Uniwersytet Warszawski
Wydział Matematyki, Informatyki i Mechaniki

Łukasz Chmielewski
Nr albumu: 189379

Wydajne protokoły obliczeń wielopodmiotowych

Praca magisterska
na kierunku INFORMATYKA
w zakresie OPROGRAMOWANIA I METOD INFORMATYKI

Praca wykonana pod kierunkiem
dra Stefan Dziembowski
Instytut Informatyki

Wrzesień 2005

Oświadczenie kierującego pracą

Oświadczam, że niniejsza praca została przygotowana pod moim kierunkiem i stwierdzam, że spełnia ona warunki do przedstawienia jej w postępowaniu o nadanie tytułu zawodowego.

Data Podpis kierującego pracą

Oświadczenie autora (autorów) pracy

Świadom odpowiedzialności prawnej oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie i nie zawiera treści uzyskanych w sposób niezgodny z obowiązującymi przepisami.

Oświadczam również, że przedstawiona praca nie była wcześniej przedmiotem procedur związanych z uzyskaniem tytułu zawodowego w wyższej uczelni.

Oświadczam ponadto, że niniejsza wersja pracy jest identyczna z załączoną wersją elektroniczną.

Data Podpis autora (autorów) pracy

Streszczenie

Inspiracją dla tej pracy są wydajne protokoły obliczeń wielopodmiotowych dla problemu `set-intersection` (części wspólnej zbiorów) i innych problemów pokrewnych zaprezentowanych w pracy Michael J. Freedmana, Kobbi Nissima i Benny Pinkasa: „Efficient Private Matching and Set Intersection".

Wielopodmiotowe obliczenie funkcji jest wykonywane przez pewien z góry określony zbiór graczy (można rozważać: dwóch lub n uczestników), którzy przy pomocy wzajemnej komunikacji obliczają pewną publicznie znaną funkcję ich wejść. Dla każdego takiego obliczenia można rozważać przeciwnika, który - przez gromadzenie informacji i korumpowanie graczy - stara się je zakłócić lub zgromadzić pewne tajne informacje (np.: poznać wejścia funkcji, które są dostarczane przez uczciwych graczy).

Istnieją protokoły obliczające wielopodmiotowo funkcję, bazujące na ją realizującym układzie arytmetycznym (czyli istnieją protokoły realizujące dowolny układ złożony z bramek dodawana i mnożenia). Takie rozwiązania można stosować do bardzo szerokiego wachlarza funkcji. Głównym problemem takich rozwiązań jest ich stosunkowo duża złożoność (zarówno czasowa, jak i komunikacyjna). W celu uzyskania lepszej wydajności konstruuje się protokoły obliczające wielopodmiotowo konkretne funkcje.

W tej pracy przedstawię podstawowe pojęcia dotyczące obliczeń wielopodmiotowych, a następnie zaprezentuję ostatnio opublikowane protokoły obliczania problemu części wspólnej zbiorów (`set-intersection`) dla dwóch i więcej graczy w modelu z przeciwnikiem pasywnym i aktywnym. Następnie zaprezentuję protokoły obliczeń wielopodmiotowych mojego autorstwa rozwiązujące problem `fuzzy-matching`, który jest zdefiniowany w pracy Freedmana, Nissima i Pinkasa . Jest to problem związany z problemem części wspólnej zbiorów.

Słowa kluczowe

kryptografia, bezpieczne obliczenia wielopodmiotowe, wydajne protokoły kryptograficzne, część wspólna zbiorów, „fuzzy matching"

Dziedzina pracy (kody wg programu Socrates-Erasmus)

11.3 Informatyka

Klasyfikacja tematyczna

94A60

Spis treści

Wprowadzenie 5
Zawartość pracy 6

1. Obliczenia wielopodmiotowe - definicje 7
1.1. Definicja obliczeń wielopodmiotowych 7
1.2. Bezpieczeństwo 8
1.3. Kryptosystem homomorficzny ze względu na dodawanie 10
1.4. Obliczeniowa nierozróżnialność 11
1.5. Funkcje haszujące 12
1.6. Dwuznaczne zobowiązania 13
1.7. Pierścienie wielomianów 14
1.7.1. Algorytmy operujące na zaszyfrowanych wielomianach 14
1.7.2. Wielomiany jako reprezentacja zbioru 14
1.7.3. Maskowanie miejsc zerowych wielomianów 15

2. Protokoły obliczające części wspólne zbiorów 17
2.1. Protokół części wspólnej zbiorów (`set-intersection`) dla dwóch graczy . . . 17
2.1.1. Definicja problemu 17
2.1.2. Bezpieczeństwo 18
2.1.3. Reprezentacja zbiorów 18
2.1.4. Wersja dla przeciwnika pasywnego 18
2.1.5. Wersja dla przeciwnika aktywnego 23
2.1.6. Wersja dla aktywnego **Server**a 23
2.1.7. Wersja dla aktywnego **Client**a 24
2.1.8. Obsługa aktywnego **Client**a i **Server**a 24
2.1.9. Podsumowanie 25
2.2. Protokół części wspólnej zbiorów (`set-intersection`) dla n graczy 26
2.2.1. Definicja problemu 26
2.2.2. Wersja dla przeciwnika pasywnego 26
2.2.3. Wersja dla przeciwnika aktywnego 27
2.2.4. Podsumowanie 29

3. Protokoły obliczające część wspólną z błędami – „fuzzy matching” . . . 35
3.1. Definicja problemu 35
3.2. Protokół - praca oryginalna 36
3.2.1. Problem 36
3.2.2. Podsumowanie 38
3.3. Protokoły `fuzzy-matching` mojego autorstwa 38

3.3.1. Protokół `fuzzy-matching` – rozwiązanie bazujące na układzie arytmetycznym 38
3.3.2. Poprawiony protokół `fuzzy-matching` 38
3.3.3. Alternatywny protokół `fuzzy-matching` 42

4. Podsumowanie 51

Bibliografia 53

Wprowadzenie

W tej pracy zaprezentuję problem wydajnych protokołów obliczeń wielopodmiotowych na przykładzie rozwiązań problemu `set-intersection` zaczerpniętych z prac: [FNP04] i [KST05] (protokoły z tej pracy są analogiczne z protokołami z pracy [LKTIP]; jednak w pracy [LKTIP] jest zastosowane nieco inne podejście do prezentacji tych protokołów). Są to protokoły rozwiązujące problem bezpiecznego wielopodmiotowego obliczenia części wspólnej zbiorów (a często również multi-zbiorów). W pracach tych są zaprezentowane protokoły dla 2 i n graczy (są od siebie istotnie różne). Zaprezentuję protokoły bezpieczne w różnych modelach przeciwnika, pasywnym i aktywnym (przy każdym zaprezentowanym protokole będzie zdefiniowane w jakim modelu jest on bezpieczny).

Te protokoły obliczeń wielopodmiotowych (ukierunkowane na rozwiązanie jednego problemu: `set-intersection`, lub innego pokrewnego problemu – np.: obliczenia wielkości części wspólnej zbiorów - `cardinality-set-intersection`) mają praktyczne zastosowania ze względu na swoją wydajność. Mają zwłaszcza szczególne zastosowania dla problemów dzielenia poufnych i prywatnych informacji, które np.: pojawiają się w medycznych bazach danych, internetowym wyszukiwaniu randek, rozproszonym monitorowaniu sieci.
Podam kilka przykładów:

- w przypadku klienta i serwera (2 graczy):
  klient posiada zbiór plików (np.: filmów) ściągniętych z sieci na swoim komputerze i chce się dowiedzieć jakie pliki z tego zbioru dzieli z jakąś bazą danych (np.: filmową bazą danych). Jednak istotne jest wymaganie bezpieczeństwa by baza nie poznała prywatnych danych klienta – plików (klient może posiadać np.: dziwne filmy), a klient poznał tylko współdzielone z bazą pliki. Porównywanie plików może być realizowane przez porównywanie nazw lub / i CRC (suma kontrolna pliku), a w ostateczności dla krótkich plików przez treść plików. Taka sytuacja może zaistnieć gdy klient prosi o nazwy współdzielonych plików, a nie ma pełnego zaufania do bazy danych oraz nie opłacił jeszcze pełnego dostępu;

- w przypadku wielu graczy:

  – grupa ludzi przed obejrzeniem / ściągnięciem / kupnem filmu (do np.: sieci lokalnej) chce ustalić jakie filmy ściągnąć; w tej sytuacji protokół dla problemu `set-intersection` pozwala ludziom o dziwnym guście uniknąć zmieszania, gdy ujawniają swoje preferencje (jeżeli wszyscy mają dziwny gust to film zostaje wybrany);

  – protokół `set-intersection` mógłby być używany przez apteki do znajdowania przestępców realizujących kilka razy te same recepty (uczciwi klienci aptek nie byliby narażeni na upowszechnienie informacji o ich chorobach);

  – na lotniskach w USA podczas kontroli pasażerowie samolotów są sprawdzani czy są na liście terrorystów; protokół `set-intersection` może być użyty do stwierdzenia

czy na pokładzie samolotu znajduje się terrorysta (tożsamość zwykłych pasażerów nie jest narażona na ujawnienie);

– modyfikacja protokołu `set-intersection` (protokół `over-threshold-set-intersection` opisany w [KST05]) może być użyta do przeprowadzenia sondażu na popularnego artystę muzycznego (nietypowe gusta uczestników nie zostają ujawnione);

– sieci lokalne grupy wielu organizacji są atakowane przez hakerów powodując poważne straty (alternatywnie: są „zasypywane" spamem); można rozwiązać ten problem stosując modyfikację protokołu `set-intersection` (protokół `threshold-set-intersection` opisany w [KST05]);

Zawartość pracy

W kolejnych rozdziałach tej pracy omówię:

Obliczenia wielopodmiotowe - definicje - w tym rozdziale zawarte są główne definicje (lub odnośniki do nich) związane z obliczeniami wielopodmiotowymi, które będą używane w tej pracy; w tym rozdziale znajdą się definicje:

- obliczeń wielopodmiotowych
- użytych modeli bezpieczeństwa
- homomorficznego szyfru (ze względu na operację dodawania)
- funkcji haszujących
- dwuznacznych zobowiązań bitowych

Zaprezentowane są również sposoby dowodzenia (w wybranych modelach) użyte do dowodzenia bezpieczeństwa i poprawności protokołów zaprezentowanych w tej pracy.

W tym rozdziale nie są podane definicje problemów rozwiązywanych w protokołach (te definicje są podawane przy opisie działania konkretnego protokołu).

Protokoły obliczające części wspólne zbiorów - w tym rozdziale zaprezentowane są protokoły rozwiązujące problem części wspólnej zbiorów; będą opisane protokoły dla 2 i n graczy w modelu aktywnym i pasywnym (wraz ze szkicami lub odnośnikami do dowodów poprawności i bezpieczeństwa); te protokoły pochodzą z prac: [FNP04] i [KST05];

Protokoły obliczające część wspólną z błędami – „fuzzy matching" - w tym rozdziale zaprezentuję i zdefiniuję protokół dla problemu "fuzzy matching"; następnie zaprezentuję protokół podany w pracy [FNP04], oraz moje rozważania na jego temat; na koniec podam moje protokoły rozwiązujące ten problem wraz z odpowiednimi dowodami;

Podsumowanie – jest to rozdział podsumowujący tą pracę;

Rozdział 1

Obliczenia wielopodmiotowe - definicje

W tym rozdziale są przedstawione główne definicje, modele i techniki dotyczące liczenia wielopodmiotowego funkcji, użyte w tej pracy.

1.1. Definicja obliczeń wielopodmiotowych

Przez wielopodmiotowe obliczenie funkcji (często zwane protokołem) przez n graczy rozumiem obliczenie pewnej z góry zadanej funkcji (definicja oparta na [RA99] i [DK04]):

$$f(\mathrm{N} \times D^n \times \{0,1\}^*) \longrightarrow C^n$$

gdzie:

- pierwszy argument funkcji to parametr bezpieczeństwa;
- ostatni argument jest losowy (ograniczony wielomianowo przez parametr bezpieczeństwa);
- D i C to ustalone wcześniej języki nad alfabetem $\{0,1\}^*$;

Wejścia i wyjścia funkcji są ograniczone wielomianowo ze względu na parametr bezpieczeństwa. Każde D na liście argumentów oznacza wejście jakiegoś gracza, a odpowiednie C wyjście danego gracza. Interesujące są funkcje obliczalne w wielomianowym czasie ze względu na parametr bezpieczeństwa.

Na początku protokołu każdy z n graczy (modelowanych za pomocą probabilistycznej maszyny Turinga) dostaje swoje wejście, prywatną losowość i parametr bezpieczeństwa. Następnie dochodzi do komunikacji (ilość przesłanych bitów jest wielomianowo ograniczona ze względu na parametr bezpieczeństwa). Zakładam, że między każdą parą graczy istnieje kanał komunikacyjny i cała wymiana informacji jest bezpieczna i w pełni autentykowana. Po zakończeniu komunikacji wyjściem każdego gracza jest obliczony przez niego wynik.

Do tego modelu należy dodać przeciwnika (modelowanego za pomocą probabilistycznej maszyny Turinga), który stara się uzyskać dane prywatne graczy lub mieć wpływ na wynik funkcji (w tym celu trzeba by rozszerzyć definicję). Może on w zależności od modelu korumpować w różny sposób graczy (użyte modele te są opisane w jednym z kolejnych podrozdziałów). Celem dobrego protokołu jest danie przeciwnikowi zaniedbywalnej szansy sukcesu (dowiedzenia się jakiś prywatnych informacji lub zakłócenia wyliczenia wartości funkcji). Dodanie jednego scentralizowanego przeciwnika (który potrafi korumpować wielu graczy) do

modelu jest wygodniejsze niż dodawanie wielu nieuczciwych graczy i rozpatrywanie wielu przeciwników. Intuicyjnie jeden przeciwnik, który ma dostęp do danych wszystkich oszustów może najwięcej wywnioskować. Celem jest skonstruowanie protokołu, w którym ten najsilniejszy przeciwnik nie może osiągnąć nielegalnych korzyści.

Często tę grupę – koalicję oszukujących graczy będę utożsamiał ze scentralizowanym przeciwnikiem korumpującym daną grupę (ta konwencja jest szeroko używana w literaturze).

Proste przykłady obliczeń wielopodmiotowych funkcji

W tej sekcji przedstawię proste i dobrze znane przykłady obliczeń wielopodmiotowych:

- Alicja i Bob żywią do siebie uczucia (miłość lub obojętność) i chcieliby sprawdzić unikając upokorzenia czy oboje siebie kochają. Jeśli oboje siebie kochają to oboje dowiadują się tego faktu, a w przeciwnym przypadku: nie kochający nie dowiaduje się niczego, a kochający dowiaduje się, że partner go nie kocha. Zapisując miłość = 0, a obojętność = 1 funkcję (wejście Alicji i Boba to $a, b \in \{0, 1\}$), która powinna być wykonana przez protokół, można zapisać następująco:

$$f(a, b) = ((a \cap b), (a \cap b))$$

- problem milionerów (zaproponowany przez Yao, przytaczam za [RI04]):
dwóch milionerów chce się dowiedzieć, który z nich jest bogatszy, ale nie chcą ujawniać stanu swoich kont. Zakładając, że danymi wejściowymi każdego milionera ($m_1, m_2 \in D$) jest stan jego konta, to funkcja, którą chcą obliczyć można zapisać następująco:

$$f(m_1, m_2) = ((m_1 > m_2), (m_2 > m_1))$$

W zapisach argumentów powyższych funkcji umieściłem tylko wejścia graczy (czyli tak naprawdę podaje funkcjonalności, które mają być realizowane przez protokoły).

1.2. Bezpieczeństwo

W tym rozdziale podam definicję bezpieczeństwa. Zaprezentuję ją w sposób w jaki jest zaprezentowana w pracy [KSC05].

Bezpieczeństwo można rozpatrywać względem różnych definicji przeciwnika i w różnych modelach. Protokoły, które zaprezentuję w tej pracy są rozpatrywane w modelu obliczeniowym. Główne założenia tego modelu to: istnienie jednokierunkowych permutacji oraz modelowanie przeciwnika przez wielomianową zrandomizowaną maszynę Turinga. Protokoły będę rozpatrywał względem następujących przeciwników (definicje są sformalizowane w [GF02]):

pasywny (uczciwy-ale-ciekawy) Jest to typ przeciwnika, który musi postępować zgodnie z protokołem. To znaczy, że skorumpowani przez niego gracze realizują protokół, ale przeciwnik ma dostęp do wszystkich danych, którymi oni dysponują (danych wejściowych, otrzymanych wiadomości i danych wyjściowych).

W tym modelu bezpieczeństwo jest proste do zdefiniowania: żaden gracz lub koalicja graczy (skorumpowanych) nie potrafi uzyskać informacji, które nie wynikają w naturalny sposób z funkcjonowania protokołu. Formalnie definiuje się idealny model (świat), w którym istnieje zaufana strona trzecia (ang.: *Trusted Third Party*: **TTP**). **TTP** pobiera od wszystkich graczy ich wejścia, następnie liczy zadaną funkcję i zwraca

odpowiedni wynik każdemu graczowi. Wymaganie bezpieczeństwa jest takie, że przeciwnik uczy się w rzeczywistej sytuacji tyle samo co w tej samej sytuacji w idealnym świecie. Zatem przeciwnik (gracz lub koalicja graczy) potrafi rozróżnić: czy znajduje się w rzeczywistym świecie, czy w idealnym z zaniedbywalnym prawdopodobieństwem (definicja tego pojęcia jest umieszczona później w tym rozdziale).

Podam przykład dla funkcji liczącej `set-intersection` (dla dwóch graczy: **A** i **B** z wejściami: a i b, gdzie a i b to zbiory). Funkcja, która ma być realizowana przez protokół wygląda następująco:

$$f(a,b) = (czesc\_wspolna(a,b), czesc\_wspolna(b,a))$$

W tej sytuacji: jeśli element zbioru x nie pojawia się w $czesc\_wspolna(a,b)$, to jeśli x nie pojawia się w a, to **A** nie potrafi stwierdzić, czy x znajduje się w b.

aktywny (złośliwy) Ten rodzaj przeciwnika może zmieniać sposób zachowania skorumpowanych graczy w dowolny sposób, by uzyskać jakieś dodatkowe informacje podczas działania protokołu lub zakłócić poprawność wyliczania funkcji (gracze mogą postępować niezgodnie z protokołem). W szczególności skorumpowani gracze mogą wybierać dowolne dane wejściowe i przerwać swój udział w protokole w dowolnym momencie.

Dobrze znanym faktem jest, że jeśli co najmniej połowa graczy jest skorumpowana to przerywając uczestnictwo w protokole potrafią oni zatrzymać protokół (czyli taka sytuacja występuje w protokołach z dwoma uczestnikami – a takie protokoły również omawiam w tej pracy).

W przypadku tego przeciwnika definicja bezpieczeństwa również polega na porównywaniu sytuacji rzeczywistej z sytuacją idealną. Jeśli protokół jest bezpieczny, wtedy można skonstruować symulator, który tłumaczy każdą strategię przeciwnika z rzeczywistego modelu do modelu idealnego w ten sposób, że przeciwnik uzyskuje obliczeniowo nierozróżnialną informacje w obu scenariuszach (to sformułowanie będzie objaśnione później) – czyli przeciwnik nie potrafi zdecydować w jakim modelu się znajduje. Symulator (reprezentujący uczciwych graczy – lub ich grupę) działa w modelu idealnym w następujący sposób (zapewniając by przeciwnik nie domyślił się, że znajduje się w modelu idealnym):

1. Komunikuje się z nieuczciwymi graczami (udając, że jest jednym z uczciwych graczy lub grupą uczciwych graczy, w ten sposób by przeciwnik nie mógł się domyślić, że znajduje się w idealnym modelu). W tym kroku symulator uzyskuje wejścia skorumpowanych graczy.
2. Przesyła do **TTP** wejścia graczy, których reprezentuje, oraz wejścia odpowiadające skorumpowanym graczom.
3. Po otrzymaniu odpowiedzi od **TTP** symulator przesyła wynik do skorumpowanych graczy (tak, by się nie zorientowali w jakim modelu się znajdują) oraz do uczciwych graczy.

Symulator podczas swojego działania może korzystać ze „specjalnych" właściwości symulatora (np.: może udawać losową wyrocznię w modelu z losową wyrocznią).

Istotne przy definiowaniu bezpieczeństwa dla przeciwnika aktywnego jest określenie co taki przeciwnik może zawsze zrobić (bez względu na bezpieczeństwo protokołu). Przeciwnik aktywny może:

- zmienić w dowolny dopuszczalny sposób swoje dane wejściowe (zanim zostaną one użyte podczas działania protokołu);
- przerwać uczestnictwo w protokole w dowolnym momencie wykonania protokołu;

Dodatkowe założenie dla protokołów z n graczami: przeciwnik może skorumpować nie więcej niż c ($c < n$) graczy (gdzie c jest z góry dane). Taki model jest nazywany modelem progowym.

Przypadek dla 2 graczy jest nieco inny, zatem bezpieczeństwo będzie nieco inaczej dowodzone (ze względu na specyfikę graczy: **Client**a i **Server**a). Szczegóły zostaną podane przy protokołach.

