol. Het algoritme kan gebruikt worden voor twee of meer partijen.

6.4.2.1 Veronderstellingen in verband met de attributen

In tegenstelling tot de vrij zwakke veronderstellingen in het vorige algoritme, weten de partijen hier zo weinig mogelijk. Er wordt vanuit gegaan dat de partijen alleen hun eigen attributen kennen. Ze weten dus niets over de attributen van de andere partijen.
Verder wordt er ook verondersteld dat maar één partij het class attribuut kent. De decision tree die de partijen leren bevat ook alleen maar hun eigen attributen. De andere knopen bevatten alleen maar informatie over welke partij het attribuut bezit dat zich op die knoop bevindt.

6.4.2.2 Het privacy preserving ID3 algoritme

Het ID3 algoritme bestaat uit drie verschillende stappen. Deze onderdelen kunnen beveiligd worden door het gebruik van het secure sum protocol en het secure size of set intersection protocol. De grootte van de verzamelingen die de partijen doorsturen bij het secure size of set intersection protocol moet in dit geval gelijk zijn aan $|T|$.

R is leeg Om te bepalen of R leeg is, wordt een variant van het secure sum protocol gebruikt. Elke partij geeft een waarde als input. Wanneer een partij geen attributen meer overhoud is die waarde een 0, in het andere geval een 1. Partij 0 telt een random waarde op bij zijn 0 of 1.
Wanneer de laatste partij zijn waarde bij de som heeft opgeteld genereren de eerste en de laatste partij een one-way commutatieve, deterministische encryptie key. Partij 0 encrypteert de random waarde en stuurt die naar de laatste partij. De laatste partij encrypteert de som en stuurt die naar partij 0. De laatste partij ontvangt de geëncrypteerde random waarde van partij 0 en encrypteert die waarde met zijn key. Dit stuurt hij terug naar partij 0.
Partij 0 kan de geëncrypteerde som verder encrypteren met zijn key en dan heeft hij twee waardes in zijn bezit die met dezelfde twee encryptie keys zijn geëncrypteerd. Wanneer deze twee waardes gelijk zijn, hebben alle partijen een 0 als input gegeven aan het protocol en blijven er dus geen attributen meer over. Indien de waardes niet gelijk zijn, heeft minstens één partij een 1 als input gegeven en is R niet leeg. Indien R leeg is, moet de meest voorkomende class waarde gezocht worden. Elke partij houdt hiervoor twee sets bij, namelijk een *constraint set* en een *transaction set*. De constraint set van een partij bevat voor elk attribuut van die partij een waarde of een '?'. Op deze manier wordt bijgehouden aan welke eigenschappen de transacties die beschouwd worden in een bepaalde knoop moeten voldoen. Merk op dat in het vorige algoritme ook al gebruik gemaakt gemaakt werd van dergelijke sets

om de vectoren op te stellen. Het werd er alleen niet expliciet zo genoemd. De transaction set bevat dan alle transacties die voldoen aan de eigenschappen voorgeschreven door de constraint set. Met behulp van het secure size of set intersection protocol kan er per class waarde bepaald worden hoeveel transacties die waarde hebben. De class waarde die het meeste voorkomt wordt dan gekozen.

Alle transacties in T hebben dezelfde class waarde Het bepalen of alle transacties in T dezelfde class waarde hebben verloopt analoog aan het bepalen welke class waarde het meeste voorkomt. Per class waarde wordt een secure size of set intersection protocol uitgevoerd. Alle transacties in T hebben dezelfde class waarde als alle resultaten 0 zijn, behalve één. Het resultaat dat niet 0 is, geeft dan de class waarde.

Anders In het andere geval moet het attribuut bepaald worden dat de transacties in T het beste classificeert. Elke partij zal de information gain berekenen voor al zijn attributen. Het tellen van transacties kan gedaan worden met het secure size of set intersection protocol. De partijen kiezen dan uit hun eigen attributen de hoogste information gain en bepalen het maximum van deze waardes. Het attribuut met de hoogste gain wordt gekozen. De volledige werking van het algoritme kan gevonden worden in Algoritme 23.

Merk op dat de partijen bij dit algoritme niet de volledige boom leren. De waardes in de knopen worden vervangen door id's. Hierdoor leren de partijen alleen welke partij verantwoordelijk is voor welke knoop.
Dit algoritme is onmiddellijk een stuk veiliger dan het vorige. Dankzij de strengere privacy eisen is de kans dat er toch iets geleerd wordt veel kleiner. Ook het uitvoeren van een probing attack heeft hier geen zin meer. De partijen weten bij het uitvoeren van het secure size of set intersection protocol nooit over welke attributen de andere partijen het hebben.

6.4.2.3 Evaluatie van een nieuwe transactie

Wanneer de decision tree moet gebruikt worden om een nieuwe transactie te classificeren moeten de partijen samenwerken. Dit is zo omdat de partijen niet de volledige boom leren. Ze weten alleen welke partij verantwoordelijk is voor welke knoop. Het classificeren van een nieuwe transactie gebeurt dan ook stap per stap.
De eerste stap is het kiezen van de juiste tak in de wortel. Er is maar één partij die dit kan doen en dat is de partij met verantwoordelijkheid over die knoop. Die partij beslist welke tak er moet gevolgd worden en geeft de

Algoritme 23 Het privacy preserving ID3 algoritme

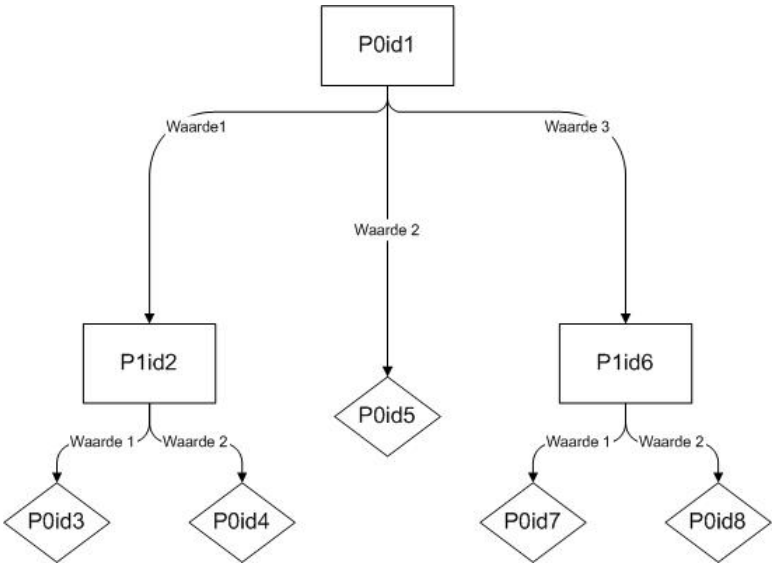
Require: Partijen $0, ..., k-1$, de k partijen waarbij P_{k-1} het class attribuut bezit.
Require: R, de set van attributen.
Require: C, het class attribuut.
Require: T, de set van transacties.

if De partijen testen of R leeg is met behulp van een variant op het secure sum protocol **then**
    Partij P_{k-1} berekent met behulp van het secure size of set intersection protocol welke class waarde c_i het meeste voorkomt.
    Partij P_{k-1} construeert een blad met als class waarde c_i en als id i.
    Return id i.
else if De partijen gebruiken het secure size of set intersection protocol om te testen of alle transacties in T dezelfde class waarde c_i hebben **then**
    Partij P_{k-1} construeert een blad met als class waarde c_i en als id i.
    Return id i.
else
    Bepaal het attribuut A dat de transacties in T het beste classificeert: Met behulp van het secure size of set intersection protocol wordt het beste attribuut bepaald.
    Partij P_i, die het beste attribuut bezit, construeert een knoop met als waarde A en als id i.
    Partij P_i verdeelt T in m delen $T(a_1), ..., T(a_m)$ zodat $a_1, ..., a_m$ de verschillende waardes van attribuut A zijn.
    Partij P_i construeert voor elke a_i een tak met $ID3(R - \{A\}, C, T(a_i))$.
    Return id i.
end if

controle aan de volgende knoop. Dit gaat zo door tot een blad van de decision tree bereikt wordt. De partij met het class attribuut kan dan de classificatie leren en eventueel bekend maken.

6.4.2.4 Voorbeeld

Beschouw opnieuw de data in Tabel 1.4 uit Hoofdstuk 1. Veronderstel dat de attributen Outlook en Temperature zich bij partij 0 bevinden en de attributen Humidity en Wind bij partij 1. Het class attribuut Play Tennis bevindt zich bij partij 0.
Het algoritme heeft als resultaat de decision tree in Figuur 6.4. Hierin staat alleen vermeld welke partij verantwoordelijk is voor welke knoop. Veronderstel dat een transactie (Outlook = Sunny, Temperature = Cool, Humidity = High, Wind = Strong) moet geclassificeerd worden. De wortel P0id1 bevindt zich bij partij 0. Partij 0 behandelt het Outlook attribuut en gaat zo verder naar P1id2. Partij 1 behandelt het Humidity attribuut en dit leidt naar het blad P0id3. Partij 0 bezit het class attribuut en kan het resultaat van de classificatie leren, namelijk No.



Figuur 6.4: De privacy preserving decision tree.

6.5 Naïve Bayes in horizontaal verdeelde data: twee of meer partijen

In [46] wordt een algoritme voorgesteld om op een veilige manier gebruik te maken van de Naïve Bayes classifier, en dit in het geval van twee of meer partijen. Het algoritme maakt gebruik van het secure sum protocol.

6.5.1 Het privacy preserving Naïve Bayes algoritme

Aangezien de werking van het Naïve Bayes algoritme afhankelijk is van de aard van de attributen, is ook de privacy preserving methode verschillend voor enerzijds nominale attributen en anderzijds numerieke attributen.

6.5.1.1 Nominale attributen

Naïve Bayes maakt gebruik van de kans op een bepaalde waarde bij een attribuut samen met een bepaalde class waarde. Aangezien de horizontale verdeling heeft elke partij een deel van de nodige waardes. Er zal voor elk attribuut en voor elke class waarde een kans moeten berekend worden. Dit gebeurt door eerst te bepalen hoeveel transacties de partijen hebben met een bepaalde waarde bij een attribuut en een bepaalde class waarde. Elke partij kan dit voor zijn eigen transacties bepalen. Van al deze waardes kan dan de som berekend worden met behulp van het secure sum protocol.
Vervolgens moet er nagegaan worden hoeveel transacties in totaal die bepaalde class waarde hebben. Dit kan op dezelfde manier gebeuren. Elke partij controleert zijn eigen transacties en dan wordt met het secure sum protocol het totaal berekend. De twee berekende waardes moeten dan gedeeld worden om de kans te leren. De volledige beschrijving van dit algoritme kan gevonden worden in Algoritme 24.

6.5.1.2 Numerieke attributen

In het geval van numerieke attributen wordt het gemiddelde en de variantie gebruikt om een nieuwe transactie te classificeren. Voor elk attribuut moet er voor elke mogelijke class waarde een gemiddelde en een variantie geleerd worden.
Om het gemiddelde te berekenen kunnen de partijen hun eigen transacties onderzoeken. Ze kunnen voor een bepaald attribuut en een bepaalde class waarde de som maken van de attribuut waardes en bijhouden hoeveel transacties er zijn met die bepaalde class waarde. Van deze twee waardes moet dan telkens de som gemaakt worden. Alle partijen werken hiervoor mee aan

Algoritme 24 Het privacy preserving Naïve Bayes algoritme

Require: Partijen $0, ..., k-1$.
Require: n transacties en c waardes voor het class attribuut.

```
1: for Elke class waarde c_i do
2:   for Elk attribuut do
3:     for Elke attribuut waarde a_j do
4:       for l = 0...k-1 do
5:         Partij l berekent het aantal transacties n_j^l met waarde a_j bij het
           attribuut en c_i als class waarde.
6:         Partij l berekent het aantal transacties n^l met c_i als class waar-
           de.
7:       end for
8:       De partijen bepalen n_j = sum_{i=1}^{k} n_j^i met behulp van het secure sum
         protocol.
9:       De partijen bepalen n = sum_{i=1}^{k} n^i met behulp van het secure sum
         protocol.
10:      De kans is gelijk aan n_j/n.
11:    end for
12:  end for
13: end for
```

een secure sum protocol. De twee bekomen waardes moeten dan gedeeld worden om het gemiddelde te leren dat bij dat attribuut en die bepaalde class waarde hoort.
Het berekenen van de variantie is eveneens niet moeilijk. Alle partijen kunnen het verschil berekenen tussen hun waardes bij een bepaald attribuut en het bijhorende gemiddelde. Dit is mogelijk omdat het hier om een horizontale verdeling gaat en de partijen dus weten welke classificatie bij hun transacties hoort. De berekende verschillen moeten dan gekwadrateerd worden. De bekomen waardes die bij dezelfde class waarde horen, worden opgeteld. Dit moet dan nog eens gedeeld worden door het aantal transacties met die bepaalde class waarde. Op deze manier leren alle partijen de variantie per attribuut en bijhorende class waarde. De volledige beschrijving van dit algoritme kan gevonden worden in Algoritme 25 en 26.

6.5.2 Moeilijkheden in het geval van twee partijen

Een belangrijke opmerking is het feit dat dit algoritme vaak gebruik maakt van het secure sum protocol. Dit protocol heeft, zoals reeds vermeld, veiligheidsproblemen in het geval van twee partijen. In dit algoritme wordt telkens

Algoritme 25 Het privacy preserving Naïve Bayes algoritme

Require: Partijen $0, ..., k-1$.
Require: n transacties en c waardes voor het class attribuut.

for Elk attribuut **do**
  for Elke class waarde c_i **do**
    for $l = 0...k-1$ **do**
      Partij l berekent s^l de som van de waardes bij dat attribuut in zijn transacties.
      Partij l berekent het aantal transacties n^l met c_i als class waarde.
    end for
    De partijen bepalen $s = \sum_{i=1}^{k} s^i$ met behulp van het secure sum protocol.
    De partijen bepalen $n = \sum_{i=1}^{k} n^i$ met behulp van het secure sum protocol.
    Het gemiddelde is dan gelijk aan s/n.
  end for
end for

Algoritme 26 Het privacy preserving Naïve Bayes algoritme

Require: Partijen $0, ..., k-1$.
Require: n transacties en c waardes voor het class attribuut.

for Elk attribuut **do**
  for Elke class waarde c_i **do**
    for $l = 0...k-1$ **do**
      Partij l berekent het verschil tussen de attribuut waardes van de transacties die c_i als class waarde hebben en het bijhorende gemiddelde.
      Partij l berekent s^l, de gekwadrateerde som van de verschillen.
    end for
    De partijen bepalen $s = \sum_{i=1}^{k} s^i$ met behulp van het secure sum protocol.
    De variantie is gelijk aan s/n.
  end for
end for

een breuk berekend waarvan de teller en de noemer uit een som bestaan. Er zijn twee mogelijke beveiligingen voor deze aanpak. Een eerste protocol dat gebruikt kan worden is het secure division computation protocol. Dit protocol berekent een breuk waarvan zowel teller als noemer verdeeld zijn over twee partijen. Een tweede oplossing is het gebruik van het secure $ln(x)$ protocol. Merk op dat x/y ook kan geschreven worden als $exp(ln(x/y))$. Dit is dan weer gelijk aan $exp(ln(x) - ln(y))$. Op deze manier kan de breuk op een veilige manier berekend worden met behulp van het secure $ln(x)$ protocol.
Het is eveneens mogelijk om deze protocollen te gebruiken in het geval van meerdere partijen. Hiervoor wordt het secure sum protocol gebruikt. Maar in plaats van het protocol volledig te volgen, wordt er gestopt voor de laatste stap. Partij 0 heeft op dat ogenblik een random getal in zijn bezit. De laatste partij heeft de volledige som, nog steeds vermeerderd met het random getal in zijn bezit. Wanneer de laatste partij nu deze som als input geeft aan de protocollen, en partij 0 geeft het random getal vermenigvuldigt met -1 als input, bekomen de partijen eveneens het gewenste resultaat.

6.5.3 Evaluatie van een nieuwe transactie

Aangezien beide partijen alle waardes leren, is het niet nodig om samen te werken als er een nieuwe transactie moet geclassificeerd worden. De partijen kunnen dit individueel doen.

6.5.4 Voorbeeld

Beschouw de data in Tabel 1.4 uit Hoofdstuk 1. Veronderstel dat er drie partijen zijn. De transacties 1 tot 5 bevinden zich bij partij 0, de transacties 6 tot 10 bij partij 1 en de transacties 11 tot 14 bij partij 2.
De partijen zullen voor alle attributen de nodige kansen berekenen. Voor het attribuut Outlook zal dit als volgt verlopen:

- Partij 0:
  - Aantal transacties met Outlook Sunny en Play Tennis Yes = 0.
  - Aantal transacties met Outlook Sunny en Play Tennis No = 2.
  - Aantal transacties met Outlook Overcast en Play Tennis Yes = 1.
  - Aantal transacties met Outlook Overcast en Play Tennis No = 0.
  - Aantal transacties met Outlook Rain en Play Tennis Yes = 2.
  - Aantal transacties met Outlook Rain en Play Tennis No = 0.

  - Aantal transacties met Play Tennis Yes = 3.
  - Aantal transacties met Play Tennis No = 2.

- Partij 1:
  - Aantal transacties met Outlook Sunny en Play Tennis Yes = 1.
  - Aantal transacties met Outlook Sunny en Play Tennis No = 1.
  - Aantal transacties met Outlook Overcast en Play Tennis Yes = 1.
  - Aantal transacties met Outlook Overcast en Play Tennis No = 0.
  - Aantal transacties met Outlook Rain en Play Tennis Yes = 1.
  - Aantal transacties met Outlook Rain en Play Tennis No = 1.

  - Aantal transacties met Play Tennis Yes = 3.
  - Aantal transacties met Play Tennis No = 2.

- Partij 2:
  - Aantal transacties met Outlook Sunny en Play Tennis Yes = 1.
  - Aantal transacties met Outlook Sunny en Play Tennis No = 0.
  - Aantal transacties met Outlook Overcast en Play Tennis Yes = 2.
  - Aantal transacties met Outlook Overcast en Play Tennis No = 0.
  - Aantal transacties met Outlook Rain en Play Tennis Yes = 0.
  - Aantal transacties met Outlook Rain en Play Tennis No = 1.

  - Aantal transacties met Play Tennis Yes = 3.
  - Aantal transacties met Play Tennis No = 1.

Door de nodige sommen te maken kunnen dan de volgende kansen berekend worden:

- P(Sunny|Yes) = (0+1+1)/(3+3+3) = 2/9.
- P(Sunny|No) = (2+1+0)/(2+2+1) = 3/5.
- P(Overcast|Yes) = (1+1+2)/(3+3+3) = 4/9.
- P(Overcast|No) = (0+0+0)/(2+2+1) = 0/5.
- P(Rain|Yes) = (2+1+0)/(3+3+3) = 3/9.

- P(Rain|No) = (0+1+1)/(2+2+1) = 2/5.

Voor de andere attributen kunnen deze kansen op dezelfde manier berekend worden. Met behulp van deze waardes kan dan een nieuwe transactie geclassificeerd worden.

6.6 Naïve Bayes in verticaal verdeelde data: twee of meer partijen

Een privacy preserving versie van de Naïve Bayes methode wordt voorgesteld in [47]. Dit algoritme kan gebruikt worden voor twee of meer partijen. Aangezien er gebruik gemaakt wordt van het $ln(x)$ protocol is het resultaat een approximatie van de Naïve Bayes classifier.

6.6.1 Veronderstellingen in verband met de attributen

Er wordt hier verondersteld dat de partijen alleen hun eigen attributen kennen. Verder wordt er ook vanuit gegaan dat het class attribuut maar door één partij gekend is.

6.6.2 Het privacy preserving Naïve Bayes algoritme

Net zoals bij de horizontale verdeling is ook hier weer een onderscheid te maken tussen enerzijds nominale attributen en anderzijds numerieke attributen.

6.6.2.1 Nominale attributen

Het algoritme laat de partijen verschillende matrices berekenen. Elke partij krijgt per attribuut een matrix. De partij met het class attribuut heeft voor elke van deze matrices een overeenkomstige matrix. De som van twee van die matrices geeft dan een overzicht van de kansen in verband met alle waardes die dat attribuut kan aannemen in combinatie met alle mogelijke class waardes. De kansen worden verdeeld over telkens twee partijen. Dit wordt uiteraard gedaan om de veiligheid te bewaren.
Om de matrices te berekenen stellen beide partijen een vector op. De partij met het class attribuut stelt voor elke waarde c_i die het class attribuut kan hebben een vector $\vec{X}$ op. Deze vector heeft dezelfde grootte als de transactie database. Dit wil dus zeggen dat de vector evenveel elementen bevat als er transacties zijn. Indien transactie i de gezochte class waarde c_i heeft dan krijgt de vector op positie i de waarde $\frac{1}{aantal\ transacties\ met\ c_i\ als\ class\ waarde}$. In

het andere geval is positie i gelijk aan 0. Op dezelfde manier stelt de partij met het andere attribuut een vector $\vec{Y}$ op. De partij stelt voor elk attribuut een reeks van vectoren op. Dit gebeurt zo dat elke mogelijke attribuut waarde een vector krijgt. Zo een vector heeft weer evenveel posities als er transacties zijn. Positie i krijgt de waarde 1 indien transactie i die bepaalde attribuut waarde heeft, en waarde 0 als dit niet zo is. Elke partij heeft dan voor elk attribuut enkele vectoren. Van al deze vectoren wordt het scalair product berekend met de vector van het class attribuut. Aangezien dankzij de verticale verdeling een attribuut zich altijd volledig bij één partij bevindt, zal een secure scalair product protocol voor twee partijen voldoende zijn.
Op deze manier kan er voor elk attribuut een matrix worden opgesteld. Deze matrix heeft dan evenveel kolommen als er waardes zijn voor het class attribuut en evenveel rijen als dat attribuut mogelijke waardes heeft. Om de veiligheid van de gegevens te bewaren worden de scalair producten niet aan één partij meegedeeld maar opgesplitst over de partij met het class attribuut en de partij met het andere attribuut. De som van deze matrices drukken dan de kansen uit die nodig zijn om de Naïve Bayes classifier te gebruiken. De volledige beschrijving van dit algoritme kan gevonden worden in Algoritme 27.

6.6.2.2 Numerieke attributen

Om Naïve Bayes toe te passen op numerieke attributen moet het gemiddelde en de variantie gekend zijn. Om de gegevens te beschermen zal het algoritme de partijen elk een share van het resultaat geven.
Voor elk numeriek attribuut wordt het gemiddelde berekend voor elke class waarde. Dit gebeurt op een analoge manier als het opstellen van de matrices bij de nominale attributen. De partij met het class attribuut stelt zijn vector $\vec{X}$ op dezelfde manier op. De partij met het andere attribuut stopt zijn data in een vector $\vec{Y}$. Deze vectoren worden als input gegeven aan het secure scalair product protocol zodat beide partijen een share van het gemiddelde leren. Net zoals bij nominale attributen zal dit protocol nooit voor meer dan twee partijen gebruikt worden. Voor elk attribuut leren de partijen een vector $\vec{M}$ met evenveel elementen als er class waardes zijn. Elke vector bevat dan shares van de gemiddeldes voor de verschillende class waardes.
Het berekenen van de variantie gebeurt met behulp van probabilistische, homomorfe encryptie. De partij met het attribuut genereert een encryptie key E en een decryptie key D. Zowel de data vector $\vec{Y}$ als de vector met de shares van de gemiddeldes $\vec{M}$ worden geëncrypteert met behulp van de key E. Deze geëncrypteerde waardes worden dan samen met de encryptie key E naar de partij met het class attribuut gestuurd.

