Uwierzytelnianie w aplikacjach webowych z użyciem kryptografii asymetrycznej

Obecnie najbardziej rozpowszechnionym schematem autoryzacji i autentykacji w Internecie jest wymaganie podania prawidłowej nazwy użytkownika i hasła. Niekiedy wprowadza się dwuskładnikowe uwierzytelnianie za pomocą fizycznego tokenu albo aplikacji w smartfonie. Celem tego artykułu jest zaprezentowanie implementacji znanych metod uwierzytelniania opartych o kryptografię asymetryczną, które do tej pory występowały wyłącznie poza aplikacjami internetowymi, a obecnie możliwa jest ich implementacja przy użyciu Web Crypto API [1].

WPROWADZENIE

Web Crypto API jest zunifikowanym interfejsem służącym do wykonywania podstawowych operacji kryptograficznych na poziomie skryptów JavaScript działających po stronie klienta. Możliwości obecnych wersji uwzględniają między innymi szyfrowanie, deszyfrowanie, generowanie kluczy, hashowanie, podpisywanie i weryfikację podpisów. Wspomniane operacje są wspierane dla wybranych algorytmów kryptografii symetrycznej (m.in. AES), asymetrycznej (m.in. RSA, ECDSA, ECDH) i/lub funkcji mieszających (m.in. SHA).

Jednym z proponowanych przez W3C przypadków użycia Web Crypto API jest „uwierzytelnianie wieloskładnikowe" [2], również w zakresie zastąpienia mało przyjaznego użytkownikom i znacznie trudniejszego w konfiguracji mechanizmu uwierzytelniania klienta na poziomie TLS.

Z punktu widzenia uwierzytelniania najbardziej istotnymi operacjami będą te związane z podpisywaniem, przy wykorzystaniu algorytmów kryptografii asymetrycznej. W takim przypadku jedna ze stron komunikacji posiada klucz prywatny (umożliwiający podpisywanie wiadomości), a druga jedynie klucz publiczny (umożliwiający weryfikację poprawności tych podpisów).

Kryptografia symetryczna i asymetryczna

Czym się różnią? Najbardziej intuicyjny będzie tutaj przykład szyfrów symetrycznych (np. AES), czyli takich, w których operacje szyfrowania i deszyfrowania są wykonywane za pomocą jednego i tego samego klucza. Zupełnie inaczej działają algorytmy szyfru asymetrycznego, albowiem najpierw konieczne jest wygenerowanie spójnej pary kluczy, gdzie jeden z nich nazywamy „publicznym", a drugi „prywatnym". Klucze te są ze sobą powiązane w taki sposób, że jedną z operacji (szyfrowanie/deszyfrowanie) da się wykonać wyłącznie kluczem prywatnym, a operację do niej odwrotną – wyłącznie kluczem publicznym.

Dlaczego w ogóle potrzebujemy algorytmów kryptografii asymetrycznej?

Największą bolączką związaną w praktyce z szyframi symetrycznymi jest konieczność uzgodnienia klucza pomiędzy rozmówcami (klientem i serwerem) w bezpieczny sposób. Czy wyobrażalne jest, żeby Google rozsyłał listy tradycyjne ze zdrapką zawierającą indywidualny klucz AES do każdego ze swoich klientów? Jest na to znacznie lepszy sposób.

Część klucza traktowaną jako publiczną można w dowolny sposób udostępniać osobom postronnym. Dzięki znajomości klucza publicznego będą one w stanie szyfrować wiadomości skierowane do jego właściciela, ale nikt inny poza nim nie będzie w stanie ich odszyfrować, nawet jeżeli je przechwyci. Dzięki takim właściwościom komunikacji możliwe jest np. uzgodnienie klucza symetrycznego w sposób uniemożliwiający jego przechwycenie.

Przykładowy schemat uwierzytelniania, przy założeniu, że komunikacja jest realizowana w sposób stanowy, może wyglądać tak:

1. Serwer generuje losowy łańcuch binarny (określoną liczbę losowych bajtów składającą się na tzw. *challenge nonce*) i przesyła go do klienta, żądając podpisania tego łańcucha.
2. Klient używa swojego klucza prywatnego, aby podpisać losowy ciąg otrzymany od serwera, i przesyła do niego swoją nazwę użytkownika oraz utworzony podpis.
3. Serwer odnajduje klucz publiczny przypisany do użytkownika o podanej nazwie i weryfikuje poprawność podpisu. Jeżeli podpis jest poprawny, to użytkownik uwierzytelnił się prawidłowo (udowodnił, że jest właścicielem klucza).

Przedstawiony scenariusz uwierzytelniania *challenge-response* będzie prawidłowy przy założeniu, że klient zweryfikował wcześniej tożsamość serwera i transmisja między nimi jest już zabezpieczona np. poprzez TLS. Bez takiego zapewnienia istnieje ryzyko, że klient połączy się ze „złośliwym serwerem", który będzie pośredniczył w komunikacji z prawdziwym serwerem, przeprowadzając atak *man-in-the-middle*.

Bardzo istotną właściwością takiego rozwiązania jest to, że serwer nie musi przechowywać żadnych wrażliwych danych związanych z uwierzytelnianiem klienta. Kłopotliwy jest np. wyciek hashowanych haseł z bazy danych na serwerze czy też wyciek współdzielonych sekretów wykorzystywanych do generowania kodów w aplikacjach mobilnych (two-factor authentication). Wyciek klucza publicznego jest stosunkowo mało kłopotliwy, bowiem jeżeli wszyscy klienci używają silnych kluczy (znacznie prostsze do zapewnienia niż silne hasła), taka informacja nie daje atakującemu żadnej znaczącej przewagi.

CO WNOSI WEB CRYPTO API DO ZABEZPIECZEŃ?

Wszystko, co zostało opisane do tej pory, może zostać zrealizowane przez dowolną bibliotekę JavaScript, która w prawidłowy sposób implementuje odpowiednie algorytmy (np. [3]). Pora na przedstawienie unikalnych korzyści wynikających z wykorzystania Web Crypto API: