Projekt: Bartłomiej Rosa, współautor anonimowy

Bezpieczny magazyn plików „w chmurze”
Komunikacja z kodem uwierzytelnienia wiadomości
HMAC_MD5

1. Wprowadzenie

Celem projektu jest zaprojektowanie i zaimplementowanie dwóch aplikacji, które będą realizowały bezpieczny magazyn plików w sieci lokalnej. Pierwsza z aplikacji będzie pełniła rolę serwera działającego na komputerze PC, druga klienta dla systemu mobilnego. Bezpieczeństwo zostanie zapewnione przez komunikację z kodem uwierzytelnienia wiadomości HMAC_MD5.

1.1. Keyed-Hash Message Authentication Code (HMAC)

Technika HMAC polega na utworzeniu skrótu wiadomości, za pomocą którego będziemy mogli zweryfikować oprócz integralności przesyłanych danych także ich autentyczność, dzięki użyciu klucza tajnego przy tworzeniu tegoż skrótu. Nie jest możliwe odzyskanie danych na podstawie wyniku funkcji skrótu, zabezpieczenie polega na tym że strony znając

wysłaną jawnie wiadomość oraz klucze prywatne generują kod HMAC otrzymanej wiadomości i porównują go z tym dołączonym do wiadomości. Jeśli obliczone skróty są zgodne możemy mieć pewność że wiadomość została wysłane przez osobę będącą w posiadaniu klucza tajnego. O sile zabezpieczeń mechanizmu HMAC stanowi przede wszystkim zastosowana w niej funkcja haszująca. W naszym projekcie zamierzamy wykorzystać sposób tworzenia kodu HMAC opisany w RFC 2104 [1]:

$$HMAC(K,m) = H\left((K \oplus opad) | H((K \oplus ipad)|m)\right)$$

H - kryptograficzna funkcja haszująca
K - klucz prywatny
m - przesyłana wiadomość
| - konkatenacja bitowa
⊕ - XOR
opad - ustalona wartość (0x5c5c5c…5c5c) o długości bloku na którym działa algorytm
ipad - ustalona wartość (0x363636…3636) o długości bloku na którym działa algorytm
B - blok danych (pewnego stałego rozmiaru dla danego algorytmu)

Opis słowny tworzenia kodu HMAC

1 - Jeśli długość K = B, to: K_0 = K oraz idź do kroku 4.

2 - Jeśli długość K > B, to oblicz hash K (o długości L bajtów) oraz dodaj na końcu B-L zer żeby otrzymać B bajtowy ciąg K_0 oraz idź do kroku 4.

3 - Jeśli długość K < B, to dodaj na końcu zera w ciągu K - otrzymujemy K_0.

4 - Wykonaj K XOR ipad - otrzymujemy ciąg B bajtowy - K ⊕ ipad.

5 - Dodaj ciąg m do ciągu z kroku 4 - (K ⊕ ipad) | m.

6 - Wykonaj funkcję H na ciągu z kroku 5 - H((K ⊕ ipad) || text).

