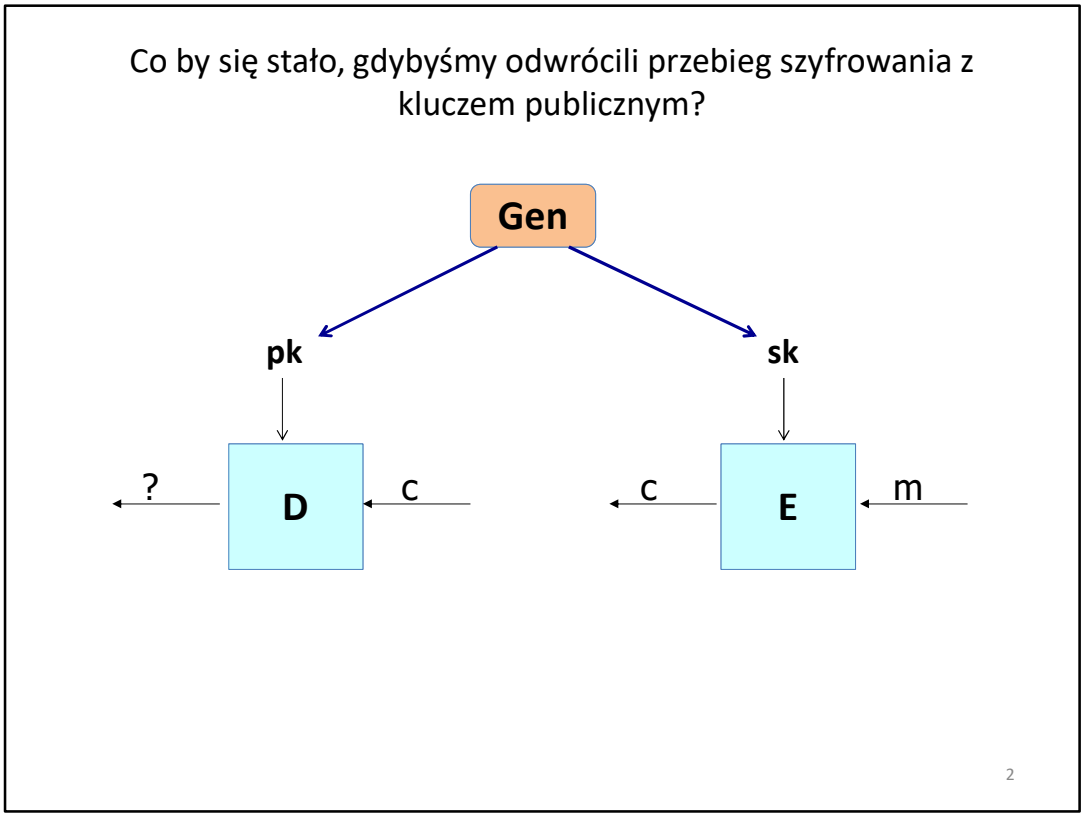
Kryptografia i bezpieczeństwo danych - podpis elektroniczny

Sławomir Samolej
ssamolej.kia.prz.edu.pl
ssamolej@prz.edu.pl



Na początek zastanówmy się, co by się stało, gdybyśmy dla odmiany zaszyfrowali jakąś wiadomość za pomocą naszego klucza sekretnego i pozwolili na jej odszyfrowanie za pomocą klucza publicznego? Na pierwszy rzut oka takie działanie wydaje się dziwne, ale po chwili zastanowienia możemy przeprowadzić następujące rozumowanie. Bob szyfruje wiadomość swoim kluczem sekretnym. Praktycznie każdy (w tym Alice) może tę wiadomość odszyfrować. Uzyskujemy tu jednak nową cechę: ponieważ tylko Bob mógł zaszyfrować tę wiadomość, to znaczy, że pochodzi ona **tylko od niego**, co więcej odpowiednio zorganizowany system szyfrowania może zapewnić **integralność** wiadomości oraz **niezaprzeczalność**, że jest się autorem wiadomości.

Atak na taki system polega na próbie podszycia się pod nadawcę wiadomości i oszukania odbiorcy, że sfałszowana wiadomość pochodzi od danej osoby i zawiera określoną treść. Bezpieczny system podpisu elektronicznego powinien zapobiegać takim atakom.

Odniesienie do MAC

- Podpis elektroniczny i MAC mają na celu zapewnienie integralności wiadomości
- Zastosowanie podpisu elektronicznego:
  - Upraszcza uwierzytelnienie wiadomości (nie ma potrzeby wymiany klucza symetrycznego z każdą osobą, dla której chcemy udowodnić integralność wiadomości)
  - Zapewnia publiczną weryfikowalność (odpowiednie mechanizmy (np. PKI) pozwalają na sprawdzenie, czy dane zostały podpisane przez daną osobę)
    <mechanizm niedostępny dla MAC>
    W konsekwencji:
    - Podpis może być przenoszony i weryfikowany w różnych miejscach
  - Wprowadza możliwość niezaprzeczalności podpisania dokumentu (osoba podpisująca nie morze się wyprzeć, że taki podpis złożyła)
- MAC są krótsze i 2-3 wydajniejsze, jeśli chodzi o czas obliczeń. W konsekwencji, jeśli w rozwiązaniu nie chodzi o zapewnienie publicznej weryfikowalności, możliwości transferu i niezaprzeczalności oraz komunikacja odbywa się z jednym odbiorcą, to należy stosować MAC.

3

Znaczenie podpisu elektronicznego

- Weryfikowalność podpisu zależy do możliwości niezawodnego i możliwego do zweryfikowania udostepnienia klucza publicznego.
- Taka procedura jest kosztowna, także ze względów organizacyjnych, ale zwykle wykonywana jest tylko raz, a potem każdy łatwo może uzyskać klucz i zweryfikować podpis.
- Okazuje się, że właśnie schemat podpisu elektronicznego sam w sobie jest doskonałym mechanizmem dystrybucji kluczy publicznych.
- Tak więc publikowanie kluczy publicznych odbywa się z zastosowaniem podpisów elektronicznych i jest podstawą do tzw. Infrastruktury Klucza Publicznego (PKI – Public Key Infrastructure)

Definicja podpisu elektronicznego

Podpis elektroniczny: (Gen, Sign, Vrfy)

- Gen: algorytm generacji pary kluczy (pk i sk)
- Sign: algorytm podpisujący

  (m, ds) ←Sign(sk,m)
- Vrfy: algorytm weryfikujący

  Vrfy(pk, m, ds)

  Zwraca 1, gdy podpis jest prawidłowo zweryfikowany, 0 gdy weryfikacja się nie udała.

Dla wszystkich legalnych wiadomości zachodzi:

Vrfy(pk,m,Sign(sk,m)) = 1.

5

Zasada wykonywania podpisu elektronicznego wygląda następująco. Osoba podpisująca generuje parę kluczy (sk i pk) [tak jak w kryptografii klucza publicznego: jeden klucz służy do szyfrowania, a drugi do odszyfrowywania]. Klucz publiczny osoby podpisującej jest udostępniany. Jeśli osoba chce podpisać jakąś wiadomość szyfruje ją z zastosowaniem klucza sekretnego sk i wysyła do odbiorcy parę (m, ds). Odbiorca znający klucz publiczny może odszyfrować podpis i sprawdzić, czy otrzymana wartość jest identyczna z nadesłaną wiadomością m. Wtedy dowiadujemy się 2 rzeczy: 1) wiadomość była od konkretnego nadawcy, 2) nie została zmodyfikowana (odszyfrowana wiadomość jest identyczna z nadesłaną).

Dalej, podobnie jak w przypadku MAC, system nie jest odporny na ataki powtórzeniowe.

W systemie zakłada się również, że istnieje jakiś mechanizm pewnego przekazywania kluczy. To znaczy, że odbiorca otrzymując klucz publiczny jest pewny, że jest to klucz od konkretnej osoby.

Jeśli potrafimy w sposób pewny dostarczyć właściwy klucz publiczny, to po co w ogóle tworzyć podpisy? Okazuje się, że takie pewne dostarczenie kluczy jest procesem trudnym i kosztownym, ale w schemacie podpisu elektronicznego musi być wykonane tylko raz. Potem można przekazywać podpisy nieskończoną liczbę razy.

Poza tym mechanizm podpisu elektronicznego służy do efektywnego przekazywania samych kluczy publicznych (np. PKI – public key infrastructure)

Bezpieczeństwo schematów podpisywania elektronicznego

- Atakujący dysponuje wiadomościami M, kluczem publicznym pk oraz podpisami DS.
- System jest bezpieczny, jeśli atakujący nie jest w stanie wygenerować nowej (innej, ot tych które przejął) wiadomości m', którą można prawidłowo zweryfikować systemem sprawdzania podpisu elektronicznego

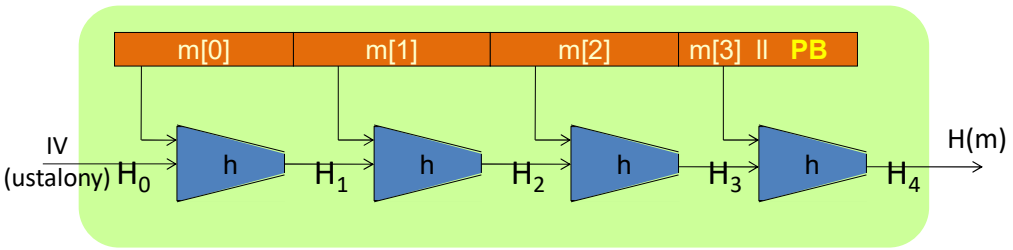
6

Paradygmat Hash-and-Sign (oblicz hash, potem podpisuj)

- Zaszyfrowanie całych, zwłaszcza długich wiadomości jest kosztowne obliczeniowo
- Popularne jest rozwiązanie „hybrydowe”:
  - Najpierw stosuje się funkcję hash do „streszczenia” wiadomości
  - Potem szyfrowanie przeprowadza się tylko na skrócie/streszczeniu wiadomości (ang. digest)
  - Rozwiązanie jest odpowiednikiem schematu hash-then-MAC

7

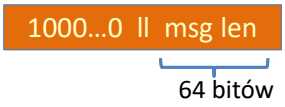
Hash wiadomości - przypomnienie



Dana jest $h: T \times X \longrightarrow T$ (funkcja kompresująca)

otrzymujemy $H: X^{\leq L} \longrightarrow T$. H_i - zmienna łańcuchowa

PB: padding blok



Jeśli nie ma miejsca dla PD, dodaj jeszcze jeden blok

8

Rozważmy konstrukcję jak na rysunku. Jest to ogólna konstrukcja, według której buduje się funkcje skrótu. Zakładamy, że dysponujemy funkcją h (hash) odporną na kolizje dla małych wiadomości. Funkcja nazywana jest też funkcją kompresującą. Wiadomość jest dzielona na bloki. W konstrukcji stosowany jest również IV (ang. Initialisation Vector), tym razem ustalany jednokrotnie raz na zawsze i wbudowywany w program (jego wartość jest ustalana na poziomie definiowania standardu). Wyjście funkcji kompresji (H_n nazywane zmienną łańcuchową) jest przekazywane na wejście kolejnej funkcji kompresji, która przekształca także kolejny blok wiadomości. Łańcuch przekształceń kolejnych porcji wiadomości jest kontynuowany do momentu osiągnięcia ostatniego bloku. Do ostatniego bloku musi zostać dodany tzw. „padding blok". Ostatnia część widomości wraz z paddingiem jest przekształcana z zastosowaniem funkcji h i otrzymujemy wyliczony hash dla długiej wiadomości. Padding blok składa się z pola 10000…0 uzupełniającego wiadomość do odpowiedniej długości oraz 64 bitowego pola zawierającego długość wiadomości. Rozmiar wiadomości jest kodowany w polu o ustalonej długości. Przykładowo, we wszystkich funkcjach SHA długość wiadomości jest ograniczona do 2^{64}-1 (ostatni blok może zwierać sam padding i długość wiadomości, jeśli wiadomość ma długość równą całkowitej wielokrotności bloku). Ograniczenie wiadomości do podanego rozmiaru w rzeczywistości nie stanowi zbyt wielkiego ograniczenia (ok. 18 tys. TB).

Formalizacja konstrukcji hash-and-sign

- Niech (Gen, Sign, Vrfy) będzie schematem podpisu elektronicznego wiadomości o długości n.
- Niech (GenH, H) będzie funkcją hash zwracającą dane długości n.
- Nowy schemat podpisu elektronicznego (Gen', Sign', Vrfy') można zdefiniować:
  - Gen': uruchom Gen, aby otrzymać klucze (pk, sk), uruchom GenH, aby otrzymać s: kluczem publicznym jest <pk, s>, kluczem sekretnym <sk, s>,
  - Sign': oblicz podpis ds = Sign(sk, H(s, m))
  - Vrfy': weryfikuj podpis z zastosowaniem <pk, s>, m i ds:

    Vrfy(pk, m, H(s, m), ds)

- Tw. : Jeśli schemat podpisywania (Gen, Sign, Vrfy) jest bezpieczny dla wiadomości o długości n i funkcja (GenH, H) jest odporna na kolizje, to schemat (Gen', Sign', Vrfy') jest bezpieczny.

9

Czy można zastosować „czyste” RSA zastosowane do wygenerowania podpisu?

- Niech GenRSA definiuje schemat wykonywania podpisu w następujący sposób:
  - Gen: algorytm generujący klucze RSA (N, d, e), gdzie pk=(N, e), sk=(N, d).
  - Sign: algorytm podpisujący z zastosowaniem klucza sekretnego sk=(N, d) wiadomość m należącą do $(Z_N)^*$ tworząc podpis:
    - $ds = [m^d \bmod N]$
  - Vrfy: algorytm weryfikujący podpis z zastosowaniem klucza publicznego (N, e) wiadomość m należącą do $(Z_N)^*$ zwracający 1 wtedy i tylko wtedy, gdy
    - $m = [ds^e \bmod N]$

10

Punktem wyjścia do praktycznego generowania sygnatury będzie podstawowy algorytm RSA. Nie zapewnia on bezpieczeństwa, ale jest dobrym punktem startowym do dalszych rozważań.

Sprawdzenie poprawności sygnatury polegające na podniesieniu sygnatury do potęgi e w naturalny sposób powinna zwrócić wartość m o ile wiadomość nie została zmodyfikowana.

Konstrukcja wydaje się bezpieczna, jeśli atakujący zna tylko klucz publiczny (N, e), bo trzeba rozwiązać do przygotowania poprawnej sfałszowanej sygnatury problem RSA. Niestety nie jest to właściwe spostrzeżenie. Po pierwsze problem RSA jest trudny, jeśli na jego wejście (wiadomość do zaszyfrowania) podaje się liczbę losową o jednolitym rozkładzie prawdopodobieństwa. Nic nie wiadomo o trudności rozwiązania problemu RSA dla wiadomości o innych własnościach. RSA nie rozważa również sytuacji, kiedy atakujący zna już podpisy innych wiadomości.

Atak bez wiadomości na „czysty” RSA zastosowany do tworzenia podpisu

- Atakujący tworzy fałszywą wiadomość stosując tylko klucz publiczny (N, e):
  - Dysponujemy pk = (N, e), wybieramy losową wartość ds należącą do $(Z_N)^*$ i obliczamy
    $m = [ds^e \bmod N]$, sfałszowana wiadomość to (m, ds).
  - W rezultacie przygotowaliśmy weryfikowalną sygnaturę dla wiadomości i ją sfałszowaliśmy.

11

Na „czysty” system RSA traktowany jako technika tworzenia podpisu elektronicznego można przeprowadzić następujący atak. Można przygotować fałszywy podpis złożony z (m, ds), gdzie $m = [ds^e \bmod N]$. Uruchomienie algorytmu sprawdzania podpisu zgodnego RSA da pozytywny wynik…

Fałszowanie podpisu „czystego” RSA
z zastosowaniem ustalonej wiadomości

- Chcemy sfałszować wiadomość $m \in (Z_N)^*$ znając klucz publiczny (N, e), dwie inne wiadomości m_1, m_2 (różne od m) i ich podpisy ds_1 i ds_2.
- Sfałszowany podpis można otrzymać tworząc wiadomość $m = m_1 * m_2 \bmod N$ i podpis $[ds_1 * ds_2 \bmod N]$.
- Okazuje się, że jest to prawidłowa sygnatura dla wiadomości m, jeśli oczywiście dalej traktujemy wprost system RSA jako mechanizm służący do generowania podpisu:
- Sprawdzanie wiadomości będzie działało następująco:

$$(ds_1 * ds_2)^e = (m_1^d * m_2^d)^e = m_1^{de} * m_2^{de} = m_1 * m_2 = m$$

12

Inny atak na sygnaturę wykonaną za pomocą „czystego” RSA wymaga posiadania dwóch wiadomości m1 i m2 oraz ich dwóch sygnatur ds1 i ds2. Atakujący może wtedy stworzyć sfałszowaną wiadomość mnożąc ze sobą modulo wiadomości m1 i m2 oraz sfałszować sygnaturę mnożąc ze sobą modulo sygnatury. Zasada odszyfrowywania zgodna z RSA pokazuje, że tak spreparowana sygnatura zostanie uznana za prawidłową. Takie fałszerstwo być może nie ma znaczenia w podpisywaniu dokumentów, ale gdy weźmiemy pod uwagę podpisywanie kluczy szyfrowania, to sytuacja staje się groźniejsza…

Konstrukcja RSA-FDH (ang. RSA-Full Domain Hash)

- Niech GenRSA będzie takie, jak na wcześniejszym slajdzie. Konstrukcja schematu podpisu elektronicznego RSA-FDH wygląda w następujący sposób:
  - Gen jest generatorem kluczy (N, e , d). Klucz publiczny to (N, e), klucz sekretny to (N, d). Elementem alg. generowania kluczy jest funkcja hash H: $\{0,1\}^*$ -> $(Z_N)^*$
  - Sign bierze na wejście klucz prywatny (N, d), wiadomość m i oblicza:
    - $ds = [H(m)^d \bmod N]$
  - Vrfy: bierze na wejście klucz publiczny (N, e), wiadomość m i sygnaturę ds. i zwraca 1, jeśli
    - $sd^e = H(m)$
- Tw. Konstrukcja RSA-FDH **jest bezpieczna** jeśli funkcja H generuje ciągi losowe o jednorodnym rozkładzie (random oracle) zwracające wartości należące do zbioru $(Z_N)^*$

13

W konstrukcji RSA-FHD w stosunku do „czystego” RSA wprowadza się funkcję H (mieszającą), która przekształca wiadomość do podpisywania w jej skrót/streszczenie. Oczekuje się, że streszczenie będzie mapowało wiadomości na zbiór $(Z_N)^*$. Samo obliczenie sygnatury polega na wykonaniu szyfrowania RSA na skrócie z wiadomości. Sprawdzanie poprawności wiadomości polega na odszyfrowaniu sygnatury i porównanie jej z wartością funkcji H wykonanej na wiadomości.

RSA PKCS#1 v2.1

- Standard RSA PKCS#1 v2.1 powiela schemat generowania i sprawdzania podpisu z RSA-FDH.
- Dodatkową własnością dodaną do standardu jest *salt* (np. wartość losowa) wybierana przez osobę podpisującą w czasie generowania sygnatury. Standard dopuszcza wartość salt = 0, wtedy otrzymuje się standardowy schemat RSA-FDH.
- Duże znaczenie ma praktyczne dobranie algorytmu funkcji mieszającej i zwracanie przez nią wyników bliskich zbiorowi $(Z_N)^*$.
- Znane są ataki na tak skonstruowane podpisy, gdzie zastosowano „krótkie" funkcje mieszające np. SHA-1. Wadą tego algorytmu jest zwracanie tylko 160-bitowej wartości

14

Inne systemy generowania podpisów elektronicznych

- Generowanie podpisów z problemu dyskretnego logarytmu
- DSA – Digital Signature Algorithm
- ECDSA – Elliptic Curve Digital Signature
- Sygnatury wywodzące się z funkcji Hash.
- Rozpatruje się również łańcuchy i drzewa sygnatur…

15

Innym trudnym problemem matematycznym, który może leżeć u podstaw konstruowania podpisów cyfrowych jest rozwiązanie problemu dyskretnego logarytmu. Algorytmy DSA i ECDSA także bazują na problemie znajdowania dyskretnych logarytmów, przy czym drugi w dziedzinie krzywych eliptycznych (wystarczy stosować krótsze długości klucza). Istnieją rozwiązania stosujące tylko funkcje hash jako baza do tworzenia bezpiecznych jednorazowych podpisów.

Certyfikat

- Załóżmy, że Charlie wygenerował parę kluczy (pk_C, sk_C)
- Załóżmy też, że Bob wygenerował parę kluczy (pk_B, sk_B)
- Załóżmy, że Charlie wie, że pk_B jest kluczem publicznym Bob'a
- Wtedy Charlie może obliczyć podpis:
  $cert_{C \to B}$ = Sign[sk_C,(„To jest klucz Boba", pk_B)]
  i odesłać ten podpis Bob'owi.
- Nazywamy $cert_{C \to B}$ certyfikatem dla klucza publicznego Bob'a wystawionym przez Charliego.
- W praktyce certyfikat powinien jednoznacznie identyfikować Bob'a, czyli zawierać jego imię i nazwisko, email, stronę WWW i inne atrybuty

16

Pokazaliśmy wybrane bezpieczne konstrukcje stosujące kryptografię z kluczem publicznym. Do tej pory zakładaliśmy, że klucze publiczne są efektywnie rozdystrybuowane, co więcej odszyfrowanie sygnatury za pomocą klucza publicznego miało także potwierdzić, że wiadomość podpisała konkretna osoba (niezaprzeczalność). Okazuje się, że kryptografia klucza publicznego może również posłużyć do efektywnej i niezaprzeczalnej metody dystrybucji kluczy publicznych. To co zostanie pokazane, to że jeden klucz publiczny należący do instytucji zaufanej i rozdystrybuowany w sposób bezpieczny może „uruchomić" bezpieczną dystrybucję wielu innych kluczy publicznych. Zwłaszcza, że problem bezpiecznej dystrybucji kluczy musi być rozwiązany tylko raz. Kluczowym pojęciem jest tutaj cyfrowy certyfikat, który w istocie jest podpisem łączącym klucz publiczny z jakąś osobą/tożsamością.

Weryfikacja klucza publicznego na podstawie certyfikatu

- Teraz Bob chce skomunikować się z inną osobą/instytucją, np. Alice, która już zna pk_C.
- Może on wysłać do Alice wiadomość (pk_B,$cert_{C\rightarrow B}$), która może zweryfikować, że pk_B rzeczywiście należy do Bob'a, o czym poświadcza pk_C.
- Jeśli sprawdzenie podpisu się udało, to Alice wie, że Charlie podpisał klucz Bob'a
- Jeśli Alice ufa Charlie'mu, to może zaakceptować pk_B jako legalny klucz
- Cała wymiana potwierdzająca pk_B może się odbyć w publicznym i niechronionym kanale komunikacyjnym
- Dopóki Charlie (klucz prywatny Charlie'go) nie zostanie „skompromitowany" tak zweryfikowane klucze są uznawane za legalne i powiązane z daną osobą/instytucją.

17

Kilka pominiętych detali

- Skąd Alice zna pk_C i mu ufa?
- Skąd Charlie wie, że pk_B należy do Boba?
- Jak Alice decyduje, czy ufać Charlie'mu?

- Pełna specyfikacja, jak rozwiązać pokazane problemy opisana jest w dokumentach dotyczących Infrastruktury Klucza Publiczengo (PKI – Public Key Infrastructure)

Pojedynczy weryfikator certyfikatów (ang. Certificate Authority)

- Instytucja, której wszyscy ufają, i która wystawia certyfikaty dla każdego klucza publicznego
- Każdy, kto ma ufać tej instytucji musi jednokrotnie w bezpieczny sposób (np. pendrive), pobrać klucz publiczny tej instytucji i jej zaufać.
- Typowym sposobem bezpiecznego rozpowszechniania certyfikatów jest ich zaszywanie w kodzie oprogramowania. Legalne systemy operacyjne oraz przeglądarki internetowe są dystrybuowane z takimi kluczami, na podstawie których będą mogły potem identyfikować podpisane klucze publiczne innych programów, kanałów komunikacyjnych, czy użytkowników.
- Mechanizm wystawiania certyfikatu musi być dokładnie kontrolowany. Przykładowo, wystawienie tzw. certyfikatu kwalifikowanego wymaga zwrócenia się do właściwego CA i personalne potwierdzenie swojej tożsamości przed człowiekiem.

19

Wielu weryfikatorów certyfikatów

- Zastosowanie jednego centrum certyfikacji nie jest praktyczne.
  - Nie wszyscy chcą ufać jedynej instytucji
  - Pojedyncze centrum jest najsłabszym ogniwem infrastruktury
- W rzeczywistości stosuje się wiele centrów certyfikacji
  - Można wystąpić o podpisanie swojego klucza przez wiele centrów certyfikacji
  - Osoba chcąca korzystać z naszego klucza może wybrać, której instytucji ufa
  - Trudność ma odbiorca takiego klucza. Bezpieczeństwo komunikacji przy pomocy wielokrotnie podpisanego klucza zależy od najsłabiej zabezpieczonego centrum certyfikacji

20

Delegacja i łańcuchy certyfikatów

- Jeśli dysponujemy certyfikatem naszego klucza publicznego, możemy sami spróbować zostać centrum certyfikacji.
- Możemy wystawić certyfikat dla Alice:
  $cert_{B\to A}$ = Sign[sk_B,(”To jest klucz Alice”, pk_A)]
- Alice wysyła: pk_A, $cert_{B\to A}$, pk_B, $cert_{C\to B}$ do Dave’a.
- Ponieważ Dave ufa Charlie’mu, a Charlie ufa Bobowi, to Dave ufa Bobowi.
- Skoro Bob ufa Alice, to Dave ufa również Alice.
- Warunkiem istnienia takiego rozwiązania, jest takie zaufanie Charliego do Boba, że pozwala on mu wystawiać własne certyfikaty.
- W rozwiązaniach PKI często istnieją główne centra certyfikacji, które delegują uprawnienia do weryfikowania certyfikatów innym centrom certyfikacji.

21

Załóżmy, że Charlie jest centrum certyfikacji i wystawia certyfikat Bob’owi tak, jak omówiliśmy to wcześniej. Teraz Bob wystawia certyfikat następnej osobie. Przykładowo, wystawia certyfikat kluczowi publicznemu Alice.
Teraz Alice, która chce wymieniać dane z Dave’m wysyła mu: pk_A, $cert_{B\to A}$, pk_B, $cert_{C\to B}$.
Co może wydedukować Dave? Dave ufa Charliemu. Mając certyfikat klucza pk_C może on sprawdzić, że klucz Bob’a pk_B został podpisany przez Charlie’go, czyli że Bob’owi można ufać. Skoro tak, to jeśli Bob podpisał klucz Alice, to Alice także jest zaufana.
Tutaj Bob uzyskuje dodatkowe uprawnienia. Charlie nie tylko potwierdza tożsamość Bob’a ale również potwierdza, że Bob ma prawo sam potwierdzać tożsamość innej osoby.
W praktyce często centra certyfikacji tworzą drzewa, gdzie podstawowe centrum certyfikacji pozwala na weryfikację certyfikatów „niższym centrum”.

„Sieć zaufania”

- W tym modelu brak jest centrum certyfikacji.
- Zaufanie opiera się na certyfikatach wystawionych nawzajem sobie przez użytkowników danego systemu (np. PGP).
- Dana osoba podejmuje decyzję o zaufaniu na podstawie posiadanych certyfikatów i własnej oceny.
- Model dobry do szyfrowania i podpisywania poczty, nie nadaje się do komunikacji np. z bankiem.

22

Model sieci zaufania to przykład infrastruktury klucza publicznego bez centralnych punktów zaufania. Jej wariantem jest PGP. Tutaj każdy może wystawić certyfikat każdemu, ale każdy musi sam podjąć decyzję, czy ufa danemu certyfikatowi.
Możemy rozważyć następujący przykład. Alice ma już trzy klucze pk1, pk2 i pk3 użytkowników C1, C2 i C3. Inny użytkownik może być w posiadaniu trzech certyfikatów poświadczających jego klucz pkB: $cert_{C1\rightarrow B}$, $cert_{C3\rightarrow B}$, $cert_{C4\rightarrow B}$. Może on je wysłać (wraz ze swoim kluczem publicznym) do Alice. Alicja nie może zweryfikować $cert_{C4\rightarrow B}$, (Bo nie ma certyfikatu C4), ale może zweryfikować dwa pierwsze. Może zaufać kluczowi pkB, jeśli ufa na pewno C1, lub jeśli ufa C1 i C3 w mniejszym stopniu (Może się jej wydawać, że jeden kluczy pkC1 lub pkC3 utracił zaufanie, ale jest mało prawdopodobne, że oba).
W tym modelu spodziewa się, że użytkownicy powinni sami zbierać klucze publiczne innych oraz certyfikaty swojego klucza publicznego wystawione przez innych. Taka wymiana kluczy odbywa się na przykład na spotkaniach towarzyskich służących wymianie kluczy. Uczestnicy spotkania nie musza się dobrze znać, wystarczy, że potwierdzą czyjąś tożsamość na podstawie osobistej weryfikacji dokumentów stwierdzających tożsamość.
System PGP oferuje również centralną bazę zawierającą klucze publiczne i ich certyfikaty (http://pgp.mit.edu). Wtedy jeśli Alice chce wysłać wiadomość do Bob’a, to może wyszukać go w bazie danych i znaleźć jego klucz publiczny wraz z wstawionymi na niego certyfikatami innych osób. Teraz Alice może zdecydować, czy ufa kluczowi Boba. W praktyce Alice może znaleźć wiele kluczy Boba i certyfikaty do nich.
Przyjęty model nie wymaga istnienia centrum certyfikacji i dobrze nadaje się do utrzymywania infrastruktury kluczy do szyfrowania i podpisywania wiadomości email. Trochę jednak trudno sobie wyobrazić model wymiany informacji z bankiem, czy innymi instytucjami.

Utrata ważności certyfikatów

- Certyfikaty nie są wystawiane „na zawsze”
  - Praktyką jest ustalanie ich ważności na 2-3 lata
- Utrata ważności certyfikatu najczęściej ma dwie przyczyny:
  - Wygaśnięcie certyfikatu
  - Unieważnienie certyfikatu
- Tak naprawdę weryfikacja certyfikatu powinna obejmować 3 kroki:
  - Czy certyfikat można zweryfikować na podstawie zaufanego urzędu certyfikacji (czy dysponujemy kluczem publicznym urzędu?)
  - Czy certyfikat nie wygasł?
  - Czy certyfikat nie został unieważniony?

23

Ważną właściwością certyfikatów jest fakt, że nie są one ważne na zawsze. Pracownik może opuścić firmę. Z chwilą jego odejścia nie powinien on mieć możliwości elektronicznego podpisywania dokumentów i dostępu do zaszyfrowanych danych korporacyjnych. Klucz prywatny (np. zapisany na karcie chipowej) może być nam skradziony i (jeśli jesteśmy tego świadomi) możemy zażądać wygenerowania nowej pary kluczy wraz z nowym certyfikatem klucza publicznego. Trzeba też poinformować centrum certyfikacji o utracie klucza i spowodować, że nikt nie może się nim posługiwać. W każdym z tych scenariuszy został poruszony temat unieważnienia certyfikatu.
Rozważone zostaną dwa podstawowe mechanizmy i przyczyny unieważniania certyfikatów. Pierwszym z nich jest przedawnienie. Certyfikaty wydaje się na ustalony okres czasu i ten atrybut zapisuje się w strukturze certyfikatu. W uproszczeniu certyfikat ma więc jeszcze jedno pole: $cert_{C\to B}$ = Sign(sk_C, („To jest klucz Boba”, pk_B, data wygaśnięcia)]. Datę wygaśnięcia ustala się z wydawcą. Użytkownik, który chce przedłużyć działanie klucza musi poprosić wystawcę certyfikatu o jego przedłużenie. Oczywiście wystawca ponownie weryfikuje dane posiadacza klucza. Dostawcy certyfikatów oferują często „próbne”, zwykle darmowe certyfikaty z czasem ważności np. 2-3 miesiące. Jest to na tyle długo, że można „wytestować” oprogramowanie z takim certyfikatem, ale na tyle krótko, że częste pobieranie nowych certyfikatów staje się uciążliwe i dostawcy aplikacji chronionych decydują się na zakup certyfikatów działających na 2-3 lata…
Jeśli sobie wyobraziliśmy, że w posiadanie certyfikatu wszedł pewien pracownik i na drugi dzień opuściło on pracę, to mechanizm przedawnienia certyfikatu jest niewystarczający. W takim przypadku potrzebny jest mechanizm natychmiastowego unieważnienia certyfikatu. Urząd certyfikacji powinien mieć taką możliwość. Każdy certyfikat wystawiony przez dany urząd certyfikacji posiada swój unikatowy numer. Jego postać rozszerza się nam więc do: $cert_{C\to B}$ = Sign(sk_C, („To jest klucz Boba”, pk_B, data wygaśnięcia, ###)], gdzie ### jest numerem certyfikatu. Jeśli dany użytkownik zgłosi skradzenie klucza lub żądanie unieważnienia certyfikatu z jakiegokolwiek powodu, urząd certyfikacji „wyrzuca” certyfikat z puli zaufanych i zamieszcza go w publikowanej codziennie „liście unieważnionych certyfikatów” (ang. CRL Certificate Revocation List). Weryfikacja certyfikatu powinna przechodzić więc jeszcze jeden krok: sprawdzenie, czy certyfikat nie znajduje się na liście certyfikatów unieważnionych.

TLS

- Stosowany do transmisji chronionej pomiędzy przeglądarką a serwerem WWW
- Bazuje na protokole SSL (Netscape)
  - Ostatnia wersja SSL: 3.0 (lata 90-te)
  - Wersje TLS:
    - 1.0: 1999 (RFC 2246)
    - 1.1: 2006 (RFC 4346, RFC 4366, RFC 4680, RFC 4681)
    - 1.2: 2008 (RFC 5246)
    - 1.3: 21.03.2018 (RFC 8446)
- Ok. 50% serwerów stosuje ver. 1.0. Wszystkie ważniejsze przeglądarki wspierają ver. 1.2 (1.3?).

24

Protokół TLS (Transport Layer Security) jest stosowany do wymiany informacji z w środowisku WWW. Jest on stosowany przez przeglądarki, kiedy łączą się do stron oznaczonych nagłówkiem https. Skupimy się na aspektach kryptograficznych protokołu dopuszczając pewne uproszczenia.

Protokół TLS basuje na wcześniejszym protokole SSL (Secure Socket Layer), który został opracowany przez firmę Netscape w połowie lat 90-tych. Ostatnia wersja protokołu posiada nr. 3.0. TSL 1.0 został ogłoszony w 1999 roku, wersję 1.1 wydano w 2006. Nowa wersja została wydana 2008 roku, najnowsza została zaproponowana 21 marca 2018 roku. Ok. 50% serwerów wciąż posługuje się wersją 1.0. Wszystkie ważniejsze przeglądarki oferują wsparcie dla wersji 1.2, chociaż w pewnych wypadkach starsze wersje TLS są akceptowane.

Przeznaczenie TLS

- Wynegocjowanie pomiędzy klientem- przeglądarką (C) a serwerem-stroną HTTPS zbioru współdzielonych kluczy i wymiana danych za pomocą tych kluczy.
- Dwa etapy:
  - Połączenie (Handshake Protocol)
  - Komunikacja (Record-Layer Protocol)
- Jest możliwość obustronnej weryfikacji, ale zwykle to serwery posiadają certyfikaty kluczy.
- Klient, jeśli jest weryfikowany, to już podczas ustalonej szyfrowanej sesji.

25

TLS pozwala klientowi (np. przeglądarce) i serwerowi (np. stronie WWW) na wynegocjowanie zbioru współdzielonych kluczy a potem na ich stosowanie do szyfrowania i uwierzytelniania kolejnych komunikatów. TLS składa się z dwóch części. Protokołu ustalenia połączenia (ang. Handshake Protocol), który służy do uwierzytelnionej wymiany kluczy. Protokołu warstwy rekordów (ang. Record-Layer Protocol) zajmującego się szyfrowaniem i uwierzytelnianiem przesyłanych danych. Protokół może uwierzytelniać klientów w stosunku do serwera, ale jest przede wszystkim używany do poświadczenia, że komunikacja z serwerem odbywa się w sposób zaufany, ponieważ z reguły to serwery posiadają wystawione certyfikaty.

Handshake protocol

- C wysyła do S informację, jakie może obsłużyć protokoły i wartość nonce N_C
- S „proponuje” najwyższy z zaproponowanych przez klienta protokołów, który umie obsłużyć, dostępny zestaw prymitywów kryptograficznych, pk_S, $cert_{i \to S}$, swój nonce N_S.
- C sprawdza, czy zna klucz publiczny certyfikatu, weryfikuje certyfikat, uznaje pk_S. Tworzy pmk, szyfruje go pk_S i odsyła do S. mk = Derive(pmk, N_C, N_S), Z mk wyprowadza się klucze sesji: $k_{b \to s}$, $k_{s \to b.}$Obliczany jest skrót: r_C=S_{MAC}(mk, transcript). Odsyła r_C.
- S oblicza oblicza pmk = Enc(sk_S,c). Wyprowadza z niego $k_{b \to s}$, $k_{s \to b}$. Weryfikuje transcript: V_{MAC}(mk, r_C, transcript). Oblicza r_S = S_{MAC}(mk, transcript’), odsyła r_S.
- C weryfikuje V_{MAC}(mk, r_S, transcript’).

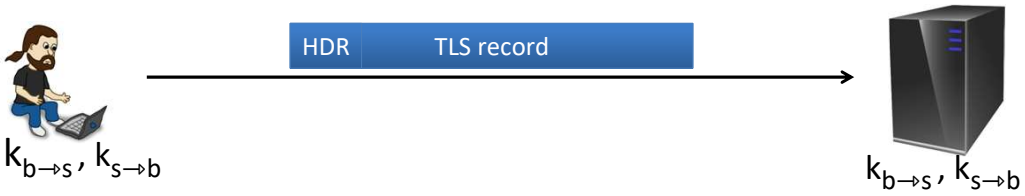
26

Połączenie pomiędzy serwerem a klientem bazuje na tym, że klient C posiada szereg certyfikatów zaufanych wystawców certyfikatów w raz z ich kluczami {pk_1, pk_2, …, pk_n}. Serwer S zaś ma parę kluczy (pk_S, sk_S) poświadczonych certyfikatem $cert_{i \to S}$ jednego z zaufanych wystawców certyfikatów. Aby klient połączył się z serwerem przeprowadzane są następujące kroki.

1. C wysyła do serwera S wiadomość informującą go o wersjach protokołów, które jest w stanie obsłużyć i losową wartość nonce N_C.
2. S odpowiada wskazując najwyższą z zaproponowanych przez klienta wersją protokołu, która jest w stanie obsłużyć oraz zestawem algorytmów szyfrowania, które obsługuje. Dodatkowo wysyła swój klucz publiczny pk_S , swój certyfikat $cert_{i \to S}$ oraz własną wartość „monce” N_S.
3. C sprawdza, czy jeden z kluczy publicznych pochodzących z centrum certyfikacji w jego posiadaniu (np. pk_i) odpowiada centrum certyfikacji, które podpisało klucz serwera S. Jeśli tak, to C weryfikuje certyfikat (i sprawdza, czy nie upłynął jego termin ważności oraz czy nie został unieważniony). Jeśli weryfikacja się udaje, to C przyjmuje do wiadomości, że pk_S jest kluczem publicznym serwera S. Następuje utworzenie wstępnego klucza symetrycznego sesji pmk (pre-master key) , który zostaje zaszyfrowany kluczem publicznym serwera S. Zaszyfrowany klucz wstępny jest odsyłany do serwera S (szyfrogram c).
   pmk służy do wyprowadzenia głównego klucza (master key) „mk” z zastosowaniem funkcji wyprowadzającej klucze, której wejściem są: pmk, N_C i N_S. Następnie klient stosuje generator liczb pseudolosowych do wyprowadzenia kluczy $k_{b \to s}$, $k_{s \to b}$.
   Na koniec C oblicza r_C=S_{MAC}(mk, transcript), gdzie transcript oznacza wszystkie wiadomości wymienione pomiędzy C i S , do tej pory. Klient C wysyła wartość r_C do serwera S.
4. S oblicza pmk = Enc(sk_S,c). Wyprowadza z niego $k_{b \to s}$, $k_{s \to b}$. Weryfikuje transcript: V_{MAC}(mk, $r_{C,}$ transcript). Jeśli weryfikacja zawodzi, to następuje przerwanie połączenia. Jeśli weryfikacja przechodzi, to następuje obliczenie r_S = S_{MAC}(mk, transcript’), gdzie transcript’ oznacza wszystkie wiadomości wymienione do tej pory pomiędzy C i S (włączając w to właśnie otrzymaną wiadomość od C). Wartość r_S. Jest odsyłana do klienta C.
5. Jeśli weryfikacja V_{MAC}(mk, r_S, transcript’) się nie powiedzie, to połączenie jest przerywane.

