Secure Card Game

Jan Klimczak
Filip Konieczny
Rafał Pyzik

24.06.2024

1 Opis projektu

Celem projektu jest implementacja podstawowych funkcjonalności związanych ze wszelkiego rodzaju grami z ustaloną talią kart, takimi jak:

1. Tasowanie talii kart,
2. Rozdawanie kart,
3. Ciągnięcie karty ze stosu,
4. Zagrywanie karty.

Oczywiście chcemy, aby wszystkie funkcjonalności były kryptograficznie bezpieczne, co między innymi implikuje, że nikt nie może ustalić kolejności kart w talii oraz by nikt nie posiadał informacji prócz tego, co wydarzyło się w grze i własnych kart na ręce.

Powyższe funkcjonalności zaprezentowane są na przykładzie prostej gry przypominającej wojnę. Przebieg rozgrywki jest następujący:

1. Talia jest tasowana.
2. Każdy z graczy ciągnie z talii 5 kart.
3. Gracze kolejno rzucają karty na stół.
4. Wygrywa karta, która ma najwyższą wartość wśród wszystkich kart w kolorze pierwszej rzuconej karty (wartości kart są standardowe 2, 3, ..., 10, J, Q, K, A).
5. Gracz, którego karta wygrała zdobywa punkt i będzie zaczynał następną turę.
6. Jeśli stos kart jest niepusty, każdy ciągnie kartę, zaczynając od osoby, którego karta wygrała.

2 Kryptografia

Karty są reprezentowane jako punkty na krzywej eliptycznej. Niech G to generator grupy na której pracujemy. Wtedy karty to $C = \{G, 2G, \ldots, 52G\}$, gdzie stosujemy notację addytywną. W wyniku procesu tasowania każdy z graczy zaszyfruje karty swoim kluczem prywatnym, w wyniku czego karta to $x_1 x_2 \ldots c$, gdzie $c \in C$, a x_i to klucze prywatne graczy (będące skalarami).

Oprócz standardowego założenia, że CDH jest trudny na naszej krzywej eliptycznej, dodatkowo zakładamy to także o decyzyjnej wersji. W szczególności, użyta w protokole krzywa eliptyczna nie może wspierać dwuliniowego parowania.

2.1 Protokoły

2.1.1 Tasowanie

Tasowanie to najbardziej skomplikowany z użytych protokołów. Wysokopoziomowo, każdy z graczy aplikuje do talii wybraną przez siebie permutację. Widzimy, że jeśli gracz nie jest w stanie poznać permutacji innego gracza, to gracz nie jest w stanie poznać końcowej permutacji talii (lepiej niż zgadując).

Potasowana talia to ciąg $(x_1^1 x_2^1 \dots c_1, x_1^2, x_2^2 \dots c_2, \dots, x_1^k x_2^k \dots c_k)$, gdzie c_i to permutacja C.

Zaimplementowaliśmy dwa protokoły. Zacznijmy od opisu prostszego:

1. Gracz A wybiera klucz prywatny A i zaszyfrowuje każdą kartę za pomocą A, a następnie tasuje talię. Każda karta jest postaci Ac, gdzie $c \in C$.
2. Kolejni gracze robią to samo.
3. Gracz A wybiera 52 klucze prywatne. Następnie ściąga szyfrowanie kluczem A i szyfruje kolejne karty kolejnymi kluczami. Każda karta jest postaci $x_1^i B \dots c_i$.
4. Kolejni gracze robią to samo.

Na końcu każdy z graczy ma ciąg $(x_1^1 x_2^1 \dots c_1, x_1^2, x_2^2 \dots c_2, \dots, x_1^k x_2^k \dots c_k)$, gdzie c_i jest permutacją zbioru C.

Protokół ten wymaga założenia o trudności DDH ze względu na pierwszy krok. Gracz drugi otrzymuje $Ai \cdot G$ dla $i \in \{1, \dots, 52\}$, więc w szczególności wśród otrzymanych kart są AG oraz $A2G$. Może on przetestować wszystkie pary kart x, y czy trójka $(x, 2G, y)$ jest trójką DH, co daje mu kandydatów na odpowiednie karty. W podobny sposób może on odtworzyć całą permutację zastosowaną przez poprzednią osobę.

Przy założeniu DDH oraz modelu semi-honest adwersarza powyższy protokół realizuje zamierzony cel. Jeśli adwersarz może robić coś niezgodnie z protokołem to pojawiają się problemy. Gracz może łatwo oszukać w kroku 3 duplikując karty. Zamiast zaszyfrować swoimi kluczami otrzymane karty, to zastępuje niektóre karty z talii inną otrzymaną kartą (np. zamiast zwrócić $x_1 a, x_2 b, x_3 c$ to zwraca $x_1 a, x_2 a, x_3 c$).

Jeśli zrobi to gracz pierwszy, to protokół się nie zakończy się błędem: inni gracze nie zauważą jego ingerencji, ich klucze prywatne z kroku 2 będą wciąż działać jak powinny, podobnie jak klucze z kroku 4 (bo jeszcze ich nie zaaplikowali). Zaskutkuje to tym, że w pewnym momencie w trakcie gry jedna karta może pojawić się wielokrotnie.

W celu naprawienia tej luki, na końcu gry wszyscy ujawniają swoje klucze i deszyfrowana jest cała talia by upewnić się, że jest permutacją zbioru C. To pozwala sprawdzić legalność przebiegu rozgrywki, jednak jest to mało satysfakcjonujące rozwiązanie.

Aby poprawić wspomniane aspekty, druga wersja protokołu narzuca na graczy dostarczenie zero-knowledge dowodów, że poprawnie wykonali wszystkie kroki.

Będziemy potrzebować kilku pomocniczych podprotokołów będących wariacjami podpisu Schnorra. Poniżej stosujemy konwencję, gdzie $Hash(\dots||X_i||\dots) = Hash(\dots||X_1||\dots||X_n||\dots)$.

Masking. Gracz ma dane publiczny punkt P oraz prywatny klucz x. Publikuje on P' i chce udowodnić $P' = xP$.

1. Losuje on r i liczy $R = rP$, $e = Hash(R||P||P')$,
2. Liczy $s = r - ex$,
3. Publikuje (e, s).

Weryfikacja polega na policzeniu $R^v = sP + eP'$ i sprawdzeniu czy $e = Hash(R^v||P||P')$. Warto zauważyć, że jest to w zasadzie dokładnie podpis Schnorra.

Multimasking. Gracz ma dane publiczne punkty $(P_1, P_2, \ldots, P_n)$ oraz prywatny klucz x. Publikuje $(P'_1, P'_2, \ldots, P'_n)$ i chce udowodnić, że $P'_i = xP_i$.

1. Losuje on r i liczy $R_i = rP_i$, $e = Hash(R_i^v||P_i||P'_i)$,
2. Liczy $s = r - ex$,
3. Publikuje (e, s).

Weryfikacja polega na policzeniu $R_i^v = sP_i + eP'_i$ i sprawdzeniu czy $e = Hash(R_i^v||P_i||P'_i)$.

1-in-2 Gracz ma dane punkty A, B oraz prywatny klucz x. Publikuje on P i chce udowodnić, że $P = xA$ lub $P = xB$, bez ujawiania, który przypadek zachodzi.

Opiszemy protokół dla przypadku $P = xA$.

1. Losuje r i liczy $R_a = rA$, $e_b = Hash(R_a||P||A||B)$,
2. Losuje s_b i liczy $R_b = e_bP + s_bB$,
3. Liczy $e_a = Hash(R_b||P||A||B)$,
4. Liczy $s_a = r - e_ax$,
5. Zwraca (e_a, e_b, s_a, s_b).

Weryfikacja polega na policzeniu $R_a^v = s_aA + e_aP$ oraz $R_b^v = s_bB + e_bP$ i sprawdzeniu czy $e_a = Hash(R_b^v||P||A||B)$ oraz $e_b = Hash(R_a^v||P||A||B)$.

