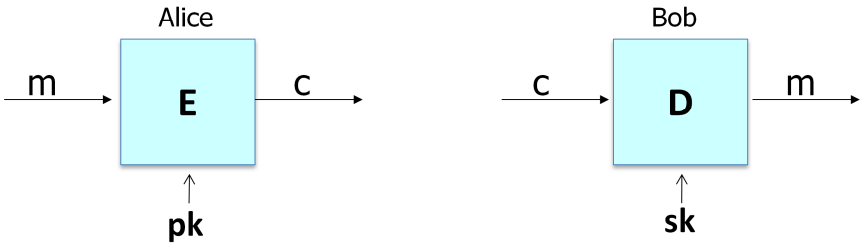
Kryptografia i bezpieczeństwo danych - Kryptografia klucza publicznego - RSA

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

Kryptografia klucza publicznego

Bob: generuje (PK, SK) i przekazuje PK do Alice

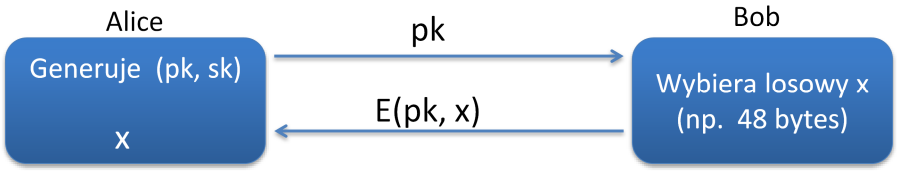


2

Czym jest kryptografia klucza publicznego? Podobnie jak w przypadku szyfrowania symetrycznego dysponujemy dwoma algorytmami: szyfrującym (E) i deszyfrującym (D). Tym razem do szyfrowania stosujemy jeden klucz, nazywany kluczem publicznym. W naszym przypadku nazwijmy go pk, ale do odszyfrowywania stosujemy inny klucz, nazywany kluczem tajnym (sk). Te dwa klucze tworzą parę kluczy. Szyfrowanie polega na uruchomieniu algorytmu szyfrującego wiadomość za pomocą klucza publicznego. Odszyfrowywanie polega na uruchomieniu algorytmu odszyfrowującego z zastosowaniem klucza tajnego.

Zastosowania

Ustanawianie sesji (jak dotąd tylko bezpieczeństwo ze względu na podsłuchiwanie, bez odporności na ataki MiTM)



Zastosowania do nieinteraktywnych aplikacji: (np. Email)

- Bob wysyła email do Alice zaszyfrowany kluczem pk_{alice}
- Uwaga: Bob potrzebuje pk_{alice} (zarządzanie kluczem publicznym)

3

Proste zastosowanie kryptografii z kluczem publicznym (nieodporne na aktywny atak) polega na udostepnieniu klucza swojego klucza publicznego. Odbiorca losuje współdzielony sekret (klucz symetryczny?) i szyfruje go kluczem publicznym nadawcy. Nadawca odszyfrowuje szyfrogram i teraz obaj uczestnicy dysponują tym samym sekretem.

Strukturę klucza publicznego zastosowaliśmy już na laboratorium do szyfrowania poczty. Znowu osoba, do której mają być wysyłane zaszyfrowane wiadomości udostępnia swój klucz publiczny. Pozostali uczestnicy mogą wysyłać do niej zaszyfrowane dane, a tylko ona może je odszyfrowywać. W prostym rozwiązaniu, które już ćwiczyliśmy klucz publiczny można umieścić na swojej stronie, lub dołączyć do swojego emaila. Rozwiązanie GPG oferuje również centralny serwer do przechowywania takich kluczy publicznych i ich zarządzania. Bardziej rozbudowanym rozwiązaniem, które stosuje uwirzytelnienie kluczy i użytkowników jest Infrastruktura Klucza Publicznego.

Szyfrowanie z kluczem publicznym

Definicja: system szyfrowania z kluczem publicznym to trzy algorytmy (G, E, D)

- G(): alg. losowy generujący parę kluczy (pk, sk)
- E(pk, m): alg. losowy biorący $m \in M$ i zwaracający $c \in C$
- D(sk,c): alg. deterministyczny biorący $c \in C$ i zwracający $m \in M$ lub $\perp$

Spójność: $\forall$(pk, sk) zwracanych przez G :

$$\forall m \in M: \quad D(sk, E(pk, m)) = m$$

Semantyczne bezpieczeństwo:

System szyfrowania z kluczem publicznym jest bezpieczny semantycznie.

4

Bardziej formalnie, Szyfrowanie z kluczem publicznym można zdefiniować, jako złożenie trzech algorytmów. Algorytmu G, który służy do generacji kluczy. Po jego działaniu otrzymujemy parę kluczy: publiczny i tajny. Algorytm E, to algorytm szyfrujący, który z zastosowaniem klucza publicznego szyfruje wiadomości. Algorytm D to algorytm odszyfrowujący, który odszyfrowuje szyfrogram z zastosowaniem klucza tajnego, lub zwraca „bottom” (znacznik, że odszyfrowanie się nie udało). Podobnie, jak to miało miejsce w szyfrowaniu symetrycznym algorytmy szyfrowania i deszyfrowania muszą spełnić własność spójności. Dla każdej pary kluczy pk i sk wygenerowanej przez algorytm G wiadomość zaszyfrowana z zastosowaniem klucza publicznego da się odszyfrować z zastosowaniem klucza tajnego.

Odniesienie do szyfrowania symetrycznego

Przypomnienie: dla szyfrów symetrycznych mamy dwie notacje dotyczące bezpieczeństwa:

- Jednokrotne bezpieczeństwo i wielokrotne bezpieczeństwo (CPA)
- Pokazaliśmy, że jednokrotne bezp. $\not\Rightarrow$ wielokrotne bezp.

Dla kryptografii klucza publicznego:

- Jednokrotne bezp. $\Rightarrow$ wielokrotne bezp. (CPA)

wynika to z faktu, że atakujący może sam szyfrować dane z zast. pk)

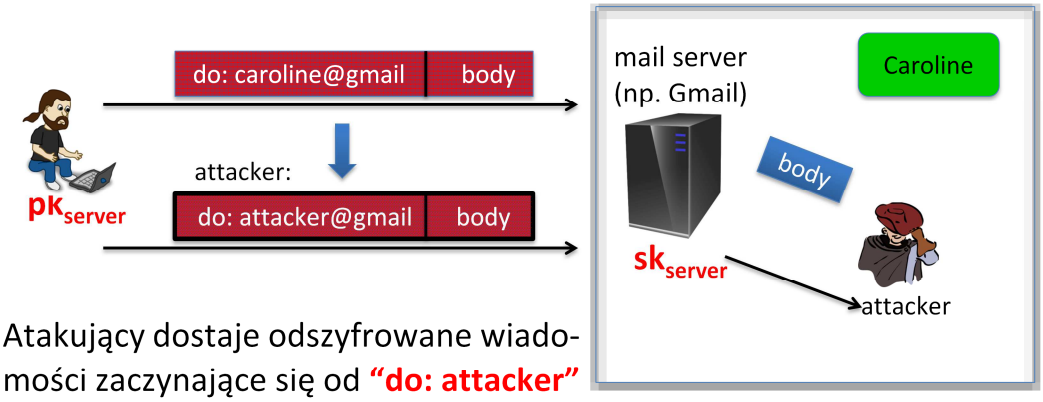
- Szyfrowanie klucza publicznego **musi** być zrandomizowane