Ostatecznie protokół uzgodnienia powoduje, że po obu stronach znajdują się wymienione klucze $k_{b \to s}$, $k_{s \to b}$.
Tylko uznany przez klienta C serwer S otrzyma pmk, na podstawie którego wygeneruje pozostałe klucze potrzebne od wymiany wiadomości i dokończenia protokołu początkowego. Zastosowanie MAC służy zabezpieczeniu przez atakiem man-in-the-middle, który może np. polegać na próbie wymuszenia stosowania starszych standardów komunikacyjnych bardziej podatnych na atak.

Protokół: TSL Record Protocol(TSL 1.2)



Jednokierunkowe klucze: $k_{b\to s}$ i $k_{s\to b}$

Szyfrowanie utrzymujące stan:

- Każda strona utrzymuje dwa 64-bitowe liczniki: $ctr_{b\to s}$, $ctr_{s\to b}$
- Liczniki są zerowane na początek sesji. Zwiększane o 1 dla każdego rekordu.
- Cel: zapobieganie atakom powtórzeniowym

27

Omówimy sobie rozwiązania szyfrowania z uwierzytelnieniem stosowane w rzeczywistości. Rozważmy więc TLS. Szyfrowanie danych w tym rozwiązaniu odbywa się z zastosowaniem „TLS Record Protocol” (rekord jest tu rozumiany jako porcja danych). Rekord danych jest poprzedzany nagłówkiem (HDR). Jego struktura zaraz będzie omówiona. Dane z nagłówkami służą do obustronnej komunikacji. W modelu komunikacji TLS największą porcję danych, jaką można przenieść w jednym rekordzie jest 16 kilobajtów. Jeśli ilość danych jest większa, to jest fragmentowana na mniejsze, max 16 kilobajtowe rekordy. TLS używa, tak zwanych jednokierunkowych kluczy. Osobny klucz służy do szyfrowania w kierunku od serwera do przeglądarki (browser), a osobny od przeglądarki do serwera. Jeden klucz służy więc wysyłaniu wiadomości, a drugi – odbieraniu. Zarówno serwer jak i przeglądarka znają oba klucze. Klucze są generowane z zastosowaniem tzw. „TLS Exchnge Key Protocol”, o którym będziemy mówili na następnych wykładach. Na chwilę obecną musimy założyć, że klucze zostały wygenerowane i (bezpiecznie) wymienione pomiędzy klientem a serwerem. Wymiana informacji pomiędzy serwerem a przeglądarką odbywa się z tzw. zachowaniem stanu. To znaczy, że na czas wymiany informacji jest utrzymywana sesja (stan) i informacja o przesyłaniu kolejnych pakietów w ramach sesji modyfikuje stan nadawcy i odbiorcy. Z punktu widzenia protokołu najważniejsze są dwa 65-bitowe liczniki utrzymywane po każdej ze stron wymieniających informacje. Jeden z liczników liczy rekordy wysłane do nadawcy, a drugi rekordy, które zostały odebrane. Liczniki są zerowane na początku nawiązania sesji, a potem zwiększane po każdym wysłaniu/odebraniu rekordu. Liczniki są zastosowane, żeby zapobiec atakom powtórzeniowym.

Rekord TLS: szyfrowanie (CBC AES-128, HMAC-SHA1)

$k_{b \to s} = (k_{mac}, k_{enc})$

Strona przeglądarki **enc($k_{b \to s}$, data, $ctr_{b \to s}$) :**

krok 1: tag ⟵ S(k_{mac} , [$++ctr_{b \to s}$ || header || data])

krok 2: pad [header || data || tag] do rozmiaru bloku AES

krok 3: szyfrowanie CBC z k_{enc} i nowym losowym IV

krok 4: dołączenie nagłówka

28

Record Protocol działa w następujący sposób. Konstrukcje jakie zastosowano, to AES-128 i HMAC-SHA-1. TLS stosuje schemat MAC-then-ENCRYPT, więc najpierw następuje obliczenie MAC, a potem zaszyfrowanie. Rozważmy sposób przesyłania danych od przeglądarki do serwera. Klucz przeglądarka-do-serwera ($k_{b \to s}$) składa się z 2 kluczy. Klucza do oblicznia MAC i klucza do szyfrowania. Są one ustalane w procesie inicjalizacji połączenia, który zostanie omówiony później. Ponieważ są osobne klucze: $k_{b \to s}$ (przeglądarka-do-serwera) i $k_{s \to b}$ (serwer-do-przeglądarki), w rezultacie w procesie przesyłania informacji biorą udział 4 klucze. Na rysunku pokazano, jak wygląda pakiet TLS. Nagłówek pakietu zwiera informację o typie pakietu, wersji protokołu oraz o długości pakietu on nie jest szyfrowany. Do procesu szyfrowania danych brany jest klucz $\mathbf{k_{b \to s}}$, dane i licznik ($\mathbf{ctr_{b \to s}}$). Najpierw obliczany jest MAC z danych połączonych z nagłówkiem oraz zwiększonym o jeden licznikiem. Wartość licznika (za wyjątkiem obliczonego z niego MAC) nigdy nie jest przesyłana. Serwer ma wiedzieć, jaka powinna być następna wartość licznika i zachowywać ją w wewnętrznym stanie. Znając ją i mając przesłany MAC może zweryfikować, czy nadesłano następny pakiet. Z punktu widzenia kryptograficznego liczniki są wartościami nonce i ponieważ obie strony wiedzą, jakiej kolejnej wartości należy się spodziewać, to nie musi ona być przesyłana. Do szyfrowania przeznaczone są nagłówek, dane oraz tag. Blok danych jest rozszerzany do rozmiaru akceptowalnego dla algorytmu AES. W tym wypadku stosujemy „prosty" pad. Jeśli w bloku brakuje 5 bajtów, to dołączamy do niego pięć bajtów, każdy z zapisaną w nim wartością 5 (…55555). Sposób uzupełniania ostatniego bloku wiadomości w szyfrowaniu blokowym (nie w generowaniu MAC!) był omówiony na wcześniejszych wykładach. Schemat szyfrowania to CBC z losowym IV. Na koniec do szyfrogramu dołączany jest jawny nagłówek (typ, wersja, długość). Daje to nam cały rekord w protokole TLS, który jest przesyłany do serwera. Dane zaznaczone ciemnym kolorem stanowią część zaszyfrowaną wiadomości, a zaznaczone na biało, to nagłówek, wcześniej zaszyfrowany i „otagowany", żeby go nie można podrobić, ale ostatecznie przesyłany w formie jawnej. Porównując to to wcześniej omówionych schematów posługujemy się tutaj rozwiązaniem MAC-then-ENCRYPT, przy czym włączamy w system licznik, zabezpieczający przed wysyłaniem powielonych wiadomości.