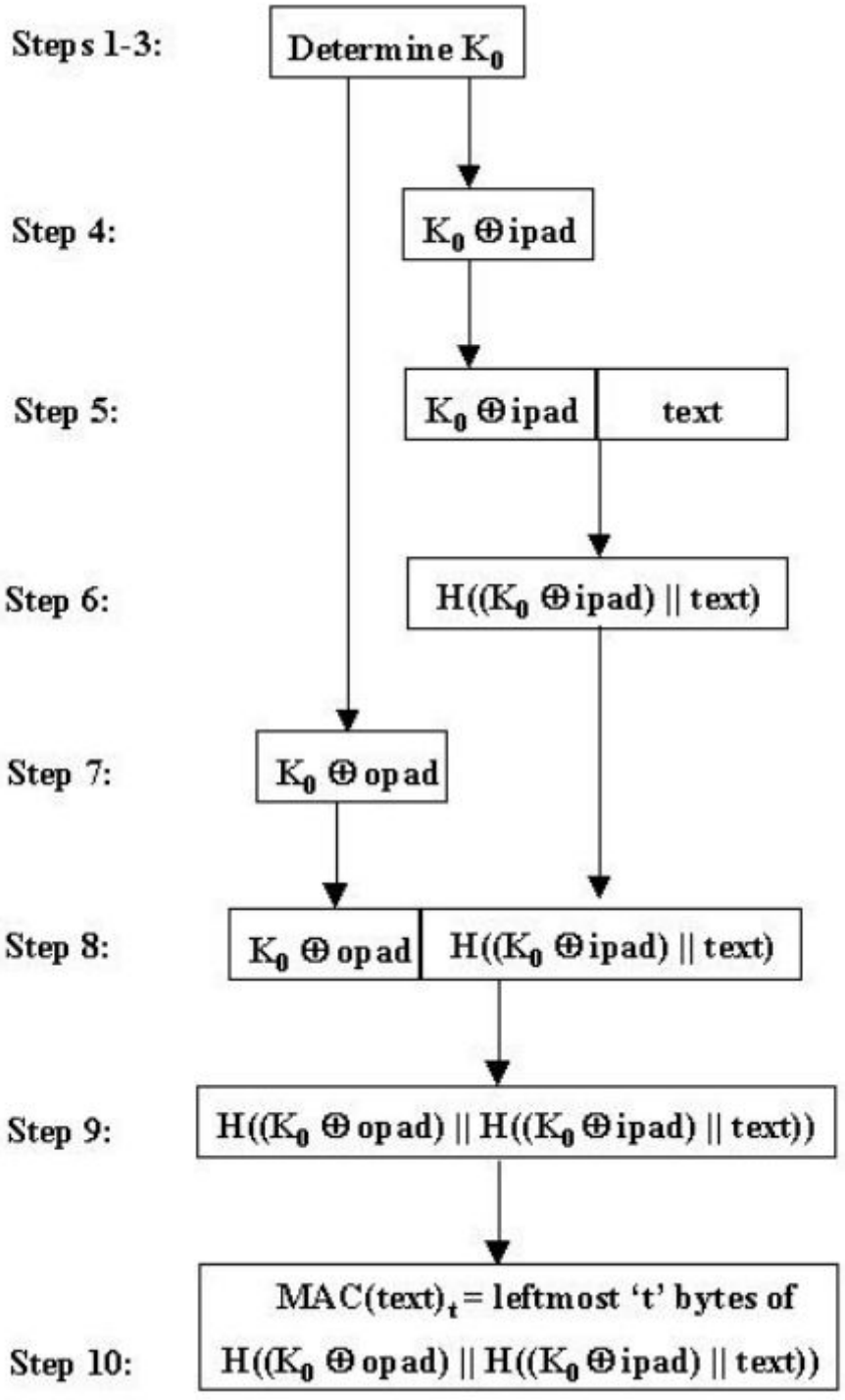
7 - Wykonaj K_0 XOR opad - K ⊕ opad.

8 - Połącz wyniki z kroków 6 oraz 7 - (K ⊕ opad) | H((K ⊕ ipad) | m).

9 - Wykonaj funkcję H na ciągu z kroku 8 - H((K ⊕ opad) | H((K ⊕ipad) | m)).

10 - Wybierz t bajtów od lewej z rezultatu punktu 9.

