Protokoły Kryptograficzne

Projekt

"Pieniądz elektroniczny"

Etap 2

Prowadzący projekt: **mgr inż. Marcin Tunia**
Semestr realizacji: **14Z**

Autorzy:
Damian Pobrotyn
Kamil Rostecki

1. Opis projektu

Tematem projektu jest pieniądz elektroniczny, który jest zdefiniowany jako wartość pieniężna przechowywana elektronicznie, z obowiązkiem jej wykupu, w celu dokonywania transakcji płatniczych, akceptowana przez podmioty inne niż wyłącznie wydawca pieniądza elektronicznego.
Naszym zadaniem jest umożliwić realizację transakcji za pomocą elektronicznego pieniądza. Transakcja taka odbywa się między dwoma stronami (Klient i Sprzedawca) przy wykorzystaniu osoby trzeciej (Bank). Wykorzystywać do tego będziemy trzy operacje: wypłaty pieniądza z konta bankowego, zapłaty za towar/usługę oraz wpłaty pieniądza na konto bankowe.

Przebieg transakcji w naszym projekcie

2. Opracowanie teoretyczne

a) Wybrany protokół

Do realizacji naszego projektu wykorzystamy protokół *"Untraceable Electronic Cash"* [1], którego autorami są: David Chaum, Amos Fiat i Moni Naor.

Protokół ten zakłady trzy strony: Alice (nasz klient), Bob (sprzedawca) oraz Bank.
Bank ustala liczbę *n*, której faktoryzację zna tylko on oraz liczbę *k*, która jest parametrem bezpieczeństwa. Bank publikuje dwie niekolizyjne funkcje *f* i *g*, które przyjmują dwa argumenty. Zgodnie z protokołem funkcja *f* działa jak czarna skrzynka, natomiast funkcja *g* jest funkcją jeden do jednego. Dodatkowo Alice posiada konto bankowe o numerze *u*, a bank posiada licznik *v* z nim związany.

W ramach naszego projektu założyliśmy, że funkcja *g* na wejściu przyjmuje dwie liczby w postaci dziesiętnej, które po zamianie na liczby binarne skleja, a jako wynik zwraca liczbę w postaci heksadecymalnej, natomiast funkcja *f* przyjmuje dwie liczby w postaci heksadecymalnej, które skleja ze sobą. Obie funkcje w celu zaszyfrowania wiadomości używają funkcji skrótu SHA-256[3], która jest dostępna w bibliotece *Security* w pakiecie *MessageDigest*[4]. Jest to funkcja skrótu działająca w jedną stronę, odporna na kolizje. Funkcja kompresji działa na 512-bitowym bloku wiadomości. Wielkość bloku wyjściowego to 512 bitów, słowo wyjściowe ma 64 znaki. Liczona jest na 32-bitowych słowach, składa się z 64 rund.

b) Przebieg protokołu / Podstawy matematyczne

✓ Operacja wypłaty pieniądza

1. Alice wybiera losowo niezależne 4 wartości: a_i, c_i, d_i oraz r_i dla i € <1;k>

2. Alice tworzy i wysyła do Banku k zaszyfrowanych wiadomości B, takich że:

$B_i = r_i^3 * f(x_i, y_i) \bmod n$ dla i € <1;k>

gdzie

$x_i = g(a_i, c_i)$ $y_i = g(a_i \text{ ^ } (u \,||\, (v+i)), d_i)$

oznaczenia: **^** - oznacza alternatywę wykluczającą (xor), **||** - oznacza sklejanie (konkatenację)

3. Bank wybiera losowy podzbiór elementów o liczności k/2. Dla uproszczenia możemy założyć, że Bank wybiera i z przedziału < k/2+1 ; k >.

4. Alice wysyła do banku wartości a_i, c_i, d_i oraz r_i dla wybranych i. Bank sprawdza czy Alice nie próbowała oszukać (wartość u || (v+i) jest znana dla Banku).

5. Jeśli wszystko się zgadza Bank daje Alice:

$\prod B_i^{1/3} \bmod n$ dla i € <1;k/2>

i daje jej monetę oraz obciąża jej konto. Dodatkowo Bank zwiększa licznik *v* o wartość *k*.

6. Alice może łatwo wydobyć elektroniczną monetę:

$C = \prod f(x_i, y_i)^{1/3} \bmod n$ dla i € <1;k/2>

✓ Operacja zapłaty

1. Alice wysyła wiadomość C do Boba.
2. Bob wybiera losowy ciąg binarny $z_1, z_2, ..., z_{k/2}$.
3. W zależności od wartości z, Alice odpowiada Bobowi dla każdego i € < 1 ; k/2 > :
   a. Dla $z_i = 1$, wysyła mu a_i, c_i oraz y_i
   b. Dla $z_i = 0$, wysyła mu x_i, a_i v (u || (v+i)) oraz d_i
4. Bob sprawdza czy C jest poprawne przed akceptacją płatności Alice.

✓ Operacja wpłaty pieniądza

1. Bob wysyła C do Banku.
2. Bank sprawdza poprawność C i czy moneta nie została już użyta.
3. Bank wrzuca monetę do bazy użytych monet, zachowuje binarny ciąg z_i oraz odpowiedzi Alice.
4. Bank zasila konto Boba.

c) Znalezienie pierwiastka 3 stopnia w grupie reszt modulo n [5]

Siłą tego algorytmu jest trudności znalezienia pierwiastka trzeciego stopnia modulo n z pewnej liczby. Ponieważ faktoryzację liczby n zna tylko bank, osobie postronnej, która przechwyci dane wysyłane przez bank, bardzo dużo czasu i pracy zajęłoby odnalezienie liczb p i q, dla których bank wykonuje algorytm, ponieważ jedynie na podstawie klucza publicznego nie da się łatwo odgadnąć liczb w kluczu prywatnym ponieważ nie jest znany wydajny algorytm rozkładu liczb na czynniki, czyli faktoryzacji liczb złożonych. Dlatego mówi się, że problem rozkładu na czynniki pierwsze jest tak ważny w kryptografii.

Będziemy szukali rozwiązania $\sqrt[3]{\mathbf{B} \bmod \mathbf{n}}$

Bank wyznacza liczbę n = p*q, której tylko on zna faktoryzację.

Po wygenerowaniu liczb pierwszych p i q, należy sprawdzić czy dla wybranej wartości e=3 liczby p-1 oraz q-1 nie mają wspólnych czynników z wybranym e, dlatego w naszym programie zakładamy:
p i q są liczbami pierwszymi postaci 3k+2, czyli: p = 3k+2 i q = 3k+2

Twierdzenie: Element odwrotny do e modulo n definiujemy jako taką liczbę d, że
e*d = 1 (mod n).
Liczba d istnieje tylko gdy e oraz n są względnie pierwsze, czyli NWD (n,e) = 1 (stąd nasze założenie odnośnie postaci p i q).
Mając NWD (n,e) = 1 można zawsze zapisać: 1 = e*d + v*n
Szukanym elementem odwrotnym e jest u nas d. Jeśli d wychodzi ujemne, trzeba zredukować d modulo n zanim użyje się go jako odwrotności.

Jak działa RSA? Po obliczeniu liczby n, będziemy ją wykorzystywać do wykonywania operacji modulo. Jednak widzimy, ze nie jest ona liczbą pierwszą, wiec nie zachodzi równość,
x^{n-1} =1 (mod n) dla 0<x<p. Aby użyć RSA musimy znaleźć taki wykładnik t by x^t=1 (mod n) dla (prawie) wszystkich x. Spójrzmy: jeśli zachodzi x^t=1 (mod n) to zachodzi również x^t=1 (mod p) oraz x^t=1 (mod q). Liczby p i q są pierwsze więc równanie x^t=1 (mod n) będzie zachodzić tylko wtedy gdy p-1 będzie dzielnikiem t oraz q-1 będzie dzielnikiem t.
Czasami używa się do tego funkcji Eulera. Funkcja ta dla liczby n=pq gdzie p i q są pierwsze wynosi f (*n*) = (*p* - 1)(*q* - 1)
Wartość tej funkcji jest wielokrotnością naszego t. Użycie jej zamiast t również da dobre wyniki. Posługując się rozszerzonym algorytmem Euklidesa obliczamy d jako odwrotność e modulo t. Aby zaszyfrować wiadomość m, nadawca oblicza tekst zaszyfrowany:
$c=m^e$ (mod n)
Aby odszyfrować tekst c, odbiorca oblicza
c^d (mod n)
co jest równe oryginalnej wiadomości m.

Rozszerzony algorytm Euklidesa:
Posłuży nam do znalezienia d oraz v.
Potrzebujemy wykładnik t: t = (p-1)(q-1)
Klucz publiczny stanowi para (n, e)
Klucz prywatny (p, q, t, d)

Używamy rozszerzonego algorytmu Euklidesa do obliczenia d jako odwrotności e modulo t
Ze względu na zależność x^t=1(mod n), wykładniki obliczeń modulo n należy brać modulo t, gdyż wielokrotności t w wykładniku operacji modulo n nie wpływają na wynik.

Przykład:
p = 53, q = 71 n = 53*71 = 3763
t = (p-1)*(q-1) = 3640

Element odwrotny do e modulo t (rozszerzony algorytm Euklidesa):
1 = d*e + v*t
d = 2427

B = 42
$\sqrt[3]{B \bmod n}$ = $B^d \bmod n = 102$
sprawdzenie: $102^3 = 1\,061\,208 = 282 * 3763 + 42$

3. Koncepcja realizacji projektu

Do realizacji projektu użyjemy platformy Android z wykorzystaniem środowiska programistycznego Android Studio (w wersji Beta 0.8.6). Aplikacja będzie tworzona pod urządzenia z wersją oprogramowania Android 4.0.3 lub nowszą. Stworzona aplikacja po uruchomieniu będzie miała do wyboru jedna z trzech stron: Bank, Klient lub Sklep. Po uruchomieniu aplikacji na dwóch lub więcej urządzeniach możliwa będzie komunikacja między stronami wykorzystująca połączenie sieciowe.

✓ Bank

Aplikacja Banku będzie miała jedynie możliwość oczekiwania na zgłoszenia Klienta lub Sprzedawcy. Po naciśnięciu przycisku "Uruchom" aplikacja będzie oczekiwała na zgłoszenia aż do momentu naciśnięcia przycisku "Zatrzymaj". Przycisk "Losuj" pozwala na nowe wygenerowanie losowych liczb *n* i *k.* Poprzez naciśnięcie przycisku "Pokaż istnieje możliwość sprawdzenia tych wartości oraz wartości *u* i *v*.

✓ Klient

Klient może za pomocą przycisku "Pobierz monetę" dokonać operacji wypłaty pieniądza z konta bankowego. Przycisk "Zapłać" umożliwia wykonanie operacji zapłaty za usługę/towar. Klient w celu wypłaty pieniądza lub wykonania zapłaty musi ustawić, po naciśnięciu przycisku "Ustawienia", odpowiednie dane do połączenia z Bankiem lub Sklepem (adres IP i port). Naciśnięcie przycisku "Losuj", wygeneruje nowe losowe wartości dla zmiennych a_i, c_i, d_i oraz r_i. Sprawdzenie tych wartości oraz numeru bankowego *u* jest dostępne po naciśnięciu przycisku "Pokaż".

✓ Sklep

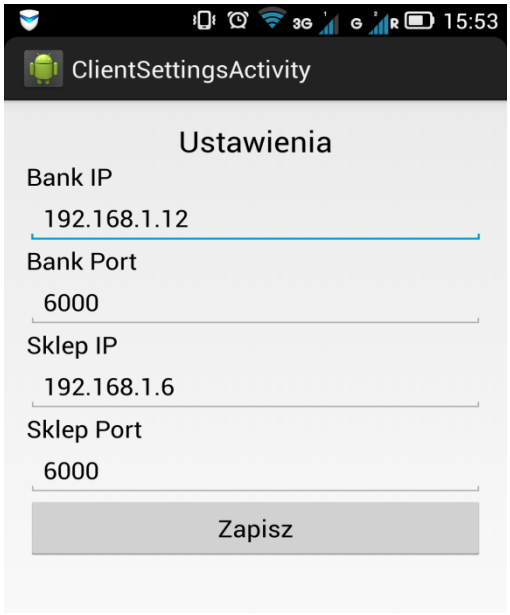
Sklep ma natomiast możliwość oczekiwania na zapłatę od Klienta (przycisk "Czekaj na klienta") oraz wpłaty otrzymanego pieniądza do Banku (przycisk "Wpłać do banku"). Podobnie jak Klient w celu wpłaty pieniądza do Banku, Sklep musi ustawić odpowiednie dane do połączenia. Naciśnięcie przycisku "Losuj", wygeneruje nowy losowy ciąg binarny z_i. Za pomocą przycisku "Pokaż" można sprawdzić wartości tego ciągu.

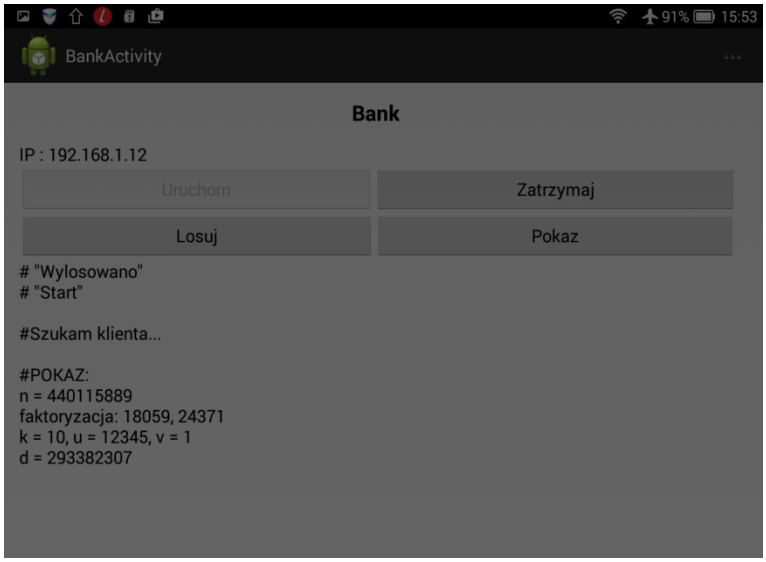
✓ Próba oszustwa

Nasz projekt zakłada również możliwość próby oszustwa. Wykorzystywana do tego jest czwarta strona (Charlie), która dostaje od Klienta (Alice) tą samą monetę (tą samą wartość C), co wcześniej Sklep (Bob). Bob, a następnie Charlie próbują wpłacić monetę do Banku, jednak w przypadku wpłaty Charlie moneta jest już "zużyta".

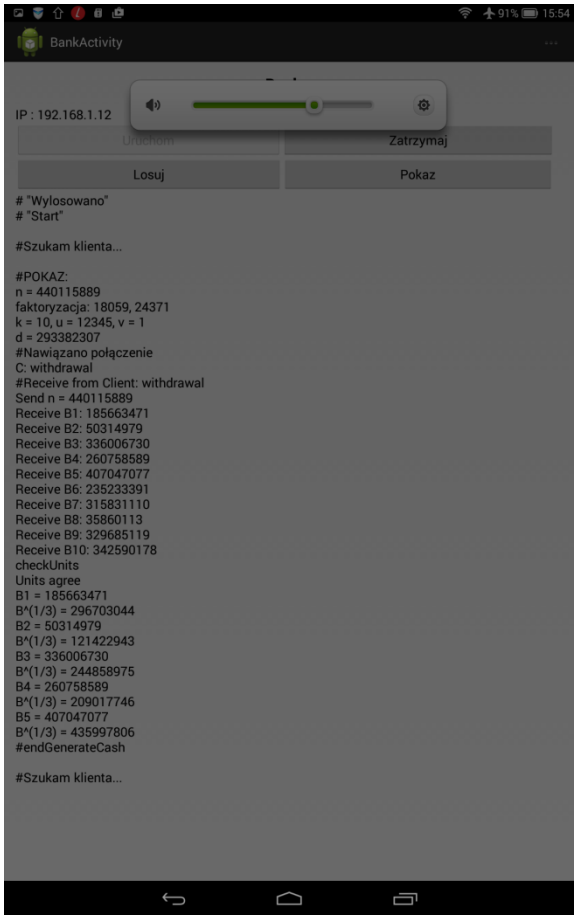
4. Przykład działania aplikacji

Na początku ustawiamy adres IP i porty, przykład ustawień Klienta, a następnie uruchamiamy aplikację Banku i wyświetlamy parametry



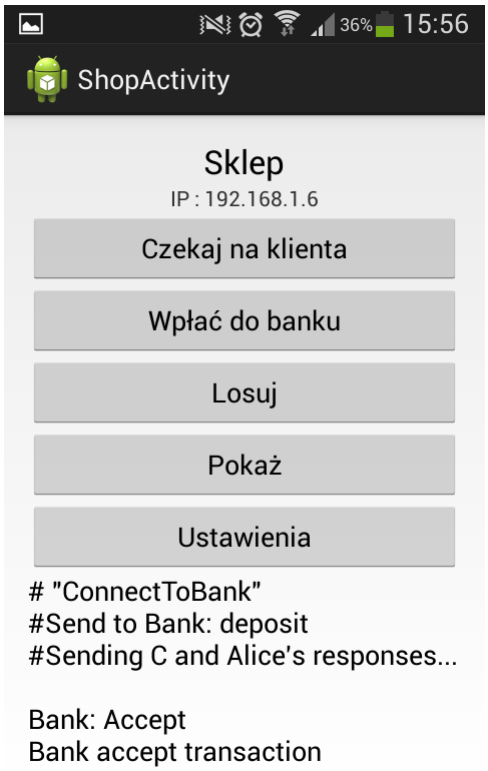


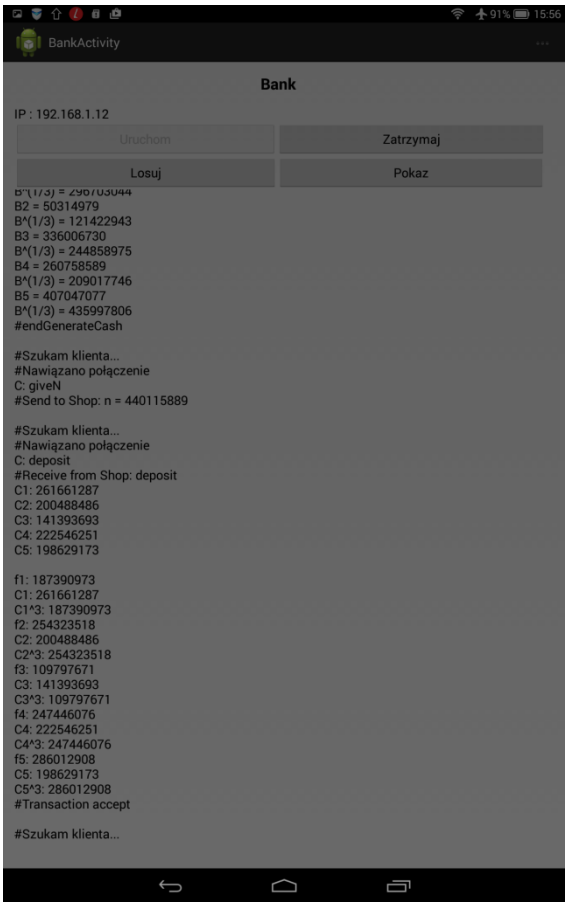
Klient pobiera monetę z Banku. Bank otrzymuje próbki Bi, sprawdza je, a następnie podpisuje je kluczem prywatnym i wysyła do Klienta. Ten otrzymuje próbki i mnoży przez 1/r.



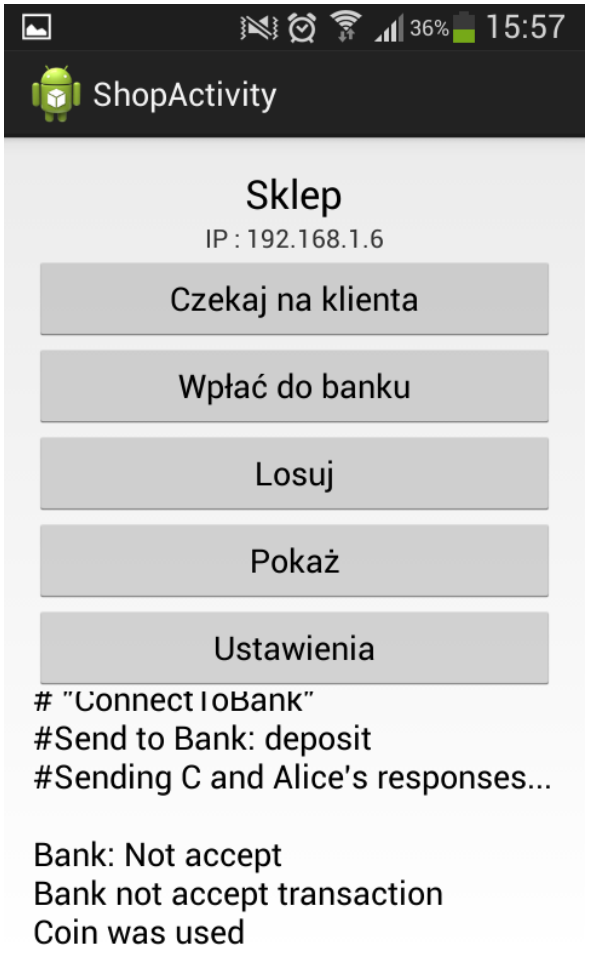
Klient po otrzymaniu pieniądza płaci nim w Sklepie, ten przed zaakceptowaniem transakcji wysyła ciąg binarny z, aby sprawdzić Klienta. Jeśli wszystko się zgadza Sklep akceptuje transakcję.

