Wybory

Demian Banakh, Jędrzej Hodor

Problem

Dane:

- n opcji (dla uproszczenia $\{0, 1, \ldots, n-1\}$) do głosowania,
- N stron (dla uproszczenia $\{0, 1, \ldots, N-1\}$) głosujących,
- p próg wybieralności.

Założenia:

- Każda strona głosuje dowolną liczbę opcji.
- Każda strona chce, aby inne strony niedowiedziały się niczego na temat jej głosów.
- Aby opcja została wybrana, musi otrzymać co najmniej p głosów.
- Na koniec każda ze stron powinna dowiedzieć się wszystkiego o wyniku głosowania i nic poza tym (to nie znaczy, że nie dowiedzie się o głosach niektórych stron po głosowaniu: np. w przypadku 3 stron, 2 opcji i jednego głosu od każdej strony, jeśli strona A zagłosowała na pierwszą opcję, a wygrała druga, to A wie, że B i C głosowali na drugą opcję).
- Zakładamy, że strony nie oszukują, czyli zawsze postępują zgodnie z protokołem, mogą jednak być "ciekawskie" i próbować wyciągnąć informacje, których nie powinny uzyskać (model semi-honest).
- Zakładamy, że nie mniej niż połowa stron jest "uczciwa" to znaczy nie jest w zmowie i nie spróbuje wyciągnąć pewnych informacji wspólnie.

Przykłady zastosowania:

- Próbujemy wyłonić "delegację" spośród n osobowej grupy. Nie ma dla nas znaczenia wielkość delegacji, a jedynie to, aby każdy uczestnik miał poparcie co najmniej p głosujących.
- Głosowanie na to, które przedmioty dodatkowe zostaną uruchomione w następnym roku akademickim. Aby przedmiot został uruchomiony (na naszym wydziale) potrzeba co najmniej 6 osób. Ustalamy więc $p = 6$ i głosujemy.
- Wybór przekąsek na imprezę - nie chcemy kupować przekąsek, których nikt nie będzie jeść, więc ustawiamy odpowiednio wysokie p. Zakładamy, że gusta kulinarne są danymi poufnymi.

Protokół połączenia

Zakładamy w naszej implementacji, że mamy do dyspozycji zaufany serwer, który będzie przekazywał informacje pomiędzy stronami. Serwer ustala jakie opcje są dostępne, ustala ilość stron, które są oczekiwaną (i jeśli wszystko pójdzie dobrze faktyczną) liczbą uczestników. Dodatkowo serwer ustala próg wybieralności. W praktyce, aby wywołać program serwera należy wykonać komendę:

```
cargo run -p voting-server -- <N> <p> "<Opt 1>,<Opt 2>,<Opt 3>,...,<Opt n>"
// opcje podajemy oddzielone przecinkami
```

Strona ustala swoje id, zakładamy, że jest ono jedną z liczb $\{0, 1, \ldots, N-1\}$ W praktyce:

```
cargo run -p voting-system -- <id>
```

Protokół warstwy komunikacji wygląda następująco:

- Każda ze stron łączy się z serwerem informując go o swoim id.
- Serwer odsyła stronie wszystkie parametry głosowania.
- Strona oddaje głos (korzystając z UI).
- Gdy wszystkie strony oddadzą swój głos, serwer informuje o tym wszystkie strony przesyłając odpowiednią wiadomość.
- Od tej pory strony wykonują protokół BGW (omówiony w innej sekcji) używając serwera jako proxy. To znaczy, za każdym razem, gdy chcemy wysłać wiadomość, wysyłamy do serwera id odbiorcy, a następnie wiadomość (to znaczy ciąg bajtów) w postaci:

  ```
  Message { from, to, gate, share }
  ```

  gdzie *gate* oznacza numer aktualnie procesowanej bramki, natomiast *share* to wartość odpowiedniego sekretu.
- Gdy wynik jest obliczony, każda ze stron wyświetla go na swoim UI i protokół się kończy.