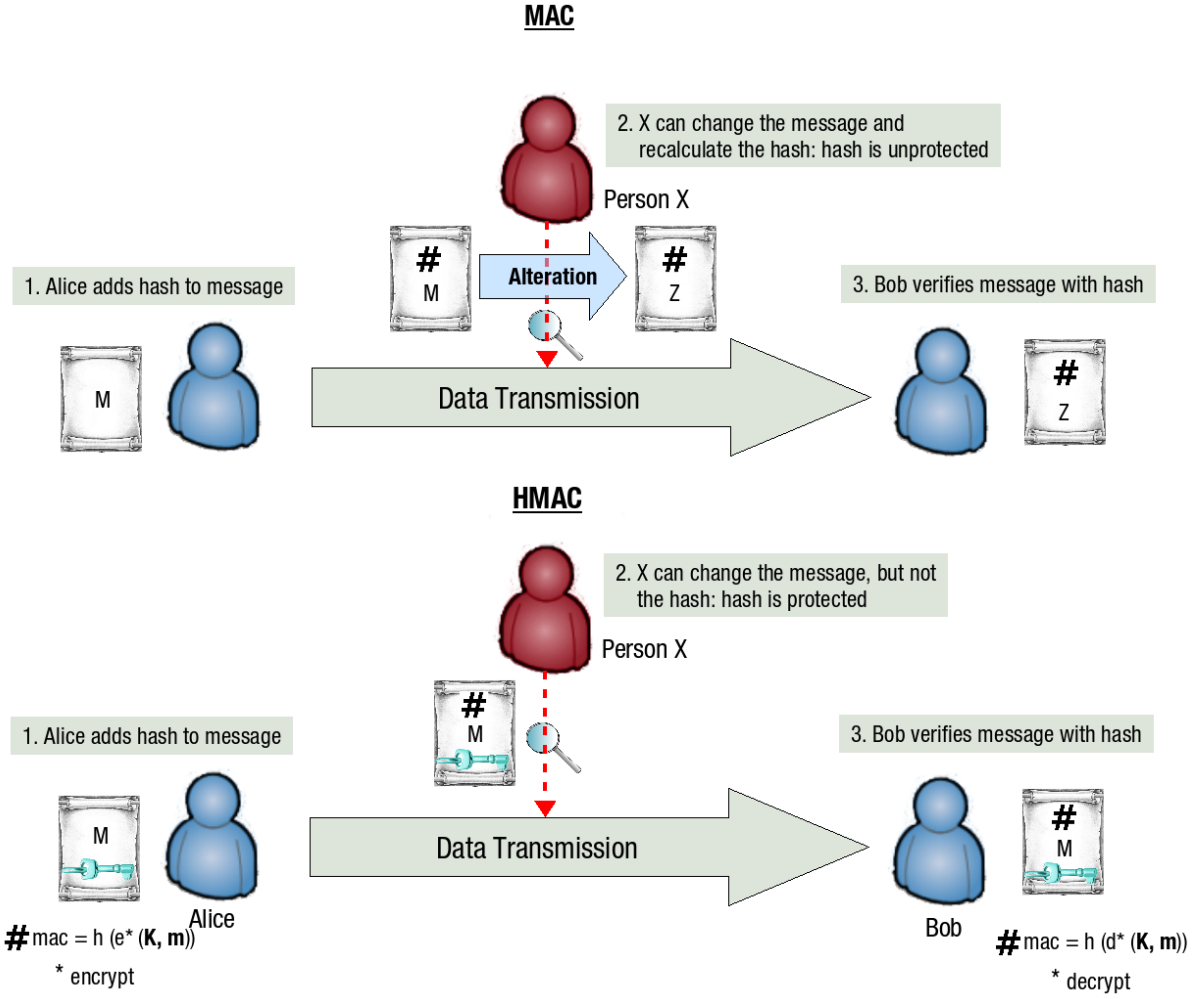
Na rysunku 1. znajduje się przedstawienie graficzne schematu tworzenia kodów HMAC [4]. Jest to ogólny schemat, gdzie można użyć różnych funkcji haszujących. W przypadku naszego projektu jest to funkcja MD5.



Rys. 1. Schemat tworzenia kodu HMAC

Zaletą HMAC nad haszowaniem bez klucza jest to, że można sprawdzić, czy osoba trzecia nie podmieniła wiadomości. Jest to przedstawione na rysunku 2. W przypadku wykorzystania funkcji MAC Alice przysyła wiadomość razem z jej haszem, Bob może tylko zweryfikować czy otrzymany hasz powstał z wiadomości, którą

dostał. Nie jest w stanie stwierdzić, czy ktoś w międzyczasie nie podmienił wiadomości oraz nie stworzył nowego haszu. Stosując HMAC Alice i Bob wykorzystują jeszcze klucz, który znają tylko oni, dzięki czemu tylko oni mogą policzyć prawidłowe kody HMAC wiadomości.



Rys. 2. Różnica między MAC, a HMAC

2. Message-Digest algorithm 5 (MD5)

W naszym projekcie funkcją haszującą H wykorzystaną w HMAC jest MD5. Algorytm MD5 to funkcja haszująca, która tworzy z wiadomości dowolnej długości 128 bitowy

skrót [2, 3]. Zakłada się, że nie jest obliczeniowo możliwe wyprodukowanie dwóch różnych wiadomości posiadających te same skróty albo wyprodukowanie jakiejkolwiek wiadomości na podstawie skrótu. Częstym zastosowaniem algorytmu MD5 jest użycie go w technice podpisu cyfrowego.

Opis algorytmu MD5

Wejściem algorytmu to komunikat M o dowolnej długości, a wyjście to 128-bitowy skrót. Przetwarzanie komunikatu na skrót odbywa się w 4 krokach. Dane w każdym kroku przetwarza się w 512-bitowych blokach, podzielonych na 16 podbloków po 32 bity.