W tej pracy będę głównie rozważał protokoły będące bezpieczne w modelu pasywnym.

1.3. Kryptosystem homomorficzny ze względu na dodawanie

W tym rozdziale podam definicję homomorficznego ze względu na dodawanie semantycznie bezpiecznego kryptosystemu. Zaprezentuję ją w sposób jaki jest zaprezentowany w pracy [KSC05].

W tej pracy zostanie użyty semantycznie bezpieczny (opisane w [SGSM84]), homomorficzny ze względu na dodawanie kryptosystem klucza publicznego. Niech $E_{pk}(.)$ oznacza szyfrogram zaszyfrowany kluczem publicznym pk (ang.: *public key*). Odszyfrować wiadomość c można jedynie używając klucza prywatnego sk (ang.: *secret key*) w następujący sposób: $D_{sk}(c)$.

Semantyczne bezpieczeństwo nieformalnie oznacza (jest to zdefiniowane w [IDC04]), że szyfrogram nie „mówi" nic o zaszyfrowanej wiadomości. Rozważa się następującą sytuację: podsłuchujący przeciwnik wybiera dowolną wiadomość m i wysyła ją do wyroczni (która dysponuje kluczem prywatnym). Wyrocznia z prawdopodobieństwem $\frac{1}{2}$ odpowiada przeciwnikowi wysyłając $E_{pk}(m)$, a z prawdopodobieństwem $\frac{1}{2}$ wysyła $E_{pk}(losowa\_wiadomosc)$. Szyfr jest nazywamy semantycznie bezpiecznym jeśli przeciwnik nie jest w stanie rozróżnić czy od wyroczni dostał szyfrogram losowej wiadomości, czy szyfrogram wiadomości m.

Homomorficzny ze względu na dodawanie kryptosystem dostarcza dodatkowo następujące operacje, które można wykonać wydajnie bez znajomości klucza publicznego:

- mając dane $E_{pk}(a)$ i $E_{pk}(b)$ (dla dowolnego a i b należących do dziedziny tekstów jawnych) można policzyć:
  $E_{pk}(a + b) = E_{pk}(a) +_h E_{pk}(b)$
- mając dane $E_{pk}(a)$ i c (dla dowolnego a i c należących do dziedziny tekstów jawnych) można policzyć:
  $E_{pk}(c \cdot a) = c \times_h E_{pk}(a)$

Po każdej takiej operacji nowo wyprodukowany szyfrogram musi być zre-randomizowany ze względu na bezpieczeństwo. Re-randomizacja oznacza transformację szyrogramu w inny szyfrogram tego samego tekstu jawnego w ten sposób, że trudne jest odkrycie, że jest to szyfrogram tego samego tekstu jawnego.

Własność re-randomizacji jest istotna ze względu na następującą niebezpieczną sytuację: przeciwnik nie powinien być w stanie rozpoznać, że ta sama wiadomość jest wysyłana kilka razy. Ma to duże znaczenie gdy wysłane są wielokrotnie pojedyncze bity (z użyciem tych samych kluczy: prywatnego i publicznego). Jeśli przeciwnik wie, że jedna wiadomość jest częściej wysyłana niż inna to może w łatwy sposób dociec kiedy ona jest wysyłana.

W tej konkretnej sytuacji jeśli przeciwnik ma wiadomości $E_{pk}(a)$ i $E_{pk}(b)$ to może policzyć $E_{pk}(a+b)$. Gdyby nie było re-randomizacji i jeśli $E_{pk}(a+b) = E_{pk}(a)$ to przeciwnik by wiedział, że $b = 0$. Zatem re-randomizacja jest potrzebna do zapewnienia bezpieczeństwa.

Wyżej wymienione własności są istotne dla rozważanych tu protokołów.

Poniżej podana własność jest przede wszystkim istotna dla sytuacji, gdy w protokole uczestniczy n graczy. Jest wymagane by kryptosystem umożliwiał bezpieczne (n,t)–progowe odszyfrowywanie. W tej sytuacji każdy z graczy dysponuje sekretnym kluczem prywatnym (ssk_i - różnym dla każdego gracza), podczas gdy klucz publiczny spk (ang.: *private key*) jest znany wszystkim stronom. Szyfrowanie wiadomości m przebiega w sposób „standartowy" (taki sam jak dla kryptosystemu z jednym kluczem prywatnym): $E_{spk}(m)$. Deszyfrowanie jest wykonywane w grupie przynajmniej t graczy posiadających dzielone klucze prywatne (ssk_i dla uczestnika o indeksie i). Jeśli graczy jest mniej niż t lub któryś z kluczy jest niepoprawny to proces odszyfrowania nie może być poprawnie wykonany (gracze nie dowiadują się niczego o odszyfrowanej wiadomości).

W prezentowanych protokołach zakładam, że na początku protokołu każdy uczestnik i posiada dzielony klucz prywatny ssk_i (i żaden inny gracz nie zna tego klucza – sekretu). Robię to dla prostoty zapisu i bez straty ogólności (gdyż istnieją wyspecjalizowane schematy wykonujące podział sekretu – dystrybucję dzielonych kluczy prywatnych; szczegóły znajdują się w [RC99]).

Do zabezpieczenia protokołu w modelu ze złośliwym graczem wymagana jest także od szyfru możliwość wykonania następujących dowodów z wiedzą zerową:

- gracz potrafi dowieść, że postępuje zgodnie z protokołem (n,t)–progowego odszyfrowania (użył prawidłowego klucza prywatnego);
- gracz potrafi dowieść, że zna tekst jawny, zaszyfrowanej wiadomości;
- gracz potrafi dowieść poprawności pewnych operacji na zaszyfrowanych wiadomościach;

Szczegóły są podane przy protokole używającym tych dowodów.

Definicje i przykłady dowodów z wiedzą zerową są podane w książce [CTP].

Szyfr Pailliera posiada podane właściwości ([PP00]). Podobne właściwości posiada również szyfr ElGamala.

1.4. Obliczeniowa nierozróżnialność

W tym podrozdziale podam formalną definicję obliczeniowej nierozróżnialności (podaję ją za [DK04] i [RA99]). To wyrażenie (obliczeniowa nirozróżnialność) jest podstawowym pojęciem używanym przy dowodzeniu bezpieczeństwa i poprawności protokołów w modelu z przeciwnikiem z ograniczoną wielomianowo mocą obliczeniową.

Definicja 1 *Zespołem (dyskretnych i skończonych) rozkładów prawdopodobieństwa nazywamy dowolny indeksowany zbiór rozkładów prawdopodobieństwa $\{X_{k,\alpha}\}_{k\in\mathrm{N},\alpha\in S}$ niezerowych jedynie na zbiorze skończonym, gdzie S jest pewną dziedziną (zwykle $S = \{0,1\}^*$).*

Definicja 2 *Mówimy, że zespoły rozkładów prawdopodobieństwa $\{X_{k,\alpha}\}_{k\in\mathrm{N},\alpha\in S}$ i $\{Y_{k,\alpha}\}_{k\in\mathrm{N},\alpha\in S}$ są równo rozłożone i piszemy $X =_d Y$, jeśli:*

$$\forall_{k\in\mathrm{N},\alpha\in S} X_{k,\alpha} = Y_{k,\alpha}$$

Definicja 3 *Funkcję* $\delta : \mathrm{N} \longrightarrow \mathrm{R}$ *nazywamy zaniedbywalną, jeśli dla każdego wielomianu* p *o wartościach w* R_+*:*

$$\exists_{N\in\mathrm{N}}\forall_{\mathrm{N}\ni n<N}|\delta(n)| < \frac{1}{p(n)}$$

Definicja 4 *Funkcję* $\delta : \mathrm{N} \longrightarrow \mathrm{R}$ *nazywamy ograniczeniem dystansu obliczeniowego zespołów rozkładów prawdopodobieństw* $\{X_{k,\alpha}\}_{k\in\mathrm{N},\alpha\in S}$ *i* $\{Y_{k,\alpha}\}_{k\in\mathrm{N},\alpha\in S}$ *jeśli dla dowolnego probabilistycznego algorytmu* D *o wielomianowej złożoności ze względu na pierwszy parametr:*

$$\exists_{K\in\mathrm{N}}\forall_{\mathrm{N}\ni k<K}\forall_{\alpha\in S}|P(D(1^k,\alpha,x)=1) - P(D(1^k,\alpha,y)=1)| < \delta(k)$$

gdzie x *i* y *są wzięte zgodnie z rozkładami* $X_{k,\alpha}$ *i* $Y_{k,\alpha}$*, a prawdopodobieństwo jest liczone po wyborach* x *i* y *oraz losowych wyborach algorytmu* D*.*

Definicja 5 *Mówimy, że zespoły rozkładów prawdopodobieństwa* $\{X_{k,\alpha}\}_{k\in\mathrm{N},\alpha\in S}$ *i* $\{Y_{k,\alpha}\}_{k\in\mathrm{N},\alpha\in S}$ *są obliczeniowo nierozróżnialne i piszemy* $X =_c Y$*, jeśli istnieje dla nich ograniczenie dystansu obliczeniowego będące funkcją zaniedbywalną* k*.*

Intuicja jest taka, że dwa rozkłady są obliczeniowo nierozróżnialne jeśli są rozpoznawalne przez wielomianowy algorytm z zaniedbywalnym prawdopodobieństwem. W dowodach poprawności protokołów zazwyczaj porównuje się dwie sytuacje: rzeczywistą (normalne wykonanie protokołu) i idealną (świat, w którym istnieje zaufana strona trzecia **TTP**, która wykonuje wszystkie obliczenia) i dowodzi, że te sytuacje są obliczeniowo nierozróżnialne. Nieformalnie znaczy to, że błąd protokołu lub „wyciek" tajnych informacji zachodzi z zaniedbywalnym prawdopodobieństwem ze względu na parametr bezpieczeństwa. Zaniedbywalne prawdopodobieństwo zazwyczaj oznacza prawdopodobieństwo wykładniczo małe ze względu na parametr bezpieczeństwa (a w tej pracy używam zawsze wykładniczo małego prawdopodobieństwa).

Definiuje również pojęcie przeważającego prawdopodobieństwa. Jakaś sytuacja zachodzi z przeważającym prawdopodobieństwem, jeśli prawdopodobieństwo, że ta sytuacja nie zachodzi jest zaniedbywalne.

1.5. Funkcje haszujące

W tej sekcji krótko omówię funkcje haszujące (więcej o funkcjach haszujących jest napisane w [HA] i [CTP]).

Funkcja haszująca to deterministyczna, wydajna i łatwa do policzenia funkcja:

$$h : \{0,1\}^* \to \{0,1\}^k$$

Funkcja haszująca jest zdefiniowana przez generator H, który dla danej wartości k zwraca opis funkcji h (opis – czyli szybki sposób liczenia wartości funkcji dla dowolnego argumentu).

Przykładowym zastosowaniem funkcji haszującej jest następująca sytuacja: wiadomość m ma być trzymana w niebezpiecznej lokacji i chcę móc sprawdzić czy wartość została zmieniona. Mogę to osiągnąć licząc $f = h(m)$ (zakładając, że mam k bitów bezpiecznej pamięci). Po odzyskaniu wiadomości m' można zweryfikować czy wiadomość była zmieniona przez sprawdzenie czy: $f = h(m')$.

Jak zdefiniować zatem bezpieczeństwo takiej funkcji? Najsilniejszym realistycznym założeniem jest to, że funkcja jest odporna na kolizje (ang.: *collision–resistant*). To znaczy,

że wielomianowo działający przeciwnik z zaniedbywalnym prawdopodobieństwem potrafi znaleźć takie dwie wiadomości m i m', że $h(m) = h(m')$ (ten fakt pociąga za sobą, że przeciwnik nie potrafi odwrócić funkcji, czyli mając daną wartość f nie potrafi znaleźć takiego x, że $h(x) = f$). Jest możliwe praktyczne konstruowanie takich funkcji (dla których nie jest znany przeciwnik, który potrafi znaleźć kolizję).

Praktyczne ograniczenie tego mechanizmu jest wyznaczane przez paradoks dnia urodzin (ang.: *birthday paradox*). Ten paradoks pokazuje, że wśród $\sqrt{2^k}$ losowych wartości z dużym prawdopodobieństwem są przynajmniej dwa identyczne elementy. Oznacza to, wśród $\sqrt{2^k}$ wartości funkcji haszującej z dużym prawdopodobieństwem są dwie równe wartości (zatem została znaleziona kolizja).

Ze względu na fakt, że funkcja haszująca jest deterministyczna (zatem teoretyczny przeciwnik mógłby dysponować z góry policzonymi wszystkimi wartościami funkcji) w dowodach bezpieczeństwa modeluje się funkcję haszującą za pomocą losowej wyroczni. Losowa wyrocznia jest tworem, któremu można zadać pytanie o wiadomość m, a on odpowiada losową wartością ze zbioru $\{0,1\}^k$. Przy czym istotne jest, że dla wielu pytań o tą samą wiadomość m wyrocznia zawsze odpowiada tą samą wartością (czyli losowanie odbywa się tylko przy pierwszym zapytaniu o jakąś wartość). Losowa wyrocznia zachowuje się jak zupełnie losowa funkcja. Model z losową wyrocznią jest „bardzo słaby", ale dowód bezpieczeństwa i poprawności w tym modelu to zawsze jest więcej niż nic. Ten model jest opisany formalnie w [ID04].

Praktycznie używaną funkcją haszującą jest **SHA1** (a jeszcze niedawno była **MD5**).

1.6. Dwuznaczne zobowiązania

W tej sekcji opiszę krótko mechanizm zobowiązań bitowych (ang.: *bit commitment*), a następnie zdefiniuje dwuznaczne zobowiązania (ang.: *equivocal commitment*).

W schemacie zobowiązania bitowego biorą udział dwie strony: Wysyłający S (ang.: *Sender*) i Odbierający R (ang.: *Receiver*). Wysyłający pobiera na wejściu bit $b \in \{0,1\}$ (łatwo to uogólnić do ciągu bitów).

Ten protokół składa się z 2 faz:

Zobowiązanie W tej fazie S wysyła do R ciąg bitów, który nie daje R (prawie) żadnej informacji o b. Ta własność nazywa się *tajnością*.

Nieformalnie można na ten krok protokołu patrzeć następująco: R wkłada b do skrzynki, zamyka ją na klucz i przesyła do R (zachowując klucz dla siebie).

Otwarcie Zobowiązania W wyniku tej fazy R poznaje b', przy czym z przeważającym prawdopodobieństwem zachodzi $b = b'$ (to wymaganie nazywane jest związaniem S z b).

Nieformalnie ten krok można opisać: S wysyła do R klucz do skrzynki.

Zatem standartowy schemat zobowiązania pozwala, każdemu graczowi dawać „zapieczętowaną kopertę", która może być później otworzona z jedną tylko wartością. Specjalną właściwością dwuznacznych zobowiązań jest to, że Symulator (z sekcji 1.2) może „złamać" to zobowiązanie ujawniając dowolną wartość, bez ujawniania się przeciwnikowi.

Mechanizm dwuznacznych zobowiązań jest użyty w przypadku protokołu 2.6 dla przeciwnika aktywnego dla n graczy. Szczegółowy opis dwuznacznych zobowiązań znajduje się w [JKRO04].

1.7. Pierścienie wielomianów

W tym rozdziale zdefiniuję operację na wielomianach i sposób ich użycia w protokołach podanych w tej pracy.

Niech R oznacza dziedzinę tekstów jawnych szyfru homomorficznego klucza publicznego (w szyfrze Pailliera R jest postaci Z_N). Definiuję pierścień wielomianów $R[x]$ jako wielomiany ze współczynnikami z R. Niech $f \in R[x]$, takie że $f(x) = \sum_{i=0}^{deg(f)} f[i] \cdot x^i$, gdzie $f[i]$ oznacza współczynniki przy x^i. Definiuję również zaszyfrowanie wielomianu f jako uporządkowaną listę szyfrogramów współczynników: $E_{pk}(f[0]), ..., E_{pk}(f[deg(f)])$.

1.7.1. Algorytmy operujące na zaszyfrowanych wielomianach

W tej sekcji podam operacje, które można wykonywać na zaszyfrowanych wielomianach.

Niech $f, g, h \in R[x]$, takie że $f(x) = \sum_{i=0}^{deg(f)} f[i] \cdot x^i$, $g(x) = \sum_{i=0}^{deg(g)} g[i] \cdot x^i$ i $h(x) = \sum_{i=0}^{deg(h)} h[i] \cdot x^i$. Można wykonywać wydajnie następujące operacje na szyfrogramach tych wielomianów (zaszyfrowanych szyfrem homomorficznym), bez znajomości klucza prywatnego:

- policzenie zaszyfrowanej wartości zaszyfrowanego wielomianu w punkcie y: $b = f(y)$ w następujący sposób:

$$E_{pk}(b) := (y^0 \times_h E_{pk}(f[0])) +_h (y^1 \times_h E_{pk}(f[1])) +_h ... +_h (y^{deg(f)} \times_h E_{pk}(f[deg(f)]))$$

- policzenie zaszyfrowanej sumy dwóch zaszyfrowanych wielomianów: $h = f + g$ w następujący sposób:

$$E_{pk}(h[i]) := E_{pk}(f[i]) +_h E_{pk}(g[i]) \quad \text{dla} \quad 0 \leq i \leq max\{deg(f), deg(g)\}$$

- policzenie zaszyfrowanego iloczynu zaszyfrowanego wielomianu f i niezaszyfrowanego wielomianu g: $h = f * g$ w następujący sposób:

$$E_{pk}(h[i]) := (g[0] \times_h E_{pk}(f[i])) +_h (g[1] \times_h E_{pk}(f[i-1])) +_h ... +_h (g[i] \times_h E_{pk}(f[0])),$$
$$\text{dla } 0 \leq i \leq deg(f) + deg(g)$$

W dwóch ostatnich przypadkach zakładam, że $f[i] = 0$, dla $i > deg(f)$ i $g[i] = 0$ dla $i > deg(g)$.

1.7.2. Wielomiany jako reprezentacja zbioru

W tej sekcji podam w jaki sposób zbiory mogą być reprezentowane przez wielomiany.

Gracz ma zbiór elementów $S = \{s_1, s_2, ..., s_{k_S}\}$. Każdy element zbioru S należy do jakiejś dziedziny: D. Bez straty ogólności zakładam, że $D = [1...N]$. Niech R oznacza dziedzinę tekstów jawnych homomorficznego kryptosystemu. R musi być większe od D, by losowy element z R ze znikomym prawdopodobieństwem należał do D. Można to uzyskać wymuszając, by element $b \in R$, reprezentujący element $a \in D$ był postaci: $b = c_m(a)||c_k(h(a))$, gdzie $||$ oznacza konkatenacji ciągów bitów, a $c_z(x)$ oznacza zakodowanie x na z bitach, a m jest liczbą bitów na których można zakodować dziedzinę D. Wtedy jeśli przeciwdziedzina $h(.)$ to $\{0,1\}^k$, to prawdopodobieństwo, że losowy element z R należy do D jest $\frac{1}{2^k}$ (czyli zaniedbywalne ze względu parametr k). W dalszej części pracy będę często utożsamiał element x z jego kodowaniem $c_m(x)$ (ze względu na wygodę).

Dla bezpieczeństwa wybieram $R = Z_p$, gdzie p to najmniejsza liczba pierwsza, taka że Z_p zawiera każdy element postaci: $x||h(x)$, dla $x \in D$. p nie jest dużo większe od największego $x||h(x)$ ze względu na twierdzenie o liczbach pierwszych ([RA, str. 168]). Twierdzenie o liczbach pierwszych mówi, że liczby pierwsze „często występują" wśród liczb naturalnych (dla każdej liczby naturalnej n liczb pierwszych mniejszych od n jest $O(\frac{n}{log(n)})$). Zatem $O(|D| \cdot param) = O(|R|)$, gdzie $param$ to parametr bezpieczeństwa.

W tej pracy będę (zazwyczaj) reprezentował zbiór jako wielomian ze zbioru $R[x]$ (jest to wielomian w grupie R), gdzie reprezentacje elementów zbioru S są miejscami zerowymi wielomianu. Załóżmy, że $S_R = \{r_1 = s_1||h(s_1), r_2 = s_2||h(s_2), ..., r_3 = s_{k_S}||h(s_{k_S})\}$. Wtedy reprezentujący zbiór S wielomian f wygląda następująco:

$$f(x) = \Pi_{1 \leq i \leq k_S}(x - r_i)$$

Wielomian będę reprezentował jako uporządkowaną listę współczynników. Zaszyfrowany wielomian będę reprezentował jako uporządkowaną listę zaszyfrowanych współczynników.

1.7.3. Maskowanie miejsc zerowych wielomianów

W tej sekcji podam użyteczne twierdzenia o wielomianach (będę z nich korzystał w niektórych protokołach).

Twierdzenie 1 ***Liczenie części wspólnej za pomocą dodawania wielomianów.***
Gdy dwa wielomiany f i g są dodawane to wspólne miejsca zerowe wielomianów są zachowane: $(f(a) = 0) \cap (g(a) = 0)$ to $(f + g)(a) = 0$.

Twierdzenie 2 ***Liczenie sumy za pomocą dodawania wielomianów.***
Gdy dwa wielomiany f i g są mnożone to miejsca zerowe obu wielomianów są zachowane: $(f(a) = 0) \cap (g(b) = 0)$ to $(f * g)(a) = 0 \cap (f * g)(b) = 0$. Zachodzi również: $(f(a) = 0) \cap (g(a) = 0)$ to $(x - a)^2|(f * g)$

By ukryć miejsca zerowe, które nie są dzielone między wielomianami f i g, można wybrać dwa losowe wielomiany r i s z $R[x]$ posiadające wystarczająco wysoki stopień i policzyć $f * r + g * s$. Wynikowy wielomian ukrywa wszystkie informacje poza wspólnymi miejscami zerowymi f i g. Formalnie zapisane jest to w poniższym twierdzeniu, którego dowód znajduje się w dodatku A w pracy: [LKTIP, str. 30].

Twierdzenie 3 *Niech f i g będą wielomianami należącymi do $R[x]$ (gdzie R jest pierścieniem), $deg(f) = deg(g) = \alpha$ i $gcd(f, g) = 1$. Niech $f(x) = \sum_{i=0}^{\beta} f[i] \cdot x^i$ i $s(x) = \sum_{i=0}^{beta} s[i] \cdot x^i$, gdzie każde $r[i]$ i $s[i]$ (dla $0 \leq i \leq \beta$) są wybierane (niezależnie) losowo z R i $\beta \geq \alpha$.*
Wtedy, jeśli

$$u = f * r + g * s = \sum_{i=0}^{\alpha+\beta} u[i] \cdot x^i$$

to $\forall_{0 \leq i \leq \alpha+\beta} u[i]$ są rozłożone jednostajnie i niezależnie w R.

Rozdział 2

Protokoły obliczające części wspólne zbiorów

W tym rozdziale przedstawię protokoły służące liczeniu części wspólnej zbiorów w bezpieczny sposób przez wielu graczy. Jest to przykład obliczeń wielopodmiotowych (ang.: *multi-party computation*). Przedstawię protokół dla 2 graczy (dwupodmiotowe obliczenia – ang.: *2-party computation*) w modelu z przeciwnikiem aktywnym i z przeciwnikiem pasywnym (protokoły pochodzące z pracy [FNP04]) oraz dla n graczy w modelu z przeciwnikiem pasywnym i aktywnym (protokoły pochodzące z pracy [KST05]).

2.1. Protokół części wspólnej zbiorów (`set-intersection`) dla dwóch graczy

W tym rozdziale przedstawię protokół liczący część wspólną dwóch zbiorów (jest to protokół zaczerpnięty z pracy [FNP04]).

2.1.1. Definicja problemu

Protokół `set-intersection` dla dwóch graczy (opisany w pracy [FNP04], gdzie jest nazywany *private matching (PM)*) jest dwupodmiotowym obliczeniem funkcji liczącej część wspólną dwóch zbiorów. W protokole uczestniczą dwaj uczestnicy: **Client** i **Server**.

Protokół liczy część wspólną dwóch zbiorów:

- zbioru należącego do gracza **Client** (**C**): $X = \{x_1, ..., x_{k_C}\}$, gdzie $x_i \in D$ (dla $1 \leq i \leq k_C$);
- zbioru należącego do gracza **Server** (**S**): $Y = \{y_1, ..., y_{k_S}\}$, gdzie $y_i \in D$ (dla $1 \leq i \leq k_S$);

gdzie D to dziedzina wartości elementów zbioru ($|D| = N$). Dla uproszczenia i bez straty ogólności zakładam, że $D = [1..N]$.

Wyjście protokołu (z przeważającym prawdopodobieństwem) jest zdefiniowane następująco:

- wyjściem **C** jest zbiór $X \cap Y$, czyli $\{x_u | \exists v, x_u = y_v\}$;
- wyjście **S** jest puste (**S** nie poznaje niczego);

2.1.2. Bezpieczeństwo

Bezpieczeństwo w tym modelu definiuje się podobnie jak to zrobiłem w rozdziale 1.2. Różnica polega na specyfice graczy: funkcjonalności obu uczestników znacznie się różnią. Istotne jest również założenie, że podczas wykonywania protokołu tylko jeden gracz może oszukiwać (w przeciwnym przypadku nie ma sensu analizowanie sytuacji, gdyż wtedy przeciwnik może zrobić co chce, skoro wszyscy oszukują).

Dla tego protokołu definiuję dwa rodzaje bezpieczeństwa w modelu pasywnym (w zależności od typu gracza):

- bezpieczeństwo klienta - nierozróżnialność: serwer **S** z przeważającym prawdopodobieństwem nie dowiaduje się niczego o wejściu klienta **C** podczas działania protokołu (nie potrafi rozróżnić sytuacji, w których klient ma różne wejścia);

- bezpieczeństwo serwera - porównanie do modelu idealnego: klient z przeważającym prawdopodobieństwem nie dowiaduje się niczego więcej poza wynikiem funkcji. Formalnie jest to dowodzone za pomocą porównywania sytuacji w modelu rzeczywistym i modelu idealnym - klient z przeważającym prawdopodobieństwem nie potrafi rozróżnić tych sytuacji;

Bezpieczeństwo w modelu aktywnym tego protokołu jest analogiczne do opisanego w sekcji 1.2.

2.1.3. Reprezentacja zbiorów

Zakładam, że zbiory są reprezentowane w ten sposób jak w rozdziale 1.7.2. Czyli utożsamiam każdy element pochodzący ze zbioru X lub Y (należący do D) z jego kodowaniem (należącym do R). Kodowanie to może być wykonywane następująco: $element \in D$ jest kodowany jako $(element||h(element)) \in R$.

2.1.4. Wersja dla przeciwnika pasywnego

Protokół liczenia części wspólnej dwóch zbiorów jest przedstawiony na rysunku: protokół 2.1. Intuicyjnie protokół 2.1 działa, gdyż wielomian przesłany przez **C** w kroku 4 przybiera wartości zerowe tylko dla argumentów ze zbioru X. Zatem dla $y \in X$, **S** w kroku 6 protokołu wysyła do **C**: $E_{pk}(r \cdot f(y)+y) = E_{pk}(y)$ i wtedy **C** potrafi rozpoznać, że **S** ma także element y w swoim zbiorze. W przeciwnym przypadku – jeśli $y \notin Y$ to $r \cdot f(y)+y$ jest kompletnie losowe. Jest tak, gdyż r jest wybierany losowo, a co za tym idzie $r \cdot f(y)$ jest losowe (bo $f(y) \neq 0$ i $R = Z_p$). Skoro $r \cdot f(y)+y$ jest losowe w R to należy ono ze znikomym prawdopodobieństwem do D, a zatem **C** rozpoznaje tą wartość w kroku 7 protokołu (po rozszyfrowaniu) również ze znikomym prawdopodobieństwem. **S** nie dowiaduje się niczego, bo wszystko co otrzymuje od **C** (z wyjątkiem klucza publicznego i parametrów kryptosystemu) jest zaszyfrowane, a szyfr jest semantycznie bezpieczny.

Uwaga do protokołu 2.1 Ten protokół jest również bezpieczny przeciwko aktywnie oszukującemu klientowi. Jedyny sposób w jaki **Client** mógłby oszukiwać to wysłanie wielomianu z więcej pierwiastkami niż k_C. Mógłby to osiągnąć jedynie przez wysłanie wszystkich współczynników zerowych. Jednakże nie może tego zrobić, gdyż współczynnik wielomianu $f[0] = 1$ nie jest przesyłany (tylko odtwarzany po stronie **Server**a).

Protokół 2.1:

PROTOKÓŁ-SI-PASYWNY-wersja1:		
Client:		**Server**:
Wejście **C**: $X = \{x_1, ..., x_{k_C}\}$		Wejście **S**: $Y = \{y_1, ..., y_{k_S}\}$
C wykonuje następujące akcje:		**S** wykonuje następujące akcje:
1. Wybiera klucz prywatny sk, klucz publiczny pk i parametry $parametry$ dla homomorficznego szyfru.		
2. Przesyła $\{pk, parametry\}$	$\Longrightarrow$	
3. Za pomocą interpolacji liczy współczynniki wielomianu $f(x) = \sum_{i=0}^{k_C} f[i] \cdot x^i$, stopnia k_C, z pierwiastkami: $\{x_1, ..., x_{k_C}\}$ i $f[0] = 1$.		
4. Przesyła $\{E_{pk}(f[1]), ..., E_{pk}(f[k_C)\}$	$\Longrightarrow$	
		5. Dla każdego $y_k \in Y$:
		5.a) Liczy $E(f(y_k)) = E_{pk}(\sum_{i=0}^{k_C} f[i] \cdot {y_k}^i)$, zakładając, że $f[0] = 1$.
		5.b) Wybiera losowe $r \in R \setminus \{0\}$ i liczy $w_k = E_{pk}(r \cdot f(y_k) + y_k)$.
		6. Wysyła:
	$\Longleftarrow$	$losowo\_spermutowane(\{w_1, ..., w_{k_S}\})$
7. Deszyfruje wszystkie k_S tekstów, które otrzymał. Jeśli odszyfrowana wartość należy do X, to dodaje ją do wyniku funkcji.		

Optymalizacje protokołu

Protokół 2.1 można w kilku miejscach istotnie zoptymalizować. W tym rozdziale zaprezentuję te optymalizacje.

Liczenie wartości wielomianu w punkcie: Podczas liczenia zaszyfrowanej wartości wielomianu w kroku 5.a) może być użyta reguła Hornera. Można policzyć $E_{pk}(f(y))$ następująco:

$$f(y) = f[0] + y \cdot (f[1] + y \cdot (f[2] + ...y \cdot (f[k_C - 1] + y \cdot f[k_C])...))$$

Można też policzyć to „standartową" metodą:

$$f(y) = f[0] + f[1] \cdot y + f[2] \cdot y^2 + ... + f[k_C] \cdot y^{k_C}$$

jednak po policzeniu każdego $f[i] \cdot y^i$ algorytm zapamiętuje y^i i używa tej wartości do policzenia kolejnego $f[i+1] \cdot y^i \cdot y_i$.

Te ulepszania są znaczące w porównaniu do naiwnego rozwiązania tego problemu.

Zmniejszenie dziedziny: Teraz podam dwie zmiany dziedziny, które istotnie mogą przyspieszyć operacje na wielomianach.

Zazwyczaj liczba elementów w zbiorach jest znacznie mniejsza od dziedziny elementów zbioru. Zatem zbiory wejściowe mogą być przyporządkowane do zbioru wielkości $m \approx 2 * log(max\{k_C, k_S\})$. Można w celu tego przekształcenia użyć funkcji haszującej parami niezależnej (co oznacza małe prawdopodobieństwo kolizji).

Można zmienić dziedzinę R. Załóżmy, że każdy element z D może być zapisany na m bitach. Wtedy element z $a \in D$, można zakodować w R w następujący sposób: $0^k||c_m(a)$ (gdzie $c_m(a)$ oznacza zakodowanie a w m bitach, a k jest parametrem bezpieczeństwa). Wtedy prawdopodobieństwo, że losowy element z R należy do D jest $\frac{1}{2^k}$ (czyli tak samo jak w rozdziale 1.7.2). R definiuje jako Z_p, gdzie p to najmniejsza liczba pierwsza większa od każdego elementu z R (czyli od $2^{m \cdot k}$).

Stosując dwa wyżej podane w tym paragrafie podejścia uzyskuję, że podczas liczenia wartości wielomianu w punkcie za pomocą metody Hornera (krok protokołu 2.1) wartość y zapisać można na m bitach (co jest istotnie mniejsze niż modułu Z_p). Ze względu na specyfikę szyfru Pailliera znacznie przyspiesza to działanie tej operacji (w tym szyfrze mnożenie przez stałą oznacza podnoszenie do takiej potęgi). To ulepszenie znacznie zmniejsza czynnik ukryty w złożoności O (ale nie zmienia złożoności w sensie notacji O).

Użycie haszowania w celu zmniejszenia kosztu operacji na wielomianie

Najlepsza optymalizacja dotyczy użycia haszowania w celu rozbicia wielomianu na wiele mniejszych wielomianów. Dla jednego wielomianu stopnia k_C policzenie współczynników na podstawie punktów jest dość kosztowne. Również policzenie wartości wielomianu stopnia k_C w danym punkcie jest kosztowne.

Zmniejszenie tych złożoności uzyskuję przez zmniejszenie stopnia wielomianu. Definiuję proces, który wrzuca elementy z X do B skrzynek zawierających co najwyżej M elementów.

Client definiuje wielomian stopnia M dla każdej takiej skrzynki: wszystkie elementy przyporządkowane do tej skrzynki przez jakąś funkcje h są pierwiastkami tego wielomianu. Reszta pierwiastków tego wielomianu uzupełniam zerami (czyli dla l elementów przyporządkowanych do tej skrzynki i stworzeniu wielomianu stopnia l, mnożę ten wielomian przez x^{M-l}). Dzięki temu wielomian uzyskany dla każdej skrzynki jest stopnia M. Istotne jest to, że 0 nie należy do dziedziny elementów zbioru.

Zmiana w protokole podstawowym polega na tym, że **C** wysyła funkcję haszującą i współczynniki B wielomianów stopnia co najwyżej M do **S** (miejscami zerowymi tych wielomianów są wartości z X które są przyporządkowane przez funkcję h do któregoś z B wielomianów), zamiast wysyłać jeden wielomian stopnia k_C. **Server** dla każdego $y \in Y$, znajduje skrzynkę (wielomian), do której y jest przyporządkowany (za pomocą h) i liczy wartość jak poprzednio: $E_{pk}(r \cdot f_y(y) + y)$. Poprawiony protokół jest przedstawiony na rysunku: protokół 2.2.

Wrzucanie elementów do skrzynek - zrównoważona alokacja. Przyporządkowanie elementów do skrzynek jest wykonane za pomocą losowej funkcji haszującej h z przeciwdziedziną B. Dobra alokacja może być uzyskana używając zrównoważoną alokację haszującą zaprezentowaną w pracy Azara ([YAAB99]). W tej pracy używa się połączenia kilku funkcji haszujących do uzyskania dobrego rezultatu. Teoria 1.1 w [YAAB99] pokazuje, że z prawdopodobieństwem $1 - o(1)$ maksymalna liczba elementów przyporządkowanych do skrzynki jest:

$$M = (1 + o(1)) \cdot \frac{ln(ln(B))}{ln(2)} + \Theta\left(\frac{k_C}{B}\right)$$

Ustawiając $B = \frac{k_C}{ln(ln(k_C))}$ uzyskuję $M = O(ln(ln(k_C)))$.

Zatem prawdopodobieństwo by liczba przyporządkowanych elementów do skrzynki była większa od M jest $o(1)$. W pracy [ABMM01] jest pokazane oszacowanie, które dowodzi, że

Protokół 2.2: wersja zoptymalizowana; uwaga: wejścia są brane z R, jak to jest opisane w paragrafie o zmniejszaniu dziedziny;

PROTOKÓŁ-SI-PASYWNY-wersja2:		
Client:		**Server**:
Wejście **C**: $X = \{x_1, ..., x_{k_C}\}$		Wejście **S**: $Y = \{y_1, ..., y_{k_S}\}$
C wykonuje następujące akcje:		**S** wykonuje następujące akcje:
1. Wybiera klucz prywatny sk, klucz publiczny pk i parametry $parametry$ dla homomorficznego szyfru.		
2. Wybiera losowo klucz k_h do funkcji haszującej h, wykonującej zrównoważoną alokację.		
3. Wysyła $\{pk, parametry, k_h\}$	$\Longrightarrow$	
4. Za pomocą interpolacji liczy współczynniki dla każdego z B wielomianów stopnia M (pierwiastkami, każdego wielomianu są wartości ze zbioru X które zostały przyporządkowane do danego wielomianu przez funkcję h, reszta pierwiastków to 0). Jeśli jakaś skrzynka pozostała bez przyporządkowanych elementów to wielomian jest ustawiany na $f(x) = 1$.		
5. $\{zaszyfrowane\_współczynniki\_wielomianów\}$	$\Longrightarrow$	
		6. Dla każdego $y_k \in Y$: 6.a) Liczy $E(f_{y_k}(y_k)) = E_{pk}(\sum_{i=0}^{k_C} f_{y_k}[i] \cdot {y_k}^i)$, gdzie f_{y_k} to wielomian odpowiadający danemu y_k (jest znaleziony za pomocą funkcji h dla klucza k_h). 6.b) Wybiera losowe r i liczy: $w_k = E_{pk}(r \cdot f_{y_k}(y_k) + y_k)$.
	$\Longleftarrow$	7. $losowo\_spermutowane(\{w_1, ..., w_{k_S}\})$
8. Deszyfruje wszystkie k_S tekstów, które otrzymał. Jeśli odszyfrowana wartość należy do X, to dołącza ją do wyniku funkcji.		

takie funkcje alokacji mogą być używane w praktyce. Autorzy [FNP04] przeprowadzili również praktyczne testy, które pokazują, że to rozwiązanie jest praktyczne (dla: $B = \frac{k_C}{ln(ln(k_C))}$ i $M = O(ln(ln(k_C)))$).

Dobry wynik można uzyskać również stosując losową funkcję haszującą (jest to funkcja haszująca, która pobiera mały losowy klucz).

Złożoność

Złożoność transmisyjna protokołu zoptymalizowanego 2.2 jest taka sama w sensie O jak złożoność protokołu 2.1. Złożoność transmisji to przesłanie funkcji haszującej i przesłanie wszystkich zaszyfrowanych współczynników wielomianów oraz przesłanie zaszyfrowanych wartości

wielomianów w punktach (gdzie λ oznacza ilość bitów potrzebna do zapisania R):

$$O(klucz\_funkcji\_haszujacej + B * M * \lambda + k_S * \lambda) =$$
$$= O(klucz\_funkcji\_haszujacej + (k_S + k_C) * \lambda) = O((k_S + k_C) * \lambda)$$

Jako, że funkcja haszująca jest wybierana losowo z rodziny funkcji haszujących to $klucz\_funkcji\_haszujcej$ nie wpływa istotnie na złożoność (bo $klucz\_funkcji\_haszujcej$ zależy bezpośrednio od parametru bezpieczeństwa).

Ponieważ $O(\lambda) = O(log(|R|) \cdot param1) = O(log|D| \cdot param1 \cdot param2)$ (gdzie $param1$ i $param2$ to parametry bezpieczeństwa) to można złożoność transmisyjną zapisać jako $O(k_S + k_C)$ (w tym zapisie pominięty jest parametr bezpieczeństwa).

Złożoność czasowa **Client**a to: $B = k_C/ln(ln(k_C))$ interpolacji (zakładam czas kwadratowy interpolacji - za [TCCLRR]) oraz sprawdzenie listy odebranej od **Server**a:

$$B * M^2 + k_S = O(k_C/ln(ln(k_C)) * ln^2(ln(k_C)) + k_S) = O(k_C * ln(ln(k_C)) + k_S) =$$

$$O(k_C * log(log(k_C)) + k_S)$$

Server wykonuje k_S razy obliczenie zaszyfrowanej wartości wielomianu - koszt tej operacji to M operacji mnożenia przez tekst jawny z małej dziedziny (z D – jest to dokładniej opisane w paragrafie o zmniejszaniu dziedziny). Zatem złożoność czasowa **Server**a jest

$$O\left(k_S * \left(1 + ln(ln(k_C)) * \frac{m}{\lambda}\right)\right) = O\left(k_S + k_S * \frac{ln(ln(k_C)) \cdot m}{\lambda}\right)$$

ze względu na ilość operacji na liczbach rozmiaru modułu λ. m oznacza liczbę bitów potrzebnych do zapisania elementów z dziedziny D. 1 w obliczeniach jest wzięta ze względu na jedno losowe obliczenie: r jest wybrane losowo z λ bitów i następnie przemnożone przez szyfrogram. Ponieważ $O(\lambda) = O(m \cdot param)$, gdzie $param$ to parametr bezpieczeństwa, to złożoność **Server**a można zapisać następująco:

$$O(k_S * ln(ln(k_C)))$$

Poprawność i bezpieczeństwo protokołu `PROTOKÓŁ-SI-PASYWNY`

W tym podrozdziale pokażę poprawność i bezpieczeństwo protokołu `PROTOKÓŁ-SI-PASYWNY` (protokół 2.1) w modelu z przeciwnikiem pasywnym (te dowody są pominięte w pracy [FNP04]).

Lemat 1 ***(Poprawność)*** *Protokół `PROTOKÓŁ-SI-PASYWNY` liczy funkcję (`set-intersection`) z przeważającym prawdopodobieństwem.*

Dowód:

C wysyła do **S** zaszyfrowane współczynniki wielomianów (których pierwiastki reprezentują zbiór X) i funkcję haszującą. Następnie **S** liczy dla każdego $y_k \in Y$: $E_{pk}(r \cdot f_{y_k}(y_k) + y_k)$. Jeśli $y \in X$ to wartość $E_{pk}(r \cdot f_y(y) + y) = E_{pk}(y)$ i po odszyfrowaniu **C** rozpozna tę wartość (funkcja haszująca wybiera właściwy wielomian, gdyż dokładnie tak samo robił to **C** wcześniej – w kroku 6.a) protokołu 2.2). Jeśli $y \notin X$ to: $f_y(y) > 0$ (bo y nie jest pierwiastkiem tego wielomianu), a zatem $r \cdot f_y(y) + y$ jest losowe w R (a $R = Z_p$, gdzie p jest liczbą pierwszą). Prawdopodobieństwo, że losowy element z R należy do D jest zaniedbywalne (nieformalnie: wykładniczo małe ze względu na jakiś parametr, napisałem o tym w rozdziale 1.7.2).

Czyli prawdopodobieństwo, że funkcja zostanie policzona niepoprawnie jest zaniedbywalne (uwaga: jedyny możliwy błąd - to policzenie zbyt dużego zbioru).
□

Lemat 2 ***(Bezpieczeństwo Clienta jest zachowane)*** *Jeśli schemat szyfrowania jest semantycznie bezpieczny to* ***S*** *nie jest w stanie rozróżnić sytuacji, w której* ***C*** *ma różne wejścia.*

Dowód:

Załóżmy, że istnieją takie dwa zbiory, że **S** potrafi rozróżnić te sytuacje na podstawie szyfrogramów, które otrzymuje (potrafi zdecydować w jakiej sytuacji się znajduje). Jednak taka umiejętność przeciwnika jest sprzeczna z definicją szyfru semantycznie bezpiecznego (a jest założone, że taki jest używany). Sprzeczność.

□

Lemat 3 ***(Bezpieczeństwo Servera jest zachowane)*** *Dla każdego klienta C^*, który działa w modelu rzeczywistym istnieje klient* ***C*** *działający w modelu idealnym taki, że dla każdego wejścia* ***S****: Y sytuacja stron* ***C, S*** *w idealnym modelu jest obliczeniowo nierozróżnialna od widoku stron C^* i* ***S*** *w rzeczywistym modelu.*

Dowód:

Dla gracza działającego w modelu rzeczywistym C^* z wejściem X istnieje gracz **C** w modelu idealnym, który przesyła wejście X do zaufanej strony trzeciej **TTP**. W idealnym modelu **C** dostaje od **TTP** zbiór $X \cap Y$ (gdzie Y to dowolne dane wejściowe wysłane przez **S**). W modelu rzeczywistym C^* dostaje k_S wartości. Jeśli $x \in X$ to $E_{pk}(x)$ znajduje się w tym zbiorze. Inne wartości są losowe (jak to zostało okazane przy dowodzie poprawności). Zatem C^* mógłby wygenerować te losowe wartości sam.

□

2.1.5. Wersja dla przeciwnika aktywnego

W tym rozdziale przedstawię protokół dla przeciwnika aktywnego (ten protokół pochodzi z pracy [FNP04]). Przedstawię też intuicję bezpieczeństwa protokołu. Dokładniejszej analizy nie przedstawię, gdyż w pracy [KSC05] jest przedstawione rozwiązanie bez używania metody „podziel-i-wybierz" (ang.: *cut-and-choose*). Zatem rozwiązanie z pracy [KSC05] jest wydajniejsze.

Protokół przedstawię osobno dla przeciwnika aktywnego: serwera (**S**) i klienta (**C**). Następnie podam sposób połączenia tych protokołów (by uzyskać pełne bezpieczeństwo).

2.1.6. Wersja dla aktywnego Servera

W tym rozdziale przedstawię protokół, który zapewnia bezpieczeństwo **Client**a przeciwko aktywnemu przeciwnikowi – **Server**owi. Protokół jest przedstawiony w modelu losowej wyroczni (ang.: *random oracle model*). Krótko piszę o tym modelu w rozdziale 1.5. Model losowej wyroczni jest mechanizmem służący do modelowania i formalnego dowodzenia sytuacji, gdy używane są funkcje haszujące (szczegóły są zawarte w [ID04]).

Protokoły z rozdziału 2.1.4 umożliwiają aktywnemu serwerowi atak na poprawność protokołu (**Server** nie może zaatakować bezpieczeństwa protokołu, gdyż nie może odczytać żadnych informacji klienta, bo wszystko co otrzymuje jest zaszyfrowane). Przykładowo **Server** może liczyć w kroku 6.b) protokołu 2.2: $E_{pk}(r \cdot (f_{y_k}(y_k) + f_{y_{k+1}}(y_{k+1})) + y_k)$, co jak widać nie jest zgodne z definicją części wspólnej (w tym przykładzie **C** dowiaduje się, że y_k jest w części wspólnej wtedy i tylko wtedy, gdy $y_k, y_{k+1} \in X$).

Protokół bezpieczny w modelu z aktywnym przeciwnikiem, który może korumpować **Server**a jest umieszczony na rysunku: protokół 2.3. Jest to przystosowany do przeciwnika aktywnego protokół 2.2.

Intuicja poprawności protokołu 2.3 polega na fakcie, że **Client** może zweryfikować obliczenia **Server**a (gdyż posiada dane wystarczające do weryfikacji czy **Server** zachowywał się poprawnie – dzięki losowym wyroczniom).

Bezpieczeństwo protokołu `PROTOKÓŁ-SI-AKTYWNY-server`

Lemat 4 *((**Bezpieczeństwo Clienta**)) Dla każdego serwera S^* operującego w modelu rzeczywistym istnieje serwer **S** operujący w idealnym modelu, taki że widok stron **C** i **S** w modelu idealnym jest obliczeniowo nierozróżnialny od widoku **C** i S^* w modelu rzeczywistym.*

Dowód tego twierdzenia znajduje się w [FNP04, str. 12-13]

2.1.7. Wersja dla aktywnego Clienta

W tym rozdziale przedstawię protokół, który zapewnia bezpieczeństwo **Server**a przeciwko aktywnemu przeciwnikowi – **Client**owi.

Jedyną możliwością oszustwa **Client**a w przypadku protokołu zoptymalizowanego (protokół 2.2) z rozdziału 2.1.4 jest wysyłanie któregoś wielomianu z zerowymi współczynnikami (wtedy może dowiedzieć się więcej informacji od **Server**a). Zatem **Server** musi być w stanie zweryfikować, że ilość miejsc zerowych (różnych od 0) wszystkich wielomianów przesłanych przez **Client**a jest k_C. Aby to uzyskać jest użyta metoda „podziel-i-wybierz" (ang.: *cut-and-choose*). Z dodatkowym wydłużeniem działania protokołu L–krotnie można uzyskać prawdopodobieństwo błędu wykładniczo małe względem L.

Uwaga: Protokół 2.1 jest bezpieczny przeciwko atakom aktywnego **Client**a (jest to opisane w rozdziale 2.1.4).

By dowieść formalnie bezpieczeństwa protokołu 2.4 dla aktywnego **Client**a trzeba pokazać, że dla każdego możliwego zachowania klienta w rzeczywistym modelu można znaleźć dane wejściowe rozmiaru k_C, które klient wysyła z idealnym modelu do **TTP** w taki sposób, że obie sytuacje są obliczeniowo nierozróżnialne.

Protokół bezpieczny w modelu z aktywnym przeciwnikiem, który może korumpować **Client**a jest umieszczony na rysunku: protokół 2.4.

Nieformalnie bezpieczeństwo **Server**a można uzasadnić następująco. **C** uczy się o elemencie y wtedy i tylko wtedy, gdy y jest pierwiastkiem któregoś z wielomianów we wszystkich $\frac{L}{2}$ kopiach wielomianów wysłanych przez **Client**a. Zatem by oszukać **Client** musi wysłać $\frac{L}{2}$ kopii posiadających więcej niż k_C pierwiastków. Prawdopodobieństwo, że wszystkie wielomiany posiadające więcej niż k_C pierwiastków są niesprawdzone przez **S** jest wykładniczo małe ze względu na L.

Bezpieczeństwo protokołu `PROTOKÓŁ-SI-AKTYWNY-client`

Lemat 5 *(**Bezpieczeństwo Servera**) Dla każdego **Client**a C^* operującego w modelu rzeczywistym istnieje klient **C** operujący w idealnym modelu, taki że widok stron **S** i **C** w modelu idealnym jest obliczeniowo nierozróżnialny od widoku **S** i C^* w modelu rzeczywistym.*

Dowód tego twierdzenia znajduje się w [FNP04, str. 12].

2.1.8. Obsługa aktywnego Clienta i Servera

W tym podrozdziale krótko i zwięźle opiszę jak połączyć dwa wyżej podane protokoły (protokoły 2.3 i 2.4) w jeden, który będzie sobie „radził" z oboma typami aktywnego przeciwnika – **Server**a i **Client**a.

C zachowuje się podobnie jak w protokole 2.4 – generuje L kopii, gdzie każda kopia składa się z B skrzynek – wielomianów. Dla każdej skrzynki B_i, **C** generuje wielomian stopnia M, taki że $P(z) = 0$ dla $z \in B_i$. z jest w B_i wtedy, gdy: z jest przyporządkowane przez funkcję haszującą do B_i (gdzie $z = F_{s_{kopia}}(G(x))$, dla $x \in X$ i s_{kopia} jest kluczem funkcji pseudolosowej charakterystycznym dla kopii) lub gdy $z = 0$ i jest dodane by uzupełnić stopień wielomianu do M. Gdy **C** skończy szykować L różnych kopii (charakteryzowanych przez klucz funkcji losowej s i klucze homomorficznego szyfru) to wysyła je zaszyfrowane do **S** (wszystkie kopie są szyfrowane innymi kluczami). Można to porównać do wysłana L zobowiązań do wielomianów.

Następnie **S** „otwiera" losowo wybrane $\frac{L}{2}$ zaszyfrowanych kopii (za pomocą kluczy przesłanych przez **C**) i weryfikuje je (sprawdza czy wielomiany mają k_C miejsc zerowych różnych od 0). Jeśli weryfikacja jest zakończona niepowodzeniem to protokół jest zakończony. W przeciwnym przypadku **S** wybiera losowe s i dzieli je na $\frac{L}{2}$ udziałów. Następnie zachowuje się jak w protokole 2.3, z tym że:

- używa losowych udziałów jako „wkładu" (ang.: *payload*) w kroku 6.b) protokołu 2.3: $E_{pk}(r' \cdot f_{y_k}(y_k) + s_{udzial})$ (gdzie s_{udzial} reprezentuje jeden z udziałów s);
- $H_1(s)$ użyte jest jako losowość w protokole (po stronie **Server**a);
- $H_2(r'', y)$ jest dodawane do wiadomości (do $E_{pk}(r' \cdot f_{y_k}(y_k) + s_{udzial})$);

Na koniec **C** otrzymuje listę $\frac{L}{2}$ nie otwartych kopii. Liczy kandydatów na udziały s i odzyskuje z nich s. Używa procedury podobnej do tej z kroku 8 protokołu 2.3 do sprawdzenia poprawności policzenia przez serwer przesłanych wartości.

2.1.9. Podsumowanie

W tej sekcji przedstawiłem protokoły z pracy [FNP04] rozwiązujące problem `set-intersection` dla 2 graczy. Należy zauważyć, że rozwiązanie dla aktywnego klienta nie jest najwydajniejsze, gdyż użyta jest metoda „podziel-i-wybierz". W pracy [KST05] jest przedstawiona wydajniejsza wersja (chociaż protokoły w niej przedstawione są na innym poziomie abstrakcji – są opisane mniej szczegółowo).

W pracy [FNP04] są przedstawione również protokoły:

- protokół aproksymujący przecięcie dwóch zbiorów (a w zasadzie problem związany z przecięciem zbiorów); ten protokół ma jednak wiele istotnych wad;
- dwie modyfikacje problemu przecięcia zbiorów:
  - protokół, w którym klient poznaje ilość elementów w zbiorze części wspólnej dwóch zbiorów;
  - protokół, w którym klient poznaje, czy ilość elementów w zbiorze części wspólnej dwóch zbiorów jest większa od jakiejś stałej t; ten protokół jednak używa w pewnym momencie rozwiązań bazujących na protokołach obliczeń wielopodmiotowych używających układów arytmetycznych (a te rozwiązania są potencjalnie wolne – wspominam o nich w rozdziale 3.3.1);
- protokół liczący część wspólną dla n graczy w modelu z przeciwnikiem pasywnym (wydajniejszy protokół, oraz protokół w modelu aktywnym jest zaproponowany w [KST05]);

- protokół dla problemu „fuzzy matching” (szczegóły przedstawione później w tej pracy); moim zdaniem protokół podany w pracy [FNP04] dla problemu „fuzzy matching” jest błędny (pokaże to w rozdziale 3.2);
- są również przedstawione pewne teoretyczne rozważania na temat protokołów rozwiązujących problem `set-intersection`;

2.2. Protokół części wspólnej zbiorów (`set-intersection`) dla n graczy

W tym rozdziale zaprezentuje protokół zaczerpnięty z pracy [KST05] rozwiązujący problem części wspólnej n zbiorów (z n uczestnikami).

2.2.1. Definicja problemu

Protokół `set-intersection` dla n ($n \geq 2$) graczy (w pracy [KST05] nazywany `set-intersection`) jest wielopodmiotowym obliczeniem funkcji liczącej część wspólną n zbiorów (gdzie każdy zbiór jest wejściem jednego z graczy).

Wejściem gracza i ($1 \leq i \leq n$) jest zbiór $S_i = \{(S_i)_1, ..., (S_i)_k\}$, gdzie $(S_i)_j \in D$ (dla $1 \leq j \leq k$), gdzie D to dziedzina wartości elementów zbioru ($|D| = N$). Dla uproszczenia i bez straty ogólności zakładam, że $D = [1..N]$. Dziedzinę szyfru homomorficznego R definiuję podobnie jak w poprzedniej sekcji (czyli prawdopodobieństwo, że losowo wybrany element z R należy do D jest zaniedbywalne).

Wyjściem protokołu dla każdego gracza jest zbiór: $S_1 \cap ... \cap S_n$ (czyli część wspólna wszystkich zbiorów).

Ten problem może być rozpatrywany w dwóch podejściach:

- każdy S_i jest zbiorem i wynikiem jest też zbiór;
- każdy S_i jest multi-zbiorem i wynikiem jest multi-zbiór o następującej własności: jeśli element a jest w liście wejściowej każdego gracza co najmniej b razy to element a występuje w wyniku każdego gracza co najmniej b razy;

Protokół jest uniwersalny i nie zakłada, czy wejściem jest zbiór czy multi-zbiór.

W tej sekcji zbiory będą definiowane podobnie jak w poprzedniej sekcji 2.1.

Dla tego protokołu bezpieczeństwo jest definiowane jak w sekcji 1.2. Przeciwnik może skorumpować w aktywny lub pasywny sposób co najwyżej c przeciwników (gdzie c jest parametrem protokołu i $c < n$).

2.2.2. Wersja dla przeciwnika pasywnego

W tym podrozdziale prezentuję protokół dla problemu `set-intersection` dla n graczy w modelu z przeciwnikiem pasywnym. Ten protokół jest zaprezentowany na rysunku 2.5.

W pierwszym kroku tego protokołu każdy gracz liczy wielomian f_i reprezentujący jego zbiór i wysyła jego zaszyfrowanie do c innych graczy (zatem $c + 1$ graczy ma zaszyfrowany wielomian f_i). Każdy z tych graczy $i + j$ ($0 \leq j \leq c$) wybiera losowy wielomian $r_{i+j,i}$ i oblicza zaszyfrowanie wielomianu $f_i * r_{i+j,i}$. Ważne jest to, że nie może być skorumpowana grupa wielkości $c + 1$, a zatem przeciwnik nie jest w stanie poznać wszystkich losowych wielomianów pomnożonych przez f_i. Z tego wynika, że wielomian $(\sum_{j=0}^{c} r_{i+j,j})$ nie jest znany przeciwnikowi i jest losowy. Następnie każdy z graczy i ($1 \leq i \leq n$) dodaje swoje

wielomiany: $f_{i-j} * r_{i,i-j}$ $(0 \leq j \leq c)$ by uzyskać ϕ, a te wielomiany są później dodane (w kroku 2-3 protokołu 2.5) by uzyskać $p = \sum_{i=1}^{n} \phi_i = \sum_{i=1}^{n} f_i * (\sum_{j=0}^{c} r_{i+j,j})$.

Jeśli element a jest pierwiastkiem wielomianu f_i to jest także pierwiastkiem wielomianu $f_i * r$ dla losowego wielomianu r. Z tego powodu wszystkie elementy prywatnych zbiorów wejściowych są zachowane jako pierwiastki, gdy wielomiany są mnożone przez losowe wielomiany. Jeśli a jest pierwiastkiem wielomianów f i g to jest także pierwiastkiem wielomianu $f + g$. Dlatego jeśli wielomiany $f_i * (\sum_{j=0}^{c} r_{i+j,j})$ (dla $1 \leq i \leq n$) zostaną dodane to jeśli jakiś element znajdował się we wszystkich prywatnych zbiorach wejściowych (a f_i to reprezentacje tych zbiorów wejściowych) i jego reprezentacja znajduje się w ostatecznym wielomianie. Jeśli jakiś element nie należał do wszystkich zbiorów wejściowych to użycie losowych wielomianów zapewnia (z przeważającym prawdopodobieństwem), że jest on ukryty i że nie pojawia się w zbiorze wynikowym. Wynika to z twierdzenia 1 z sekcji 1.7.3. Następnie gracze grupowo odszyfrowują końcowy wielomian (krok 5 protokołu – (n, n)–odszyfrowanie) i sprawdzają czy reprezentacje ich elementów ze zbioru wejściowego są pierwiastkami tego wielomianu. Elementy, które są pierwiastkami końcowego wielomianu, należą do zbioru wynikowego. Zatem każdy gracz uczy się części wspólnej wszystkich zbiorów.

Protokół działa dla zbiorów, jak również w naturalny sposób dla multi-zbiorów (w sposób, który jest opisany w sekcji 2.2.1).

W prezentowanym protokole 2.5 zakładam, że na początku protokołu każdy uczestnik i posiada dzielony klucz prywatny ssk_i, który jest jego udziałem w kluczu prywatnym sk (szczegółowo opisałem dlaczego to zakładam w sekcji 1.3).

Bezpieczeństwo protokołu `PROTOKÓŁ-SI-N-GRACZY-PASYWNY`

Lemat 6 ***(Poprawność działania protokołu w modelu z przeciwnikiem pasywnym)*** *W protokole* `PROTOKÓŁ-SI-N-GRACZY-PASYWNY` *(protokół 2.5), każdy gracz z przeważającym prawdopodobieństwem uczy się zbioru będącego częścią wspólną zbiorów wejściowych wszystkich graczy:* $I = S_1 \cap S_2 \cap ... \cap S_n$.

Dowód tego twierdzenia znajduje się w [KST05, str. 30].

Lemat 7 ***(Prywatność graczy w modelu z przeciwnikiem pasywnym)*** *W protokole* `PROTOKÓŁ-SI-N-GRACZY-PASYWNY` *(protokół 2.5) przeciwnik, który może skorumpować w pasywny sposób dowolną grupę co najwyżej c graczy dowiaduje się nie więcej niż by się dowiedział używając tych samych zbiorów wejściowych w modelu idealnym.*

Dowód tego twierdzenia znajduje się w [KST05, str. 30].

2.2.3. Wersja dla przeciwnika aktywnego

W tym podrozdziale prezentuję protokół dla problemu `set-intersection` dla n graczy w modelu z przeciwnikiem aktywnym. Ten protokół jest zaprezentowany na rysunku: protokół 2.6.

Ten protokół w dużym stopniu bazuje na rozwiązaniu z rozdziału 2.2.2. Do protokołu 2.5 dodane są dowody z wiedzą zerową weryfikowane przez wszystkich graczy (są dodane by zapewnić poprawność i bezpieczeństwo obliczeń w modelu z przeciwnikiem aktywnym). Najpierw zaprezentuję notację dowodów z wiedzą zerową (użytych w tym rozdziale), a następnie protokół bezpieczny względem aktywnego przeciwnika.

Dowody z wiedzą zerową

Krótko napisałem o dowodach z wiedzą zerową w rozdziale 1.3.

Przedstawiam notację dla użytych w protokole dowodów z wiedzą zerową. Dla szyfru Pailliera można wydajnie skonstruować te dowody bazując na wynikach z prac [CEGP86, JC97].

W protokole 2.6 są użyte następujące dowody z wiedzą zerową:

- POPK{ $E_{pk}(x)$ } jest dowodem wiedzy zerowej, który dla danego szyfrogramu $E_{pk}(x)$ udowadnia, że dany gracz zna odpowiadający mu tekst jawny.

- ZKPK{ $f : p' = f *_h E_{pk}(p)$ } jest skrótową notacją dowodu wiedzy z wiedzą zerową, pokazującego że „udowadniacz" (ang.: *prover*) zna wielomian f taki, że zaszyfrowany wielomian $p' = f *_h E_{pk}(p)$ dla danego zaszyfrowanego wielomianu p' i $E_{pk}(p)$.