BGW

Protokół BGW został dokładnie omówiony na wykładzie, więc tutaj zrobimy to tylko w skrócie podkreślając, których rozwiązań użyliśmy. Protokół pozwala na wspólne obliczanie pewnego obwodu arytmetycznego nad ciałem $\mathbb{Z}_q$. Korzystaliśmy z typu *u16*, aby reprezentować liczby używane w protokole. Można policzyć, że największa liczba reprezentowana tym typem to $M = 65535$. Aby uniknąć przepełnienia arytmetycznego, należy zapewnić, że $a \cdot b \leqslant M$ (największa możliwa wartość przed wzięciem modulo q. Dlatego mając na uwadzę, że pierwiastek kwadratowy z M to około 255 użyliśmy najmniejszej większej liczby pierwszej $q = 251$.

Fundamentem protokołu BGW jest protokół "Shamir's secret sharing" bazujący na tym, że interpolacja wielomianu stopnia $t - 1$ jest możliwa wtedy i tylko wtedy, gdy znamy jego wartości w co najmniej t punktach. Każdy input i każda obliczona wartość w protokole jest współdzielona z pomocą f-SSS, gdzie f jest równe $\left\lfloor \frac{N-1}{2} \right\rfloor$. Dodawanie i mnożenie przez stałą mogą być wykonane bardzo prosto lokalnie. Jedynym problemem jest mnożenie, gdyż naiwne pomnożenie części sekretów przez siebie daje wielomian stopnia dwa razy większego. Na wykładzie przedstawiono dwie idee poradzenia sobie z tym problemem, wybraliśmy drugą. Polega ona na tym, że w fazie setup generujemy nieznany nikomu współdzielony losowy sekret r, który współdzielimy za pomocą dwóch wielomianów, jednego stopnia f, a drugiego stopnia $2f$. Następnie w momencie, gdy chcemy uzyskać f-współdzielenie iloczynu sekretów a, b wykonujemy następującą procedurę:

$$\begin{aligned} \text{dane :} & [a]_f, [b]_f \\ & [c]_{2f} = [a]_f [b]_f \\ & [g]_{2f} = [c]_{2f} + [r]_{2f} \\ & [g]_{2f} \to g \\ & [c]_f = g - [r]_f = [g - r]_f. \end{aligned}$$

Co by się stało, gdyby kilka razy użyć tego samego sekretu? Mamy wtedy $[c]_f - [c']_f = g - g'$, czyli bylibyśmy w stanie poznać różnicę miedzy dwoma sekretami, co nie jest bezpieczne. Z tego powodu należy wygenerować r dla każdej bramki mnożenia. Dlaczego wydajnie jest zrobić to w czasie setupu? Ponieważ w teorii można operacje te zrównoleglić. Nasza implementacja działa w jednym wątku, jednak jest otwarta na takie rozszerzenie.

Budowa obwodu

Załóżmy, że mamy trzy wejścia boolowskie x, y, z i chcemy sprawdzić, czy co najmniej dwa są wartościowane na 1. Nie jest trudno sprawdzić, że wtedy formuła boolowska rozwiązująca ten problem to:

$$(x \wedge y) \vee (y \wedge z) \vee (z \wedge x).$$

Całe wyrażenie jest prawdą tylko jeśli, któraś grupa dwóch zmiennych daje prawdę, czyli obie są prawdziwe. Powyższą prostą obserwację można uogólnić.

Lemat 1 *Niech $x_1, \ldots, x_k \in \{0,1\}$ i dla $A \subset [k]$ mamy: $\varphi_A = \bigwedge_{i \in A} x_i$. Wtedy dla $t \leqslant k$ istnienie conajmniej t indeksów i takich, że $x_i = 1$ jest równoważne temu, że:*

$$\bigvee_{A \subset \binom{[k]}{t}} \varphi_A = 1.$$

Dowód: W jedną stronę wybieramy t odpowiednich zmiennych do A i otrzymujemy $\varphi_A = 1$, natomiast w drugą stronę zauważamy, że zbiór A, na którym $\varphi_A = 1$ jest zbiorem zmiennych, którego szukaliśmy. □

Konstruujemy n obwodów $O_1, \ldots, O_n$, gdzie obwód O_i będzie sprawdzał, czy opcja i dostała wystarczającą ilość głosów. Obliczając obwód O_i strona j ma input 1 wtedy, gdy wybrała opcję i, a 0 w przeciwnym przypadku. Korzystając z lematu widzimy, że obwód O_i obliczy 1 tylko jeśli na opcje i zostało oddanych conajmniej p głosów.