Kroki
1 - Wiadomość dopełniamy bitami, żeby jej długość była 64 bity mniejsza od wielokrotności 512 bitów. Uzupełniane bity zaczynają się bitem 1, a kolejne to 0.

2 - Dodajemy 64 bity reprezentujące liczbę K, czyli pierwotną długość wiadomości M. Jeśli K jest większe niż 2^64, to pierwotną długość obliczamy modulo 2^64. W rezultacie otrzymujemy L bloków (Y0, Y1,...,YL-1) 512 bitowych. Każdy z nich możemy podzielić na 16 podbloków 32 bitowych.

3 - Operacje, realizowane przez algorytm MD5, polegają na modyfikacji wartości czterech zmiennych A, B, C i D, zapamiętanych w 4 rejestrach 32-bitowych. W kroku 3. zmiennym tym nadaje się następujące wartości początkowe, zapisane w kodzie szesnastkowym: A=01234567, B=89ABCDEF, C=FEDCBA98, D=76543210.

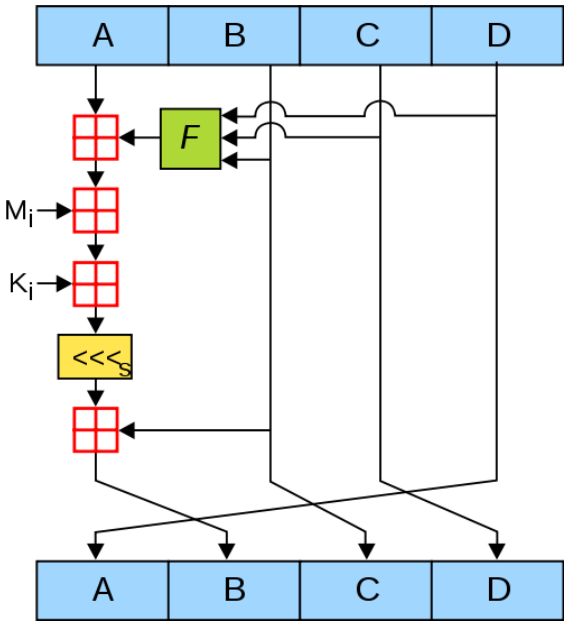
4 - W tym kroku używa się czterech funkcji pomocniczych, których argumentami są trzy 32-bitowe słowa, a wyjściem jedno słowo 32-bitowe. Są to następujące funkcje:

$$F(X,Y,Z) = (X \text{ and } Y) \text{ or } ((\text{ not } X) \text{ and } Z), \quad (1)$$
$$G(X,Y,Z) = (X \text{ and } Z) \text{ or } (Y \text{ and } (\text{ not } Z)), \quad (2)$$
$$H(X,Y,Z) = X \text{ xor } Y \text{ xor } Z, \quad (3)$$
$$I(X,Y,Z) = Y \text{ xor } (X \text{ or } (\text{ not } Z)). \quad (4)$$

Funkcje te działają na odpowiadających sobie bitach bloków X, Y, Z zgodnie z tablicą:

x	y	z	F	G	H	I
0	0	0	0	0	0	1
0	0	1	1	0	1	0
0	1	0	0	1	1	0
0	1	1	1	0	0	1
1	0	0	0	0	1	1
1	0	1	0	1	0	1
1	1	0	1	1	0	0
1	1	1	1	1	1	0

Właściwy algorytm został przedstawiony na rysunku 3.

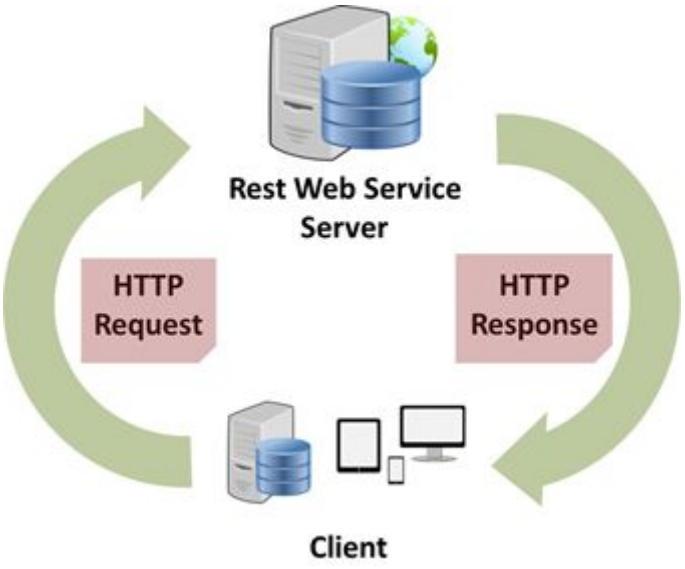


Rys.3. Schemat tworzenia MD5

Skrótem MD5 jest sekwencja 4 32-bitowych słów A,B,C,D.

3. Architektura aplikacji

Stworzony system będzie opierał się na klasycznej architekturze klient-serwer. Aplikacja kliencka znajdowała się będzie na urządzeniu mobilnym, aplikacja serwerowa natomiast będzie zawierała główną logikę biznesową systemu. Do zbudowania części serwerowej wykorzystamy popularne w ostatnim czasie rozwiązanie *RESTful Service.* Opiera się ono na komunikacji protokołem http za pośrednictwem struktury JSON.



Rys. 4. Architektura aplikacji

4. Podstawowe założenia i funkcje aplikacji

Stworzony przez nas system będzie umożliwiał obsługę wielu użytkowników. Każdy z nich po zarejestrowaniu się (stworzeniu konta na naszym serwerze) będzie miał możliwość wykonania następujących operacji:

- podglądu listy plików aktualnie przechowywanych przez niego na serwerze
- wysłania pliku
- pobrania pliku
- usunięcia pliku

Każda wykonywana operacja, a dokładniej każdy komunikat jej dotyczący przesyłany pomiędzy klientem a serwerem, będzie zabezpieczany za pomocą dodania do niej kodu wyznaczonego za pomocą HMAC_MD5. Strony znające klucz użyty do tworzenia, będą mogły dzięki temu zweryfikować czy wiadomość została w rzeczywistości wysłana przez jednostkę identyfikującą się daną tożsamością. Projekt przewiduje korzystanie z sieci lokalnej a więc do poprawnej pracy programu Użytkownik będzie musiał znać adres IP serwera, oraz port na którym jest on uruchomiony. Serwer i klienci będą posiadał swoje certyfikaty stworzone w prostym Urzędzie Certyfikacji (za pomocą narzędzia openSSL).

Scenariusz działania aplikacji

(1) Uwierzytelnienie się podmiotów

Uwierzytelnianie jest sprawdzeniem czy podmiot (osoba, program) jest tym za kogo się podaje.–Jednym z przypadków potrzeby potwierdzenia swej tożsamości jest komunikacja między dwoma osobami.
Na potrzeby projektu, że podmioty są uwierzytelnione.

(2) Ustalenie klucza prywatnego

Podmioty poza aplikacjami wymieniają się kluczem. Zakładamy, że robią to w sposób bezpieczny.

(3) Wymiana wiadomości

Serwer i klient mogą wymieniać ze sobą wiadomości, w tym pliki, które są z dodanymi kodami HMAC_MD5. Struktura wiadomości został przedstawiona na poniższym schemacie:

timestamp
command
argument
filename
hmac

Objaśnienie zawartości poszczególnych pól:
timestamp - data utworzenia wiadomości - mierzona w milisekundach od 1 stycznia 1970 roku (pozwala na zabezpieczenie się przed atakami powtórzeniowymi)
command - rodzaj wiadomości
argument - dodatkowa informacja/argument dla konkretnego rodzaju wiadomości(np. plik)
filename - nazwa pliku, w przypadku jego przesyłania
hmac - HMAC wiadomości, wyliczony na podstawie konkatenacji powyższych pól oraz klucza prywatnego.

5. Wykorzystane technologie

Język programowania:

- część serwerowa: serwer napisany w Python z wykorzystaniem biblioteki tornado
- część kliencka: Windows Phone 8+

Baza danych: tinyDB
Wykorzystane narzędzia:

- Visual Studio
- Windows Management Studio
- openSSL

6. Zastosowanie praktyczne

Stworzony system może pełnić rolę tzw. chmury obliczeniowej. Są to aplikacje umożliwiające dostęp do wcześniej zmagazynowanych plików, niezależnie od miejsca pobytu. Rozwiązanie to ma równie wielu zwolenników co krytyków. Głównym argumentem za są prostota oraz łatwość dostępu. Najczęściej zarzucanymi mankamentami są brak bezpieczeństwa, a także uzależnienie od dostępu do sieci Internet.

Chmura obliczeniowa (ang. *cloud computing*) jest modelem, w którym przechowywanie aplikacji oraz plików znajduje się na zewnętrznych serwerach, a nie na maszynie użytkownika. Dzięki temu uzyskujemy dostęp do danych z dowolnego miejsca podłączonego do Internetu. Usługodawca zapewnia konsumentowi korzystanie z konkretnej usługi bez konieczności instalowania oprogramowania po stronie klienta. Mechanizm ten jest obecnie jednym z najpopularniejszych trendów w branży IT.
Opisana w dokumentacji aplikacja może pełnić rolę naszego domowego sejfu komputerowego, w którym możliwe jest przechowywanie cennych dokumentów.

7. Instrukcja użytkownika programu

Do poprawnego użytkowania systemu konieczne jest w pierwszej kolejności uruchomienie aplikacji serwerowej, dlatego pierwszym krokiem jest pobranie środowiska uruchomieniowego *Python 2.7* na maszynie hostującej. Możemy to zrobić używając oficjalnej strony projektu w zakładce *Downloads*. Link, którego możemy bezpośrednio użyć do pobierania został przedstawiony poniżej.
https://www.python.org/downloads/release/python-2710/ .
Po poprawnym zainstalowaniu środowiska *Python* możemy przystąpić do uruchomienia serwera. Start aplikacji serwerowej odbywa się poprzez uruchomienie programu *python.exe,* którego argumentem jest ścieżka do głównego pliku aplikacji o nazwie *Main.py.* Przykładowe zapytanie uruchamiające aplikację serwerową zostało podane poniżej.

```
C:\Projects\MKOI\mkoi\serwer>C:\Python27\python.exe Main.py
```

W następnym kroku możemy przejść do uruchomienia aplikacji mobilnej. Do poprawnego użytkowania konieczny będzie aparat z systemem *Windows Phone 8* lub nowszy. Po zainstalowaniu aplikacji mamy dwie możliwości:

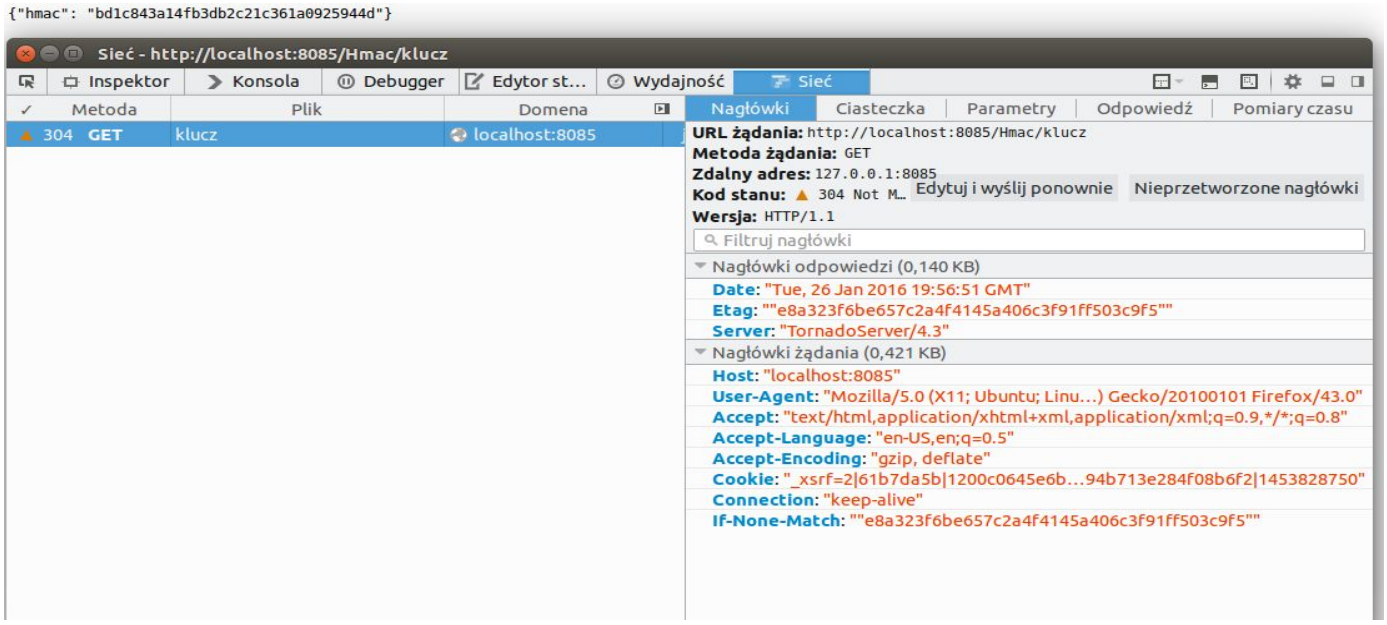
- stworzenie nowego użytkownika - opcja wywołująca interfejsy serwerowe w celu utworzenia nowego konta łącznie z nowym loginem oraz hasłem, została dodana do aplikacji dla celów testowych. W ostatecznej wersji systemu zostanie usunięta, a możliwość dodawania/usuwania użytkowników będzie dostępna wyłącznie dla administratora systemu.
- logowanie - żadne interfejsy serwerowe nie są wywoływane, użytkownik wpisuje ID oraz klucz uzgodniony wcześniej z serwerem i przechodzi do dalszej części aplikacji