- ZKPK{ $f : (p' = f *_h E_{pk}(p)) \cap (y = E_{pk}(f))$ } jest dowodem ZKPK{ $f : p' = f *_h E_{pk}(p)$ } z dodatkowym warunkiem: $y = E_{pk}(f)$ (y jest szyfrogramem wielomianu f) przy danych: zaszyfrowanym wielomianie p', niezaszyfrowanym wielomianie y i $E_{pk}(p)$.

W prezentowanym protokole 2.6 zakładam, że na początku protokołu każdy uczestnik i posiada dzielony klucz prywatny ssk_i, który jest jego udziałem w kluczu prywatnym sk (szczegółowo opisałem dlaczego to zakładam w sekcji 1.3).

Protokół liczący część wspólną wielu zbiorów w modelu z przeciwnikiem aktywnym jest przedstawiony na rysunku: protokół 2.6. Jego działanie w dużym stopniu jest podobne do protokołu dla pasywnego przeciwnika (protokół 2.5). Zobowiązania do informacji $\Lambda(r_{i,j})$ są wykonywane wyłącznie ze względu na dowód poprawności i bezpieczeństwa protokołu (używając dwuznacznych zobowiązań i dowodów znajomości tekstu jawnego zaszyfrowanej wiadomości Symulator potrafi odtworzyć tekst jawny przesyłanego szyfrogramu).

Dodanie dowodów z wiedzą zerową chroni protokół przed następującymi sposobami oszukiwania:

- wybierania szyfrogramów dla współczynników wielomianu f_i bez ich znajomości;

- wykonywania niewłaściwie mnożenia wielomianów $f_j * r_{i,j}$;

- niewłaściwego wykonywania grupowego odszyfrowywania (np.: używania niewłaściwego dzielonego klucza prywatnego);

Wybranie k-tego współczynnika (wielomianu stopnia k) jako 1 chroni bezpieczeństwo protokołu przed atakiem polegającym na wybraniu zerowego wielomianu przez jakiegoś gracza by zyskać nieuprawnioną wiedzę. Analogicznie w protokole 2.1 współczynnik przy potędze 0 był ustawiany na 1 – nie ma znaczenia, który współczynnik jest ustawiany, istotne jest to że jest on z góry ustalony i nie był przesyłany.

Zatem gracze są w tym protokole zmuszeni do postępowania poprawnie – jak w protokole działającym w modelu bezpiecznym przeciwko przeciwnikowi pasywnemu (protokół 2.5). Jeśli jakiś uczestnik zachowuje się niewłaściwie to zostaje on wykryty przez uczciwych graczy (i protokół zostaje zatrzymany).

Protokół może zyskać na wydajności dzięki znajomości maksymalnego rozmiaru koalicji skorumpowanych graczy c.

Bezpieczeństwo protokołu `PROTOKÓŁ-SI-N-GRACZY-AKTYWNY`

Lemat 8 ***(Poprawność działania protokołu w modelu z przeciwnikiem aktywnym)*** *W protokole* `PROTOKÓŁ-SI-N-GRACZY-AKTYWNY` *(protokół 2.6), każdy gracz z przeważającym prawdopodobieństwem uczy się zbioru będącego częścią wspólną zbiorów wejściowych wszystkich graczy:* $I = S_1 \cap S_2 \cap ... \cap S_n$.

Dowód tego twierdzenia znajduje się w [KST05, str. 30] i jest taki sam jak dowód poprawności protokołu działającego w modelu z przeciwnikiem pasywnym.

Lemat 9 ***(Prywatność graczy w modelu z przeciwnikiem aktywnym)*** *W protokole* `PROTOKÓŁ-SI-N-GRACZY-AKTYWNY` *(protokół 2.6), w którym istnieje przeciwnik* Γ*, który może skorumpować w aktywny sposób dowolną grupę co najwyżej* $n-1$ *graczy, istnieje gracz (lub grupa graczy)* G *działający w idealnym modelu, taki że widok graczy w idealnym modelu jest obliczeniowo nierozróżnialny od widoku graczy uczciwych i* Γ *w modelu rzeczywistym.*

Dowód tego twierdzenia znajduje się w [KST05, str. 30-33].

2.2.4. Podsumowanie

W tej sekcji przedstawiłem protokoły z pracy [KST05] rozwiązujące problem `set-intersection` dla n graczy w modelu z przeciwnikiem pasywnym i w modelu z przeciwnikiem aktywnym. Te rozwiązania są przedstawione na wyższym poziomie abstrakcji niż rozwiązania z pracy [FNP04] (np.: nie są rozważane optymalizacje protokołów, nie są określone konkretne algorytmy operujące na zaszyfrowanych wielomianach). Jest to spowodowane tym, że w pracy [KST05] bardziej jest podkreślony aspekt bezpieczeństwa tych protokołów w modelu aktywnym dla n graczy, gdzie dowolna grupa c graczy może „oszukiwać". Natomiast w pracy [FNP04] główny nacisk położony jest na prostszą sytuację - 2 graczy.

Istotnym rozwinięciem pracy [FNP04] zamieszczonym w pracy [KST05] jest podanie:

- protokołu dla problemu `set-intersection` dla 2 graczy w modelu z przeciwnikiem aktywnym nie korzystających z metody "podziel-i-wybierz"(ang.: *cut-and-choose*)
- protokołu rozwiązującego problem `set-intersection` dla n graczy w modelu z przeciwnikiem aktywnym
- protokołów obliczeń wielopodmiotowych dla wielu problemów pokrewnych do problemu części wspólnej zbiorów:
  - problemu ilości elementów w części wspólnej zbiorów wszystkich uczestników (nazywane w pracy `cardinality-set-intersection`) dla n uczestników. Wyjściem protokołu jest dla każdego gracza: $|S_1 \cap S_2 \cap ... \cap S_n|$.
  - problemu „progowej" ilości elementów w części wspólnej zbiorów wszystkich uczestników (nazywane w pracy `treshold-set-intersection`) dla n uczestników. Jest to protokół w wyniku którego każdy gracz dowiaduje się (w zależności od wersji protokołu):
    - elementów z jego zbioru, które występują w części wspólnej wszystkich zbiorów t razy (gdzie t to parametr protokołu) – wersja jest nazywana ang.: *threshold contribution*.

    - elementów, które występują w części wspólnej wszystkich zbiorów t razy (gdzie t to parametr protokołu). W pracy są umieszczone dwie wersje tak zdefiniowanego protokołu, różniące się detalami użytej definicji bezpieczeństwa. Nazywane są *perfect* i *semi-perfect* `treshold-set-intersection`.
- problemu „ponad progowej" ilości elementów w części wspólnej zbiorów wszystkich uczestników (nazywane w pracy `over-treshold-set-intersection`) dla n graczy. Jest to protokół w wyniku którego każdy gracz uczy się wszystkich elementów zbiorów, które występują w części wspólnej wszystkich zbiorów co najmniej t razy (gdzie t to parametr protokołu) oraz ilości wystąpień danego elementu.

Podane protokoły są przedstawione zarówno w modelu z przeciwnikiem pasywnym jak i aktywnym. Czasami protokoły są przedstawione w wielu różnych wersjach w zależności od przyjętych założeń odnośnie modelu (np.: standartowy model, model z wyrocznią losową) lub szyfru. Założone jest że przeciwnik może skorumpować dowolną grupę co najwyżej c uczestników ($c \leq n - 1$). W dodatku pracy [KST05] są również zaprezentowane dowody poprawności protokołów. Wymienione protokoły działają również w „naturalny" sposób dla multi-zbiorów jako danych wejściowych.

Protokół 2.3:

PROTOKÓŁ-SI-AKTYWNY-server:		
Client:		**Server**:
Wejście **C**: $X = \{x_1, ..., x_{k_C}\}$		Wejście **S**: $Y = \{y_1, ..., y_{k_S}\}$
Losowe wyrocznie: H_1, H_2.		
C wykonuje następujące akcje:		**S** wykonuje następujące akcje:
1. Wybiera klucz prywatny sk, klucz publiczny pk i parametry $parametry$ dla homomorficznego szyfru.		
2. Wybiera losowo klucz k_h do funkcji haszującej h, wykonującej zrównoważoną alokację.		
3. Wysyła $\{pk, parametry, k_h\}$	$\Longrightarrow$	
4. Za pomocą interpolacji liczy współczynniki dla każdego z B wielomianów stopnia M (pierwiastkami, każdego wielomianu są wartości ze zbioru X które zostały przyporządkowane do danego wielomianu przez funkcję h, reszta pierwiastków to 0). Jeśli jakaś skrzynka pozostała bez przyporządkowanych elementów to wielomian jest ustawiany na $f(x) = 1$.		
5. Wysyła $\{zaszyfrowane\_współczynniki\_wielomianów\}$	$\Longrightarrow$	
		6. Dla każdego $y_k \in Y$: 6.a) Wybiera losowe s i liczy $r = H_1(s)$. 6.b) Wybiera losowe r i liczy: $(e, h)_k = (E_{pk}(r' \cdot f_{y_k}(y_k) + s), H_2(r'', y))$, gdzie r jest „sparsowane" na r_1 i r_2 (oraz wszystkie losowości użyte w tych obliczeniach).
		7. Wysyła
	$\Longleftarrow$	$losowo\_spermutowane(\{w_1, ..., w_{k_S}\})$
8. Dla każdej odszyfrowanej pary (e, h): 8.a) Odszyfrowuje $\mathfrak{s} = D_{sk}(e)$ i liczy $\mathfrak{r} = H_1(\mathfrak{s})$. 8.b) Parsuje $\mathfrak{r}$ na $\mathfrak{r}'$ i $\mathfrak{r}''$. 8.c) Sprawdza, czy istnieje $x \in X$, takie że: $h = H_2(\mathfrak{r}'', x)$ i $e = \mathfrak{r}' * P(y) + \mathfrak{s}$. Jeśli istnieje to dołącza x do wyniku funkcji.		

Protokół 2.4:

PROTOKÓŁ-SI-AKTYWNY-klient:

Client:		**Server**:
Wejście **C**: $X = \{x_1, ..., x_{k_C}\}$		Wejście **S**: $Y = \{y_1, ..., y_{k_S}\}$
C wykonuje następujące akcje:		**S** wykonuje następujące akcje:
1. Wybiera losowo klucz k_h do funkcji haszującej h, wykonującej zrównoważoną alokację.		
2. Wysyła k_h	$\Longrightarrow$	
3. Powtarza L razy (produkuje L kopii współczynników):		
3.a) Wybiera losowo klucz s_i dla pseudo-losowej funkcji F, i liczy pseudo-tożsamości: (ang.: *pseudo-identities*) $F_{s_i}(G(x))$, gdzie G jest kryptograficznie bezpieczną funkcją haszującą.		
3.b) Za pomocą interpolacji liczy współczynniki dla każdego z B wielomianów stopnia M (pierwiastkami, każdego wielomianu są wartości ze zbioru $\{F_{s_i}(G(x))\|x \in X\}$, które zostały przyporządkowane do danego wielomianu przez funkcję h, a reszta pierwiastków to 0). Jeśli jakaś skrzynka pozostała bez przyporządkowanych elementów to wielomian ją reprezentujący jest ustawiany na $f(x) = 1$.		
3.c) Wybiera klucz prywatny sk_i, klucz publiczny pk_i i parametry *parametry* dla homomorficznego szyfru.		
3.d) Wysyła $\{pk_i, parametry\}$	$\Longrightarrow$	
3.e) Szyfruje współczynniki wielomianów kluczem pk_i.		
3.f)$\{zaszyfrowane\_współczynniki\_wielomianów\}$	$\Longrightarrow$	
	$\Longleftarrow$	5. Wybiera losowo $\frac{L}{2}$ kopii i przesyła informację, które kopie mają zostać otwarte.
6. Odpowiada na to:		
6.a) wysyłając $\frac{L}{2}$ odpowiednich (tych, które **S** wybrał) kluczy prywatnych.	$\Longrightarrow$	
6.b) wysyłając $\frac{L}{2}$ kluczy funkcji pseudo-losowej s_i (dla $1 \leq i \leq \frac{L}{2}$) dla kopii, o które **S** się nie pytał.	$\Longrightarrow$	
		7. Weryfikuje, czy każdy z $\frac{L}{2}$ odszyfrowanych wielomianów ma w sumie k_C pierwiastków różnych od 0. Jeśli nie ma, to zatrzymuje działanie protokołu.
		8. W przeciwnym przypadku dla każdego $y_k \in Y$:
		8.a) Tworzy swoje pseudo-tożsamości $(F_{s_i}(G(y_k)))$ używając funkcji haszującej G i otrzymanych kluczy s_i.
	$\Longleftarrow$	8.b) Liczy dla każdego otrzymanego s_i (jest ich $\frac{L}{2}$): $E_{pk}(r' \cdot f_{F_{s_i}(G(y_k))}(F_{s_i}(G(y_k))) + y_{k,udzial})$ i wysyła do **C**, gdzie $y_{k,udzial}$ jest losowe, przy czym $\oplus$ (xor) wszystkich $y_{k,udzial}$ wynosi y_k.
9. Wynik, który otrzymuje to lista k_S zbiorów rozmiaru $\frac{L}{2}$. Deszyfruje tą listę i liczy $\oplus$ każdego zbioru i porównuje ze swoim wejściem. Jeśli wartość należy do X, to dołącza ją do wyniku funkcji.		

Protokół 2.5:

Protokół: `PROTOKÓŁ-SI-N-GRACZY-PASYWNY`
Wejście: $n \geq 2$ graczy, przy czym co najwyżej c graczy może być skorumpowanych w sposób pasywny (gdzie $c < n$). c jest parametrem protokołu. Każdy z graczy ma zbiór wejściowy S_i, gdzie $|S_i| = k$. Gracze dzielą klucz prywatny sk homomorficznego szyfru, dla którego pk jest odpowiadającym kluczem publicznym (jest możliwe (n, n)–odszyfrowywanie, które jest opisane w sekcji 1.3).

1. Każdy gracz $i = 1, ..., n$ wykonuje:
   (a) liczy wielomian odpowiadający swojemu zbiorowi wejściowemu: $f_i = (x - (S_i)_1) \cdot ... \cdot (x - (S_i)_k)$
   (b) wysyła zaszyfrowany wielomian f_i do graczy $i + 1, ..., i + c$
   (c) wybiera $c + 1$ losowych wielomianów stopnia k: $r_{i,0}, ..., r_{i,c}$ należących do $R[x]$ (losowy wielomian oznacza, że wszystkie współczynniki są wybrane losowo z R)
   (d) liczy zaszyfrowanie wielomianu:
   $$\phi_i = f_{i-c} * r_{i,i-c} + ... + f_{i-1} * r_{i,i-1} + f_i * r_{i,0}$$
   za pomocą algorytmu z sekcji 1.7.1.
2. Gracz numer 1 wysyła zaszyfrowanie wielomianu $\lambda_1 = \phi_1$ do gracza numer 2.
3. Każdy gracz $i = 2, ..., n$ po kolei
   (a) otrzymuje zaszyfrowany wielomian λ_{i-1} otrzymany od gracza $i - 1$
   (b) liczy zaszyfrowane $\lambda_i = \lambda_{i-1} + \phi_i$ (za pomocą algorytmu z sekcji 1.7.1)
   (c) jeśli $i < n$ to gracz wysyła zaszyfrowanie wielomianu λ_i do gracza numer $i + 1$; jeśli $i = n$ to gracz wysyła zaszyfrowanie wielomianu λ_i do gracza numer 1
4. Gracz numer 1 wysyła zaszyfrowanie wielomianu
   $$p = \lambda_n = \sum_{i=1}^{n} f_i * \left(\sum_{j=0}^{c} r_{i+j,j} \right)$$
   do wszystkich pozostałych graczy.
5. Wszyscy gracze wykonują zbiorowe odszyfrowanie danych i uzyskują wielomian p.
6. Każdy gracz $i = 1, .., n$ liczy, które z elementów jego zbioru są w zbiorze wynikowym (części wspólnej wszystkich zbiorów) w następującym sposób:
   dla danego elementu $(S_i)_j$ jeśli $p((S_i)_j) = 0$ to ten element jest w zbiorze wynikowym, a w przeciwnym przypadku nie jest.

Protokół 2.6:

Protokół: PROTOKÓŁ-SI-N-GRACZY-AKTYWNY
Wejście: $n \geq 2$ graczy, przy czym co najwyżej c graczy może być skorumpowanych w sposób aktywny (gdzie $c < n$). Każdy z graczy ma zbiór wejściowy S_i, gdzie $|S_i| = k$. Gracze dzielą klucz prywatny sk homomorficznego szyfru, dla którego pk jest odpowiadającym kluczem publicznym (jest możliwe (n, n)–odszyfrowywanie, które jest opisane w sekcji 1.3). Zobowiązanie użyte w protokole jest dwuznacznym zobowiązaniem (jest opisane w sekcji 1.6).
Wszyscy gracze weryfikują poprawność dowodów z wiedzą zerową zaraz po ich otrzymaniu. Przestają uczestniczyć w protokole jeśli weryfikacja dowodu zawiodła.

Każdy gracz $i = 1, ..., n$:

1. (a) liczy wielomian odpowiadający swojemu zbiorowi wejściowemu:
   $f_i = (x - (S_i)_1) \cdot ... \cdot (x - (S_i)_k)$
   (b) wysyła zaszyfrowany wielomian $\delta_i = E_{pk}(f_i)$ do wszystkich graczy wraz z dowodem znajomości wszystkich współczynników, z wyjątkiem współczynnika o indeksie k: (POPK{ $(\delta_i)_j$ }, $0 \leq j < k$).
   (c) dla każdego j takiego, że $1 \leq j \leq n$:
      i. wybiera losowy wielomian stopnia k: $r_{i,j}$ należący do $R[x]$ (losowy wielomian oznacza, że wszystkie współczynniki są wybrane losowo z R).
      ii. wysyła zobowiązanie $\Lambda(r_{i,j}) = E_{pk}(r_{i,j})$ do wszystkich graczy.

2. dla każdego j takiego, że $1 \leq j \leq n$:
   (a) otwiera zobowiązanie $\Lambda(r_{i,j})$.
   (b) weryfikuje dowód znajomości tekstu jawnego dla zaszyfrowanych współczynników f_i.
   (c) ustawia współczynnik o indeksie k (współczynnik przy x^k) na znany szyfrogram 1.
   (d) liczy μ - zaszyfrowanie wielomianu $p_{i,j} = f_j * r_{i,j}$ oraz dowód właściwego przemnożenia wielomianów: ZKPK{ $r_{i,j} | (\mu = r_{i,j} *_h \delta_j) \cap (\Lambda(r_{i,j}) = E_{pk}(r_{i,j}))$ }, a następnie przesyła to wszystko do wszystkich innych graczy.

3. Wszyscy gracze:
   (a) liczą zaszyfrowanie wielomianu.

$$p = \sum_{i=1}^{n} \sum_{j=1}^{n} p_{i,j} = \sum_{i=1}^{n} f_i * \left(\sum_{j=1}^{n} r_{j,i} \right)$$

   za pomocą algorytmu z sekcji 1.7.1 i weryfikują wszystkie dołączone dowody
   (b) wykonują grupowe odszyfrowanie by uzyskać wielomian p i przesyłają dowody poprawnego odszyfrowania.

4. Każdy gracz $i = 1, .., n$ liczy, które z elementów jego zbioru są w zbiorze wynikowym (części wspólnej wszystkich zbiorów) w następującym sposób:
   dla danego elementu $(S_i)_j$ jeśli $p((S_i)_j) = 0$ to ten element jest w zbiorze wynikowym, a w przeciwnym przypadku nie jest.

Rozdział 3

Protokoły obliczające część wspólną z błędami – „fuzzy matching”

W tym rozdziale przedstawię rozwiązania dla bezpiecznego wielopodmiotowego liczenia problemu „fuzzy matching”. Problem ten jest zdefiniowany w pracy „Private Matching and Set Intersection” napisanej przez Freedmana, Nissima i Pinkasa ([FNP04]). Najpierw zdefiniuję problem, a następnie podam rozwiązanie podane przez autorów wyżej wymienionej pracy. Następnie podam powód oraz przykład, dla którego ten protokół nie działa (ten protokół nie działa dla niektórych „rozsądnych” danych wejściowych). W kolejnej sekcji podam protokół działający w tym samym czasie (ze względu na ilość przesłanych bitów i złożoność czasową uczestników) mojego autorstwa. To rozwiązanie używa podobnej techniki jak to z pracy [FNP04]. Na koniec zaprezentuję protokół używający nieco innego podejścia, zmieniający złożoność komunikacyjną (dla niektórych danych znacznie ją zmniejszający).

3.1. Definicja problemu

Protokół `fuzzy-matching` dla dwóch graczy (w pracy nazywany [FNP04] *fuzzy matching*) jest dwupodmiotowym obliczeniem funkcji liczącej część wspólną dwóch zbiorów dopuszczającym pewną możliwość błędu przy porównywaniu elementów. W protokole uczestniczy 2 graczy: **Client** i **Server**.

Wejście i wyjście gracza **Client**a jest zdefiniowane następująco:

- wejściem **Client**a jest: $X = \{X_1, ..., X_{k_C}\}$ oraz t, gdzie $X_i = \{X_i^1, ..., X_i^T\}$ (X_i będę nazywał tuplą), a $X_i^j \in D$ (X_i^j będę nazywał atrybutem);
- wyjściem **Client**a są: elementy (tuple) ze zbioru **Server**a (zbioru Y), które są zgodne z jakimś elementem z jego zbioru na co najmniej t atrybutach. Dwa elementy X_i i Y_j są zgodne na t atrybutach jeśli: $t \leq |\{k : X_i^k = Y_j^k \cap (1 \leq k \leq T)\}|$;

Wejście i wyjście gracza **Server**a jest zdefiniowane następująco:

- wejściem **Server**a jest: $Y = \{Y_1, ..., Y_{k_S}\}$ oraz t, gdzie $Y_i = \{Y_i^1, ..., Y_i^T\}$, a $Y^j{}_i \in D$;
- wyjście **Server**a jest puste (**Server** nic nie poznaje);

D oznacza dziedzinę wartości elementów zbioru ($|D| = N$). Dla uproszczenia i bez straty ogólności zakładam, że $D = \{0, ..., N-1\}$. Dziedzinę szyfru homomorficznego do zakodowania elementu z D definiuję podobnie jak w sekcji 1.7.2. Zakładam również, że każdy element z D można zakodować na m bitach. Definiuję dziedzinę D^T jako dziedzinę elementów zbioru X i zbioru Y: X_i i Y_i. Zakładam, że $C(X_i)$ reprezentuje zakodowaną

wartość argumentów jako liczbę (zakodowaną na $T \cdot m$ bitach). Zapisuję to kodowanie jako $C(X_i) = C(X_i^1, ..., X_i^T) = c_m(X_i^1)||...||c_m(X_i^T)$ (gdzie $c_m(x)$ oznacza kodowanie x na m bitach). Zatem $D^T = \{0, ...N^T - 1\}$. Dziedzinę szyfru homomorficznego R dla kodowania D^T wybieram odpowiednio większą od D^T. Robię to by zachodziła następująca własność: prawdopodobieństwo, że losowo wybrany element z R należy do D^T jest zaniedbywalne (ze względu na parametr bezpieczeństwa). Dodatkowo zakładam, że $R = Z_p$, gdzie p to najmniejsza liczba pierwsza taka, że $p > N^T$. Zakładam też, że X i Y są zbiorami (interpretując X_i (dla $1 \leq i \leq k_C$) i Y_i (dla $1 \leq i \leq k_S$) jako elementy zbioru; zakładam, że elementy są równe gdy wszystkie atrybuty są takie same).

Można zdefiniować też podobny problem do `fuzzy-matching`: `fuzzy-search`. Jest to wersja protokołu dla, której $k_C = 1$. W [FNP04] jest zasugerowane rozwiązanie tego problemu (będące modyfikacją protokołu 2.1). Niestety ta sugestia jest zapisana bardzo chaotycznie.

3.2. Protokół - praca oryginalna

W tym rozdziale zaprezentuję protokół wzięty z pracy [FNP04]. Zaprezentuję go zgodnie z pracą: dla $T = 3$ i $t = 2$. Ten protokół jest przedstawiony w modelu z przeciwnikiem pasywnym. Następnie podam przykład, dla którego protokół nie działa i krótką dyskusję na ten temat. Ten protokół korzysta z podobnych technik jak protokół 2.1 i jest zaprezentowany na rysunku: protokół 3.1.

Poprawność protokołu 3.1. Intuicja działania tego protokołu jest następująca: wielomiany P_1, P_2 i P_3 są stopnia $k_C - 1$, zatem równość pomiędzy dwoma wielomianami (dla dowolnych argumentów) nie może zachodzić w więcej niż $k_C - 1$ punktach. Czyli dla dowolnych $i, j \in \{1, 2, 3\}$ zachodzi:

$$|(x, y) : (P_i(x) = P_j(y)) \cap (x, y \in D)| \leq k_C - 1$$

Jest to uzasadnienie wzięte z oryginalnej pracy – [FNP04].
Uwaga: To stwierdzenie nie jest prawdziwe (przykład: dwa wielomiany będące prostymi równoległymi). Przypuszczam, że do naprawy tego problemu wystarczyłoby dodać losowy punkt (z R) do interpolacji tych wielomianów (może to sugerować niespójność przy definiowaniu stopnia wielomianu w protokole - patrz „uwaga" w kroku 2 protokołu 3.1). Jednak nie zajmę się tym problemem gdyż występuje w tym protokole inny poważniejszy problem (taki, dla którego nie widzę żadnego rozwiązania).

Uogólnienie protokołu 3.1 dla większych T i t. Niestety nie widzę prostego sposobu uogólnienia tego konkretnego protokołu dla większych $T > 3$ i $t > 2$.

Złożoność W kroku 3 protokołu 3.1 zachowanie **S** koresponduje do sprawdzania wszystkich kombinacji $\binom{T}{t} = \binom{3}{2}$.

3.2.1. Problem

W tej sekcji podam przykład, gdy protokół w ogóle nie kończy działania.
Dla następujących poprawnych danych wejściowych:

Wejście **Client**a:	Wejście **Server**a:
$[1, 2, 2]$, $[1, 3, 2]$	$[5, 6, 7]$

Protokół 3.1: wersja z [FNP04]

Protokół: `fuzzy-matching` „2-zgodne-z-3" („2–out–of–3")
Wejście: Zbiorem wejściowym **Client**a jest X ($|X| = k_C$), a **Server**a Y ($|Y| = k_S$), gdzie wielkość tupli jest $T = 3$. Dwie tuple są zgodne, jeśli co najmniej $t = 2$ atrybuty są takie same. Każdy atrybut należy do dziedziny D. Dziedzina tekstów jawnych szyfru R jest zdefiniowana jak w sekcji 3.1.

1. **C** wybiera klucz prywatny sk, klucz publiczny pk i *parametry* dla homomorficznego szyfru i wysyła pk i *parametry* do **S**.
2. **C** tworzy 3 wielomiany: P_1, P_2, P_3, gdzie wielomian P_j jest użyty do zakodowania j-tych atrybutów tupli: X_i^j, dla $1 \leq i \leq k_C$.
   Dla każdego i (takiego, że $1 \leq i \leq k_C$), **C** wybiera losowe $r_i \in R$ i ustawia $r_i = P_1(X_i^1) = P_2(X_i^2) = P_3(X_i^3)$.
   Za pomocą interpolacji liczy współczynniki tych wielomianów: (P_1, P_2, P_3) i przesyła ich zaszyfrowanie do **S**.
   Uwaga: Te wielomiany mają stopień $k_C - 1$ (a nie jak podane w pracy [FNP04] — k_C).
3. **S** odpowiada **C** (podobnie jak w przypadku protokołu 2.1) wysyłając:
   $E_{pk}(r \cdot (P_1(Y_i^1) - P_2(Y_i^2)) + C(Y_i))$,
   $E_{pk}(r' \cdot (P_2(Y_i^2) - P_3(Y_i^3)) + C(Y_i))$,
   $E_{pk}(r'' \cdot (P_1(Y_i^1) - P_3(Y_i^3)) + C(Y_i))$,
   gdzie r, r', r'' jest za każdym razem losowe w R.
4. Jeśli **C** otrzymuje kodowanie Y_i, które jest podobne do jakiejś tupli z jej zbioru (i nie było w wyjściu **C** do tej pory) to dodaje je do swojego wyjścia.

w kroku 2 protokołu 3.1 wielomiany P_1, P_2, P_3 są zdefiniowane (przez **C**) następująco:

P_1:	P_2:	P_3:
$P_1(1) = r_1 \cap P_1(1) = r_2$	$P_2(2) = r_1 \cap P_2(3) = r_2$	$P_3(2) = r_1 \cap P_3(2) = r_2$

Następnie wykonywana jest interpolacja na tych punktach. Z przeważającym prawdopodobieństwem $r_1 \neq r_2$ (bo r_1 i r_2 są wybierane losowo), a wtedy para punktów dla P_1 nie definiuje wielomianu, a zatem nie można policzyć współczynników. Taka sama sytuacja jest dla P_3. Zatem dla właściwych danych wejściowych protokół nie może skończyć działania. Co więcej dość łatwo można zdefiniować rodzaj danych wejściowych dla, których protokół nie działa. Protokół nie działa dla danych wejściowych **Client**a o następującej własności:

$$\exists((1 \leq i \leq k_C) \cap (1 \leq j \leq k_S) \cap (j \neq i) \cap (1 \leq index \leq T)) \text{ takie, że } X_i^{index} = X_j^{index}$$

Czyli protokół nie działa, wtedy gdy istnieje taka para tupli, że dla jakiegoś indeksu wartości atrybutów są równe. Wtedy z przeważającym prawdopodobieństwo wielomian jest definiowany niewłaściwie (dla wartości wspomnianego atrybutu są przyporządkowane 2 różne wartości).

3.2.2. Podsumowanie

Opisany powyżej protokół z pracy [FNP04, str. 16-17] nie działa. Poza tym cała sekcja poświęcona temu problemowi (poza definicją) jest opisana w niejasny i chaotyczny sposób.

Podany protokół ma w złożoności komunikacyjnej (a także czasowej obu graczy) czynnik $\binom{T}{t} = \binom{3}{2}$. W pracy jako problem otwarty jest zostawione usunięcie ze złożoności komunikacyjnej czynnika $\binom{T}{t}$ lub zmiana protokołu z modelu z pasywnym przeciwnikiem do modelu z przeciwnikiem aktywnym.

3.3. Protokoły `fuzzy-matching` mojego autorstwa

W tym rozdziale zaprezentuję protokoły rozwiązujące problem „fuzzy matching" mojego autorstwa. Najpierw napiszę trochę o protokole `fuzzy-matching` korzystającym z wielopodmiotowych obliczeń bazujących na układach arytmetycznych. Następnie zaprezentuję protokół realizujący `fuzzy-matching` poprawiający błędy protokołu z rozdziału 3.2 (ale nadal posiadający w złożoności czynnik $\binom{T}{t}$) działający dla dowolnych T i t. Opiszę również pewne ulepszenia tego protokołu. Wszystkie protokoły są zaprezentowane w modelu z przeciwnikiem pasywnym.

3.3.1. Protokół `fuzzy-matching` – rozwiązanie bazujące na układzie arytmetycznym

Jest dobrze znanym faktem, że każda funkcja, która może być reprezentowana jako obwód arytmetyczny, może być zrealizowana wielopodmiotowo dla 2 graczy ze złożonością komunikacyjną: $O(|C| \cdot param)$, gdzie C jest układem arytmetycznym ją realizującym, $|C|$ liczbą bramek w tym układzie, a $param$ parametrem bezpieczeństwa. Powyższe twierdzenie (i konstrukcja rozwiązania) jest podane w wielu pracach (np.: [GF02]).

Jak wygląda układ rozwiązujący problem `fuzzy-matching`?
W kolejnych akapitach opiszę wygląd tego rozwiązania.

Następujący algorytm rozwiązuje problem `fuzzy-matching`:

Dla każdej pary: (X_i, Y_j), dla $1 \leq i \leq k_C$ i $1 \leq j \leq k_S$:

1. algorytm sprawdza czy X_i i Y_j są zgodne na co najmniej t atrybutach (spośród T atrybutów);
2. jeśli X_i i Y_j są zgodne to dodaje Y_j do wyniku $\mathbf{C}$;

Złożoność komunikacyjna tego algorytmu wynosi $O(k_C \cdot k_S \cdot T \cdot |D|)$ bitów przesłanych. Jeśli interpretować $T \cdot |D|$ jako długość słowa to wtedy złożoność można zapisać jako $O(k_C \cdot k_S)$ (w sensie operacji na słowach).

Liczba bramek, potrzebnych do zasymulowania tego algorytmu to jego złożoność w sensie O, czyli: $O(k_C \cdot k_S \cdot T \cdot |D|)$ bramek. Zatem obliczenie tej funkcjonalności (`fuzzy-matching`) zajmuje czas $O(k_C \cdot k_S \cdot T \cdot |D| \cdot parametr\_bezpieczenstwa)$, gdzie $parametr\_bezpieczenstwa$ jest parametrem bezpieczeństwa. Problemem tego rozwiązania jest jednakże jego współczynnik ukryty w O (jest on bardzo duży). To rozwiązanie można traktować jako punkt odniesienia.

3.3.2. Poprawiony protokół `fuzzy-matching`

W tym rozdziale zaprezentuję protokół, który jest poprawioną wersją protokołu z pracy [FNP04] (w tym sensie, że jest zainspirowany przez protokół z tej pracy). To rozwiązanie ma

czynnik $\binom{T}{t}$ w złożoności komunikacyjnej. Ten protokół działa dla dowolnego T i t. Jest on przedstawiony na rysunku: protokół 3.2.

W tym protokole są użyte podobne techniki do użytych w protokole 2.1.

Protokół 3.2:

Protokół: `fuzzy-matching` wersja_1
Wejście: Zbiorem wejściowym **Client**a jest X ($|X| = k_C$), a **Server**a Y ($|Y| = k_S$), gdzie wielkość tupli jest T. Dwie tuple są zgodne, jeśli co najmniej t atrybuty są takie same. Każdy atrybut należy do dziedziny D. Dziedzina tekstów jawnych homomorficznego szyfru – R jest zdefiniowana jak w 3.1.

1. **C** wybiera klucz prywatny sk, klucz publiczny pk i *parametry* dla homomorficznego szyfru i wysyła pk i *parametry* do **S**.
2. Dla każdej kombinacji t elementów spośród T elementów: $K_i = \{K_i^1, K_i^2 ..., K_i^t\}$, dla $i \in \{1, ..., \binom{T}{t}\}$ (traktując i jako indeks kombinacji jednoznacznie identyfikujący kombinację), gdzie $K_i^j \in \{1, ..., T\}$ (dla $1 \leq j \leq t$), **C** wykonuje następujące czynności:
   (a) konstruuje wielomian:
   $P_i(x) =$
   $(x - C(X_1^{K_i^1}, X_1^{K_i^2}, ..., X_1^{K_i^t})) \cdot (x - C(X_2^{K_i^1}, X_2^{K_i^2}, ..., X_2^{K_i^t})) \cdot ... \cdot (x - C(X_{k_C}{}^{K_i^1}, X_{k_C}{}^{K_i^2}, ..., X_{k_C}{}^{K_i^t}))$
   (b) Ten wielomian ma stopień k_C. Za pomocą interpolacji **C** liczy współczynniki tego wielomianu i przesyła ich zaszyfrowanie do **S**.
3. dla każdego $Y_i \in Y$ (dla $1 \leq i \leq k_S$), **S** wykonuje następujące czynności:
   (a) dla każdego otrzymanego wielomianu $E_{pk}(P_j)$, dla $j \in \{1, ..., \binom{T}{t}\}$ (wielomian reprezentuje jakąś kombinację K_j):
      i. ewaluuje wielomian $E_{pk}(P_i)$ w punkcie $a_i^j = C(Y_i^{K_j^1}, Y_i^{K_j^2}, ..., Y_i^{K_j^t})$ i liczy: $E_{pk}(w_i^j) = E_{pk}(r * P_j(a_i^j) + C(Y_i))$, gdzie $r \in R$ jest wybierane za każdym razem losowo.
      ii. wysyła $E_{pk}(w_i^j)$ do **C**.
4. **C** odszyfrowuje otrzymane wiadomości i jeśli odszyfrowana wiadomość w_i^j należy do D^T i jest podobne do jakiejś tupli ze zbioru X (i nie pojawiło się do tej pory) to dodaje w_i^j do swojego wyjścia.

Poprawności i bezpieczeństwo protokołu 3.3

W tym rozdziale przedstawię argumenty dowodzące poprawności i bezpieczeństwa protokołu 3.2.

Poprawność protokołu. W tym akapicie podam intuicję poprawności protokołu. **Client** w tym protokole tworzy $\binom{T}{t}$ wielomianów stopnia k_C. Każdy z wielomianów reprezentuje jakąś kombinację t atrybutów spośród T atrybutów, a w pewnym sensie również możliwości podobieństwa. Miejscami zerowymi wielomianu są odpowiadające danej kombinacji atrybuty słów (tupli) z X (kodowane jako liczby). Zatem gdy w zbiorze Y jest element Y_j podobny

do jakiegoś $X_i \in X$, to w kroku 3(a)i wykonania protokołu wartość wielomianu w „punkcie podobieństwa" (jakieś kombinacji, która jest taka sama dla obu tupli) jest 0, a wtedy $C(Y_j)$ jest przesłane do klienta. Jeśli natomiast te elementy nie są podobne, to do klienta zostaje wysłana losowa wartość.

Teraz podam lematy formalnie definiujące poprawność i bezpieczeństwo protokołu 3.2.

Lemat 1 ***(Poprawność)*** *Protokół* `fuzzy-matching` *wersja_1 (protokół 3.2) liczy funkcję* `fuzzy-matching` *z przeważającym prawdopodobieństwem.*

Dowód:

W świecie idealnym gdzie **TTP** pobiera zbiory od graczy ,a następnie liczy wynik i przesyła go do **Client**a, zachodzi następująca własność: jeśli dla dowolnego Y_j $(1 \leq j \leq k_S)$ istnieje X_i (dla $1 \leq i \leq k_C$) takie, że Y_j i X_i są podobne, oznaczane $Y_j \approx_t X_i$ (co najmniej t atrybutów jest równych), to Y_j należy do wyjścia **Client**a. W przeciwnym przypadku Y_j nie należy do wyjścia **Client**a (wynika to z definicji wyjścia klienta).

W świecie rzeczywistym jeśli dla elementu Y_j istnieje element X_i taki, że $X_i \approx_t Y_j$ (dla $1 \leq i \leq k_C$ i $1 \leq j \leq k_S$) to istnieje kombinacja t elementów K_z ($K_z = \{K_z^1, ..., K_z^t\}$, gdzie $K_z^l \in \{1, ..., T\}$ dla $1 \leq l \leq t$), która jest taka sama w X_i i Y_j. Zatem jeden z wielomianów P_z $(z \in \{1, ..., \binom{T}{t}\})$ ma miejsce zerowe będące:

$$kombinacja = C\left(X_i^{K_z^1}, X_i^{K_z^2}, ..., X_i^{K_z^t}\right) = C\left(Y_j^{K_z^1}, Y_j^{K_z^2}, ..., Y_j^{K_z^t}\right)$$

Wtedy w kroku 3(a)i protokołu 3.2 zostaje policzone:

$$E_{pk}(w_j^z) = E_{pk}(r * P_z(kombinacja) + C(Y_j)) = E_{pk}(C(Y_j))$$

Zatem **C** otrzyma zaszyfrowane i zakodowane Y_j. Ponieważ jest ono podobne na t atrybutach do X_i (atrybuty z kombinacji K_z są takie same) to zostanie ono dołączone do wyniku. Nie zostaje dodane do wyniku tylko wtedy, gdy zostało już dołączone do niego wcześniej, a to znaczy, że jest podobne również do innego elementu z X.

W świecie rzeczywistym jeśli dla danego Y_j (dla $1 \leq j \leq k_S$) dla wszystkich X_i (dla $1 \leq i \leq k_C$): $X_i \not\approx_t Y_j$ to nie istnieje kombinacja elementów Y_j będąca miejscem zerowym jakiekolwiek wielomianu P (zgodnie z definicją wielomianów P). Zatem dla każdej kombinacji K_z $(z \in \{1, ..., \binom{T}{t}\})$ wartość:

$$dowolna\_kombinacja = C\left(Y_j^{K_z^1}, Y_j^{K_z^2}, ..., Y_j^{K_z^t}\right)$$

nie jest pierwiastkiem żadnego z wielomianów P. Zatem w kroku 3(a)i protokołu 3.2 zawsze zostaje policzony szyfrogram losowej wiadomości:

$$E_{pk}(w_j^z) = E_{pk}(r * P_z(dowolna\_kombinacja) + C(Y_j))$$

gdyż $P_z(dowolna\_kombinacja) > 0$ i $r \in R$ jest losowe. Zatem dla $Y_j \notin X$ **Client** uzyskuje tylko losowe wartości.

Jakie jest prawdopodobieństwo, że jedna z losowych wiadomości będzie należała do X „przez przypadek"? Prawdopodobieństwo, że jedna losowa wartość z R należy do D^T jest zaniedbywalne. Zatem można dobrać parametr bezpieczeństwa tak, by prawdopodobieństwo, że dla wszystkich losowych wiadomości z R przesłanych podczas działania protokołu, przynajmniej jedna należała do D^T było zaniedbywalne.

Zatem protokół 3.2 liczy funkcję `fuzzy-matching` z przeważającym prawdopodobieństwem.

□

Bezpieczeństwo protokołu. W tym podrozdziale pokażę bezpieczeństwo protokołu 3.2 w modelu pasywnym (pokażę bezpieczeństwo **Client**a i **Server**a). Bezpieczeństwo tego protokołu definiuje i dowodzi się podobnie jak dla protokołu `PROTOKÓŁ-SI-PASYWNY` (który jest przedstawiony w sekcji 2.1.4).

Intuicyjnie protokół jest bezpieczny, bo uczestnicy nie dowiadują się nieuprawnionych informacji:

- **Server** otrzymuje tylko zaszyfrowane wiadomości (zatem nie dowiaduje się niczego);
- **Client** dowiaduje się tylko podobnych do jego elementów tupli, gdyż dla niepodobnych elementów otrzymuje losowe dane (jak to jest opisane powyżej w dowodzie lematu 1);

Lemat 2 ***(Bezpieczeństwo Clienta jest zachowane w protokole 3.2)*** *Jeśli schemat szyfrowania jest semantycznie bezpieczny to w protokole 3.2* ***S*** *nie jest w stanie rozróżnić sytuacji, w której* ***C*** *ma różne wejścia.*

Dowód:

Załóżmy, że istnieją takie dwa zbiory, że **S** potrafi rozróżnić te sytuacje na podstawie szyfrogramów, które otrzymuje (potrafi zdecydować w jakiej sytuacji się znajduje). Jednak taka umiejętność przeciwnika jest sprzeczna z definicją szyfru semantycznie bezpiecznego (a jest założone, że taki jest używany). Sprzeczność.
□

Lemat 3 ***(Bezpieczeństwo Servera jest zachowane w protokole 3.2)*** *W protokole 3.2 dla każdego klienta C^*, który działa w modelu rzeczywistym istnieje klient* ***C*** *działający w modelu idealnym taki, że dla każdego wejścia* ***S****: Y widok stron* ***C*** *i* ***S*** *w idealnym modelu jest obliczeniowo nierozróżnialny od widoku stron C^* i* ***S*** *w rzeczywistym modelu.*

Dowód:

Dla gracza działającego w modelu rzeczywistym C^* z wejściem X jest gracz **C** w modelu idealnym, który przesyła wejście X do zaufanej strony trzeciej **TTP**. W idealnym modelu **C** dostaje od **TTP** zbiór $X \approx_t Y$ (oznacza to zbiór elementów zbioru Y, które sązgodne na t atrybutach z jakimś elementem z X), gdzie Y to dowolne dane wejściowe wysłane przez **S**. W modelu rzeczywistym C^* dostaje $k_S \cdot \binom{T}{t}$ wartości. Jeśli dla Y_j istnieje element X_i taki, że $X_i \approx_t Y_j$ (dla $1 \leq i \leq k_C$ i $1 \leq j \leq k_S$) to $E_{pk}(Y_i)$ znajduje się wśród otrzymanych elementów (zostało to pokazane w dowodzie poprawności protokołu 3.2) i C^* dowiaduje się legalnej informacji. Inne wartości (dla „niepodobnych" do X elementów z Y), które C^* otrzymuje są losowe (zatem C^* mógłby je wygenerować sam).
□

Optymalizacje

Zauważmy, że w tej sytuacji można zastosować wszystkie optymalizacje użyte w sekcji 2.1.4, w tym najistotniejszą optymalizację – użycie zrównoważonej alokacji za pomocą haszowania (i „podziału" jednego wielomianu na $\frac{k_C}{log(log(k_C))}$ „małych" wielomianów stopnia $O(log(log(k_C)))$). Każda taka optymalizacja powinna być użyta osobno do każdego z $\binom{T}{t}$ wielomianów.

Aktywny przeciwnik

By zapewnić bezpieczeństwo w modelu aktywnym można użyć technik podobnych do tych z sekcji 2.1.5 (dla wersji zoptymalizowanej – metoda "podziel-i-wybierz"(ang.: *cut-and-choose*)) i sekcji 2.2.3 (jednak wtedy trzeba nieco przerobić protokół, żeby uzyskać odpowiednią funkcjonalność: ustawić $n = 2$ i zapewnić, by **Server** nic się nie dowiedział). Bezpieczeństwo powinno być zapewnione osobno do każdego z $\binom{T}{t}$ wielomianów (dla każdego wielomianu z osobna sytuacja jest analogiczna do sytuacji z protokołu 2.1).

Złożoność protokołu 3.2

Opisuję złożoność protokołu 3.2 bez optymalizacji i w modelu z przeciwnikiem pasywnym. Krok 1 protokołu 3.2 nie ma znaczenia dla ogólnych złożoności w sensie O.

Złożoność komunikacyjna. W kroku 2a protokołu 3.2 **Client** (**C**) przesyła $\binom{T}{t}$ wielomianów stopnia k_C, zatem w tym kroku zostaje przesłanych $O(k_C \cdot \binom{T}{t} \cdot \lambda)$ bitów (gdzie λ oznacza ilość bitów potrzebna do zapisania szyfrogramu wiadomości z R).

W kroku 3(a)ii protokołu 3.2 **Server** (**S**) wysyła dla każdego $Y_i \in Y$: $\binom{T}{t}$ wartości do **C**. Zatem w tym kroku zostaje przesłanych $O(k_S \cdot \binom{T}{t} \cdot \lambda)$ bitów.

Zatem w sumie zostaje przesłanych $O((k_S + k_C) \cdot \binom{T}{t} \cdot \lambda)$ bitów. Ponieważ $O(\lambda) = O(param\_1 \cdot log(|R|)) = O(param\_1 \cdot param\_2 \cdot log|D|)$, gdzie $param\_1$ i $param\_2$ są parametrami bezpieczeństwa, to złożoność transmisyjną można zapisać jako $O(max(k_S, k_C) \cdot \binom{T}{t} \cdot param)$ (gdzie $param$ to parametr bezpieczeństwa). Można złożoność zapisać również jako $O(max(k_S, k_C) \cdot \binom{T}{t})$ (pomijając parametr bezpieczeństwa – traktując go jako stałą).

Złożoność czasowa. W kroku 2a protokołu 3.2 **Client** (**C**) wykonuje $\binom{T}{t}$ interpolacji wielomianów stopnia k_C, a to kosztuje: $O({k_C}^2 \cdot \binom{T}{t} \cdot \lambda)$ operacji na bitach. Natomiast w kroku 4 **C** weryfikuje $\binom{T}{t}$ wartości (sprawdza czy należą one do X) co kosztuje go: mniej niż operacje w kroku 2a. Zatem złożoność czasowa **Client**a to $O({k_C}^2 \cdot \binom{T}{t} \cdot \lambda) = O({k_C}^2 \cdot \binom{T}{t})$ (pomijając parametr bezpieczeństwa i ilość bitów potrzebną do zapisania tupli).

Całkowita złożoność czasowa **Server**a pochodzi z kroku 3(a)i, gdy **S** ewaluuje wielomiany $\binom{T}{t}$ razy (dla każdego $Y_i \in Y$), co kosztuje go w sumie: $O(k_C \cdot k_S \cdot \binom{T}{t} \cdot \lambda) = O(k_C \cdot k_S \cdot \binom{T}{t})$.

Zatem w obu złożonościach czasowych oprócz k_C i k_S występuje $\binom{T}{t}$.

Podsumowanie

Protokół zaprezentowany w tej sekcji (3.3.2) jest zainspirowany niedziałającym protokołem z pracy [FNP04] (zaprezentowany w tej pracy w sekcji 3.2). Jest protokołem do którego będę się odnosił w kolejnym protokole zaprezentowanym w tej pracy.

Czas jego działania wynosi (złożoność komunikacyjna): $O(max(k_S, k_C) \cdot \binom{T}{t})$, czyli dla odpowiednich danych (małej ilości błędów) może być mniejsza od złożoności O „naiwnego" protokołu bazującego na układzie arytmetycznym (sekcja 3.3.1), która wynosi $O(k_C * k_S)$. Jednak dla „naiwnego" protokołu nie należy zapominać o dużym współczynniku ukrytym w notacji O.

3.3.3. Alternatywny protokół `fuzzy-matching`

Protokół zaprezentowany w tym rozdziale jest alternatywną wersją protokołu z sekcji 3.3.2. Czynnik $\binom{T}{t}$ został usunięty ze złożoności komunikacyjnej (ale zastąpiony zostaje

czynnikiem $T \cdot min(k_C, k_S)$ lub jeśli osłabić nieco bezpieczeństwo czynnikiem: $min(k_C, k_S)$). Ten protokół działa dla dowolnego T i t. Jest on przedstawiony na rysunku: protokół 3.3.

Dziedzina szyfru homomorficznego R'

Dziedzina szyfru homomorficznego R' (używana w kroku 1 protokołu 3.3), ze względu na bezpieczeństwo musi się równać dziedzinie dzielonych kluczy prywatnych i zawierać w sobie D (by wielomiany były jednoznacznie zdefiniowane). Dla uproszczenia i bez straty ogólności zakładam, że $R' = Z_p$ (gdzie p jest pierwsze). Zatem definicja ta w pewnym stopniu definiuje również R (gdyż definiuje dziedzinę dzielonych kluczy prywatnych). Jednak nie jest to tak istotne, gdyż jeśli dziedzina R' jest za duża to można założyć, że przy (T, t)–deszyfrowaniu jest użyta tylko część bitów z ssk_i (dla $1 \leq i \leq T$), np.: mniej znacząca połowa bitów. Dzięki temu policzona w kroku 3(a)iiA zaszyfrowana wartość wygląda dla **C** zupełnie losowo (przynajmniej dopóki **C** nie wykona (T, t)–odszyfrowania).

Dzielenie sekretu

W krokach 3(a)i i 3(b)i protokołu 3.3 użyty jest szyfr umożliwiający (T, t)-odszyfrowanie. Zamiast tego może być użyty dowolny schemat dzielenia sekretu (ang.: *secret sharing*) lub weryfikowalnego dzielenia sekretu (ang.: *veryfing secret sharing - VSS*) o następujących własnościach:

- sekret (w tym przypadku Y_z) jest dzielony na T dzielonych sekretów (w protokole 3.3 dzielenie sekretu odbywa się w kroku 3(a)i).
- t dzielonych sekretów wystarcza do rekonstrukcji sekretu (w tym przypadku Y_z), w przeciwnym przypadku (jeśli dysponuje się mniej niż t właściwymi sekretami) nie można zrekonstruować sekretu i jest to możliwe do stwierdzenia – w protokole 3.3 rekonstrukcja sekretu odbywa się w kroku 3(a)i.

Używając do dzielenia sekretu szyfru z (T, t)-odszyfrowaniem podział sekretu polega na stworzeniu T dzielonych kluczy prywatnych (t z nich jest potrzebne do wykonania operacji odszyfrowania). Natomiast odtwarzanie sekretu polega na odszyfrowaniu wiadomości używając t dzielonych kluczy prywatnych (**C** sprawdza wszystkie kombinacje t dzielonych kluczy spośród T otrzymanych od **S** wiadomości). W przypadku, gdy któryś z t kluczy przy odszyfrowaniu jest nieprawidłowy (jest losowy - bo tak jest wybierany przez **S**) to wynik odszyfrowania jest pseudolosową wiadomością (przeciwnik - modelowany przez wielomianowy algorytm nie powinien odróżniać tej wiadomości od losowej) w zbiorze tekstów jawnych szyfru. Prawdopodobieństwo , że losowa wiadomość z R należy do dziedziny do której należą sekrety ($Y_z \in D^T$) jest zaniedbywalne.

Jednak co do tej pory jest uzyskane to tylko pseudolosowość wiadomości. Jeśli to jest wystarczającym warunkiem bezpieczeństwa (przyjęcie, że w tym przypadku pseudolosowość oznacza losowość) to protokół może być wydajniejszy bo dziedzina ssk_i może być relatywnie mała – porównywalna z D i parametrem bezpieczeństwa.

By uzyskać pełną losowość trzeba użyć kluczy długości co najmniej wielkości szyfrowanej wiadomości (dziedzina wiadomości: D^T), albo użyć schematu dzielenia sekretu takiego jak np.: schemat Shamira (opisany w [RC99]). Ale wtedy dziedzina udziałów – dzielonych kluczy jest proporcjonalna do D^T.

Będę zakładał przy dowodach poprawności i bezpieczeństwa protokołu 3.3, że wiadomość uzyskana po odszyfrowaniu przy użyciu t kluczy, gdy przynajmniej jeden z nich jest

losowy, jest losowa (opisałem w poprzednim akapicie jak to można osiągnąć). A skoro prawdopodobieństwo, że losowa wiadomość należy do dziedziny do której należą sekrety ($Y_z \in D^T$) jest zaniedbywalne to **C** potrafi stwierdzić, czy używał dobrych udziałów, czy nie. Zatem **C** wie, czy udało mu się zrekonstruować sekret.

Ulepszenie. Potencjalnym ulepszeniem protokołu może być użycie innego protokołu dzielenia sekretu. Ulepszeniem protokołu byłoby użycie schematu, który umożliwiałby weryfikację czy **C** dysponuje t poprawnymi udziałami (spośród T) szybciej niż sprawdzając wszystkie kombinacje ($\binom{T}{t}$ kombinacji).

Reprezentacja protokołu za pomocą macierzy

Tutaj zamieszczam matematyczny opis niektórych kroków protokołu 3.3. Robię to, gdyż uważam, że to w dobry sposób wizualizuje ten protokół.

Tutaj prezentuję opis macierzowy kroku 2:

$$\begin{bmatrix} X_1^1 & X_1^2 & \cdots & X_1^T \\ X_2^1 & X_2^2 & \cdots & X_2^T \\ \vdots & \vdots & \ddots & \vdots \\ X_{k_C}^1 & X_{k_C}^2 & \cdots & X_{k_C}^T \end{bmatrix} \overset{E_{pk}(.)}{\Longrightarrow} \begin{bmatrix} E_{pk}(X_1^1) & E_{pk}(X_1^2) & \cdots & E_{pk}(X_1^T) \\ E_{pk}(X_2^1) & E_{pk}(X_2^2) & \cdots & E_{pk}(X_2^T) \\ \vdots & \vdots & \ddots & \vdots \\ E_{pk}(X_{k_C}^1) & E_{pk}(X_{k_C}^2) & \cdots & E_{pk}(X_{k_C}^T) \end{bmatrix} \overset{\text{Server}}{\Longrightarrow}$$

Tutaj prezentuję opis macierzowy kroku 3a:

$$\forall_{Y_z \in Y}: \left(\begin{bmatrix} E_{pk}(r_1^1 \cdot (X_1^1 - Y_z^1) + ssk_1^1) & \cdots & E_{pk}(r_1^T \cdot (X_1^T - Y_z^T) + ssk_1^T) \\ E_{pk}(r_2^1 \cdot (X_2^1 - Y_z^1) + ssk_2^1) & \cdots & E_{pk}(r_2^T \cdot (X_2^T - Y_z^T) + ssk_2^T) \\ \vdots & \ddots & \vdots \\ E_{pk}(r_{k_C}^1 \cdot (X_{k_C}^1 - Y_z^1) + ssk_{k_C}^1) & \cdots & E_{pk}(r_{k_C}^T \cdot (X_{k_C}^T - Y_z^T) + ssk_{k_C}^T) \end{bmatrix} \begin{bmatrix} E_{spk_1}(Y_z) \\ E_{spk_2}(Y_z) \\ \vdots \\ E_{spk_{k_C}}(Y_z) \end{bmatrix} \right) \overset{\text{Client}}{\Longrightarrow}$$

Poprawność i bezpieczeństwo protokołu

W tej sekcji przedstawię argumenty dowodzące poprawności i bezpieczeństwa protokołu 3.3.

Poprawność protokołu. W tym akapicie podam intuicję poprawności protokołu 3.3. **Client** w protokole szyfruje wszystkie słowa (tuple) atrybut po atrybucie i przesyła do **Server**a. **Server** dla każdego słowa z jego zbioru: dzieli to słowo na T udziałów i dla każdego otrzymanego od **C** słowa liczy $E_{pk}(r * (X_i^j - Y_z^j) + udzial)$ i przesyła uzyskane szyfrogramy od **C**. Jeśli $X_i^j = Y_z^j$ to **C** otrzymuje dobry udział, a w przeciwnym przypadku losową wartość (bo r jest losowe w R'). Jednak **C** nie potrafi rozróżnić w jakiej sytuacji się znalazł (prawdziwy udział i losowa wartość wyglądają dla niego nierozróżnialnie). Następnie **C** sprawdza wszystkie możliwości t elementów spośród T: sprawdza czy jakiekolwiek t udziałów jakie otrzymał wystarcza do otrzymania sekretu – Y_z. Jeśli wystarcza to wie, że Y_z jest podobne do jakiegoś słowa z jego zbioru, w przeciwnym przypadku: wszystkie wiadomości jakie otrzymał wyglądają dla niego losowo.

Uwaga. Dla bezpieczeństwa jest istotny warunek dotyczący deszyfrowania z dzielonymi kluczami prywatnymi. Mianowicie istotne jest by podczas procesu (T, t)-odszyfrowania wiadomość odszyfrowana była losowa, jeśli chociaż jeden klucz prywatny był losowy. W paragrafie: „Dzielenie sekretu" opisałem tą sytuację. W dowodach będę zakładał losowość wiadomości po odszyfrowaniu, jeśli przynajmniej jeden z kluczy był losowy.

Teraz podam lematy formalnie definiujące poprawność protokołu 3.3.

Lemat 4 ***(Poprawność)*** *Protokół 3.3 liczy funkcję* `fuzzy-matching` *z przeważającym prawdopodobieństwem.*

Dowód:

W świecie idealnym gdzie **TTP** pobiera zbiory od graczy (a następnie liczy wynik i przesyła go do **Client**a), zachodzi następująca własność: jeśli dla dowolnego Y_j $(1 \leq j \leq k_S)$ istnieje X_i (dla $1 \leq i \leq k_C$), takie że $Y_j \approx_t X_i$, to Y_j należy do wyjścia **Client**a. W przeciwnym przypadku Y_j nie należy do wyjścia **Client**a (wynika to z definicji wyjścia klienta).

W świecie rzeczywistym jeśli dla elementu Y_j istnieje element X_i taki, że $X_i \approx_t Y_j$ (dla $1 \leq i \leq k_C$ i $1 \leq j \leq k_S$) to w kroku 3a protokołu 3.3 **Server** przesyła do **Client**a następujący ciąg szyfrogramów:

$$\left(E_{pk}(r_i^1 \cdot (X_i^1 - Y_j^1) + ssk_i^1), E_{pk}(r_i^2 \cdot (X_i^2 - Y_j^2) + ssk_i^2), \cdots E_{pk}(r_i^T \cdot (X_i^T - Y_j^T) + ssk_i^T)\right)$$

Spośród tych szyfrogramów co najmniej t jest postaci: $E_{pk}(ssk_i^g)$ (gdzie g jest jedną z t pozycji i $g \in \{1, ..., T\}$), gdyż co najmniej t atrybutów jest równych. Zatem również zachodzi $X_i^g - Y_j^g = 0$. Jeśli $X_i^g - Y_j^g \neq 0$ to **C** dostaje na pozycji g losową wartość. Czyli **C** dostaje ciąg wartości, które są albo postaci $E_{pk}(ssk_i^g)$ (jest ich co najmniej t), albo postaci $E_{pk}(losowe)$. Po odszyfrowaniu **C** dysponuje co najmniej t poprawnymi kluczami, które są postaci: ssk_i^g (ale nie potrafi ich odróżnić od losowych wartości – jest to opisane w sekcji „Dziedzina szyfru homomorficznego R'"). Zatem sprawdzając wszystkie kombinacje odszyfrowanych wiadomości **C** uzyskuje Y_j. Nieprawidłowe odszyfrowania wyglądają losowo (bo przynajmniej jeden klucz jest losową wartością), zatem należą one ze znikomym prawdopodobieństwem do X (bo losowy element z R należy ze znikomym prawdopodobieństwem do D^T). Zatem w tym przypadku Y_j zostanie dodane do wyjścia **C** (chyba, że wcześniej zostało dodane do wyjścia bo było podobne do innego elementu z X).

W świecie rzeczywistym jeśli dla danego Y_j (dla $1 \leq j \leq k_S$) dla wszystkich X_i (dla $1 \leq i \leq k_C$): $X_i \not\approx_t Y_j$, to **C** otrzymuje od **S** $k_C \cdot (T + 1)$ wiadomości dotyczących Y_i. $T + 1$ wiadomości dotyczy „podobieństwa" między dowolnym X_i oraz Y_j (wysłane są one w kroku 3a). Te wiadomości to T szyfrogramów postaci:

$$\left(E_{pk}(r_i^1 \cdot (X_i^1 - Y_j^1) + ssk_i^1), E_{pk}(r_i^2 \cdot (X_i^2 - Y_j^2) + ssk_i^2), \cdots E_{pk}(r_i^T \cdot (X_i^T - Y_j^T) + ssk_i^T)\right)$$

oraz $E_{spk_i}(Y_j)$. By odszyfrować $E_{spk_i}(Y_j)$ **C** potrzebuje przynajmniej t dzielonych kluczy prywatnych. Ale w powyższym ciągu wiadomości mniej niż t wiadomości jest postaci: $E_{pk}(ssk_i^g)$ (gdzie g jest jedną z t pozycji i $g \in \{1, ..., T\}$), gdyż $X_i \not\approx_t Y_j$. Reszta zaszyfrowanych wiadomości jest losowa. Zatem **C** nie może odszyfrować $E_{spk_i}(Y_j)$ i wiadomość uzyskana w wyniku odszyfrowania jest losowa. Losowa wiadomość należy do D^T znikomym prawdopodobieństwem. A zatem prawdopodobieństwo, że **C** uda się uzyskać jakiś element należący do D^T podczas rekonstrukcji ((T, t)–deszyfrowania) „przez przypadek" jest zaniedbywalne.