Algoritme 27 Het privacy preserving Naïve Bayes algoritme

Require: Partijen $0, ..., k-1$.
Require: P_c, de partij met het class attribuut.
Require: P_d, de partij met het gewone attribuut.
Require: n transacties en c waardes voor het class attribuut.

for Elke class waarde c_i **do**
  P_c genereert een vector $\vec{X}$.
  for Elke transactie $1...n$ **do**
    if De transactie heeft c_i als class waarde **then**
      $x_i = \frac{1}{aantal\ transacties\ met\ c_i\ als\ class\ waarde}$.
    else
      $x_i = 0$.
    end if
  end for
  for Elk attribuut **do**
    for Elke attribuut waarde a_j **do**
      P_d genereert een vector $\vec{Y}$.
      for Elke transactie $1...n$ **do**
        if De transactie heeft a_j als waarde bij dat attribuut **then**
          $y_i = 1$.
        else
          $y_i = 0$.
        end if
      end for
      Bereken shares van het scalair product van de vectoren $\vec{X}$ en $\vec{Y}$ met behulp van het secure scalair product protocol.
    end for
  end for
  Met de shares kan elke partij een matrix vormen. Deze matrix heeft evenveel rijen als er waardes zijn voor het attribuut en evenveel kolommen als er waardes zijn voor het class attribuut.
end for
De partijen hebben dan per attribuut een matrix waarin shares van de kansen te vinden zijn voor alle mogelijke attribuut waardes en alle mogelijke class waardes.

Deze partij zal nu de variantie berekenen. Aangezien de partij het class attribuut heeft kan dit zonder problemen. Om te beginnen worden de waardes in de data vector verminderd met het gemiddelde van de class waarde en met een random waarde. Het gemiddelde bestaat uit twee delen. Het eerste deel komt van de partij met het andere attribuut, het tweede deel heeft de partij met het class attribuut in zijn bezit. Door de homomorfe encryptie te gebruiken kan deze berekening gemaakt worden zonder de waardes te decrypteren. Het resultaat van deze berekeningen komt in een vector $\vec{V}$ terecht. Deze vector wordt teruggestuurd naar de partij met het gewone attribuut. Deze partij kan de vector decrypteren met behulp van de decryptie key D.
Beide partijen hebben op deze manier een share van het verschil tussen de waardes en het gemiddelde geleerd. Van deze som moet de tweede macht gekend zijn. Dit is nodig om de variantie te berekenen. Dit wordt gedaan door gebruik te maken van het secure square computation protocol. Hierdoor kunnen de partijen de vectoren $\vec{T'}$ en $\vec{T''}$ opstellen.
In een volgende stap worden de vectoren $\vec{X}$ en $\vec{T''}$ als input gegeven aan het secure scalair product protocol. Dit moet gebeuren voor elke class waarde. De partij met het class attribuut leert hierdoor zijn share van de variantie σ_i''. De andere partij leert een waarde *temp*. Met deze waarde kan de partij zijn share berekenen. Dit kan door het scalair product van de vectoren $\vec{X}$ en $\vec{T'}$ te vermeerderen met de waarde *temp*. Aangezien $\sigma_i' + \sigma_i'' = \vec{T'}.\vec{X} + temp + \vec{T''}.\vec{X} - temp = \vec{T'}.\vec{X} + \vec{T''}.\vec{X}$ hebben beide partijen een deel geleerd van de variantie. Dit wordt berekend voor elk attribuut en voor elke mogelijke class waarde. De volledige beschrijving van dit algoritme kan gevonden worden in Algoritme 28 en Algoritme 29.

6.6.3 Evaluatie van een nieuwe transactie

Aangezien de resultaten van de berekeningen verdeeld zijn over verschillende partijen moet er samengewerkt worden om een nieuwe transactie te classificeren. De Naïve Bayes formule kan als volgt herschreven worden:

$$\begin{aligned} C_{MAP} &= argmax_{c_i} P(c_i) \prod_j P(a_j|c_i) \\ &= argmax_{c_i} (ln(P(c_i) \prod_j P(a_j|c_i))^2) \\ &= argmax_{c_i} (2.ln(P(c_i)) \sum_j ln(P(a_j|c_i))^2) \end{aligned}$$

Nominale attributen Voor nominale attributen moet een aantal keren $ln(x)$ berekend worden. De waarde x kan bij één partij zitten, namelijk de partij die het class attribuut bezit, of verspreid over twee partijen. Dit kan op een veilige manier gebeuren met behulp van het secure $ln(x)$ protocol.

Algoritme 28 Het privacy preserving Naïve Bayes algoritme

Require: Partijen $0, ..., k-1$.
Require: P_c, de partij met het class attribuut.
Require: P_d, de partij met het gewone attribuut.
Require: n transacties en c waardes voor het class attribuut.

1: Berekenen van het gemiddelde.
2: **for** Elke class waarde c_i **do**
3: P_c genereert een vector $\vec{X}$.
4: **for** Elke transactie $1...n$ **do**
5: **if** De transactie heeft c_i als class waarde **then**
6: $x_i = \frac{1}{aantal\ transacties\ met\ c_i\ als\ class\ waarde}$.
7: **else**
8: $x_i = 0$.
9: **end if**
10: **end for**
11: **for** Elk attribuut **do**
12: P_d genereert een data vector $\vec{Y}$ die de waardes van het attribuut bevat.
13: Bereken shares van het scalair product van de vectoren $\vec{X}$ en $\vec{Y}$ met behulp van het secure scalair product protocol.
14: **end for**
15: Met de shares kan elke partij een vector $\vec{M}$ vormen. Deze vector heeft evenveel elementen als er waardes zijn voor het attribuut.
16: **end for**
17: De partijen hebben dan per attribuut een vector waarin shares van het gemiddelde te vinden zijn voor alle mogelijke class waardes.

Algoritme 29 Het privacy preserving Naïve Bayes algoritme

Require: Partijen $0, ..., k-1$.
Require: P_c, de partij met het class attribuut.
Require: P_d, de partij met het gewone attribuut.
Require: n transacties en k waardes voor het class attribuut.

1: Berekenen van de variantie.
2: **for** Elk attribuut **do**
3: P_d genereert een probabilistische, homomorfe encryptie key E en een decryptie key D.
4: P_d encrypteert de data vector $\vec{Y}$ met E: $E(\vec{Y})$.
5: P_d encrypteert de gemiddeldes vector $\vec{M'}$ met E: $E(\vec{M'})$.
6: P_d stuurt de geëncrypteerde vectoren samen met de encryptie key E naar P_c.
7: P_c encrypteert zijn gemiddeldes vector $\vec{M''}$ met E: $E(\vec{M''})$.
8: P_c construeert een vector $\vec{V} = (v_i, ..., v_n)$.
9: **for** $j = 1...n$ **do**
10: P_c genereert een random getal r_j.
11: P_c berekent $v_j = E(y_j)/(E(m'_j).E(m''_j).E(r_j))$
12: **end for**
13: P_c stuurt $\vec{V}$ naar P_d.
14: P_d decrypteert $\vec{V}$ met de decryptie key D en verkrijgt zo $\vec{W}$ zodat $w_j = (y_j - m_j - r_j)$.
15: **for** $j = 1...n$ **do**
16: Bereken shares t'_j en t''_j van $(r_j + w_j)^2$ met behulp van het secure square computation protocol.
17: **end for**
18: **for** Elke class waarde c_i **do**
19: Bereken shares $temp$ en σ''_i van $\vec{T''}.\vec{X}$ met behulp van het secure scalair product protocol. $\vec{X}$ is de vector die P_c heeft opgesteld voor het berekenen van het gemiddelde en $\vec{T''}$ is de vector die P_d geleerd heeft. P_c verkrijgt $temp$ en P_d verkrijgt σ''_i.
20: P_c berekent $\sigma'_i = \vec{T'}.\vec{X} + temp$.
21: **end for**
22: Met de shares kan elke partij een vector vormen. Deze vector heeft evenveel elementen als er waardes zijn voor het attribuut.
23: **end for**
24: De partijen hebben dan per attribuut een vector waarin shares van de variantie te vinden zijn voor alle mogelijke class waardes.

Numerieke attributen In het geval van numerieke attributen wordt er gebruik gemaakt van het gemiddelde en de variantie. Dit gebeurt op de volgende manier:

$$\begin{aligned} ln(P(a_j|c_i))^2 &= \frac{1}{\sqrt{2\pi\sigma}} e^{\frac{-(x-\mu)^2}{2\sigma^2}} \\ &= -ln(2\pi\sigma) - \frac{-(x-\mu)^2}{2\sigma^2} \\ &= -ln(2\pi) - ln(\sigma) - \frac{-(x-\mu)^2}{2\sigma^2} \end{aligned}$$

Shares van σ^2 zijn gekend bij de partijen. Shares van $ln(\sigma^2)$ kunnen dan berekend worden met het secure $ln(x)$ protocol. Met behulp van het secure square computation protocol is het mogelijk om shares van $(x-\mu)^2$ te berekenen. Het secure division protocol, als laatste, kan gebruikt worden om shares te berekenen van $\frac{-(x-\mu)^2}{2\sigma^2}$.

Om dan de classificatie van de nieuwe transactie te bepalen worden de nodige shares als input gegeven aan een Yao circuit. Dit circuit telt de shares op en output dan de beste classificatie.

6.6.4 Voorbeeld

Beschouw de data in Tabel 1.4 uit Hoofdstuk 1. Veronderstel dat de attributen Outlook en Temperature zich bij partij 0 bevinden en de attributen Humidity en Wind bij partij 1. Het class attribuut Play Tennis bevindt zich bij partij 0.
Omdat alle attributen nominaal zijn, zullen de nodige matrices moeten geconstrueerd worden. Aangezien partij 0 zowel het class attribuut heeft als enkele andere partijen hoeven er hier niet echt matrices opgesteld worden. Partij 0 kan zelf bepalen wat de kans is op P(Yes), P(No), P(Sunny|Yes), P(Sunny|No), P(Overcast|Yes), P(Overcast|No), P(Rain|Yes), P(Rain|No), P(Hot|Yes), P(Hot|No), P(Cool|Yes), P(Cool|No), P(Mild|Yes) en P(Mild|No).
Voor de attributen van partij 1, namelijk Humidity en Wind, moeten wel matrices geconstrueerd worden. Zo zal de matrix voor het attribuut Humidity als volgt geconstrueerd worden:

- Humidity High: (1,1,1,1,0,0,0,1,0,0,0,1,0,1)
  - Play Yes $(0, 0, 1/9, 1/9, 1/9, 0, 1/9, 0, 1/9, 1/9, 1/9, 1/9, 1/9, 0)$
    $\Rightarrow$ Scalair product is 3/9.
  - Play No $(1/5, 1/5, 0, 0, 0, 1/5, 0, 1/5, 0, 0, 0, 0, 0, 1/5)$
    $\Rightarrow$ Scalair product is 4/5.

- Humidity Normal: (0,0,0,0,1,1,1,0,1,1,1,0,1,0)
  - Play Yes $(0, 0, 1/9, 1/9, 1/9, 0, 1/9, 0, 1/9, 1/9, 1/9, 1/9, 1/9, 0)$
    $\Rightarrow$ Scalair product is 6/9.
  - Play No $(1/5, 1/5, 0, 0, 0, 1/5, 0, 1/5, 0, 0, 0, 0, 0, 1/5)$
    $\Rightarrow$ Scalair product is 1/5.

De matrix zal er dan als volgt uitzien:

	Yes	**No**
High	3/9	4/5
Normal	6/9	1/5

Hierin staat 3/9 voor P(High|Yes), 4/5 voor P(High|No), 6/9 voor P(Normal|Yes) en 1/5 voor P(Normal|No). In realiteit is deze matrix uiteraard verdeeld over de twee partijen. Merk ook op dat het niet nodig is om de attributen van de andere partijen te kennen. Er kan net zoals bij decision trees een bepaald id toegekend worden aan de attributen. De andere partij weet dan alleen van welk id de gegevens zijn en niet van welk attribuut.
Analoog kan zo ook de matrix voor het attribuut Wind bepaald worden:

	Yes	**No**
Weak	6/9	2/5
Strong	3/9	3/5

Met deze gegevens kan een nieuwe transactie dan geclassificeerd worden.

6.7 Partition based cluster analyse in verticaal verdeelde data: twee of meer partijen

In [48] wordt een algoritme besproken om op een veilige manier partition based clusters te bepalen. Dit algoritme is gebaseerd op het k-means algoritme en kan gebruikt worden voor twee of meer partijen.

6.7.1 Veronderstellingen in verband met de attributen

De partijen kennen alleen hun eigen attributen. Ze weten dus niet welke attributen de andere partijen bezitten.

6.7.2 Het privacy preserving k-means algoritme

Het k-means algoritme bestaat hoofdzakelijk uit drie stappen. Als eerste worden er random k cluster centers vastgelegd. De partijen kunnen dit individueel doen. Elke partij bepaalt het deel van de cluster centers dat overeenkomt met zijn attributen. Daarna moeten de transacties toegewezen worden tot een cluster. Hiervoor moeten de partijen samenwerken. Als laatste worden de nieuwe cluster centers berekend. Dit kunnen de partijen ook weer alleen berekenen aangezien ze alleen de delen van de cluster centers moeten kennen die overeenkomen met hun eigen attributen. Dit alles gebeurt tot dat het verschil tussen de oude en de nieuwe cluster centers klein wordt. Om dit te testen is weer samenwerking tussen de partijen nodig.

Transacties toewijzen aan een cluster Om de transacties te verdelen in clusters moeten de partijen samenwerken. De afstanden van de transacties tot de cluster centers moeten bepaald worden. De partijen hebben per attribuut dat ze bezitten een deel van die afstanden. Elke partij zal per transactie een vector berekenen. Deze vector $\vec{X_i}$ bevat k elementen. Deze elementen staan voor de afstanden tussen de transactie en de k cluster centers. Met deze vectoren zullen de partijen dan bepalen welke transactie aan welke cluster wordt toegewezen.
De eerste partij begint met een aantal random vectoren met k elementen te genereren. Hij construeert één vector $\vec{V_i}$ voor elke partij. De som van de deze vectoren moet gelijk zijn aan 0. Die eerste partij genereert eveneens een permutatie Π voor de k elementen van de vectoren. Partij 0 werkt dan met al de andere partijen samen. Ze maken gebruik van het secure permutation protocol. Partij 0 geeft de vector $\vec{V_i}$ en de permutatie Π als input. De andere partij geeft zijn afstandsvector $\vec{X_i}$ als input. Partij 0 kan $\Pi(\vec{X_0} + \vec{V_0})$ zelf berekenen. De andere partijen bekomen $\Pi(\vec{X_i} + \vec{V_i})$ met behulp van het secure permutation protocol.
Alle partijen behalve partij 1 sturen hun gepermuteerde vector som naar de laatste partij. Deze partij berekent dan de som van alle vectoren die hij aankrijgt en zijn eigen vector. Dit is de som van alle vectoren behalve die van partij 1. Om nu de kleinste afstand te zoeken moet de laatste partij samenwerken met partij 1. Ze moeten de overeenkomstige elementen van de vectoren optellen en de kleinste waarde wordt gekozen.
Merk op dat alle waardes beschermd blijven door de random vectoren. Dankzij het feit dat de vector som van die vectoren gelijk is aan 0 wordt het resultaat ook niet beïnvloed. De twee partijen gebruiken een Yao circuit om te bepalen welke waarde het kleinste is. De index van deze waarde is dan de cluster waaraan de transactie wordt toegekend. Aangezien al de vectoren

gepermuteerd zijn is het nodig om de index naar partij 0 te sturen. Deze partij kent immers de permutatie en kan berekenen welke index overeenkomt met de gepermuteerde waarde.
Indien er maar twee partijen zijn kan dit algoritme niet gebruikt worden. Er zijn dan ook geen hulpmiddelen nodig om de vectoren te beschermen. De twee partijen kunnen samen een Yao circuit uitvoeren. Dit circuit krijgt dan de twee vectoren als input en output de correcte index.

Oude en nieuwe cluster centers vergelijken Om te testen of het algoritme kan stoppen moet er gekeken worden naar het verschil tussen de oude en de nieuwe cluster centers. De partijen berekenen voor al waardes van de cluster centers die ze in hun bezit hebben, de afstand tussen de nieuwe en de oude waardes. Elke partij bekomt dan een afstand. De som van deze afstanden moet vergeleken worden met een vooropgestelde threshold. Als de afstand kleiner is dan de threshold mag het algoritme stoppen.
Dit kan berekend worden met behulp van het secure sum protocol. Elke partij geeft zijn berekende afstand als input. In de voorlaatste stap heeft de laatste partij de som vermeerderd met een random waarde in zijn bezit. De eerste partij kent die random waarde en vermeerdert deze waarde met de threshold. Beide partijen geven deze waardes als input aan een Yao circuit. Het circuit vergelijkt de waardes en kijkt of de threshold groter is dan de afstand. De volledige beschrijving van dit algoritme kan gevonden worden in Algoritme 30.

6.7.3 Voorbeeld

Beschouw de transactie database in Tabel 6.5. Veronderstel dat deze data verdeeld is over drie partijen. Partij 0 heeft het attribuut u, partij 1 heeft de attributen v en w en partij 2 heeft het attribuut x. Wanneer het k-means algoritme uitgevoerd wordt met $k = 2$ beginnen de partijen met random k cluster centers te bepalen. Partij 0 zal dit doen voor het attribuut u, partij 1 voor v en w en partij 2 voor x. Veronderstel dat de cluster centers worden vastgelegd op (1,1,1,1) en (2,2,2,2). Dit wil zeggen dat partij 0 (1) en (2) bepaalt, partij 1 (1,1) en (2,2) en partij 2 (1) en (2). Alle transacties moeten toegewezen worden aan een cluster. Hoe dit zal verlopen voor de eerste transactie kan gevonden worden in Figuur 6.5 en 6.6.
Elke partij berekent de nodige afstanden. Partij 0 heeft een 1 bij de eerste transactie. De vector die deze partij opstelt is dan gelijk aan (0,1). De eerste waarde staat voor de afstand tussen de transactie en de eerste cluster, de tweede waarde voor de afstand tussen de transactie en de tweede cluster. Dit zijn de afstanden $d((1),(1))$ en $d((1),(2))$. Voor de andere partijen kan dit

Algoritme 30 Het privacy preserving k-means algoritme

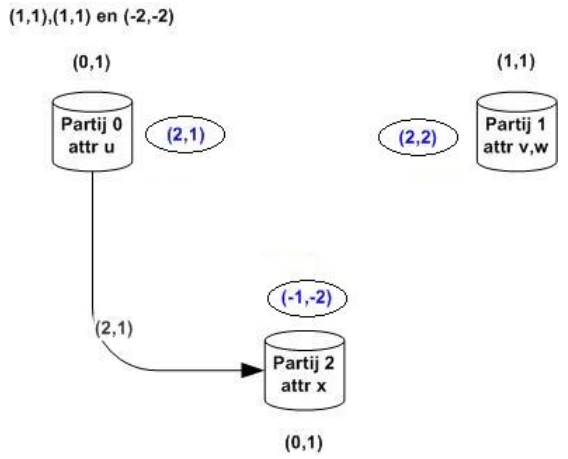
Require: Partijen $0, ..., k-1$.
Require: n transacties en k clusters.
1: **for** $i = 0...k-1$ **do**
2: Partij i construeert random de k cluster centers voor zijn attributen.
3: **end for**
4: **while** De partijen controleren of het verschil tussen de oude en de nieuwe cluster centers groter is dan een threshold **do**
5: **for** $i = 0...k-1$ **do**
6: **for** $j = 1...n$ **do**
7: Partij i berekent een afstandsvector $\vec{X}_i$ met daarin de afstanden van transactie j tot de k cluster centers.
8: **end for**
9: **end for**
10: De partijen werken samen om te bepalen welke tranactie in welke cluster geplaatst wordt.
11: **for** $i = 0...k-1$ **do**
12: Partij i berekent de nieuwe waardes van de cluster centers.
13: **end for**
14: **end while**

analoog. Zo zal partij 1 een vector hebben met de afstanden $d((2,1),(1,1))$ en $d((2,1),(2,2))$, wat (1,1) oplevert. Partij 0 construeert drie random vectoren. Dit zijn in dit geval $(1,1)+(1,1)+(-2,-2)=(0,0)$. Partij 0 stelt eveneens een permutatie Π op. De permutatie zal in dit voorbeeld niet meer doen dan de twee elementen in de vectoren omwisselen. Met deze waardes wordt dan het secure permutation protocol uitgevoerd. De resultaten hiervan staan in de figuur in een ovaaltje aangegeven.
Partij 0 zal zijn waarde naar partij 2 sturen. Partij 2 telt de vectoren bij elkaar op. Dit resultaat wordt dan samen met de vector van partij 1 aan een Yao circuit gegeven. Dit circuit telt de overeenkomstige waardes op en merkt dan dat index 2 het beste resultaat heeft. Dit moet naar partij 0 gestuurd worden. Deze partij kent de permutatie en kan deze ongedaan maken. De eerste transactie zal dan aan de eerste cluster worden toegekend. Op een analoge manier kan dan berekend worden dat de tweede transactie aan de tweede cluster wordt toegekend.
De nieuwe cluster centers worden dan berekend. In dit geval zijn dat (1,2,1,1) en (1,2,2,3). De partijen hebben hier elk hun eigen deel van bepaald. Om te testen of het algoritme mag stoppen moeten de oude waardes met de nieuwe vergeleken worden. Dit kan gevonden worden in Figuur 6.7. Elke partij be-

u	v	w	x
1	2	1	1
1	2	2	3

Tabel 6.5: Transactie database met coördinaten.

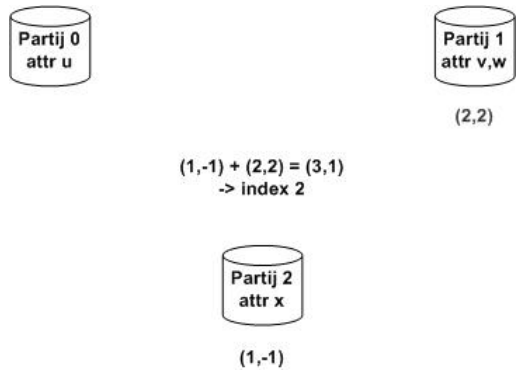
kijkt dit voor zijn eigen waardes. Zo zal partij 0 $d((1),(1)) + d((2),(1)) = 1$ bepalen, partij 1 $d((1,1),(2,1)) + d((2,2),(2,2)) = 1$ en partij 2 $d((1),(1)) + d((2),(3)) = 1$. Met deze waardes wordt dan het secure sum protocol uitgevoerd. Op het einde worden de waardes 7 en 6 met elkaar vergeleken. Aangezien 7 groter is, moet het algoritme nog een volgende iteratie uitvoeren. Dit zal dan ook de laatste zijn omdat de punten niet meer van cluster zullen veranderen.



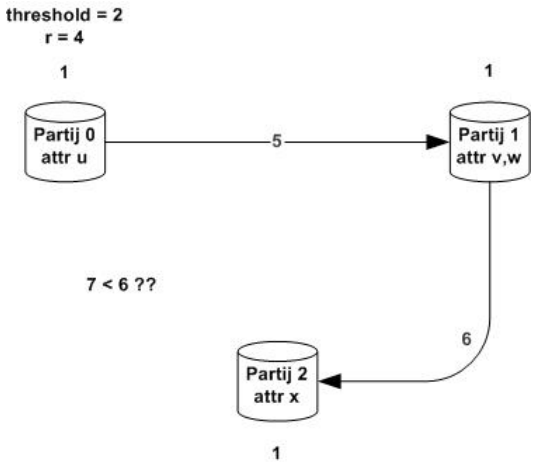
Figuur 6.5: Transactie 1 toewijzen aan cluster 1 (1).

6.8 Outlier analyse in verticaal verdeelde data: twee of meer partijen

Een algoritme om op een privacy preserving manier aan distance based outlier analyse te doen kan gevonden worden in [49]. Dit algoritme kan gebruikt worden in het geval van twee of meer partijen.



Figuur 6.6: Transactie 1 toewijzen aan cluster 1 (2).



Figuur 6.7: Oude en nieuwe cluster centers vergelijken.

6.8.1 Veronderstellingen in verband met de attributen

Het algoritme veronderstelt dat de partijen alleen hun eigen attributen kennen en dus niet die van de andere partijen.

6.8.2 Het privacy preserving distance based outlier detection algoritme

Om (p, d) outliers te zoeken moeten er afstanden bepaald worden tussen de transacties. Voor elke transactie moet geteld worden hoeveel transacties op een afstand groter dan d liggen. Als dit aantal groter is dan p dan is de transactie een outlier.
Voor elke transactie zal apart moeten gekeken worden of het een outlier is. Hiervoor moeten de nodige afstanden berekend worden. Voor alle andere transacties moet de afstand berekend worden tot de beschouwde transacties. De partijen zullen samenwerken om deze afstanden te bekomen. Ze kunnen elk een deel van zo een afstand berekenen, namelijk het deel dat overeenkomt met hun eigen attributen.
Wanneer dit gedaan is, wordt het secure sum protocol gebruikt om de totale afstand te berekenen van een transactie tot de beschouwde transactie. In de voorlaatste stap worden weer twee waardes als input gegeven aan een Yao circuit. Dit circuit bepaalt of de afstand groter is dan d. Als dit het geval is, output het circuit twee random shares van 1. In het andere geval twee random shares van 0.
Op deze manier verzamelen de eerste en de laatste partij voor elke geteste transactie een aantal shares. De som van deze shares is het aantal punten die op een afstand groter dan d liggen van die transactie. Deze shares worden dan ook aan een Yao circuit gegeven. Het circuit telt de shares op en kijkt of het resultaat groter is dan p. Indien dit het geval is dan is de transactie een outlier en zal het circuit random shares van 1 als output geven. Als dit niet zo is, geeft het circuit random shares van 0.
De eerste en de laatste partij bekomen zo een share per transactie. Wanneer een partij nu moet weten of een bepaalde transactie een outlier is dan sturen beide partijen hun overeenkomstig share naar die partij. De partij kan de shares optellen en besluiten of de transactie al dan niet een outlier is. De volledige beschrijving van dit algoritme kan gevonden worden in Algoritme 31.

Algoritme 31 Het privacy preserving distance based outlier detection algoritme

Require: Partijen $0, ..., k-1$.
Require: n transacties.

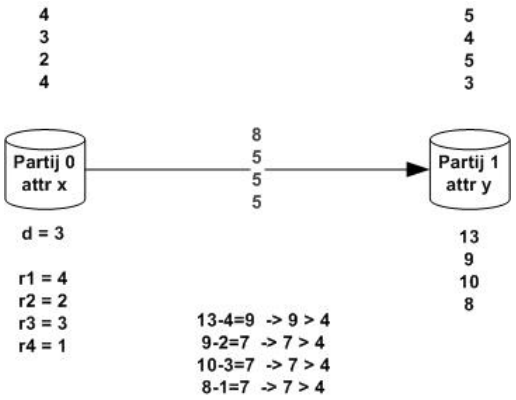
```
for Elke transactie do
    for Alle andere transacties do
        for k = 0...k − 1 do
            Partij i berekent zijn deel van de afstand tot de transactie.
        end for
        De partijen gebruiken het secure sum protocol om de afstanden op te tellen.
        Met behulp van een Yao circuit krijgen de eerste en de laatste partij shares van 0 of 1.
    end for
    De eerste en de laatste partij geven hun shares van de beschouwde transactie als input aan een Yao circuit. Dit circuit output shares van 0 of 1.
end for
```

6.8.3 Voorbeeld

Beschouw de transactie database in Tabel 6.6. Een transactie wordt als een outlier beschouwd wanneer er meer dan drie transacties op een afstand groter dan vier liggen. Om te testen of de vijfde transacties een outlier is zullen de partijen elk hun deel van de afstand bepalen. Voor partij 0 zijn dat de volgende vier afstanden: $d((1),(5)) = 4$, $d((2),(5)) = 3$, $d((3),(5)) = 2$ en $d((1),(5)) = 4$. De afstanden van partij 1 kunnen op een analoge manier bepaald worden. Dit geeft 5, 4, 5 en 3. Deze zullen met het secure sum protocol verwerkt worden. Dit wordt weergegeven in Figuur 6.8. Het Yao circuit bepaalt of de afstanden groot genoeg zijn. Dit zal voor de vier transacties telkens het geval zijn. Beide partijen krijgen telkens een share van 1. De som van deze shares zal gelijk zijn aan vier. Aangezien dit groter is dan de waarde die werd vastgelegd zal de transactie een outlier zijn.

x	y
1	1
2	2
3	1
1	3
5	6

Tabel 6.6: Transactie database met coördinaten.



Figuur 6.8: Bepalen of transactie 5 een outlier is.

Hoofdstuk 7

Nieuwe privacy preserving algoritmes

In het vorige hoofdstuk ontbreken nog een aantal belangrijke gevallen om voor alle data mining algoritmes uit Hoofdstuk 1 een privacy preserving variant te hebben. De ontbrekende situaties zijn de volgende:

- Decision trees in horizontaal verdeelde data voor meer dan twee partijen.
- Partition based cluster analyse in horizontaal verdeelde data voor twee of meer partijen.
- Outlier analyse in horizontaal verdeelde data voor twee of meer partijen.

Al deze gevallen ontbreken in de literatuur. We hebben de verschillende situaties onderzocht en besloten dat het mogelijk is om ook voor deze gevallen algoritmes te construeren die veilig zijn. Deze algoritmes zijn opgesteld en gebruiken eveneens de beschreven bouwstenen. Ze zullen verder besproken worden in dit hoofdstuk.

7.1 Decision trees in horizontaal verdeelde data: meer dan twee partijen

7.1.1 Het privacy preserving ID3 algoritme

Het ID3 algoritme bestaat uit drie stappen. Net zoals in de andere situaties is het ook hier mogelijk om al deze onderdelen te beveiligen met behulp van de bouwstenen.

R is leeg Dankzij de horizontale verdeling en het feit dat alle partijen het tussenresultaat van de boom kennen, kunnen ze bepalen of R leeg is of niet. Wanneer dit het geval is, moet de meest voorkomende class waarde bepaald worden. Dit kunnen de partijen berekenen door voor elke class waarde gebruik te maken van het secure sum protocol. Elke partij geeft als input het aantal transacties met een bepaalde class waarde in zijn bezit.
Om het algoritme extra te beveiligen kan er gebruik gemaakt worden van een Yao circuit. De sommen worden niet volledig berekend. Er wordt telkens gestopt wanneer de laatste partij de som vermeerderd met een random getal in zijn bezit heeft. De eerste partij kent dan die random waarde. De partijen houden deze waardes bij voor elke mogelijke class waarde. Als al deze waardes berekend zijn, worden ze als input gegeven aan een Yao circuit. Het circuit vermindert de sommen met de random waardes en output welke class waarde de grootste som heeft.

Alle transacties in T hebben dezelfde class waarde Om te bepalen of alle transacties in T dezelfde class waarde hebben, kan er gebruik gemaakt worden van een variant op het secure union protocol. Alle partijen geven één waarde als input aan het protocol. Indien een partij maar één class waarde heeft in zijn transacties geeft hij die waarde als input. Als er nog meerdere zijn dan wordt een vastgelegd symbool $\perp$ als input gegeven.
Het protocol verloopt dan analoog aan het secure union protocol tot de stap waarin er moet gedecrypteerd worden. Partij 0 heeft alle waardes in zijn bezit en hij haalt hier de dubbels uit. Er zijn nu twee mogelijkheden. Ofwel blijft er maar één waarde over. Dit moet dan de waarde zijn die partij 0 zelf als input gegeven heeft. Dat zal ofwel een class waarde zijn ofwel het $\perp$ symbool. Als het een class waarde is dan wilt dit zeggen dat alle partijen die waarde aan het protocol hebben gegeven en dat alle transacties in T dezelfde class waarde hebben. In het andere geval hebben alle partijen nog meer dan één class waarde over.
De tweede mogelijkheid is dat er na het verwijderen van de dubbels nog meer dan één waarde overblijft. In dit geval is het zeker dat er nog meerdere class waardes zijn. Het protocol moet dan gestopt worden. Er mag immers niet gedecrypteerd worden want dan worden de waardes vrijgegeven.

Anders In het andere geval moet het attribuut bepaald worden dat de transacties in T het beste classificeert. Er moet hier een aantal keer $xln(x)$ berekend worden waarbij de waarde van x verdeeld zit over de partijen. Het is mogelijk om dit op te lossen met het secure sum protocol. De partijen geven als input hun deel van x aan het protocol. De som die de laatste par-

tij kent en de random waarde van de eerste partij vermenigvuldigt met -1 worden als input gegeven aan het $xln(x)$ protocol. Dit circuit zal shares van het resultaat als output geven. Deze waardes kunnen dan als input gegeven worden aan een circuit dat de overeenkomstige waardes optelt en output welk attribuut de transacties het beste classificeert. De volledige werking van het algoritme kan gevonden worden in Algoritme 32.

Algoritme 32 Het privacy preserving ID3 algoritme

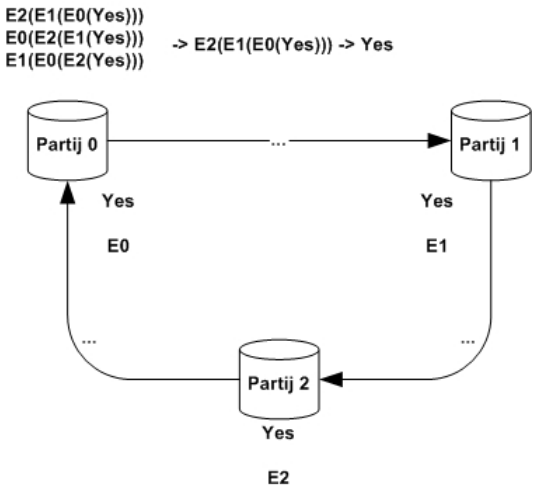
Require: R, de set van attributen.
Require: C, het class attribuut.
Require: T, de set van transacties.

if De partijen testen of R leeg is **then**
    Met behulp van het secure sum protocol wordt berekend welke class waarde c_i het meeste voorkomt.
    Return een blad met als class waarde c_i.
else if De partijen gebruiken het secure union protocol om te testen of alle transacties in T dezelfde class waarde c_i hebben **then**
    Return een blad met als class waarde c_i.
else
    Bepaal het attribuut A dat de transacties in T het beste classificeert: Met behulp van het secure sum protocol en het $xln(x)$ protocol wordt het beste attribuut bepaald.
    Verdeel T in m delen $T(a_1), ..., T(a_m)$ zodat $a_1, ..., a_m$ de verschillende waardes van attribuut A zijn.
    Return een boom met als wortel een knoop A en m takken $a_1, ..., a_m$ zodat tak i $ID3(R - \{A\}, C, T(a_i))$ bevat.
end if

7.1.2 Voorbeeld

Beschouw de data in Tabel 1.4 uit Hoofdstuk 1. Veronderstel dat er drie partijen zijn. De transacties 1 tot 5 bevinden zich bij partij 0, de transacties 6 tot 10 bij partij 1 en de transacties 11 tot 14 bij partij 2.
De wortel van de boom is het attribuut Outlook. Voor de tak met Outlook gelijk aan Overcast moet bepaald worden of alle transacties dezelfde class waarde hebben. Dit kan gevolgd worden in Figuur 7.1. De drie partijen bestuderen hun eigen transacties en nemen dan deel aan het secure union protocol. Ze zullen alledrie Yes als input geven omdat dat de enige class waarde is die ze hebben bij de transacties met Outlook Overcast. Partij 0

zal alle waardes ontvangen en merken dat ze gelijk zijn. Dit wil zeggen dat alle partijen Yes hebben ingegeven. Er mag dus besloten worden dat alle transacties dezelfde class waarde hebben en dat die waarde gelijk is aan Yes.



Figuur 7.1: Bepalen of alle transacties dezelfde class waarde hebben.

7.2 Partition based cluster analyse in horizontaal verdeelde data: twee of meer partijen

7.2.1 Het privacy preserving k-means algoritme

Het k-means algoritme bestaat hoofdzakelijk uit drie stappen. Dankzij de horizontale verdeling kunnen de partijen veel van de berekeningen individueel uitvoeren. De k cluster centers zijn gekend door de partijen. De oorspronkelijke k cluster centers worden door één partij gegenereerd en bekend gemaakt aan de andere partijen. Ze kunnen dan ook zelf bepalen tot welke clusters hun transacties horen. Om de nieuwe cluster centers te berekenen moeten de partijen wel samenwerken.
Per cluster moeten ze voor elk attribuut twee waardes kennen. Namelijk de som van de attribuut waardes die tot die cluster behoren en het aantal transacties die tot die cluster behoren. Op deze manier kunnen ze de nieuwe cluster centers berekenen. Het testen of er mag gestopt worden, kunnen de partijen ook individueel doen.

Beide waardes zijn sommen waarvan elke partij een deel heeft. Ze kunnen dan ook berekend worden met behulp van het secure sum protocol. Merk op dat dezelfde problemen als bij het Naïve Bayes algoritme hier aan bod komen. De problemen van het gebruik van het secure sum protocol beschreven in sectie 6.5.2 zijn ook hier van toepassing. Ze kunnen op dezelfde manier opgelost worden namelijk door gebruik te maken van het secure division protocol of het secure $ln(x)$ protocol. Beide protocollen kunnen ook in het geval van meer dan twee partijen toegepast worden als extra beveiliging. Dit kan op analoge manier als beschreven bij het Naïve Bayes algoritme. De volledige werking van het algoritme kan gevonden worden in Algoritme 33.

Algoritme 33 Het privacy preserving k-means algoritme

Require: Partijen $0, ..., k-1$.
Require: n transacties en k clusters.
1: **for** $i = 0...k-1$ **do**
2: Partij i construeert random de k cluster centers voor zijn attributen.
3: **end for**
4: **while** De partijen controleren of het verschil tussen de oude en de nieuwe cluster centers groter is dan een threshold **do**
5: **for** $i = 0...k-1$ **do**
6: Partij i kent zijn transacties toe aan de beste cluster.
7: **end for**
8: De partijen werken samen om de nieuwe cluster centers te bepalen met behulp van het secure sum protocol.
9: **end while**

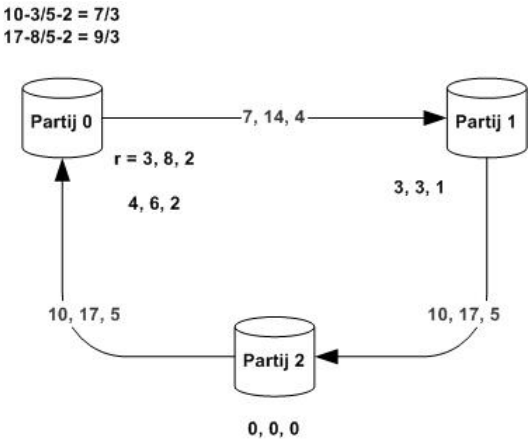
7.2.2 Voorbeeld

Beschouw de data in Tabel 1.5 uit Hoofdstuk 1. Veronderstel dat er drie partijen zijn. De transacties 1 en 2 bevinden zich bij partij 0, de transacties 3 en 6 bij partij 1 en de transacties 4 en 5 bij partij 2.
Wanneer het k-means algoritme uitgevoerd wordt met $k = 2$ kiezen de partijen random twee cluster centers. Alle transacties worden dan toegekend aan een cluster. Veronderstel dat transacties 1, 2 en 3 bij cluster 1 worden geplaatst en transacties 4, 5 en 6 bij cluster 2. Er moeten dan nieuwe cluster centers berekend worden. Hiervoor zullen de partijen de nodige waardes berekenen. Het verloop van deze berekeningen kan gevolgd worden in Figuur 7.2. Alle clusters bepalen in dit geval drie waardes:

- De som van de x waardes die tot de eerste cluster behoren.

- De som van de y waardes die tot de eerste cluster behoren.
- Het aantal transacties die tot de eerste cluster behoren.

Voor partij 0 zullen dit 4, 6 en 2 zijn. Voor partij 1 3, 3 en 1 en voor partij 2 0, 0 en 0. Deze waardes worden als input gegeven aan het secure sum protocol. Het resultaat hiervan geeft het nieuwe cluster center: $(\frac{7}{3}, \frac{9}{3})$. Voor de tweede cluster zal dit analoog verlopen.



Figuur 7.2: De nieuwe cluster centers bepalen.

7.3 Outlier analyse in horizontaal verdeelde data: twee of meer partijen

7.3.1 Het privacy preserving distance based outlier detection algoritme

Om te bepalen of een transactie een (p, d) outlier is, moeten er afstanden berekend worden. De transactie die getest wordt, is altijd volledig aanwezig bij één partij. Deze partij kan zonder problemen de afstanden berekenen van de behandelde transactie tot al zijn andere transacties. Die partij kan dan bijhouden hoeveel van zijn transacties op een afstand groter dan d liggen van de beschouwde transactie.
De afstanden tot de transacties die zich bij andere partijen bevinden, kunnen berekend worden met behulp van een oblivious polynomial evaluation. De partij die de transactie bezit kan een vergelijking opstellen die het gebied

uitdrukt waarin de punten liggen die op een afstand minder dan d van de transactie liggen. Voor een transactie (x_1, y_1) en een afstand d zal die vergelijking er als volgt uitzien: $(x - x_1)^2 + (y - y_1)^2 - d^2$. Wanneer hier een punt wordt ingevuld kan uit het resultaat worden afgeleid of het punt al dan niet binnen het beschreven gebied ligt.
De partij die de behandelde transactie in zijn bezit heeft voert een oblivious polynomial evaluation uit voor al de andere transacties. Om te voorkomen dat de afstand echt geleerd wordt, wordt het protocol aangepast. In de voorlaatste stap heeft de ene partij de afstand vermeerderd met een random waarde. De andere partij heeft die random waarde. Deze waardes worden als input gegeven aan een Yao circuit. Het circuit berekent de echte afstand en vergelijkt die waarde met d. Als de afstand groter is dan d output het circuit 1, als dit niet zo is geeft het 0 als output.
De partij met de beschouwde transactie telt deze waardes op bij het aantal transacties dat hij zelf al gevonden had. Indien er in totaal meer dan p transacties geteld zijn kan de transactie als een outlier beschouwd worden. De volledige werking van het algoritme kan gevonden worden in Algoritme 34.

Algoritme 34 Het privacy preserving distance based outlier detection algoritme

---

Require: Partijen $0, ..., k - 1$.
Require: n transacties.
1: **for** Elke transactie **do**
2: Partij i die de beschouwde transactie bezit, telt het aantal transacties in zijn bezit die op een afstand groter dan d van de beschouwde transactie liggen.
3: Partij i stelt een vergelijking op.
4: **for** $j = 0...i - 1, i + 1...k - 1$ **do**
5: Partij i voert met partij j oblivious polynomial evaluations uit en telt het aantal transacties die op een afstand groter dan d liggen van de beschouwde transactie.
6: **end for**
7: Indien er meer dan p transacties geteld zijn, is de beschouwde transactie een outlier.
8: **end for**

7.3.2 Voorbeeld

Beschouw de data in Tabel 6.6 uit Hoofdstuk 6. Veronderstel dat er twee partijen zijn. De transacties 1, 2 en 3 bevinden zich bij partij 0 en de transacties 4 en 4 bij partij 1.
Partij 1 wil testen of zijn tweede transactie een $(3, 4)$ outlier is. Hij test zelf zijn andere transactie. Deze transactie ligt op een afstand groter dan 4 en dus zet hij zijn teller op 1. Partij 1 voert dan drie oblivious polynomial evaluations uit met partij 0. Hier krijgt hij telkens een 1 terug. Hij vermeerdert zijn teller met 3 en kan besluiten dat de transactie een outlier is.

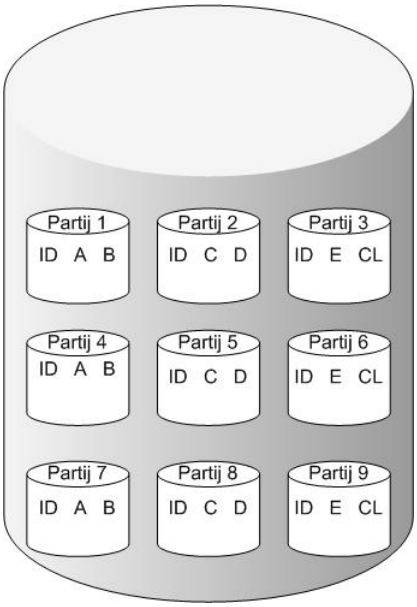
Hoofdstuk 8

Situatie met horizontaal en verticaal verdeelde data

Een speciale situatie die nergens beschouwd wordt, is het geval waarin de data zowel horizontaal als verticaal verdeeld is. We hebben deze situatie onderzocht en besloten dat er ook in dergelijke gevallen gebruik gemaakt kan worden van de bouwstenen om een privacy preserving data mining algoritme te verkrijgen. Deze aanpak zal in dit hoofdstuk verder geïllustreerd worden aan de hand van het ID3 algoritme om decision trees te bouwen.

8.1 Een motiverend voorbeeld

Het heeft uiteraard geen zin om algoritmes uit te werken die nergens voor nodig zijn. Dit is in deze situatie zeker niet het geval. Veronderstel dat er een grootschaals onderzoek wordt gedaan naar gezondheidsproblemen en hun oorzaken. Mogelijke resultaten zouden kunnen zijn de woonplaats, bepaalde voedingsgewoonten, werk en nog veel meer. Hiervoor zal data van verschillende bronnen gecombineerd moeten worden.
Algemene gegevens over personen kunnen gevonden worden in het stadhuis van hun woonplaats, voor het vinden van verbanden met bepaalde voedingsmiddelen is data van warenhuizen nodig, de gezondheidsgegevens kunnen gevonden worden bij huisdokters en ziekenhuizen. De data zal dus zowel horizontaal als verticaal verdeeld zijn. Horizontaal omdat er meerdere stadhuizen, warenhuizen en ziekenhuizen zullen meewerken. Verticaal omdat al de partijen verschillende gegevens bezitten. Een schematisch voorbeeld van een dergelijke situatie kan gevonden worden in Figuur 8.1. De data is hier verdeeld onder negen partijen. Er zijn vijf attributen, namelijk A, B, C, D en E en een class attribuut CL.



Figuur 8.1: Horizontaal en verticaal verdeelde data.

8.2 Mogelijke oplossingsmethodes

Er zijn twee voor de hand liggende aanpakken om dit probleem op te lossen:

- De data horizontaal samenbrengen om ze daarna verticaal verder uit te werken.
- De data verticaal samenbrengen om ze daarna horizontaal verder uit te werken.

Het idee hierachter is om telkens één verdeling weg te werken zodat een aantal van de partijen daarna kunnen samenwerken volgens de andere verdeling. Deze twee mogelijkheden zullen achtereenvolgens besproken en geanalyseerd worden.

8.2.1 Data horizontaal samenbrengen en verticaal verder uitwerken

De drie stappen van het ID3 algoritme zullen apart behandeld worden.

R is leeg Om te bepalen of er nog attributen overblijven moeten er net zoveel partijen samenwerken als er verticale verdelingen zijn. In het voorbeeld kunnen partijen P_1, P_2 en P_3 bepalen of er nog attributen overblijven. De

partijen bekijken hoeveel attributen ze bezitten die nog in aanmerking komen en geven deze waarde als input aan het secure sum protocol. Op het einde van het protocol worden de som vermeerderd met de random waarde en de random waarde zelf als input gegeven aan een Yao circuit. Dit circuit test of de som gelijk is aan 0 en R inderdaad leeg is.
Indien dit het geval is, moet er bepaald worden welke class waarde het meeste voorkomt. Per verticale groep wordt een secure union protocol uitgevoerd. In dit geval zullen partijen P_1, P_4 en P_7, P_2, P_5 en P_8 en P_3, P_6 en P_9 een unie berekenen. De partijen die naast elkaar liggen zullen dezelfde encryptie key gebruiken. Hier zijn dat P_1, P_2 en P_3, P_4, P_5 en P_6 en P_7, P_8 en P_9. De partijen maken dan een set met daarin de transacties die tot op het huidig niveau raken. Van deze sets worden de unies berekend. De partijen met het class attribuut moeten per class waarde een set en een unie berekenen.
Het is echter niet nodig om de waardes te decrypteren. Dit zou uiteraard te veel vrijgeven. Er wordt dus gestopt op het moment dat partijen P_1, P_4 en P_7 de volledig geëncrypteerde unies in hun bezit hebben. Omdat deze waardes met dezelfde encryptie keys geëncrypteerd zijn, zijn overeenkomsten in de geëncrypteerde unies ook overeenkomsten in de echte unies. Om te weten welke class waarde moet gekozen worden, werken de drie partijen samen. Er moeten doorsnedes berekend worden van de gemaakte unies. Merk op dat het niet nodig is om het secure size of set intersection protocol volledig uit te voeren. De unies zijn immers al geëncrypteerd. Daarom kunnen de partijen hun geëncrypteerde unies gewoon naar de partij sturen die het class attribuut bezit. In dat geval zal dat P_3 zijn. Deze partij zal per class waarde een doorsnede berekenen. De class waarde met het grootste resultaat is de waarde die het meeste voorkomt. P_3 kan een blad construeren met een bepaald id. De waarde van het blad wordt meegedeeld aan de partijen die de class waarde bezitten en het id aan de andere partijen.

Alle transacties in T hebben dezelfde class waarde Het nagaan of alle transacties in T dezelfde class waarde hebben kan op een analoge manier als het bepalen van de class waarde die het meeste voorkomt. In deze situatie leert P_3 de verschillende doorsnedes. Indien alle doorsnedes gelijk zijn aan 0, behalve één hebben alle transacties in T die bepaalde class waarde. Deze partij zal dan een blad construeren met een bepaald id, en dat id meedelen aan de andere partijen. Aan de partijen die het class attribuut bezitten wordt de echte waarde bekend gemaakt.

Anders In het andere geval moet het attribuut bepaald worden dat de transacties in T het beste classificeert. Hiervoor moeten transacties geteld

worden. Om deze aantallen te leren, stellen de partijen de nodige sets op en berekenen op dezelfde manier als in de eerste stap enkele unies. Van deze unies worden doorsnedes berekend. De resultaten worden geleerd door één van de bovenste partijen, namelijk de partij die het attribuut in zijn bezit heeft waarvoor de information gain bepaald wordt. Met deze waardes kan die partij de nodige berekeningen maken. Er wordt dan berekend welk attribuut de hoogste gain heeft. De bovenste partij met dat attribuut in zijn bezit construeert een knoop met een bepaald id. Deze partij deelt de waarde van de knoop mee aan de partijen die dat attribuut bezitten en het id aan de andere partijen.

8.2.2 Data verticaal samenbrengen en horizontaal verder uitwerken

De drie stappen van het ID3 algoritme zullen apart behandeld worden.

R is leeg Het nagaan of R leeg is, kan moeilijk op een efficiënte manier gebeuren wanneer de data verticaal wordt samengebracht. Partijen die een horizontale groep vormen, kunnen elkaar niets leren aangezien ze dezelfde attributen bezitten. Daarom wordt dit op dezelfde manier gedaan als bij het horizontaal samenbrengen van de data.
Bepalen welke class waarde het meeste voorkomt kan wel door de data verticaal samen te brengen. Elke partij stelt een set op met de transacties die tot op het huidig niveau geraken. De partijen met het class attribuut moeten per class waarde een set berekenen. Per horizontale groep worden net zoveel secure size of set intersection protocollen uitgevoerd als er class waardes zijn. De partijen P_3, P_6 en P_9 leren de resultaten van deze protocollen. Per class waarde wordt dan een secure sum protocol uitgevoerd. De sommen vermeerdert met de random waardes worden samen met de random waardes als input gegeven aan een Yao circuit. Dit circuit berekent de sommen en kijkt welke class waarde het meeste voorkomt. Op deze manier leert P_3 welke waarde er moet gekozen worden. Deze partij kan een blad construeren met een bepaald id. De waarde van het blad wordt meegedeeld aan de partijen die de class waarde bezitten en het id aan de andere partijen.

Alle transacties in T hebben dezelfde class waarde Het nagaan of alle transacties in T dezelfde class waarde hebben kan op een analoge manier als het bepalen van de class waarde die het meeste voorkomt. Het Yao circuit kan testen of alle sommen gelijk zijn aan 0, behalve één. Als dit zo is, hebben alle transacties in T dezelfde class waarde. P_3 zal dan een blad construeren met

een bepaald id, en dat id meedelen aan de andere partijen. Aan de partijen die het class attribuut bezitten wordt de echte waarde bekend gemaakt.

Anders In het andere geval moet het attribuut bepaald worden dat de transacties in T het beste classificeert. Hiervoor moeten transacties geteld worden. Deze tellingen gebeuren door een aantal secure size of set intersection protocollen uit te voeren, gevolgd door een aantal secure sum protocollen. De partijen stellen de nodige sets op en berekenen per horizontale groep de nodige doorsnedes. De partijen die het attribuut bezitten waarvoor de information gain bepaald wordt, leren de waardes en kunnen de sommen berekenen met behulp van enkele secure sum protocollen. De som vermeerderd met de random waarde en de random waarde vermenigvuldigt met -1 worden als input gegeven aan het $xln(x)$ protocol. Dit circuit zal shares van het resultaat als output geven. Deze waardes kunnen dan als input gegeven worden aan een circuit dat de overeenkomstige waardes optelt en output welk attribuut de transacties het beste classificeert. Op deze manier kan dan bepaald worden welk attribuut de hoogste gain heeft. De bovenste partij met dat attribuut in zijn bezit construeert een knoop met een bepaald id. Deze partij deelt de waarde van de knoop mee aan de partijen die dat attribuut bezitten en het id aan de andere partijen.

Merk op dat in beide gevallen net zoals in het privacy preserving algoritme voor verticaal verdeelde data ook hier niet de volledige boom geleerd wordt. Hierdoor wordt er minder vrijgegeven. Dit heeft ook als gevolg dat de partijen moeten samenwerken om een nieuwe transactie te classificeren. Hiervoor kunnen de bovenste rij partijen samenwerken, aangezien zij de knopen en bladeren aanmaken en weten welke waardes bij welke id's horen.
Verder is het ook belangrijk dat indien er een knoop met een bepaald attribuut geconstrueerd wordt, alle partijen die dat attribuut bezitten dat weten. Dit is nodig om te bepalen welke transacties tot op het huidige niveau in de boom geraken. Als ze niet weten welke attributen al voorgekomen zijn, is dit niet mogelijk.

8.3 Vergelijking tussen de methodes

Een voor de hand liggende vraag is nu welke methode de beste is. Om op deze vraag een antwoord te krijgen, zal er een analyse gemaakt worden van de complexiteit van de algoritmes. De gebruikte notaties in deze analyse kunnen gevonden worden in Tabel 8.1.

Notatie	Betekenis
k	Het aantal partijen.
h	Het aantal horizontale groepen.
v	Het aantal verticale groepen.
$\|T\|$	Het aantal transacties.
$\|R\|$	Het aantal attributen.
c	Het aantal waardes voor het class attribuut.
i	Het aantal waardes voor een attribuut.

Tabel 8.1: Notaties voor de complexiteitsanalyse.

Het belangrijkste onderdeel in het ID3 algoritme is het bepalen van het attribuut met de hoogste information gain. De analyse zal dan ook gebaseerd zijn op deze stap.

8.3.1 Complexiteiten van de bouwstenen

In de privacy preserving data mining algoritmes is er sprake van twee soorten complexiteiten. Namelijk de computation complexiteit en de communication complexiteit. Bij de computation kost draait het om de berekeningen, bij de communication kost om de berichten die moeten doorgestuurd worden.

8.3.1.1 Het secure union protocol

De volgende waardes gelden voor een secure union protocol dat uitgevoerd wordt met k partijen. De partijen sturen sets door niet groter dan $|T|$. Verder maken ze gebruik van een encryptie key met lengte t.
De factor t^3 bij de computation kost staat hier voor de kost van het encrypteren van de sets. Elke partij moet k^2 sets encrypteren. De waarde t in de communication kost duidt op de grootte van de sets die worden doorgestuurd. Er worden k^2 berichten doorgestuurd met als grootte $|T|.t$.

Computation kost $O(k^2.|T|.t^3)$

Communication kost $O(k^2.|T|.t)$

8.3.1.2 Het secure size of set intersection protocol

De complexiteit van het secure size of set intersection protocol is gelijk aan die van het secure union protocol.

Computation kost $O(k^2.|T|.t^3)$

Communication kost $O(k^2.|T|.t)$

8.3.1.3 Het secure sum protocol

De complexiteit van het secure sum protocol is gebaseerd op een uitvoering met k partijen. De partijen geven als input aan het protocol waardes niet groter dan $|T|$ en berekenen telkens een som. De berichten die worden doorgestuurd bedragen eveneens maximaal $|T|$.

Computation kost $O(k.log|T|)$

Communication kost $O(k.log|T|)$

8.3.1.4 Het secure xln(x) protocol

Het secure $xln(x)$ protocol wordt steeds uitgevoerd met twee partijen. De waardes die als input aan het circuit worden gegeven bedragen maximaal $|T|$. Verder is het protocol afhankelijk van een waarde n. Het protocol berekent een Taylor reeks en deze waarde geeft aan in welke mate de reeks ontwikkelt wordt.
Het protocol bestaat uit een Yao circuit. Eén van de twee partijen stelt een geëncrypteerd circuit op en verwerkt zijn eigen input in het circuit. Dit circuit moet doorgestuurd worden naar de andere partij. Het doorsturen zorgt hier voor de communication complexiteit. De factor n speelt hier uiteraard een rol in aangezien hij bepaalt hoe groot het circuit is.
Als de tweede partij dit ontvangt moet hij zijn eigen input nog in het circuit verwerken. Hiervoor wordt per bit van zijn input een oblivious transfer uitgevoerd. Deze stap zorgt voor de computation complexiteit.

Computation kost $O(log|T|)$

Communication kost $O(n.log|T|.t)$

8.3.2 Complexiteit van het horizontaal samenbrengen van de data

Om te bepalen welk attribuut de transacties het beste classificeert moeten er per attribuut enkele unies bepaald worden over h partijen. Hoeveel unies er

exact moeten bepaald worden, is afhankelijk van welk attribuut er behandeld wordt. Indien het een attribuut is dat zich bevindt bij de partij die het class attribuut bezit zijn het er $(v + c + i - 1)$. Indien het attribuut zich bij een andere partij bevindt zijn het er $(v + c + i - 2)$.
Ook het aantal unies die moeten doorgestuurd worden, hangt af van het attribuut. Voor attributen die zich bij de partij bevinden die het class attribuut bezit moeten er $(v-1)$ worden doorgestuurd, voor de andere attributen $(v + c - 2)$.

Computation kost $O(|R|.(v + c + i).(h^2.|T|.t^3))$

Communication kost $O(|R|.(v + c).(h^2.|T|.t))$

Merk op dat de sterkte in dit protocol het feit is dat er wordt vanuit gegaan dat de partijen die naast elkaar liggen gebruik maken van dezelfde encryptiekeys. Om deze reden is het niet nodig om het secure size of set intersection protocol te gebruiken om de doorsnedes te berekenen. De eerste fase van dat protocol is immers al gebeurd door het encrypteren met dezelfde keys bij het berekenen van de unies.
Dit is hier mogelijk omdat de data zowel horizontaal als verticaal verdeeld is. Wanneer de data alleen verticaal verdeeld is zou het ook mogelijk zijn om de partijen hier enkele encryptie keys te laten afspreken en het protocol op deze manier te vereenvoudigen.

8.3.3 Complexiteit van het verticaal samenbrengen van de data

Per attribuut moeten er waardes berekend worden. Dit zijn er in totaal $(1 + c + i + c.i)$:

- Het aantal transacties tot in de huidige node: 1.
- Het aantal transacties tot in de huidige node per class waarde: c.
- Het aantal transacties tot in de huidige node per attribuut waarde: i.
- Het aantal transacties tot in de huidige node per class waarde en per attribuut waarde: $(c.i)$.

Al deze waardes kunnen berekend worden aan de hand van een secure size of set intersection protocol dat telkens wordt uitgevoerd door v partijen. Deze tellingen moeten gebeuren per horizontale groep dus in totaal geeft dit

$h.(1+c+i+c.i)$ uitvoeringen van het secure size of set intersection protocol. Met deze waardes wordt dan verder gewerkt. In het geval van twee horizontale groepen is dat met het $xln(x)$ protocol, in het andere geval met het secure sum protocol gevolgd door het secure $xln(x)$ protocol.

Computation kost $O(|R|.(h.(1+c+i+c.i)).(v^2.|T|.t^3)) + O(|R|.(1+c+i+c.i).(log|T|)) + O(|R|.log|T|)$ $[+O(h.log|T|)]$

Communication kost $O(|R|.(h.(1+c+i+c.i)).(v^2.|T|.t)) + O(|R|.(1+c+i+c.i).n.(log|T|.t)) + O(|R|.log|T|.t)$ $[+O(h.log|T|)]$

8.4 Besluit

Een eerste opmerking die kan gemaakt worden, is dat het logischer lijkt om de data eerst horizontaal samen te brengen en dan verticaal verder uit te werken. De stap in het algoritme waarin getest wordt of de verzameling van attributen leeg is kan namelijk alleen maar op die manier op een efficiënctie wijze gebeuren.
Verder geeft het secure $xln(x)$ protocol een benadering van de echte waarde. De kans dat dit echt sterk verschillende resultaten geeft is uiteraard klein. Toch is het eenvoudig kiezen tussen een resultaat dat zeker correct is, of een resultaat dat een kleine afwijking kan vertonen.
Het secure $xln(x)$ protocol maakt ook gebruik van een zware circuitberekening. In de praktijk is het altijd handiger om zulke situaties te vermijden.
Wat de complexiteit betreft, blijkt dat het horizontaal samenbrengen van de data ook een stuk voordeliger is. Zoals reeds opgemerkt is dit het gevolg van het optimaal benutten van encryptie. Door bepaalde partijen dezelfde encryptie keys te geven, is het niet nodig om na de secure union protocollen nog secure size of set intersection protocollen uit te voeren en dat scheelt uiteraard heel wat in de complexiteit.

Hoofdstuk 9

Besluit

9.1 Nieuwe bijdragen

Deze thesis bevat naast een literatuurstudie van het domein ook nog een aantal uitbreidingen. Er werd onderzocht welke problemen nog geen concrete oplossingen hadden. Door gebruik te maken van de bouwstenen werden nieuwe algoritmes geconstrueerd om deze open problemen op te lossen. Verder werd er ook een speciale situatie bekeken waarin de data zowel horizontaal als verticaal verdeeld is over de verschillende partijen. Aan de hand van het ID3 algoritme wordt aangetoond dat ook deze situatie kan opgelost worden met behulp van de bouwstenen.

9.2 Waarom de cryptografie gebaseerde methodes gebruiken?

Deze hele thesis handelt over cryptografie gebaseerde privacy preserving methodes. Een belangrijke vraag is nu natuurlijk waarom deze aanpak verkozen wordt boven de reconstructie gebaseerde methodes. Het antwoord op deze vraag blijkt uit deze thesis. Met een toolkit van een tiental bouwstenen kunnen een heel aantal data mining algoritmes omgevormd worden naar privacy preserving data mining algoritmes. Het is mogelijk om de juiste verhouding tussen efficiëntie en privacy te verkrijgen. Er zijn verschillende extra beveiligingen mogelijk zoals het toevoegen van een Yao circuit aan het secure sum protocol waardoor het algoritme iets minder efficiënt is maar wel veel veiliger. Op deze manier kunnen de partijen zelf uitmaken hoe belangrijk privacy is en of ze iets meer willen vrijgeven als ze daardoor een efficiënter algoritme verkrijgen.

Verder zijn de cryptografie gebaseerde methodes ook meestal 100% correct. Sommige protocollen geven een benadering maar dit kan niet voor grote verschillen zorgen. Over de reconstructie gebaseerde methodes zijn minder resultaten bekend. Aangezien er met verminkte data gewerkt wordt, is de kans groter dat er minder betrouwbare resultaten bekomen worden.
Een laatste belangrijk punt is het feit dat de partijen hun data hier niet echt moeten vrijgeven. Wanneer privacy voor de partijen uiterst belangrijk is, zullen ze het waarschijnlijk sneller aanvaaarden om enkele waardes vrij te geven (waarvan ze weten dat de andere partijen er niets mee kunnen doen) dan hun hele database, zoals in de reconstructie gebaseerde methodes gevraagd wordt.

9.3 Eindigt het hier?

De literatuur bevat reeds heel wat privacy preserving data mining algoritmes die volgens de cryptografie gebaseerde methodes zijn opgebouwd. Na het onderzoeken van de ontbrekende gevallen bleek dat ook deze situaties zonder problemen kunnen opgelost worden met de bestaande bouwstenen. Natuurlijk eindigt het hier niet. Er zullen nog meer data mining algoritmes kunnen geconstrueerd worden. Denk bijvoorbeeld aan density bases clustering. Het DBSCAN algoritme vertoont bepaalde overeenkomsten met het distance based outlier detection algoritme. In beide gevallen moeten er afstanden bepaald worden en transacties geteld worden. Een core punt kan op dezelfde manier bepaald worden als een outlier. Het enige verschil is dat er hier transacties moeten geteld worden die op een afstand *niet* groter dan een bepaalde waarde liggen.
Verder kan er ook onderzocht worden of de protocollen misschien nog extra kunnen beveiligd worden. Wat er wordt vrijgegeven hangt vaak samen met het feit dat de partijen weten van wie ze berichten aankrijgen. Een mogelijke oplossing zou kunnen zijn om gebruik te maken van zogenaamde anonymizing protocollen afkomstig uit de internetbeveiliging waardoor de afkomsten van de berichten beschermd blijft.
Ook kan er gekeken worden hoe de partijen kunnen valsspelen en hoe de algoritmes kunnen beveiligd worden wanneer er wordt uitgegaan van een malicious in plaats van een semi-honest model.

Bibliografie

[1] Usama Fayyad, Gregory Shapiro, and Padhraic Smyth. From data mining to knowledge discovery in databases. *AI Magazine*, 1996.

[2] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to Data Mining*. Addison-Wesley, 2005.

[3] Tom Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[4] Kirsten Wahlstrom and John Roddick. On the impact of knowledge discovery and data mining. *Selected Papers from the Second Australian Institute of Computer Ethics Conference*, 2001.

[5] Csilla Farkas and Sushil Jajodia. The inference problem: A survey. *SIGKDD Explorations*, 2002.

[6] Latanya Sweeney. *Computational Disclosure Control: A Primer on Data Privacy Protection*. PhD thesis, Massachusetts Institute of Technologie, 2001.

[7] Claus Boyens, Oliver Gunther, and Maximilian Teltzrow. Privacy conflicts in crm services for online shops: A case study. *Proceedings of the IEEE ICDM Workshop on Privacy, Security and Data Mining*, 2002.

[8] Workshop on privacy, security, and data mining - how do we mine data when we aren't allowed to see it? `http://www.cs.purdue.edu/homes/clifton/icdm02/psdm.html`.

[9] Mike Atallah, Elise Bertino, Ahmed Elmagarmid, Mohames Ibrahim, and Vassilios Verykios. Disclosure limitation of sensitive rules. *Proceedings of the IEEE Knowledge and Data Engineering Workshop*, 1999.

[10] Elena Dasseni, Vassilios Verykios, Ahmed Elmagarmid, and Elisa Bertino. Hiding association rules by using confidence and support. *Proceedings of the 4th Information Hiding Workshop*, 2001.

[11] Yucel Saygin, Vassilios Verykios, and Chris Clifton. Using unknows to prevent discovery of association rules. *SIGMOD*, 2001.

[12] Yucel Saygin, Vassilios Verykios, and Ahmed Elmagarmid. Privacy preserving association rule mining. *Proceedings of the 12th International Workshop on Research Issues in Data Engineering*, 2002.

[13] Rakesh Agrawal and Ramakrishnan Srikant. Privacy preserving data mining. *Proceedings of the ACM SIGMOD Conference on Management of Data*, 2000.

[14] Dakshi Agrawal and Charu Aggarwal. On the design and quantification of privacy preserving data mining algoritms. *Proceedings of the 20th ACM Symposium on Principles of Database Systems*, 2001.

[15] Alexandre Evfimievski, Ramakrishnan Srikant, Rakesh Agrawal, and Johannes Gehrke. Privacy preserving mining of association rules. *Proceedings of the 8th ACM SIGKDDD International Conference on Knowledge Discovery and Data Mining*, 2002.

[16] Shariq Rizvi and Jayant Haritsa. Maintaining data privacy in association rule mining. *Proceedings of the 28th International Conference on Very Large Databases*, 2002.

[17] Nationale privacy commissie. `http://www.privacy.fgov.be/`.

[18] Deadly combination. `http://www.sptimes.com/News/webspecials/firestone/`.

[19] Inside the ford/firestone fight. `http://www.time.com/time/business/article/0,8599,128198,00.html`.

[20] Chris Clifton. Privacy preserving distributed data mining. 2001.

[21] Chris Clifton, Wenliang Du, and Mikhail Atallah. Distributed data mining to protect information privacy. 2003.

[22] Chris Clifton, Murat Kantarcioglu, Jaideep Vaidya, Xiaodong Lin, and Michael Zhu. Tools for privacy preserving distributed data mining. *SIGKDD Explorations*, 2002.

[23] Vassilios Verykios, Elisa Bertino, Igor Fovino, Loredana Provenza, Yucel Saygin, and Yannis Theodoridis. State of the art in privacy preserving data mining. 2004.

[24] Andrew Yao. How to generate and exchange secrets. *Proceedings of the 27th IEEE Symposium on Foundations of Computer Science*, 1986.

[25] Oded Goldreich, Silvio Micali, and Avi Widgerson. How to play any mental game. *Proceedings of the 19th Annual ACM Symposium on the Theory of Computing*, 1987.

[26] Oded Goldreich. Secure multiparty computation. 1998.

[27] Yehuda Lindell and Benny Pinkas. A proof of Yao's protocol for secure two-party computation. 2004.

[28] Kun Liu. Basic introduction to secure multiparty computation.

[29] Ronald Cramer. Introduction to secure computation. *Lecture Notes In Computer Science*, 1999.

[30] Benny Pinkas. Cryptographic techniques for privacy preserving data mining. *SIGKDD Explorations*, 2002.

[31] Alfred Menezes, Paul van Oorschot, and Scott Vanstone. *Handbook of applied cryptography*. CRC Press, 1996.

[32] Andrew Tanenbaum. *Computer netwerken*. Prentice-Hall, 2003.

[33] Stephen Pohlig and Martin Hellman. An improved algorithm for computing logarithms over GF (p) and its cryptographic significance. *IEEE Transactions on Information Theory*, 1978.

[34] Pascal Paillier. Public key cryptosystems based on composite degree residuosity classes. *Advances in Cryptology*, 1999.

[35] David Naccache and Jacques Stern. A new public key cryptosystem based on higher residues. *The 5th ACM Conference on Computer and Communications Security*, 1998.

[36] Ivan Damgard and Mads Jurik. A generalisation, a simplification and some applications of paillier's probabilistic public-key system. *Public Key Cryptography 2001*, 2001.

[37] Yan-Cheng Chang and Chi-Jen Lu. Oblivious polynomial evaluation and oblivious neural learning. *Proceedings of the 7th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology*, 2001.

[38] Moni Naor and Benny Pinkas. Efficient oblivious transfer protocols. *Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms*, 2001.

[39] Moni Naor, Benny Pinkas, and Reuben Sumner. Privacy preserving auctions and mechanism design. *Proceedings of the 1st ACM Conference on Electronic Commerce*, 1999.

[40] David Cheung, Jiawei Han, Vincent Ng, Ada Fu, and Yongjian Fu. A fast distributed algorithm for mining association rules. *Proceedings of the 4th International Conference on Parallel and Distributed Information Systems*, 1996.

[41] Wenliang Du and Mikhail Atallah. Privacy preserving statistical analysis. *Proceedings of the 17th Annual Computer Security Applications Conference*, 2001.

[42] Murat Kantarcioglu and Chris Clifton. Privacy preserving distributed mining of association rules on horizontally partitioned data. *ACM SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery*, 2002.

[43] Jaideep Vaidya and Chris Clifton. Privacy preserving association rule mining in vertically partitioned data. *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2002.

[44] Yehude Lindell and Benny Pinkas. Privacy preserving data mining. *Lecture Notes in Computer Science*, 2000.

[45] Wenliang Du and Zhijun Zhan. Building decision tree classifier on private data. *Proceedings of the IEEE International Conference on Privacy, Security and Data Mining*, 2002.

[46] Jaideep Vaidya and Murat Kantarcioglu. Privacy preserving naive bayes classifier for horizontally partitioned data. *The 3th IEEE International Conference on Data Mining*, 2003.

[47] Jaideep Vaidya and Chris Clifton. Privacy preserving naive bayes classifier for vertically partitioned data. *The 2004 SIAM International Conference on Data Mining*, 2003.

[48] Jaideep Vaidya and Chris Clifton. Privacy preserving k-means clustering over vertically partitioned data. *The 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2003.

[49] Jaideep Vaidya. *Privacy Preserving Data Mining over Vertically Partitioned Data*. PhD thesis, Purdue University, 2004.

[50] Bart Goethals, Sven Laur, Helger Lipmaa, and Taneli Mielikainen. On private scalar product computation for privacy preserving data mining. 2004.