Jeśli po podaniu ID oraz klucza użytkownikowi zostaną zaprezentowane pliki dostępne na serwerze, oznacza to, że został on poprawnie zidentyfikowany.
Kolejny ekran prezentuje użytkownikowi intuicyjny interfejs, dzięki któremu możemy w prosty sposób dodawać oraz usuwać pliki z serwera.

8. Testy systemu

Pierwsza faza testów objęła algorytmy wyliczania HMAC zarówno po stronie klienckiej jak i serwerowej. Wyniki zostały sprawdzone ze stroną:

https://asecuritysite.com/encryption/hmac . Przykładowo z naszej strony serwer udostępnia dla celów testowych opcję wyliczania HMAC pod adresem /Hmac/klucz, gdzie klucz to użyty klucz (jest to liczone dla wiadomości o treści “test”). Przykładowy zrzut ekranu z takiego testu znajduje się na rysunku 5, na rysunku 6 znajduje się log serwera.

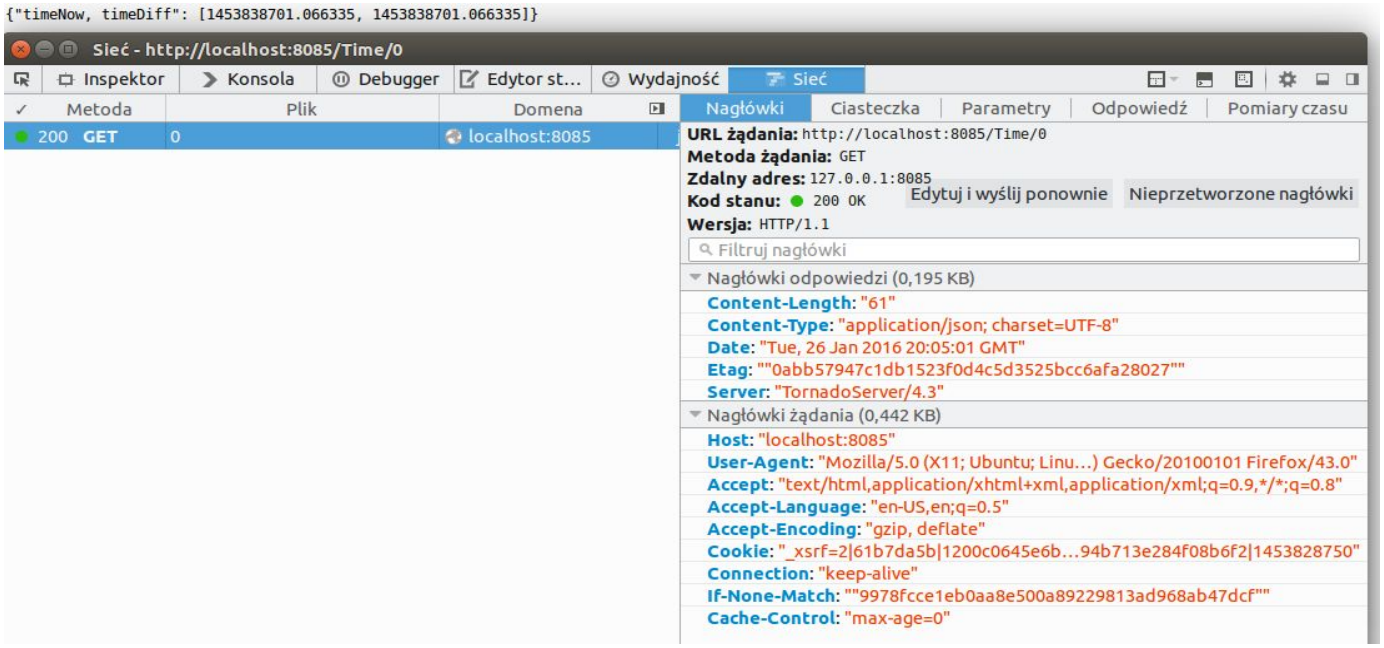


Rys. 5. Testowanie liczenia HMAC

Rys. 6. Testowanie liczenia HMAC - log serwera

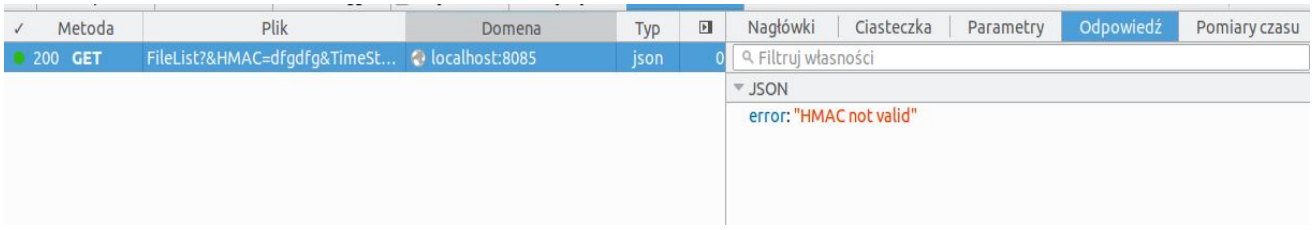
Produkty końcowe algorytmu zaimplementowanego w aplikacji klienckiej, serwerowej oraz na podanej stronie internetowej zgadzały się.