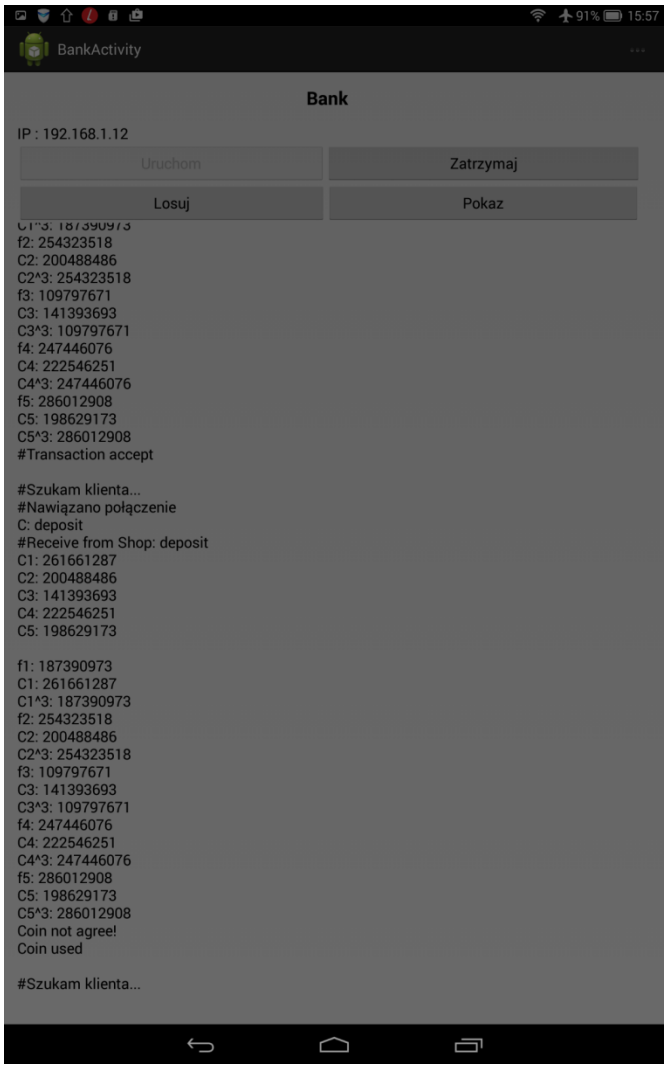
Sklep wpłaca otrzymany pieniądz do Banku, a ten sprawdza czy nie otrzymał już takiej monety oraz weryfikuje swój podpis.





Na koniec Sklep próbował oszukać Bank wpłacając drugi raz tą samą monetę.





5. Podsumowanie

✓ Właściwości[2]

Bezpieczeństwo: W tym protokole szansa oszustwa i ryzyko złapania są określone przez k/2, nawet jeśli k jest małe i równe 2, wtedy szanse są ciągle 1 do 1.