5

W kryptografii symetrycznej opisywaliśmy dwa podejścia do zastosowania klucza: kiedy jest on stosowany raz, lub kiedy jest stosowany wiele razy. I pamiętamy, że szyfrowanie jednokrotne (one time pad) było bezpieczne, jeśli klucz stosowaliśmy tylko raz, ale przestawało być bezpieczne, gdy ten sam klucz był zastosowany wielokrotnie.
Jeśli rozważymy kryptografię z kluczem publicznym, to z jej właściwości, jeśli udowodnimy, że jest ona bezpieczna dla pojedynczego użycia klucza, to automatycznie będzie bezpieczna dla jego wielokrotnego użycia. W tym przypadku atakujący, dysponujący kluczem publicznym, sam może generować wiele szyfrogramów. Jeśli udowodnimy, że system jest bezpieczny na takie ataki z tym samym kluczem, to będzie on bezpieczny, jeśli klucze będziemy zmieniać.

Bezpieczeństwo na aktywne ataki

Co jeśli atakujący może fałszować wiadomości?



Atakujący dostaje odszyfrowane wiadomości zaczynające się od **"do: attacker"**

6

Rozważmy następującą sytuację. Bob chce wysłać wiadomość do swojej koleżanki Caroline. Caroline ma kont na Gmail. Bob dysponuje kluczem serwera. Wiadomość jest zaszyfrowana, po dotarciu na serwer jest odszyfrowywana i przesyłana do znalezionego odbiorcy. Jeśli atakujący będzie w stanie zmodyfikować wiadomość, np. zmienić adresata, to wiadomość zostanie przekazana do kogoś innego. Podany schemat jest zbliżony do rzeczywistości. Rzeczywiście Gmail odbiera wiadomości zaszyfrowane z zastosowaniem protokół SSL, odszyfrowuje je i kieruje do odbiorcy. Można założyć, że do szyfrowania zastosowano schemat pozwalający zamianę części wiadomości bez wykrycia tego faktu. Np. wiadomość jest szyfrowana w trybie licznikowym. Atak może polegać na podmianie adresata wiadomości. Atakujący przechwytuje wiadomość do Caroline i zmienia jej cześć zawierającą adresata na inną (attacker). Serwer sam odszyfrowuje wiadomość i przesyła jej treść do atakującego.
Naszym celem jest teraz zbudowanie systemu z kluczem publicznym, który zapobiegnie takiemu atakowi.

Porównanie aktywnych ataków

Przypomnienie: bezpieczne szyfry symetryczne oferują **szyfrowanie z uwierzytelnieniem**

[bezpieczeństwo na atak z wybranym tekstem jawnym & integralność szyfrogramu**]**

- W skrócie: **atakujący nie może stworzyć nowych szyfrogramów**
- W konsekwencji otrzymujemy bezpieczeństwo na atak z wybranym szyfrogramem

W kryptografii klucza publicznego:

- Atakujący **może** tworzyć nowe szyfrogramy z zastosowaniem pk!!
- Więc musimy zapewnić bezpośrednio bezpieczeństwo na atak z wybranym szyfrogramem.

7

Funkcje zapadkowe

(ang. Trapdoor functions)

Def: funkcja zapadkowa $X \longrightarrow Y$ to trójka efektywnych algorytmów (G, F, F^{-1})

- $G()$: losowy algorytm zwracający parę kuczy (pk, sk)
- $F(pk,\cdot)$: deterministyczny algorytm definiujący funkcję $X \longrightarrow Y$
- $F^{-1}(sk,\cdot)$: definiuje funkcję $Y \longrightarrow X$ która odwraca $F(pk,\cdot)$

Dokładnie: $\forall(pk, sk)$ wytworzonego przez G

$$\forall x \in X: \quad F^{-1}(sk, F(pk, x)) = x$$

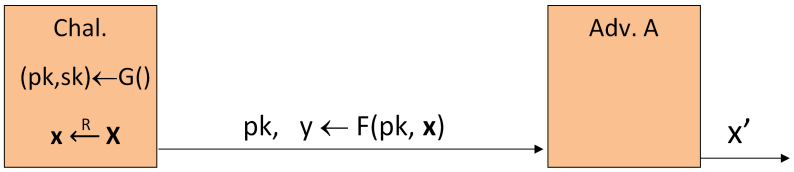
8

Wprowadźmy pojęcie funkcji zapadkowej. Funkcja zapadkowa przekształca pewien zbiór X w Y. Jest definiowana przez trójkę algorytmów. Generator, funkcję f i funkcję odwracającą f. Generator generuje parę kluczy, klucz publiczny i klucz sekretny. Klucz publiczny pozwala na zdefiniowane specyficznej funkcji, która przekształca zbiór X w Y. Wtedy klucz prywatny pozwala na zdefiniowanie funkcji odwracającej funkcję f, pozwalający na przejście ze zbioru Y na zbiór X. Można wykonać funkcję z zastosowaniem klucza pk i odwrócić jej działanie z zastosowaniem klucza sk. Dokładnie, dla każdej pary kluczy pk i sk wygenerowanej przez alg. G możemy dokonać najpierw przekształcenia pewnego zbioru wartości za pomocą funkcji f z kluczem pk, a potem odzyskać te wartości za pomocą funkcji f i klucza sk.

Bezpieczne funkcje zapadkowe

(G, F, F^{-1}) jest bezpieczna jeśli $F(pk, \cdot)$ jest funkcją „jednokierunkową”:

może być obliczona, ale nie może być odwrócona bez sk



Def: (G, F, F^{-1}) jest bezpieczną f. zapadkową, jeśli dla każdego efektywnego algorytmu A

$Adv_{OW}[A,F] = Pr[x = x'] <$ pomijalnie małe

9

Kiedy funkcja zapadkowa będzie bezpieczna? Bezpieczeństwo zachodzi, jeśli funkcja F jest jednokierunkowa. Taką funkcję jest łatwo obliczyć (zastosowanie pk), ale prawie niemożliwe jest jej odwrócenie bez dostępu do sk.

Wyprowadzenie kryptografii klucza publicznego z funkcji zapadkowych (1)

- (G, F, F^{-1}): bezpieczna funkcja zapadkowa $X \longrightarrow Y$
- (E_s, D_s) : symetryczne szyfrowanie z uwierzytelnieniem zdefiniowane na (K,M,C)
- $H: X \longrightarrow K$ funkcja hash

Konstruujemy system szyfrowania w oparciu i kryptografię klucza publicznego (G, E, D):

Generator kluczy G: taki sam jak G dla funkcji zapadkowych

10

Stosując pomysł na bezpieczne funkcje zapadkowe można sformułować pojęcie systemu szyfrowania z kluczem publicznym. Pierwszym elementem systemu jest funkcja zapadkowa. Drugim elementem – symetryczny system szyfrowania. Zakładamy, że system z kluczem symetrycznym jest bezpieczny ze względu na aktywne ataki. System szyfrowania symetrycznego pobiera na wejściu klucze ze zbioru K, a funkcja zapadkowa elementy ze zbioru X. To są różne zbiory, dlatego potrzebujemy funkcję hash. Ona przekształca wartości z ze zbioru X na wartości ze zbioru K (mapuje elementy ze zbioru X na elementy ze zbioru K).
Mając te wszystkie elementy możemy zacząć konstruować system szyfrowania z kluczem publicznym w następujący sposób. System składa się z 3 elementów (G, E, D).
Generowanie kluczy w naszym systemie będzie się odbywać dokładnie w taki sam sposób, jak w przypadku funkcji pułapkowych. Po uruchomieniu generatora kluczy otrzymamy klucz publiczny i klucz sekretny.

Wyprowadzenie kryptografii klucza publicznego z funkcji zapadkowych (2)

- (G, F, F^{-1}): bezpieczna funkcja pułapkowa $X \longrightarrow Y$
- (E_s, D_s) : system szyfrowania symetrycznego z uwierzytelnieniem (K,M,C)
- $H: X \longrightarrow K$ funkcja hash

E(pk, m) :

$x \xleftarrow{R} X$, $y \longleftarrow F(pk, x)$

$k \longleftarrow H(x)$, $c \longleftarrow E_s(k, m)$

wyjście (y, c)

D(sk, (y,c)) :

$x \longleftarrow F^{-1}(sk, y)$,

$k \longleftarrow H(x)$, $m \longleftarrow D_s(k, c)$

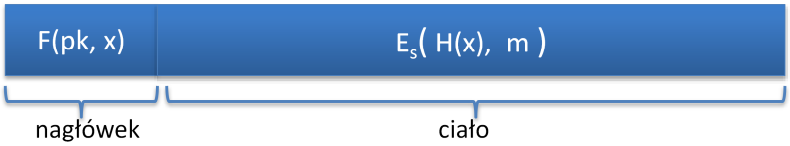
wyjście m

11

Wejściem podsystemu szyfrującego jest klucz publiczny (pk) i wiadomość (m). Najpierw generujemy losową wartość ze zbioru X, a potem stosujemy funkcję zapadkową F z kluczem pk do obliczenia wartości y. y jest przekształceniem wartości x przez funkcję F z kluczem pk. Teraz z kolei przekształcamy z zastosowaniem funkcji hash wartość x na klucz k (klucz symetryczny do „szybkiej" komunikacji szyfrowanej). Na koniec szyfrujemy wiadomość szyfrem symetrycznym z otrzymanym kluczem k. Wyjściem algorytmu są szyfrogram i wartość y. Zwróćmy uwagę, że funkcja zapadkowa jest zastosowana tylko do przekształcenia losowej wartości x w y. Szyfrowanie zaś odbywa się z zastosowaniem szyfru symetrycznego.

Rozważmy proces odszyfrowywania. Algorytm deszyfrujący bierze jako wejście klucz sekretny i parę wartości y i c. Deszyfrowanie rozpoczyna się od odwrócenia wartości y z zastosowaniem funkcji zapadkowej i klucza sk. Na podstawie otrzymanej wartości x z zastosowaniem funkcji hash oblicza się klucz. Algorytm deszyfrujący kryptografii symetrycznej odszyfrowuje szyfrogram z zastosowaniem „odzyskanego" klucza k.

Na rysunku:



<u>Twierdzenie o bezpieczeństwie</u>:

Jeśli
(G, F, F^{-1}) jest bezpieczną funkcją zapadkową,
(E_s, D_s) zapewnia szyfrowanie z uwierzytelnieniem
i **H:** X ⟶ K jest "random oracle"
to **(G,E,D)** jest bezpieczny na atak z wybranym szyfrogramem <CCA^{ro} secure>.

12

Wiadomość w takim schemacie szyfrowania składa się z dwóch pól. Nagłówka zawierającego przekształcenie wartości x z zastosowaniem funkcji zapadkowej oraz szyfrogramu zaszyfrowanego za po mocą klucza otrzymanego przez wykonanie funkcji hash na wartości x. W czasie odszyfrowywania najpierw odszyfrowujemy nagłówek z zastosowaniem odwrotności funkcji zapadkowej i klucza prywatnego, żeby obliczyć x. Następnie obliczmy skrót z x otrzymując klucz symetryczny i odszyfrowujemy wiadomość w systemie szyfrowania z kluczem symetrycznym.
Jest udowodnione twierdzenie, które mówi, że jeśli dysponujemy bezpieczną funkcją zapadkową, system szyfrowania symetrycznego z uwierzytelnieniem oraz funkcję hash generującą ciągi nieodróżnialne od ciągów pseudolosowych, to zaproponowany system jest odporny na ataki z wybranym szyfrogramem (Indeks **ro** oznacza, że bezpieczeństwo zostało wyprowadzone w modelu „random oracle" – wyidealizowanej definicji funkcji hash).
Taki system szyfrowania został ustandaryzowany przez ISO (ang. International Standard Organisation).

Nieprawidłowe posługiwanie się funkcjami zapadkowymi

Nigdy nie szyfrujmy przez zastosowanie funkcji F bezpośrednio na tekst do zaszyfrowania:

E(pk, m) :	**D(sk, c) :**
wyjście $c \leftarrow F(pk, m)$	wyjście $F^{-1}(sk, c)$

Problemy:

- Determinizm: nie można uzyskać bezpieczeństwa semantycznego !!
- Istnieje wiele opublikowanych ataków

13

Warto zwrócić uwagę, na możliwość popełnienia błędu w posługiwaniu się funkcjami zapadkowymi. Pierwszy pomysł (niestety błędny) jaki przychodzi do głowy, to zastosowanie bezpośrednio funkcji zapadkowej oraz klucza pk do zaszyfrowania wiadomości, a następnie zastosowanie funkcji odwrotnej i klucza sk do jej odszyfrowania. Dowodów na niepoprawność takiego systemu szyfrowania jest wiele. Najprościej można wykazać, że taki system jest szyfrowaniem deterministycznym (za każdym razem ta sama wiadomość uzyskuje ten sam szyfrogram). Wybrane ataki na niepoprawnie skonstruowane systemy szyfrowania z kluczem publicznym zostaną omówione później.

Przypomnienie: zapadkowe permutacje

Trzy algorytmy: (G, F, F^{-1})

- G: zwraca pk, sk. pk definiuje funkcję $F(pk, \cdot): X \rightarrow X$
- F(pk, x): oblicza funkcję dla argumentu x
- $F^{-1}(sk, y)$: odwraca funkcję z argumentem y używając sk

Bezpieczna zapadkowa permutacja:

Funkcja $F(pk, \cdot)$ jest jednokierunkowa jeśli nie znamy sk

14

Przypomnienie: arytmetyka liczb złożonych modulo

$p, q \approx \sqrt{N}$

Niech $N = p \cdot q$ gdzie p, q są liczbami pierwszymi

$Z_N = \{0,1,2,\ldots,N-1\}$; $(Z_N)^* = \{\text{odwracalne elementy w } Z_N\}$

Fakty: $x \in Z_N$ jest odwracalne $\Leftrightarrow$ $nwd(x,N) = 1$

- Liczba elementów w $(Z_N)^*$ jest równa

  $\varphi(N) = (p-1)(q-1) = N-p-q+1$

$\varphi(N) \approx N - 2\sqrt{N} \approx N$

Tw. Euler'a: $\forall\, x \in (Z_N)^* : x^{\varphi(N)} = 1$

↳ w $\mathbb{Z}_N$ (arytmetyka modulo N)

15

Przypomnijmy też najważniejsze potrzebne fakty z arytmetyki modulo. Zakładamy, że dysponujemy liczbą złożoną N otrzymaną z iloczynu dwóch liczb pierwszych. Ich rozmiary są podobne. Dla naszych rozważań możemy przyjąć ich rozmiar w przybliżeniu pierwiastkowi kwadratowemu z N. Z_N jest nazywany zbiorem liczb całkowitych od 0 do N-1. $(Z_N)^*$ oznacza zbiór odwracalnych liczb z w zbiorze Z_N. Pamiętamy, że uznajemy jakąś liczbę odwracalną, jeśli jej największy wspólny dzielnik z N wynosi 1, czyli jest względnie pierwsza z N. Przypomnijmy również, że liczba elementów w (ZN)* jest wyrażona funkcją Euler'a i dla N skonstruowanego, jak na początku slajdu wynosi (p-1)(q-1) = N – p – q – 1. Jeśli przyjmiemy, że q i p mają w przybliżeniu rozmiar pierwiastka kwadratowego z N, to wartość funkcji ϕ powinna wynosić w przybliżeniu N-2*sqrt(N), co jest w zasadzie bardzo bliskie samej wartości N. W konsekwencji, jeśli weźmiemy jakiś losowy element ze zbioru Z_N to jest bardzo prawdopodobne, że będzie on należał również do $(Z_N)^*$. Ostatni fakt, o którym należy pamiętać to to, że jeśli wezmę jakikolwiek element x ze zbioru $(Z_N)^*$ i go podniosę do potęgi ϕ(N), to otrzymam 1 (z tw. Euler'a).

Zapadkowa permutacja RSA (1)

Po raz pierwszy opublikowana w:

Scientific American, Sierpień 1977.

RSA: Ron **R**ivest, Adi **S**hamir, Leonard **A**dleman.

Bardzo szeroko stosowana:

- SSL/TLS: certyfikaty i wymiana kluczy
- Chronione e-mail i systemy plików
- … i wiele innych

16

Pułapkowa permutacja RSA była pierwszy rzaz opublikowana w 1977 roku w Scientific Armerican (po polsku: Świat Nauki). Do tej pory ta funkcja jest bardzo popularna w konstrukcjach współczesnych systemów kryptograficznych, takich jak SSL/TSL do wystawiania certyfikatów i wymiany kluczy, w ochronie poczty elektronicznej, czy plików dyskowych.

Pułapkowa permutacja RSA (1)

G(): wybierz losowe l. pierwsze p,q ≈1024 bits.
Ustal **N=p*q**.
wybierz l. całkowite **e , d** takie, że **e·d = 1 (mod φ(N))**
zwróć pk = (N, e) , sk = (N, d)

F(pk, x): $\mathbb{Z}_N^* \rightarrow \mathbb{Z}_N^*$; **RSA(x) = x^e** (w Z_N)

F^{-1}(sk, y) = y^d ; y^d = **RSA(x)^d** = x^{ed} = $x^{k\varphi(N)+1}$ = $\left(x^{\varphi(N)}\right)^k \cdot \mathbf{x}$ = x

17

Aby zdefiniować funkcję pułapkową RSA trzeba pokazać definicje funkcji generowania kluczy G, funkcji F i F^{-1}. Generowanie kluczy rozpoczyna się od wylosowania dwóch liczb pierwszych p i q, o długości zbliżonej do 1024 bity (około 300 cyfr). Liczba systemu modularnego, ma którym będziemy pracować jest iloczynem tych dwóch liczb. Wybieramy takie dwie liczby e i d, takie że ich iloczyn wynosi 1 modulo ϕ(N). To oznacza, że są względem siebie pierwsze oraz jedna jest odwrotnością drugiej (modulo ϕ(N)). Zwracamy klucz publiczny jako para (N, e) i klucz sekretny jako para (N, d). e jest nazywana wykładnikiem szyfrującym, a d – wykładnikiem deszyfrującym.
Funkcja zapadkowa przekształca zbiór $(Z_N)^*$ w $(Z_N)^*$. Dla wejścia x funkcja oblicza wartość wyrażenia x^e (w zbiorze Z_N) i je zwraca. Aby odszyfrować jakąś wartość (y) podnosimy ją do potęgi d, czyli obliczamy y^d. Oznacza to, że obliczamy x^{ed}. Ponieważ e*d = 1 (mod ϕ(N)), to wykładnik x możemy zapisać: kϕ(N)+1. Czyli wykonanie takiego potęgowania zwraca nam z powrotem x.
Trzeba sobie oczywiście odpowiedzieć na pytanie, dlaczego tak sformułowana funkcja zapadkowa staje się jednokierunkowa, jeśli nie dysponujemy kluczem sekretnym?

Założenia RSA

Założenie RSA: RSA jest jednokierunkową permutacją

Dla wszystkich efektywnych algorytmów A:

$$\Pr\left[\ A(N,e,y) = y^{1/e}\ \right] < \text{pomijalnie małe}$$

gdzie $p,q \xleftarrow{R}$ n-biowe l. piewsze, $N \leftarrow pq$, $y \xleftarrow{R} Z_N^*$

18

U podstawy bezpieczeństwa rozwiązania RSA leży **założenie**, że zaproponowane przekształcenie jest jednokierunkową permutacją. Dokładnie: zakładając, że p i q są liczbami pierwszymi, N jest ich iloczynem, y jest losową liczbą należącą do $(Z_N)^*$, to posiadając N, e (wykładnik), y nie jesteśmy w stanie znaleźć efektywnego algorytmu obliczającego odwrotność funkcji RSA w y, czyli obliczyć $y^{1/e}$.

Przypomnienie: system klucza publicznego RSA (standard ISO)

(E_s, D_s): system szyfrowania symetrycznego z uwierzytelnieniem.
$H: Z_N \rightarrow K$ gdzie K jest przestrzenią kluczy (E_s, D_s)

- **G**(): Generuj parametry RSA: pk = (N,e), sk = (N,d)
- **E**(pk, m): (1) wybierz losowy x w Z_N
  (2) $y \leftarrow RSA(x) = x^e$, $k \leftarrow H(x)$
  (3) zwróć $(y , E_s(k,m))$
- **D**(sk, (y, c)): zwróć $D_s(H(RSA^{-1}(y)) , c)$

19

Dysponując bezpieczną permutacją zapadkową, możemy ją „włączyć" do znormalizowanego schematu szyfrowania z kluczem publicznym i otrzymać nasz „pierwszy" system szyfrowania z kluczem publicznym. Naszymi „klockami" były: system szyfrowania symetrycznego z uwierzytelnieniem, funkcja hash (mieszająca) przekształcająca wartość x na przestrzeń kluczy systemu szyfrowania symetrycznego. Sam algorytm szyfrowania rozpoczyna się od uruchomienia generatora kluczy RSA. Następnie wybieramy losową liczbę w Z_N, wykonujemy na niej funkcję RSA (otrzymując wartość y), wyprowadzamy z wartości x z zastosowaniem funkcji hash klucz szyfrowania symetrycznego k. Ostatecznie zwracamy wiadomość złożoną z wartości y i zaszyfrowanej wiadomości m z zastosowaniem klucza k (szyfrowanie symetryczne). Zwykle funkcja hash, to SHA-256.
Odszyfrowywanie polega na wykonaniu funkcji odwrotnej na RSA (z kluczem sk), aby odzyskać wartość x, następnie tę wartość trzeba przekształcić z zastosowaniem funkcji hash, żeby odzyskać klucz, który z kolei służy do odszyfrowywania szyfrogramu (zaszyfrowanej wiadomości).
Przy założeniu, że: system szyfrowania symetrycznego z uwierzytelnieniem jest odporny na atak z wybranym szyfrogramem, funkcja hash daje pseudolosowe wyniki i funkcja RSA jest jednokierunkowa, to system jest bezpieczny.

Bezpośrednie szyfrowanie danych z zastosowaniem RSA nie jest bezpieczne

Bezpośrednie szyfrowanie danych z zastosowaniem RSA:

- Klucz publiczny: **(N,e)** Szyfruj: $\mathbf{c} \longleftarrow \mathbf{m}^{\mathbf{e}}$ (in Z_N)
- Klucz sekretny: **(N,d)** Deszyfruj: $\mathbf{c}^{\mathbf{d}} \longrightarrow \mathbf{m}$

System szyfrowania nie jest bezpieczny!!

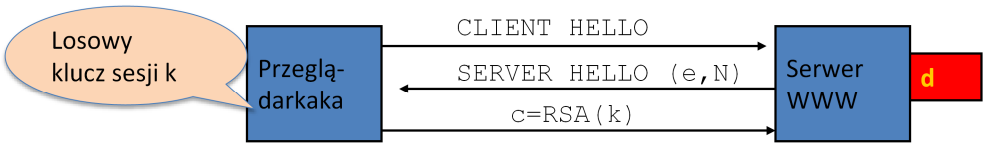
- Nie jest bezpieczny semantycznie i istnieje bardzo wiele ataków na niego

⇒ Permutacja jednokierunkowa RSA nie jest schematem szyfrującym!

20

Jeśli mamy nasz pierwszy system z kluczem publicznym, to teraz przedyskutujemy, jak go **nie używać** do szyfrowania. Po pierwsze pokażemy, że bezpośrednie stosowanie szyfrowania z kluczem publicznym (**bez generacji** x i jej skracania do k i wykonania szyfrowania symetrycznego z przekazaniem wartości y) nie jest bezpieczny.
W naszym (błędnym) podejściu stosujemy podejście RSA bezpośrednio do szyfrowania. Podnosimy wiadomość m do potęgi e (w Z_N), deszyfracja polega na podniesieniu szyfrogramu do potęgi d uzyskując wiadomość. Można znaleźć wiele książek, które właśnie w taki sposób opisują RSA. Takie rozwiązanie jest zupełnie pozbawione bezpieczeństwa, choćby dlatego, że jest to system szyfrowania deterministyczny (taka sama wiadomość otrzyma zawsze taki sam szyfrogram!). Jest bardzo wiele ataków na takie zastosowanie RSA do szyfrowania. Poza tym algorytm RSA jest tylko zapadkową permutacją, a nie schematem szyfrującym. Musi zostać obudowany schematem, jak na poprzednim slajdzie, aby uzyskać bezpieczeństwo.

Prosty atak na wiadomości bezpośrednio szyfrowane RSA



Załóżmy, że k jest 64 bitowe: $k \in \{0,\ldots,2^{64}\}$. Ewa widzi: $c = k^e$ w Z_N

Jeśli $\mathbf{k = k_1 \cdot k_2}$ gdzie $k_1, k_2 < 2^{34}$ (prawd. ≈20%) wtedy $\mathbf{c/k_1^e = k_2^e}$ w Z_N

Krok 1: zbuduj tablicę: $c/1^e, c/2^e, c/3^e, \ldots, c/2^{34e}$. czas: 2^{34}

Krok 2: dla $k_2 = 0,\ldots, 2^{34}$ sprawdź, czy k_2^e jest w tablicy. time: 2^{34}

Wyjście pasujące klucze (k_1, k_2). Czas całkowity ataku: $\approx 2^{40} << 2^{64}$