Rekord TLS: rozszyfrowywanie

(CBC AES-128, HMAC-SHA1)

Strona Serwera: **dec($k_{b\to s}$, record, $ctr_{b\to s}$)** :

krok 1: rozszyfrowanie schematu CBC z zastoswaniem k_{enc}

krok 2: sprawdzenie formatu padu:
wysłanie bad_record_mac jeśli się nie zgadza

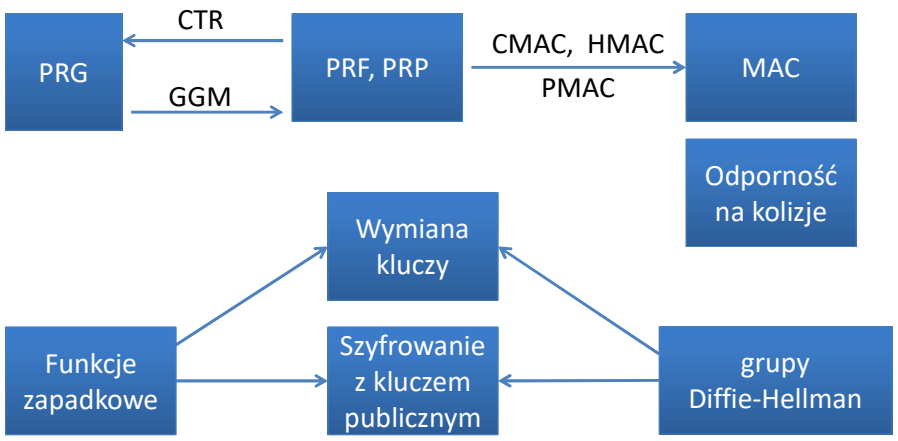
krok 3: sprawdzenie tagu bloku [++$ctr_{b\to s}$ ll header ll data]
wysłanie bad_record_mac jeśli się nie zgadza

Zapewnia szyfrowanie z uwierzytelnieniem
(nie dostarcza żadnych dodatkowych informacji podczas odszyfrowywania)

29

Odszyfrowywanie bloku danych wygląda w następujący sposób. Serwer otrzymuje blok danych (record) i posługuje się własną kopią klucza ($k_{b\to s}$) i własną kopią licznika ($ctr_{b\to s}$) do przeprowadzenia odszyfrowywania danych. W pierwszym kroku następuje uruchomienie algorytmu odszyfrowującego z kluczem k_{enc} . Potem sprawdzany jest format padu. Jeśli się nie zgadza, to odsyłany jest komunikat bad_record_mac i następuje zerwanie komunikacji. Do rozpoczęcia nowego połączenia muszą zostać wynegocjowane nowe klucze sesji. Po sprawdzeniu padu jest on odrzucany i sprawdzany jest MAC wiadomości. Znowu, jeśli sprawdzenie MAC zakończy się porażką protokół odsyła informację bad_record_mac i następuje zerwanie komunikacji. Jeśli wszystko jest OK od wiadomości odrzucany jest nagłówek i tag i przesyłany jako odszyfrowane dane. Proszę zwrócić uwagę, że jeśli ktoś przejmie jakiś blok danych i spróbuje po jakimś czasie przesłać go ponownie do serwera, to zostanie on odrzucony, ponieważ wewnętrzny stan licznika ulegnie zmianie i nie będzie się on zgadał z wartością licznika zapisaną w zaszyfrowanych danych. Liczniki okazują się eleganckim rozwiązaniem zapobiegającym atakom powtórzeniowym. Dodatkowo, ponieważ nadawca i odbiorca utrzymują sesję (w tym stan liczników) nie ma potrzeby włączania wartości liczników w przesyłany komunikat, a same liczniki nie wydłużają długości przesyłanej wiadomości. Zastosowany schemat gwarantuje zachowanie szyfrowania z uwierzytelnieniem, ponadto żadne dodatkowe informacje, poza stwierdzeniem, że odszyfrowanie się nie powiodło nie są wysyłane na zewnątrz. Istnieją ataki na TLS, jeśli system zwraca więcej informacji… W pokazanej wersji TSL znacznik bad_record_mac jest odpowiednikiem znacznika niepowodzenia w odszyfrowywaniu w szyfrowaniu z uwierzytelnieniem. Bardzo ważne jest, że odbiorca/atakujący dowiaduje się tylko , że odszyfrowanie się nie udało, ale nie jest mu podawana przyczyna niepowodzenia. Jeśli tylko dodalibyśmy, że odrzucenie odbyło się z jednej czy drugiej przyczyny, system mógłby być mocno zaatakowany (taki atak zostanie pokazany w dalszej części wykładu).

Krótkie przypomnienie: prymitywy kryptograficzne (1)



30

Krótkie przypomnienie: prymitywy (2)

Ochrona danych niezaszyfrowanych: (integralność danych)

- przy zastosowaniu małego obszaru tylko do odczytu: stosuje się funkcje hash odporne na kolizje
- brak obszaru tylko do odczytu: stosuje się MAC … wymaga posiadania klucza symetrycznego (tajnego)

Ochrona wrażliwych danych: należy używać tylko szyfrowania z uwierzytelnieniem (bezpieczeństwo na podsłuchiwanie to za mało)

Ustalenie sesji:

- Występuje interakcja: należy zastosować bezpieczny protokół wymiany kluczy
- Kiedy nie ma interakcji: kryptografia klucza publicznego

31

Nieomówione tematy

- Autentykacja użytkowników:

hasła, hasła jednorazowe, autentykacja przez wyzwanie

- Mechanizmy prywatności
- Protokoły z zerową wiedzą
- Polityki bezpieczeństwa systemów informatycznych

32

Inne zagadnienia z kryptografii warte przestudiowania

- Kryptografia w oparciu o krzywe eliptyczne
- Obliczenia kwantowe
- Nowe paradygmaty zarządzania kluczami:

Szyfrowanie oparte na identyfikacji i funkcyjne szyfrowanie

- Anonimowa cyfrowa waluta
- Prywatne systemy aukcyjne i do głosowania
- Obliczenia na szyfrogramach pełne homomorficzne szyfrowanie
- Szyfrowanie oparte na drabinkach
- Dwu i wielo instytucjonalne obliczenia

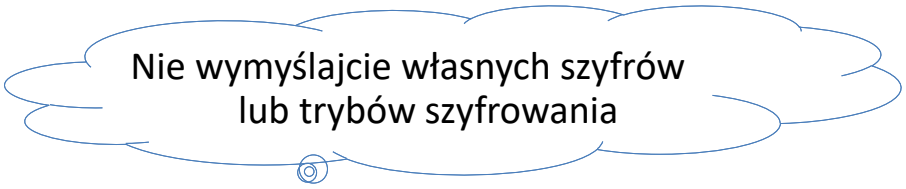
33

Na koniec

Proszę o ostrożność w stosowaniu kryptografii:

- Wspaniałe narzędzie, ale niewłaściwie zaimplementowane doprowadzi to tego, że system będzie pracował, ale będzie go można łatwo zaatakować

W czasie projektowania systemu z kryptografią wymagajcie, aby ktoś inny widział wasz projekt i kod.



Bibliografia

- Jonathan Katz, Yehuda Lindell: INTRODUCTION TO MODERN CRYPTOGRAPHY, Second Edition, CRC Press, 2015.