Jednym z najprostszych oszustw jest tutaj podwójna zapłata. W celu zminimalizowania możliwość oszustwa, kiedy bank otrzymuje monetę zachowuje jej numer w bazie danych, a następnie jeżeli ten sam numer pojawi się drugi raz bank wie, że doszło do oszustwa. Jeśli Alice jest oszustem, który zapłacił dwa razy, jest ona poddawana sprawdzeniu dwa razy. Ponieważ każde sprawdzenie jest losowym ciągiem binarnym, nowe sprawdzenie nie będzie zgadzać się ze starym (z bardzo dużym prawdopodobieństwem) na co najmniej jedynym bicie. Będą to dwa różne zestawy połówek tożsamości monety. Szansa, że jedna z połówek

tożsamości będzie pasować do innej połówki innej monety jest wysoka i tożsamość Alice zostanie ujawniona. Szansa, że Alice zostanie poproszona o podanie tych samych połówek tożsamości w obu transakcjach z tą monetą wynosi 1 do $2^{k/2}$. Dwie połówki tożsamości zostaną ujawnione i Alice zostanie złapana, ale jeśli dwie połówki tożsamości nie będą pasować do siebie i tożsamość Alice nie zostanie ujawniona to oznaczać będzie, że oszustem jest Bob.

Przenośność: Monety są łatwe do wytworzenia, są one niczym innym jak cyfrowymi numerami i jakimiś losowymi liczbami, które Alice potrzebuje użyć do zapłaty. Monety nie mogą być użyte kilka razy. Muszą być one zwrócone do banku po każdej transakcji przez Bob'a.

Podzielność: Ten system nie daje żadnej możliwości podziału monety.

✓ Ataki na protokół[2]

Atak na ten protokół jest możliwy przy współpracy Alice i Charlie. Jeśli Alice po transakcji z Bob'em wyśle tą samą zużytą monetę do Charlie z ciągiem binarnym wybranym przez Bob'a i odpowiedzi na ten ciąg, to Charlie będzie miała taką samą historię płatności jak Bob i bank nie będzie w stanie określić kto jest oszustem.

✓ Porównanie do innych protokołów[2]

Inne dwa znane protokoły elektronicznej gotówki to schemat Ferguson'a i schemat Brands'a. Właściwości protokołu Fergusona są bardzo zbliżone do protokołu Chauma, Fiata, Naora. Bezpieczeństwo obu jest na podobnym poziomie, oba systemy nie dają możliwości podziału monety. Są oparte na kluczu RSA, a atak w obu przypadkach jest taki sam (współpraca Alice i Charlie przy podwójnym użyciu monety).

Inne właściwości wykazuje natomiast protokół Brandsa. Format i istnienie monety nie zależą od tego gdzie się znajduje, ani od rodzaju pamięci masowej na jakiej się znajduje. Pozwala on na użycie systemu jako czeku lub używania do transakcji dokonywanych przez inteligentne karty. W przeciwieństwie do dwóch wcześniej wymienionych systemów nie można używać tego rodzaju pieniądza w kółko, ponieważ możliwość złapania oszusta używającego tej samej monety kilku krotnie byłaby nie możliwa. Umożliwia on również pewien specyficzny podział monety. Protokół Brandsa broni się przed atakiem, który można było zastosować w poprzednich dwóch protokołach, jednak istnieje inny atak na ten protokół, w którym Alice może użyć kilka razy tej samej monety zostając niezidentyfikowana dzięki niewłaściwemu protokołowi otwarcia konta.

System Brandsa jest trudniejszy do zrozumienia, ponieważ używa bardziej skomplikowanych zagadnień matematycznych. Jednak dzięki temu możliwości tego protokołu są bardziej zbliżone do elektronicznego pieniądza niż protokołu Chauma, Fiata, Naora. Jest on również bezpieczniejszy, ponieważ opiera się na trudności obliczenia logarytmu dyskretnego, a nie jak protokół "*Untraceable Electronic Cash*" na kluczu RSA, który zależy od faktoryzacji liczb pierwszych.

7. Bibliografia

[1] Chaum D., Fiat A., Naor M., " *Untraceable Electronic Cash*", Springer-Verlag Berlin Heidelberg 1990,
[2] Mandana Jahanian Farsi, "*Digital Cash*", Göteborg University 1997,
[3] *Descriptions of SHA-256*, *http://csrc.nist.gov/groups/STM/cavp/documents/shs/sha256-384-512.pdf*
[4] https://docs.oracle.com/javase/7/docs/api/java/security/MessageDigest.html
[5] http://michalbereta.pl/do_pobrania/dydaktyka/RSA.pdf