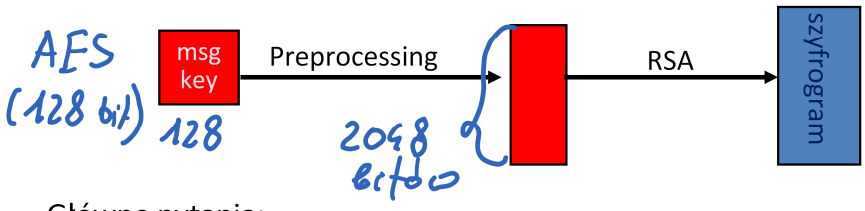
21

Załóżmy, że mamy serwer WWW, który posiada klucz d RSA. Mamy przeglądarkę, która chce ustalić bezpieczne powiązanie SSL pomiędzy klientem a serwerem. SSL rozpoczyna sesję od wysłania informacji do serwera, że chce nawiązać bezpieczne połączenie. Serwer odpowiada wiadomością zawierającą klucz publiczny serwera. Przeglądarka szyfruje wygenerowany przez siebie losowy klucz sesji i odsyła go do serwera WWW zaszyfrowany kluczem publicznym z zastosowaniem RSA. Teraz serwer i przeglądarka dysponują kluczem symetrycznym do wymiany informacji z zastosowaniem szyfrowania symetrycznego.
Zastanówmy się, co się stanie, jeśli bezpośrednio z zastosowaniem RSA „zaszyfrujemy" klucz sesji k, czyli szyfrogram klucza wynosi k^e (mod Z_N). W rozważania założymy, że klucz jest 64 bitowy i traktujemy go jako liczbę z zakresu 0 do 2^{64}, a dokładnie z zakresu 0 do 2^{64}- 1. Załóżmy także, że klucz ta się rozłożyć na dwie liczby o podobnej długości. Możemy go zapisać jako iloczyn k1 i k2, gdzie każdy z nich jest mniejszy od 2^{32}. Okazuje się, że prawdopodobieństwo trafienia na klucz, który można tak przedstawić wynosi ok. 20%. Teraz wstawiamy tak „rozłożony" klucz w zależność obliczającą szyfrogram (k^e). Jedną z części klucza „przenosimy" ma druga stronę równania i otrzymujemy: $\mathbf{c/k_1^e = k_2^e}$ w Z_N. Atakujący zna c, e i N. Ponieważ udało się nam rozdzielić klucz możemy przeprowadzić atak „spotkajmy się w środku". Najpierw przygotowujemy tablicę wszystkich możliwych ilorazów c/n^e. Po przygotowaniu tablicy porównujemy, która z wartości w tablicy odpowiada wartości k_2^e. Po znalezieniu takiej „kolizji" możemy obliczyć wartość klucza, mnożąc k1 przez k2. Jaki jest czas takiego ataku? Brutalny atak trwałby 2^{64}. Pokazany atak może się odbyć w czasie 2^{40}.
Ważny wniosek z tej analizy jest taki, że jeśli zastosujemy bezpośrednio obliczanie permutacji RSA do szyfrowania, to można taki system złamać szybciej niż za pomocą brutalnego ataku. Podkreślmy jeszcze raz, nigdy nie należy stosować RSA bezpośrednio do szyfrowania danych, kluczy itp. Powinien być częścią odpowiednio zaprojektowanej struktury kryptograficznej.

RSA w praktyce

Nie stosujemy RSA do bezpośredniego szyfrowania danych czy kluczy!

RSA w praktyce (ponieważ standard ISO nie jest często stosowany) :



Główne pytania:

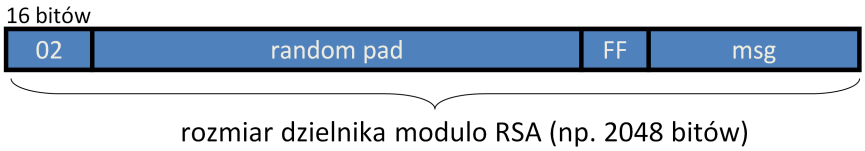
- Jak powinien być przeprowadzony „preprocessing”?
- Czy możemy udowodnić bezpieczeństwo takiego systemu?

22

Podany wcześniej ustandaryzowany system szyfrowania z kluczem publicznym rzadko jest używany w powiązaniu z RSA. RSA jest stosowany do czegoś innego. Zwykle system „osobno” generuje klucz kryptografii symetrycznej, a system RSA jest „proszony” o zaszyfrowanie tego klucza. Przykładowo, tworzymy 128-bitowy klucz AES, rozszerzamy do długości 2048 bitów i stosujemy RSA do zaszyfrowania takiego klucza. Powstaje pytanie, jak powinno zachodzić takie przekształcenie? Drugą kwestą jest, czy takie podejście jest bezpieczne?

PKCS1 v1.5

PKCS1 tryb 2: (szyfrowanie)



rozmiar dzielnika modulo RSA (np. 2048 bitów)

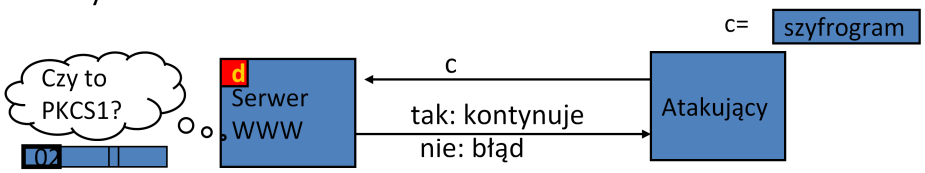
- Otrzymana wartość jest szyfrowana RSA
- Szeroko zastosowane, np. w HTTPS

23

Jednym z praktycznych realizacji pomysłu pokazanego na poprzednim slajdzie jest konstrukcja opisana w dokumencie PKCS1 v1.5 (Public Key Cryptography Standard). Tryb 2 oznacza szyfrowania, tryb 1 oznacza podpisywanie. Zaczynamy od wybrania wiadomości do zaszyfrowania, np. 128 bitowy klucz AES i umieszczamy ją jako ciąg najmniej znaczących bitów w bloku danych do zaszyfrowania. Przed wiadomością dołączamy blok 16 bitów składających się z samych jedynek. Potem dodajemy losowe pole danych, z takim zastrzeżeniem, że nigdzie w nim nie może wystąpić 16 bitów samych jedynek. Na początek pola danych zamieszcza się wartość 0x02 (16 bitów), oznaczająca, że sposób kodowania danych jest zgodny z PKCS1 w trybie 2. Rozszerzony w taki sposób klucz jest wejściem funkcji RSA i jest podnoszony do potęgi e (mod N). W rezultacie otrzymujemy szyfrogram zgodny z PKCS1 v1.5. Odbiorca odwraca funkcję RSA, odczytuje 2 pierwsze bajty, usuwa wszystkie dane wraz z 16-bitowym ciągiem 0xFF i otrzymuje wiadomość, która była zaszyfrowana. Ten mechanizm jest szeroko stosowany w wielu rozwiązaniach, między innymi w protokole HTTPS.

Atak na PKCS1 v1.5 (Bleichenbacher 1998) (1)

PKCS1 stosowany w HTTPS:



⇒ Atakujący może sprawdzić czy 16 bitów na początku to '02'

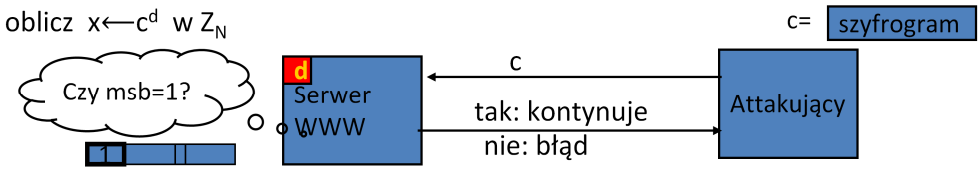
Atak z wybranym szyfrogramem: Żeby odszyfrować dany szyfrogram c wykonuj::

- Wybierz $r \in Z_N$. Oblicz $c' \longleftarrow r^e \cdot c = (r \cdot \text{PKCS1}(m))^e$
- Wyślij c' do serwera WWW i wykorzystaj odpowiedź

24

Schemat PKCS1 v1.5 był opracowany w późnych latach 80-tych. Nie było dowodu, że jest on bezpieczny. W 1998 roku został opublikowany przez Bleichenbacher'a w 1998 roku atak na taki schemat szyfrowania. Załóżmy, że atakujący może przechwycić szyfrogram c. Wiemy, że szyfrowana była wiadomość zgodna z PKCS1. Atakujący wysyła szyfrogram bezpośrednio do serwera WWW. Serwer odszyfrowuje szyfrogram i sprawdza, czy pierwsze 2 bajty zawierają wartość 0x02. Jeśli tak, to następuje interpretacja wiadomości, jeśli nie, to zwrócenie błędu. Takie działanie serwera WWW daje atakującemu informacje, czy początkowa wartość szyfrogramu zawiera, czy nie zawiera sekwencji 2 bajtów 0x02. Okazuje się, że to wystarczy do odszyfrowania każdego szyfrogramu przejętego przez atakującego z uwzględnieniem pewnego wycieku informacji wynikającej z konstrukcji RSA.
Atakujący najpierw wysyła szyfrogram bez zmian do serwera i dostaje informację, czy on się zaczyna od 0x02, czy nie. Jeśli tak, to przygotowuje on spreparowany szyfrogram następującej postaci. Wybiera losową wartość r należącą do zbioru Z_N i podnosi ją do potęgi e. Następnie wprowadza i mnoży ją przez szyfrogram. Z własności potęgowania można zauważyć, że taki zabieg jest tożsamy z pomnożeniem wiadomości niezaszyfrowanej z r (w czasie szyfrowania będzie ta wartość podnoszona do potęgi e). Taka wartość jest wysyłana jako szyfrogram do serwera. Serwer pozwala na stwierdzenie, czy pierwsze 2 bajty wynoszą 0z02, czy nie.
Spoglądając na to z innej perspektywy możemy powiedzieć, że zastanawiamy się ile wynosi pewna liczba x, taka, że pomnożona przez znaną nam liczbę (mod N) da nam na początku wartość 0x02. Okazuje się, że zadając odpowiednia liczbę pytań (1 mln) można dotrzeć do wartości x.

Uproszczony atak Bleichenbacher'a



Załóżmy, że $N = 2^n$ (nieprawidłowy dzielnik RSA). Wtedy:

- Wysłanie c ujawnia msb(x)
- Wysłanie $\mathbf{2^e \cdot c = (2x)^e}$ w Z_N ujawnia msb(2x mod N) = $\mathbf{msb_2(x)}$
- Wysłanie $\mathbf{4^e \cdot c = (4x)^e}$ w Z_N ujawnia msb(4x mod N) = $\mathbf{msb_3(x)}$
- … i tak dalej, aż do ujawnienia całego x

25

Pokażmy przykład takiego ataku w formie uproszczonej. Atakujący może wysyłać szyfrogramy, serwer WWW dysponuje kluczem sekretnym d. W uproszczonym modelu zakładamy, że serwer stwierdza, czy najstarszy bit wiadomości wynosi 1 czy 0. Drugie upraszczające założenie polega na przyjęciu $N=2^n$. Wiemy, że wysyłając szyfrogram dowiadujemy od serwera się o stanie najstarszego bitu. Teraz możemy pomnożyć szyfrogram przez 2, co w praktyce oznacza przesunięcie wiadomości o jeden bit w lewo. Serwer znowu może nam odpowiedzieć, czy bit jest, czy nie jest jedynką. Mnożąc kolejno szyfrogram przez kolejne potęgi 2 de facto przesuwamy go o jeden bit i dowiadujemy się, ile ten bit wynosi. Tak możemy uzyskać informację o całej wiadomości. „Prawdziwy atak" wymaga przeprowadzenia większej ilości zapytań, ponieważ nie analizujemy pojedynczego bitu, ale porównujemy 2 bajty.
Okazało się więc, że schemat PKCS1 v1.5 zastosowany do ochrony klucza symetrycznego można było złamać…

Obrona HTTPS (RFC 5246)

Można uniknąć ataków odkrytych przez Bleichenbacher'a, Klima i innych przez ulepszenie niewłaściwie skonstruowanego bloku danych w taki sposób, że nowy blok będzie nierozróżnialny od poprawnie sformułowanych bloków RSA. Czyli:

1. *Generujemy łańcuch **R** składający się z 48 losowych bajtów*
2. *Odszyfrowujemy wiadomość, żeby uzyskać jawny tekst*
3. *Jeśli padding PKCS#1 nie jest poprawny, to*

   *pre_master_secret = **R***

26

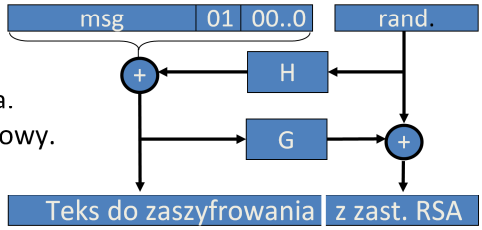
Jak możemy się obronić, przed takim atakiem? Odpowiedzią jest dokument RFC 5246. Sugeruje on następujące rozwiązanie: jeśli odszyfrowany teks tnie odpowiada formatowi danych zdefiniowanych w PKCS#1 (nie zaczyna się od 0x02), to wybieramy losowy łańcuch R 46 bajtowy i udajemy, że wiadomością, która była zaszyfrowana jest właśnie ten łańcuch. Oczywiście protokół nie będzie w stanie odszyfrować poprawnie danych, ale wydarzy się to później… Po prostu klucze sesji nie będą jednakowe.
Taka „poprawka" jest wprowadzona do współczesnych systemów HTTPS. Była łatwa do implementacji i zapobiega omówionemu wcześniej atakowi.
Powstaje pytanie, czy tak skonstruowany HTTPS jest bezpieczny na atak z wybranym szyfrogramem?

PKCS1 v2.0: OAEP

(ang. Optimal Asymmetric Encryption Padding))

Zaproponowano nową funkcję obliczającą preprocessing: OAEP [BR94]

sprawdź pad
w czasie odszyfrowywania.
odrzuć CT jeśli nieprawidłowy.



Tw [FOPS'01] : RSA jest bezpieczną zapadkową permutacją $\Rightarrow$
RSA-OAEP jest bezpieczna na atak z wybranym szyfrogramem (CCA attack) kiedy H,G są funkcjami hash generującymi ciągi losowe (random oracles)

W praktyce: używamy SHA-256 dla H i G

27

Tak dochodzimy do kolejnej, nowej wersji dokumentu PKCS#1, proponującej nowy sposób rozszerzania klucza/wiadomości, żeby ją można było skutecznie zabezpieczyć z zastosowaniem funkcji zapadkowej RSA. Wiadomość/klucz do zaszyfrowania zapisujemy na początku bloku danych, potem wstawiamy znacznik 0x01 i tą część bloku wypełniamy zerami. Długość pierwszego i drugiego bloku (o którym za chwilę) zależy od standardu. Drugi blok danych wybierany jest losowo. Razem oba bloki powinny mieć długość klucza RSA (np. 2048 bitów). Zanim dane są wprowadzane na wejście algorytmu RSA wartość losowa (rand) jest przekształcana przez funkcję hash (mieszczającą) generującą ciąg losowy o długości pierwszego bloku. Na wiadomości (z paddingiem) i przekształconej wartości losowej (H(rand)) wykonywany jest xor i przekazywany jako pierwszy blok na wejście RSA. Ten sam blok jest jeszcze raz przekształcany z zastosownaiem funkcji mieszającej (hash) G , dodawany xor do pierwotniej wartości rand i zamieszczany jako drugi blok danych na wejście algorytmu RSA.
W 2001 roku udowodniono twierdzenie, że jeśli funkcje H i G generują ciągi losowe, to zaproponowana konstrukcja jest bezpieczna na atak z wybranym szyfrogramem. W praktyce stosuje się „dobrą" funkcję SHA-256 jako przybliżenie matematycznych właściwości funkcji H i G z twierdzenia.
Nazwa „Optmalna" w podanej metodzie preprocessingu danych pochodzi ot tego, że uzyskujemy szyfrogram o długości minimalnej, równej wyjściu z funkcji RSA. W standardzie ISO ta wylosowywana wartość była dodatkowo dołączana do szyfrogramu. Udowodnione twierdzenie bazuje tylko na właściwościach funkcji zapadkowej RSA. Zastosowanie funkcji zapadkowej o innych właściwościach algebraicznych nie daje gwarancji bezpieczeństwa w takiej konstrukcji. Powstaje więc pytanie, czy można zbudować konstrukcję zapewniającą bezpieczeństwo dla innej lub dowolnej bezpiecznej funkcji zapadkowej.