Protokół BGW operuje na obwodach arytmetycznych. Musieliśmy więc przekonwertować obwód boolowski na arytmetyczny. Zakładamy, że wejścia są elementami $\{0,1\}$. Wtedy bramkę $\wedge$ można zrealizować jako mnożenie. Bramka $\vee$ jest minimalnie bardziej skomplikowana. Mamy $a \vee b = a+b-a \cdot b$. Bezpośrednia zmiana mogłaby być kłopotliwa i skomplikowana. Dlatego warto zastosować pewne proste przekształcenie. Załóżmy, że chcemy zamienić boolowskie $a \vee b \vee c \vee d$ na arytmetyczne. Mamy:

$$\begin{aligned} a \vee b \vee c \vee d &= a \vee b \vee (c + d - cd) \\ &= a \vee (b + (c + d - cd) - b \cdot (c + d - cd)) \\ &= a \vee (b + c + d - bc - cd - cd + bcd) \\ &= a + (b + c + d - bc - cd - cd + bcd) - a \cdot (b + c + d - bc - cd - cd + bcd) \\ &= a + b + c + d - ab - ac - ad - bc - cd - cd + abc + acd + abd + bcd - abcd. \end{aligned}$$

Rozumowanie indukcyjne, które można odtworzyć z powyższego ciągu równości pokazuje, że:

$$a_1 \vee a_2 \vee \cdots \vee a_m = \sum_{s=1}^{m} (-1)^{s+1} \sum_{A \subset \binom{[m]}{s}} \prod_{a \in A} a.$$

Korzystając z tej równości można w zgrabny funkcyjny sposób wygenerować odpowiedni obwód arytmetyczny dla zadanych parametrów.

Rozmiar obwodu

Interesującym, może być rozmiar użytego obwodu w zależności od parametrów. Na początek spróbujmy obliczyć po prostu liczbę bramek każdego rodzaju w jednym obwodzie O_i: (oznaczmy $m = \binom{N}{p}$)

- INPUT - N bramek,
- MUL - faza liczenia koniunkcji po podzbiorach - $\binom{N}{p}$ podzbiorów razy $p-1$ rozmiar podzbioru (bramki są binarne, więc aby pomnożyć p liczb potrzebujemy $p-1$ bramek),
- MUL - faza liczenia alternatyw - $\sum_{s=1}^{m} \sum_{A \subset \binom{[m]}{s}} |A| - 1 = \sum_{s=1}^{m} \binom{m}{s} \cdot (s-1) = 2^{m-1} \cdot m - 2^m + 1$,
- CMUL - $\sum_{s=1}^{\frac{m}{2}} \binom{m}{2s} = 2^{m-1} - 1$,
- ADD - $\sum_{s=1}^{m} \binom{m}{s} - 1 = 2^m - 2$.

Wiemy, że najbardziej interesująca jest tzw. MUL-głębokość, gdyż ona jest minimalnym ograniczeniem dla paralelizacji protokołu. W naszym przypadku do MUL-głębokości najpierw wlicza się $p-1$ z fazy liczenia koniunkcji, a następnie jedynie najdłuższy składnik sumy, czyli $m-1$. Łącznie:

$$(p-1) \cdot \left(\binom{N}{p} - 1 \right).$$

Obwody dla poszczególnych opcji głosowania można obliczać równolegle, więc nie zwiększa to MUL-głębokości. Patrząc na powyższe wyrażenia widzimy, że rozmiary obwodów rosną bardzo szybko w tempie wykładniczym. Dla ilustracji pokażemy kilka wyników w poniższej tabeli:

parametry	liczba bramek
$n = 2, N = 3, p = 1$	34
$n = 2, N = 5, p = 3$	11310
$n = 2, N = 6, p = 4$	22020184

Wartość w ostatniej komórce to więcej niż 20 milionów...