Swap Gracz ma dane punkty A, B oraz prywatny klucz x. Publikuje on (C, D) i chce udowodnić, że $(C, D) = (xA, xB)$ lub $(C, D) = (xB, xA)$ bez ujawniania, który przypadek zachodzi.

Wystarczy dwukrotnie zastosować protokół 1-in-2.

Korzystając z protokołów swap oraz multimaskingu można udowodnić, że rzeczywiście poprawnie wykonało się krok 1 poprzedniego protokołu. Można to osiągnąć poprzez użycie sieci sortującej na odwrót. Sieć sortująca jest w stanie posortować dowolną permutację za pomocą statycznego ciągu komparatorów. Przechodząc przez nią na odwrót i używając protokołu swap dla każdego komparatora jesteśmy w stanie dotrzeć do dowolnej permutacji.

Takie podejście daje algorytm wysyłający $\log^2 n$ razy liniowo wiele punktów i skalarów. Dla $n = 52$ logarytm z n jest względnie duży ($\log^2 n > n$). Nawet użycie najlepszej znanej sieci sortującej dla $n = 52$ daje nam 19 warstw sieci, co daje prawie tysiąc skalarów i punktów do wysłania. Zamiast tego zdecydowaliśmy się użyć rozszerzonej wersji protokołu 1-in-2, mianowicie 1-in-n, którego n-krotna aplikacja (jak w swap) pozwoli nam udowodnić cały shuffle za jednym zamachem wysyłając liniowo wiele punktów i kwadratowo wiele skalarów.

1-in-n Gracz ma dane punkty $P_1, P_2, \ldots, P_n$ oraz prywatny klucz x. Publikuje on P i chce udowodnić, że $P = xP_i$ dla pewnego i, bez ujawniania wybranego i.

Opiszemy protokół dla $P = xP_1$

1. Losuje r i liczy $R_1 = rP_1, e_2 = Hash(R_1||P||P_i)$,

2. Losuje s_2 i liczy $R_2 = e_2P + s_2P_2$,
3. Liczy $e_3 = Hash(R_2||P||P_i)$,
4. Losuje s_3 i liczy $R_3 = e_3P + s_3P_3$,
5. ...
6. Liczy $e_1 = Hash(R_n||P||P_i)$,
7. Liczy $s_1 = r - e_1x$,
8. Zwraca $(e_1, \ldots, e_n, s_1, \ldots, s_n)$,

Weryfikacja polega na policzeniu $R_i^v = e_iP + s_iP_i$ i sprawdzeniu $e_{i+1} = Hash(R_i^v||P||P_j)$.

2.1.2 Ciągnięcie z talii

Gracz ogłasza, że chce pociągnąć kartę z talii. Reszta weryfikuje, że ma on do tego prawo, a następnie publikuje swoje klucze prywatne związane z kartą na szczycie stosu. Gracz następnie może użyć wspomniane klucze prywatne oraz swój własny do poznania wartości karty.

Rozdawanie kart to wielokrotne zaaplikowanie tego protokołu.

2.1.3 Zagrywanie karty

Gracz ogłasza, którą kartę chce zagrać, a następnie publikuje swój klucz prywatny. Pozostałe klucze zostały już ujawnione, więc tożsamość karty staje się publiczna.

3 Implementacja i struktura projektu

- Do części kryptograficznej użyliśmy implementacji krzywych eliptycznych z crate'a z rodziny `arkworks`, a konkretnie krzywej Pallas. Poza tym, wyżej opisane protokoły zaimplementowane zostały przez nas.
- Rozgrywka, przez którą rozumiemy tu stany trwającej gry, zaimplementowana została jako automat skończony.
- Do komunikacji sieciowej został użyty `std::net`. Gracze komunikują się bezpośrednio między sobą przez połączenie TCP. Z uwagi na niemałe wielkości przesyłanych wiadomości został zaimplementowany prosty protokół: najpierw wysyłamy czterobajtową długość wiadomości, a następnie samą wiadomość.