Zatem protokół 3.3 liczy funkcję `fuzzy-matching` z przeważającym prawdopodobieństwem.

□

Bezpieczeństwo protokołu. W tym podrozdziale pokażę bezpieczeństwo protokołu 3.3 w modelu z przeciwnikiem pasywnym (pokażę bezpieczeństwo **Client**a i **Server**a). Bezpieczeństwo tego protokołu definiuje i dowodzi się podobnie jak dla protokołu `PROTOKÓŁ-SI-PASYWNY` (który jest przedstawiony w sekcji 2.1.4).

Intuicyjnie protokół jest bezpieczny, bo uczestnicy podczas działania protokołu nie dowiadują się nieuprawnionych informacji:

- **Server** otrzymuje tylko zaszyfrowane wiadomości (zatem nie dowiaduje się niczego).

- **Client** otrzymuje poprawne klucze dzielone tylko dla elementów, które są podobne do pewnych elementów ze swojego zbioru. Dla elementów ze zbioru **Server**a niepodobnych do elementów z jego zbioru, **C** otrzymuje dla każdej pary elementów X_i i Y_j mniej niż t udziałów służących do odszyfrowania Y_j (pozostałe udziały są losowe). Dysponując mniej niż t poprawnymi udziałami **Client** nie potrafi odszyfrować Y_j.

Lemat 5 ***(Bezpieczeństwo Clienta jest zachowane w protokole 3.3)*** *Jeśli schemat szyfrowania jest semantycznie bezpieczny to w protokole 3.3* ***S*** *nie jest w stanie rozróżnić sytuacji, w której* ***C*** *ma różne wejścia.*

Dowód: Załóżmy, że istnieją takie dwa zbiory będące wejściami **C**, że **S** potrafi rozróżnić te sytuacje na podstawie szyfrogramów, które otrzymuje (potrafi zdecydować w jakiej sytuacji się znajduje). Jednak taka umiejętność przeciwnika jest sprzeczna z definicją szyfru semantycznie bezpiecznego (a jest założone, że taki jest używany). Sprzeczność.
□

Lemat 6 ***(Bezpieczeństwo Servera jest zachowane w protokole 3.3)*** *W protokole 3.3 dla każdego klienta C^*, który działa w modelu rzeczywistym istnieje klient* ***C*** *działający w modelu idealnym, taki że dla każdego wejścia* ***S****: Y widok stron* ***C*** *i* ***S*** *w idealnym modelu jest obliczeniowo nierozróżnialny od widoku C^* i* ***S*** *w rzeczywistym modelu.*

Dowód:

Dla gracza działającego w modelu rzeczywistym C^* z wejściem X istnieje gracz **C** w modelu idealnym, który przesyła wejście X do zaufanej strony trzeciej **TTP** (która dostaje również wejście od **S**, liczy wynik i wysyła go do **C**). W idealnym modelu **C** dostaje zbiór $X \approx_t Y$, gdzie Y to dowolne dane wejściowe wysłane przez **S**. W modelu rzeczywistym C^* dostaje k_C+1 wartości dla każdego elementu ze zbioru Y (k_C zaszyfrowanych udziałów i jedno zaszyfrowanie odpowiadającej im tupli z Y). Jeśli dla Y_j istnieje element X_i taki, że $X_i \approx_t Y_j$ (dla $1 \leq i \leq k_C$ i $1 \leq j \leq k_S$) to **C** dostaje przynajmniej t poprawnych kluczy dzielonych potrzebnych do odszyfrowania $E_{spk_i}(Y_j)$ (jest to pokazane podczas dowodu poprawności protokołu 3.3). Zatem podczas sprawdzania wszystkich kombinacji kluczy dzielonych C^* odszyfruje $E_{spk_i}(Y_j)$ i doda Y_j do wyniku (chyba, że ta wartość już została dodana do wyniku, ale to oznacza, że Y_j jest podobne również do innego elementu z X). Dla innych wartości (dla „niepodobnych" do X elementów z Y) C^* dostaje k_C razy: $E_{spk_i}(Y_j)$ i T wiadomości reprezentujących dzielone klucze prywatne. Mniej niż t z dzielonych kluczy prywatnych jest poprawnymi kluczami (dla każdej z k_C kopii). Zatem zgodnie z definicją C^* nie potrafi odszyfrować żadnej wiadomości $E_{spk_i}(Y_j)$ posiadając niewystarczającą liczbę dzielonych kluczy prywatnych.
□

Złożoność protokołu 3.3

Opisuję złożoność protokołu 3.3. Zakładam, że dziedzina kluczy dzielonych jest proporcjonalna do D^T.

Krok 1 protokołu 3.3 nie ma znaczenia dla ogólnych złożoności w sensie O.

Złożoność komunikacyjna. W kroku 2 protokołu 3.3 **Client** (**C**) przesyła osobno zaszyfrowane wszystkie swoje atrybuty (dla wszystkich słów). Dziedzina użytego szyfru mieści w sobie dziedzinę dzielonych kluczy (czyli jest proporcjonalna do D^T). Zatem w tym kroku zostaje przesłane $O(k_C \cdot T \cdot \lambda)$ bitów, gdzie λ oznacza ilość bitów potrzebna do zapisania R'.

W kroku 3 protokołu 3.3 **Server** (**S**) odpowiada **C** wysyłając dla każdego $Y_i \in Y$: tyle bitów co w kroku 2 i jedną wiadomość $E_{spk_i}(Y_i)$ (dziedzina R). Zatem w tym kroku zostaje przesłanych:

$$O(k_S * (k_C \cdot T \cdot \lambda + \Lambda)) = O(k_S \cdot k_C \cdot T \cdot \lambda)$$

bitów, gdzie λ oznacza ilość bitów potrzebna do zapisania R', a Λ oznacza liczbę bitów potrzebna do zapisania R. Ostatnia równość wynika z faktu, że $O(\lambda) = O(\Lambda)$.

Zatem w sumie zostaje przesłanych $O(k_S \cdot k_C \cdot T \cdot \lambda)$ bitów. Ponieważ $O(\Lambda) = O(\lambda) = O(param\_1 \cdot log(|R|)) = O(param\_1 \cdot param\_2 \cdot log|D|)$, gdzie $param\_1$ i $param\_2$ to parametry bezpieczeństwa, to można złożoność transmisyjną zapisać jako $O(k_S \cdot k_C \cdot T \cdot param)$ (gdzie $param$ to parametr bezpieczeństwa), lub $O(k_S \cdot k_C \cdot T)$.

W przypadku, gdy dziedzina kluczy dzielonych jest proporcjonalna do D to złożoność komunikacyjna jest $O(k_S \cdot k_C \cdot param)$ (gdzie $param$ to parametr bezpieczeństwa), lub $O(k_S \cdot k_C)$.

Złożoność czasowa. W kroku 2 protokołu 3.2 **Client** (**C**) wysyła zaszyfrowane wszystkie swoje atrybuty (dla wszystkich słów). Kosztuje go to $O(k_S \cdot k_C \cdot T \cdot \lambda)$ operacji na bitach.

W kroku 3(b)i protokołu 3.2 **Client** (**C**) weryfikuje wszystkie kombinacje dzielonych kluczy prywatnych – sprawdza, czy odszyfrowanie wiadomości za pomocą danej kombinacji kluczy daje element należący do X. Kosztuje to go: $O(\binom{T}{t} \cdot k_S \cdot k_C \cdot T \cdot \lambda)$ operacji na bitach.

Zatem w sumie złożoność **Client**a wynosi $O(\binom{T}{t} \cdot k_S \cdot k_C \cdot T \cdot \lambda)$. Można to także zapisać jako: $O(\binom{T}{t} \cdot k_S \cdot k_C \cdot T)$ (gdyż λ występuje we wszystkich złożonościach). W przypadku, gdy dziedzina kluczy dzielonych jest proporcjonalna do D to złożoność wynosi $O(\binom{T}{t} \cdot k_S \cdot k_C)$.

Całkowita złożoność czasowa **Server**a pochodzi z kroku 3, gdy **S** dla każdego $Y_i \in Y$, liczy $k_C \cdot T$ wartości: $f_i^j = E_{pk}(r \cdot (X_i^j - Y_z^j) + ssk_i^j)$. Zatem kosztuje go w sumie: $O(k_C \cdot k_S \cdot T \cdot \lambda)$ operacji na bitach. Można to zapisać jako $O(k_C \cdot k_S \cdot T)$. W przypadku, gdy dziedzina kluczy dzielonych jest proporcjonalna do D to złożoność wynosi **Server**a wynosi $O(k_S \cdot k_C)$.

Zatem tylko w złożoności czasowej klienta występuje czynnik $\binom{T}{t}$. Dodatkowo weryfikacja może nastąpić na sam koniec protokołu (już po zakończeniu komunikacji w protokole). W paragrafie „Ulepszenie" w sekcji 3.3.3 podaję również kierunek, w którym można szukać zmniejszenia złożoności czasowej **Client**a (chodzi o przyspieszenie kroku 3(b)i - weryfikacji kluczy).

Podsumowanie

Protokół zaprezentowany w tym rozdziale (3.3.3) jest rozwiązaniem bazującym na innym pomyśle niż protokół z sekcji 3.3.2. Jest on w pewnym stopniu związany z sugestiami z pracy [FNP04] dotyczącym problemu `fuzzy-search` (jednak sugestia pochodząca z pracy [FNP04] jest bardzo chaotyczna i nie jestem pewny czego dokładnie dotyczyła).

Czas działania tego protokołu wynosi (złożoność komunikacyjna): $O(k_S \cdot k_C \cdot T)$, a przy zmniejszeniu dziedziny dzielonych kluczy prywatnych nawet $O(k_S \cdot k_C)$. Czyli jest to taka sama złożoność w sensie O jak protokołu bazującego na układzie arytmetycznym (sekcja 3.3.1). Jednak rozważając protokół bazujący na układzie arytmetycznym nie należy zapominać o dużym współczynniku ukrytym w notacji O dla tego rozwiązania.

Protokół 3.3:

Protokół: `fuzzy-matching` wersja_2
Wejście: Zbiorem wejściowym **Client**a jest X ($|X| = k_C$), a **Server**a Y ($|Y| = k_S$), gdzie wielkość tupli jest T. Dwie tuple są zgodne, jeśli co najmniej t atrybutów jest takich samych. Każdy atrybut należy do dziedziny D. R jest zdefiniowane podobnie w 3.1 i jest dziedziną homomorficznego szyfru z (n, t)–progowym deszyfrowaniem użytego w kroku 3(a)i. R' jest dziedziną szyfru homomorficznego użytego w kroku 1 (dokładnie jest opisane w paragrafie: Dziedzina szyfru homomorficznego).

1. **C** wybiera klucz prywatny sk, klucz publiczny pk i parametry *parametry* dla homomorficznego szyfru i wysyła pk i parametry do **S**.
2. Dla każdego i takiego, że $1 \leq i \leq k_C$, **C** wykonuje:
   (a) Dla każdego j takiego, że $1 \leq j \leq T$, **C** wykonuje:
      i. Liczy $c_i^j = E_{pk}(X_i^j)$, a następnie wysyła c_i^j do **S**.

   Uwaga: Ten krok jest zwizualizowany w paragrafie: Reprezentacja protokołu za pomocą macierzy.
3. Dla każdego $Y_z \in Y$ wykonywane są następujące czynności:
   (a) Dla każdego otrzymanego wiersza ($1 \leq i \leq k_C$), **S** wykonuje:
      i. Tworzy klucz publiczny spk_i i T dzielonych kluczy prywatnych ssk_i^j (dla $1 \leq j \leq T$) umożliwiających (T, t)-odszyfrowanie (jest to opisane w sekcji 1.3). Oznacza to, że by odszyfrować poprawnie wiadomość należy użyć t różnych udziałów (jeśli jakiś jest nieprawdziwy to wynik jest niepoprawny).
      Uwaga: Dyskusja na temat realizacji tego kroku znajduje się w paragrafie: Dzielenie sekretu.
      ii. Dla każdego ($1 \leq j \leq T$), **S** wykonuje:
         A. Liczy $f_i^j = E_{pk}(r \cdot (X_i^j - Y_z^j) + ssk_i^j)$, gdzie r jest każdorazowo losowe w R'. Następnie wysyła f_i^j do **S**.
      iii. Wysyła $p_z = E_{spk_i}(Y_z)$ do **S**.

      Uwaga: Ten krok jest zwizualizowany w paragrafie: Reprezentacja protokołu za pomocą macierzy.
   (b) Dla $1 \leq i \leq k_C$, **C** wykonuje:
      i. Po otrzymaniu informacji opisujących potencjalne podobieństwo X_i z Y_z wysłanych przez **S** w kroku 3a (czyli po otrzymaniu: f_i^j dla $1 \leq j \leq T$ i p_z), **C** wykonuje:
         A. Deszyfruje wszystkie f_i^j:
         $ssk'^j_i = D_{sk}(f_i^j)$ dla wszystkich j, takich, że $1 \leq j \leq T$.
         B. Dla wszystkich kombinacji t elementów spośród T elementów $K_i = \{K_i^1, K_i^2 ..., K_i^t\}$ (dla $i \in \{1, ..., \binom{T}{t}\}$) **C** wykonuje:
         deszyfruje $y' = D_{\{ssk'_{K_i^1}, ssk'_{K_i^2}, ..., ssk'_{K_i^t}\}}(p_z)$. Jeśli $y' \in X$ (i nie zostało do tej pory dodane do wyjścia **C**) to **C** dodaje y' do wyniku i kończy przeglądanie kombinacji. W przeciwnym przypadku kontynuuje przeglądanie kombinacji.
         Uwaga: Dyskusja na temat realizacji tego kroku znajduje się w paragrafie: Dzielenie sekretu.

Rozdział 4

Podsumowanie

Celem tej pracy jest przegląd metod używanych przy konstrukcji wydajnych i praktycznych protokołów obliczeń wielopodmiotowych (na przykładzie jakiegoś znanego problemu) oraz skonstruowanie protokołu rozwiązującego nowy, niezbadany jeszcze problem.

W pierwszym rozdziale (1) podaję wstęp do obliczeń wielopodmiotowych oraz definiuję inne użyte mechanizmy (jak np.: funkcje haszujące). W drugim rozdziale (2) prezentuję protokoły rozwiązujące problem części wspólnej (ang.: `set-intersection`) dla 2 oraz n graczy w modelu z przeciwnikiem pasywnym i w modelu z przeciwnikiem aktywnym. Protokoły te pochodzą z prac: [FNP04] (dla 2 graczy) i [KST05] (dla n graczy). Czyli w tym rozdziale dokonuję przeglądu technik użytych przy konstrukcji praktycznych, wyspecjalizowanych protokołów realizujących obliczenia wielopodmiotowe (na przykładzie problemu części wspólnej zbiorów – `set-intersection`).

W pracy [FNP04] jest zaprezentowany problem części wspólnej dwóch zbiorów z błędami (ang.: `fuzzy-matching`) – oznacza to, że dwa elementy zbiorów są uważane za podobne (pasujące) jeśli zgadzają się (są równe) na co najmniej t atrybutach spośród T atrybutów, które się składają na element. Moje rozważania na temat tego problemu przedstawiam w rozdziale trzecim (3). W oryginalnej pracy przedstawiony jest protokół dla problemu `fuzzy-matching` dla $T = 3$ i $t = 2$. Jako otwarte pytanie zostawione są ulepszenia tego protokołu. W tej pracy pokazuję, że ten protokół nie działa (pokazuję również dla jakiego typu danych protokół z pracy [FNP04] nie działa). W rozdziale trzecim prezentuję rozwiązanie zainspirowane protokołem z oryginalnej pracy działający dla dowolnego T i t oraz protokół rozwiązujący problem `fuzzy-matching` bazujący na innym pomyśle. W rozdziale trzecim prezentuję również krótkie rozważania na temat rozwiązania tego problemu bazującym na realizacji funkcjonalności za pomocą układu arytmetycznego. Należy zauważyć, że chociaż prezentuję protokoły w modelu z przeciwnikiem pasywnym to pierwszy podany protokół (podany w trzecim rozdziale) może być stosunkowo łatwo przerobiony by działał w modelu aktywnym (za pomocą mechanizmów przedstawionych w [FNP04] i [KST05] – opisanych przeze mnie w rozdziale drugim).

Bibliografia

[ABMM01] Andrei Z. Broder, Michael Mitzenmacher, In IEEE INFOCOM'01, strony 1454–1463, Anchorage, Alaska, April 2001, *Using multiple hash functions to improve ip lookups*,
`http://www.eecs.harvard.edu/ michaelm/NEWWORK/postscripts/iproute.pdf`

[AC] Bruce Schneier, John Wiley & Sons, Inc., *Applied Cryptography, Second Edition: Protocols, Algorthms, and Source Code in C (cloth)*

[CEGP86] David Chaum, Jan-Hendrick Evertse, Jeroen van de Graaf i Rene Peralta, In A.M. Odlyzko, editor, Advances in Cryptology – Crypto 1986, pages 200–212, Springer-Verlag, 1986, *Demonstrating possession of a discrete log without revealing it.*

[CTP] Douglas Stinson, CRC Press, CRC Press LLC, *Cryptography: Theory and Practice*

[DK04] Dominik Kamiński, 2004, *Adaptywność w wielopodmiotowych obliczeniach funkcji*,
`http://www.mimuw.edu.pl/~niwinski/Seminarium/DKaminski.pdf`

[FNP04] Michael J Freedman, Kobbi Nissim, Benny Pinkas, Advances in Cryptology - Eurocrypt '2004 Proceedings, LNCS 3027, Springer-Verlag, May 2004, pp. 1–19, *Efficient Private Matching and Set Intersection*,
`http://www.pinkas.net/PAPERS/FNP04.pdf`

[GF] Oded Goldreaich, Cambrige, *The foudations of cryptography - volume 2*

[GF02] Oded Goldreaich, October 27, 2002, *Secure Multi-Party Computation (Final (Incomplete) Draft, version 1.4)*
`http://www.wisdom.weizmann.ac.il/~oded/PS/prot.ps`

[HA] Alfred J. Menezes, Paul C. van Oorschot, Scott A. Vanstone, CRC Press, October 1996, *Handbook of Applied Cryptography*,
`http://www.cacr.math.uwaterloo.ca/hac/`

[ID04] Ivan Damgard, Cryptography Course, 2004, *Definitions and Results for Authentication Systems*
`http://www.daimi.au.dk/~ivan/Authentication.pdf`

[IDC04] Ivan Damgard, Cryptography Course, 2004, *Definitions and results for Cryptosystems*
`http://www.daimi.au.dk/~ivan/cryptosystems.pdf`

[JC97] Jan Camenisch, Technical Report 260, Dept. of Computer Science, ETH Zurich, Mar 1997, *Proof systems for general statements about discrete logarithms.*

[JKRO04] Jonathan Katz and Rafail Ostrovsky, In Advances in Cryptology – Crypto 2004, Springer-Verlag, 2004, *Round-optimal secure two-party computation*

[KSC05] Lea Kissner, Dawn Song, CRYPTO 2005, *Privacy-Preserving Set Operations*, `http://www.cs.cmu.edu/~leak/papers/set-tech-full.pdf`

[KST05] Lea Kissner, Dawn Song, Technical Report CMU-CS-05-113, Carnegie Mellon University, February 2005, *Private and Threshold Set-Intersection*, `http://reports-archive.adm.cs.cmu.edu/~anon/2004/CMU-CS-04-182.pdf`

[LKTIP] Lea Kissner, *Private and Threshold Set-Intersection*, `http://privacy.cs.cmu.edu/TIP/kissner.pdf`

[PP00] Pascal Paillier, In Proc. of Asiacrypt 2000, pages 573–84, 2000, *Public-key cryptosystems based on composite degree residuosity classes*

[RA] Rajeev Motwani, Prabhakar Raghavan, Cambridge, *Randomized Algorithms*

[RA99] Ran Canetti, 1999, *Security and Composition of Multi-party Crytptographic protocols*

[RC99] Ronald Cramer, Springer LNCS Tutorial, vol.1561, March 1999, pp. 16-62, In Lectures on Data Security - Modern Cryptology in Theory and Practice, Ivan Damgaard (Ed.), *Introduction to Secure Computation* `http://homepages.cwi.nl/~cramer/papers/CRAMER_revised.ps`

[RI04] Ronald Cramer, Ivan Damgaard Lecture Notes, 2004, *Multiparty Computation, an Introduction* `http://www.daimi.au.dk/~ivan/mpc_2004.pdf`

[SGSM84] Shafi Goldwasser, Silvio Micali, Journal of Computer and Systems Science, 28:270–99, 1984, *Probabilistic encryption*

[TCCLRR] Thomas H. Cormen, Charles E. Leiserson, Rolnald L. Rivest, WNT 2000, *Wprowadzenie do algorytmów*

[YAAB99] Yossi Azar, Andrei Z. Broder, Anna R. Karlin, Eli Upfal, SIAM Journal on Computing, 29(1):180–200, 1999, *Balanced allocations*