Podsumowując, liczba opcji do wyboru nie wpływa istotnie na wydajność, gdyż jest to tylko liniowy czynnik multiplikatywny. Jeśli chodzi o liczbę głosujących i próg wybieralności to najlepiej, gdy albo są one zbliżone, albo, gdy prób jest bardzo niski. (Tak, aby wartość $\binom{N}{p}$ była mała.) Przy $p \sim \frac{N}{2}$, złożoność rośnie bardzo szybko.

Opis kodu

Kod programu podzieliliśmy na trzy części, omówimy tutaj każdą z nich. Pierwsza część to **voting-server**. Odpowiada za pracę "zaufanego serwera" opisanego wyżej. Znajduje się tam tylko jeden plik. Do połączenia między stronami używamy *TcpStream* z biblioteki standardowej. Najistotniejszym szczegółem implementacyjnym jest to, że rozmiar struktury *Message*, która jest przekazywana między stronami może się różnić zależnie od lokalnej maszyny. Dlatego przy każdej wiadomości poza swoim id, strony wysyłają również rozmiar wiadomości.

Serwer jest jednorazowy, to znaczy, że po wykonaniu protokołu, przed następnym wykonaniem należy go zresetować.

Następną częścią projektu jest **voting-system**. Ta część odpowiada za UI klienta i wywołanie protokołu MPC. W pliku *main* nawiązujemy połączenie z serwerem i tworzymy UI za pomocą biblioteki *druid*. Plik *util* odpowiada za stworzenie odpowiedniego obwodu zależnego od parametrów, zawiera również kilka testów sprawdzających, czy zgenerowany obwód reprezentuje oczekiwaną funkcjonalność. Pozostałe pliki są szczegółami implementacyjnymi szczególnymi dla biblioteki *druid*. Odpowiadają za wywołanie protokołu BGW i związanych z nim obliczeń. Obliczenia wywoływane są asynchronicznie, tak, aby UI nie był zablokowany. Klient jest informowany o zakończeniu obliczeń przy użyciu zdarzenia typu *Command*.

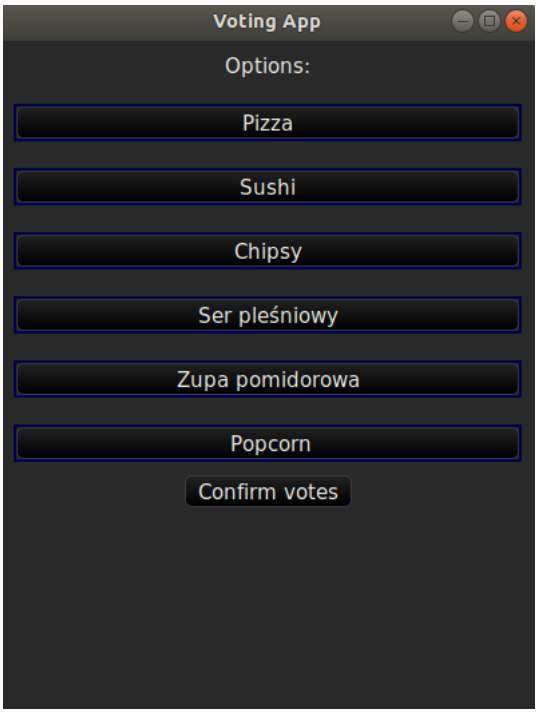
Ostatnia, najważniejsza i najobszerniejsza część to **mpc**, która zawiera implementację BGW. W pliku *field* definiujemy operacje na ciele Galois. Plik *gate* zawiera definicję struktury reprezentującej różne typy bramek. W pliku *circuit* jest implementacja obwodu - ponieważ obsługa struktur drzewiastych w języku *Rust* jest nieco trudna, używamy podejścia *arena tree* (identyfikacja dzieci przez indeksy, a nie bezpośrednie referencje). Plik *message* definiuje format wiadomości przesyłanych między stronami. Plik *polynomial* udostępnia narzędzia związane z generowaniem losowych wielomianów, oraz interpolacją. Najistotniejszy jest plik *party*, w którym definiujemy strukturę reprezentującą stronę. Strona korzysta z interfejsu do wysyłania i otrzymywania wiadomości, implementację którego udostępnia voting-system. Strona w sposób zgodny z BGW procesuje po kolei bramki obwodu. Aby żadne wiadomości się nie gubiły i nie trzeba było polegać na kolejności ich wysyłania zaimplementowaliśmy "cache", które przechowuje wszystkie wiadomości, które nie są aktualnie potrzebne, ale być może przydadzą się w przyszłości (patrz metoda *safe_recv*).