Następnie przeprowadzono testy odporności na atak powtórzeniowy. Dzięki dodanemu polu *timestamp* do każdego z zapytań, aplikacja w prosty sposób została zabezpieczona przed takimi wypadkami. Po przeprowadzeniu testów stwierdzamy, że wystarczy zmiana wartości *timestamp* o jeden, a wyliczony HMAC zmienia się zupełnie. Dla ułatwienia testowania po stronie serwera pod adresem /Time/timestamp (timestamp - liczba określająca czas w sekundach od 01.01.1970 UTC) zwraca różnicę podanego timestampa z aktualnym czasem sprawdzonym przez serwer. W ten sposób można było sprawdzić, czy dobrze liczymy.

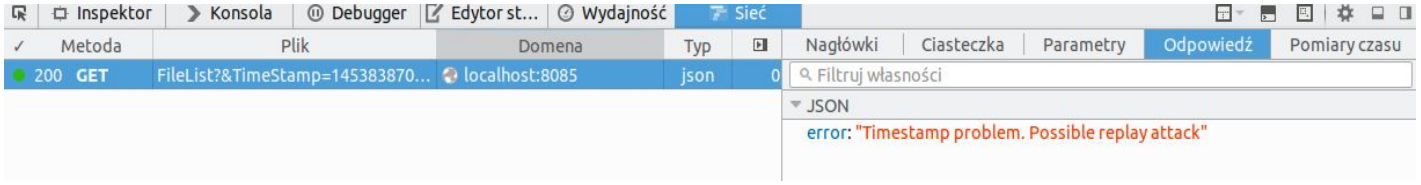


Rys. 7. Testowanie timestampa

Dodatkowo sprawdzaliśmy w praktyce, czy serwer dla źle wyliczonego HMAC lub złego timestampa zwraca dane, czy błędy. Przykładowe wyniki zawierą rysunki 8 i dalej.



Rys. 8. Widok dla przypadku złego wyliczania HMAC



Rys. 9. Widok dla przykładowego złego timestampa

9. Bibliografia

1. H. Krawczyk, *RFC 2104 HMAC: Keyed-Hashing for Message Authentication*, 1997
2. Wykłady z przedmioutu PKRY
3. Wykłady z przedmioutu MKOI
4. FIPS PUB 198, *The Keyed-Hash Message Authentication Code (HMAC)*, 2002