Bezpieczeństwo

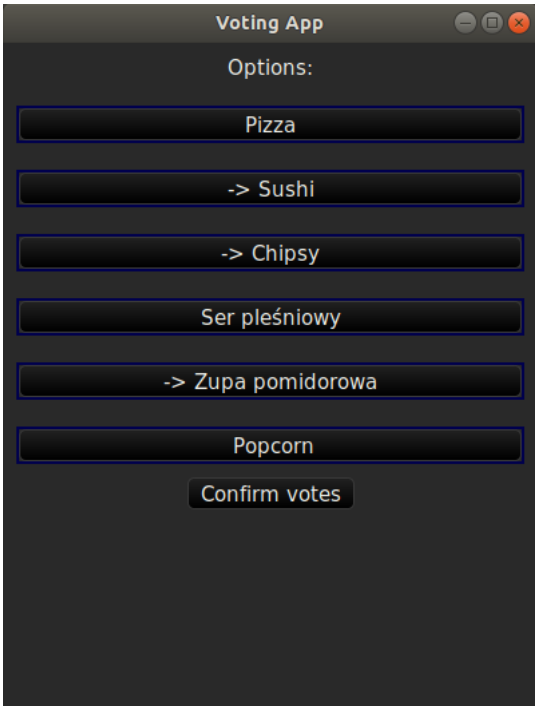
Przy założeniu modelu semi-honest, braku oszustw ze strony serwera, oraz bezpieczeństwa *TcpStream* nasz system jest bezpieczny głównie z powodu tego, że algorytm BGW jest bezpieczny. Teoretycznie każdy może "podszyć" się za głosują stronę, jednak jest to problem równoważny problemowi inputu. Dopóki nie mamy zdefiniowanych uczestników protokołu dopóty każdy może nimi być.

Gdyby zebrało się $f + 1$ partii chcących wyciągnąć jakieś informacje z protokołu i skłonnych do współpracy to wtedy interpolując odpowiednie wielomiany byliby w stanie to zrobić. Każda mniejsza liczba stron nie ma na to szans.

Interfejs użytkownika

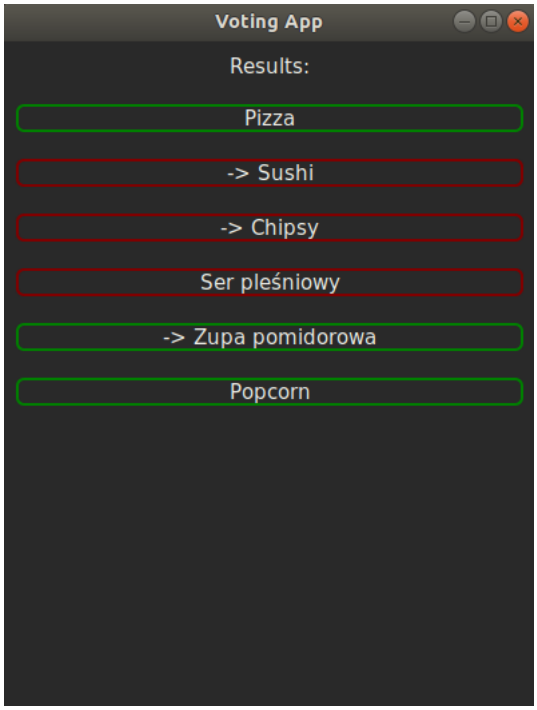


Rysunek 1: Tuż po włączeniu programu.



Rysunek 2: Wybrane opcje zostają zaznaczone strzałką.

Rysunek 3: Po potwierdzeniu głosów, czekamy na wynik. W tym momencie klikanie przycisków nie wywołuje żadnego efektu.



Rysunek 4: Zielony oznacza zwycięstwo danej opcji, natomiast czerwony przegraną.