Ulepszenia OAEP

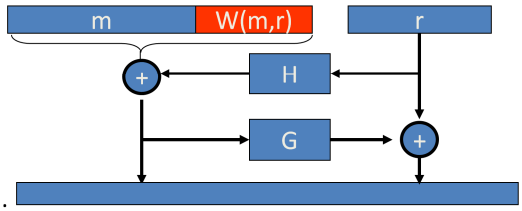
OAEP+: [Shoup'01]

∀ zapadkowej permutacji F

F-OAEP+ jest CCA bezpieczne, kiedy

H,G,W generują ciągi losowe.

Podczas odszyfrowywania sprawdź pole W(m,r).



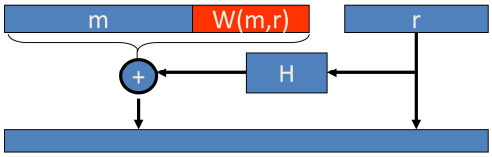
SAEP+: [B'01]

RSA (e=3) jest zapadkową permutacją

⇒

RSA-SAEP+ jest CCA bezpieczne, kiedy

H,W generują ciągi losowe.



28

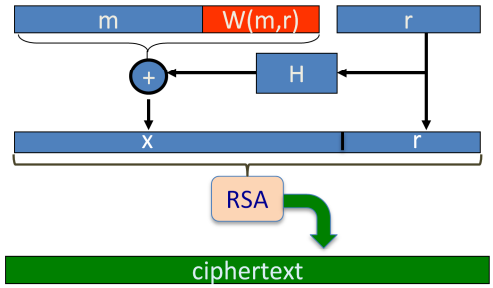
Okazuje się, że jeśli zamiast paddingu 0x01000… dołączymy do wiadomości dołączymy rezultat funkcji mieszającej (hash), dla której wejście podamy wiadomość m i wylosowaną wartość r, to przy zachowaniu schematu OAEP otrzymamy system bezpieczny na atak CCA.

Wykazano również, że jeśli do wiadomości dołączymy w pierwszym bloku wynik działania funkcji mieszającej na m i r oraz wartość e=3, to schemat, w którym stosujemy funkcję zapadkową RSA można uprościć i wciąż będzie bezpieczny ze względu na atak CCA.

Oba wymienione warianty nie są w rzeczywistości stosowane, ale warto o nich wiedzieć.

Ostatecznie standardem jest PKCS1 z OAEP.

Jak odszyfrować szyfrogram zaszyfrowany z zastosowaniem SAEP?



$(x,r) \longleftarrow RSA^{-1}(sk,ct)$, $(m,w) \longleftarrow x \oplus H(r)$, zwróć m jeśli w = W(m,r)

29

Jaki jest schemat odszyfrowywania konstrukcji SAEP? Zaczynamy od funkcji odwracającej RSA, która zwróci wartość x i r. Potem musimy obliczyć funkcję hash na r i wykonać na otrzymanej wartości xor z x. Rezultatem będzie „pierwszy" blok danych zawierający wiadomość i hash z r…
Ostatni etap deszyfrowania powinien sprawdzić, czy W(m, r) z wiadomości jest identyczny z wartością W(m, r) obliczoną przez nas. Jeśli tak, to mamy prawidłowo odszyfrowane dane, jeśli nie zgłaszamy niepowodzenie odszyfrowywania.
Proszę zwrócić uwagę, że sprawdzenie wartości W(m, r) jest bardzo ważne w naszym schemacie. Ten etap odszyfrowywania (gdy znamy już treść wiadomości) pozwala sprawdzić, czy wiadomość zachowała integralność.
Tak naprawdę sprawdzenie padu w każdym z podanych wcześniej schematów jest obowiązkowe dla wykazania integralności wiadomości.

Subtelności w implementacji OAEP (M'00)

```
OAEP-decrypt(ct):
        error = 0;
        ……..
        if  ( RSA^-1(ct) > 2^(n-1) )
              { error =1;  goto exit; }
        ……..
        if  ( pad(OAEP^-1(RSA^-1(ct))) != "01000" )
              { error = 1;  goto exit; }
```

Problem: różne czasy zgłaszania błędow stanowią wyciek informacji

$\Rightarrow$ Atakujący może odszyfrować każdy szyfrogram

Lekcja: Proszę nie implementować samodzielnie RSA-OAEP!

30

Implementacja OAEP może nieść kilka pułapek, które mogą osłabić system. Zastanówmy się, jak napisać program odszyfrowujący omawianą strukturę. Najpierw odwracamy funkcję RSA. Jeśli wynik potęgowania przekroczy wartość 2^{n-1}, to wykryliśmy błąd i dalsze przetwarzanie nie ma sensu, więc powinniśmy przerwać działanie funkcji. W kolejnym etapie sprawdzamy, czy PAD ma ustaloną przez nas postać „01000". Jeśli nie znowu wykryliśmy błąd i powinniśmy przerwać obliczenia.
Teka implementacja jest wrażliwa na atak czasowy, bo atakujący może się dowiedzieć, co spowodowało błąd: pierwszy, czy drugi etap odszyfrowywania. Atakujący z taką wiedzą może złamać system.
Proszę więc nie implementować samemu omówionego systemu szyfrowania. Może on być matematycznie poprawny, i możliwy do złamania.

Czy RSA jest jednokierunkową permutacją?

Żeby odwrócić jednokierunkową funkcję RSA (bez znajomości d) atakujący musi wyodrębnić:

$$x \quad z \quad c = x^e \pmod N.$$

Jak złożony jest problem wyliczenia pierwiastka e-tego rzędu modulo N ??

Najlepszy znany algorytm:

- Krok 1: rozłóż N na czynniki pierwsze (trudny)
- Krok 2: oblicz pierwiastek e-tego rzędu modulo p i q (łatwy)

31

Zastanówmy się, jak trudno jest odwrócić funkcję RSA, mając do dyspozycji klucz publiczny. Dysponujemy wartością x^e i poszukujemy x. Najlepszy znany algorytm rozwiązujący ten problem rozkłada wartość N na czynniki pierwsze a następnie dysponując nimi oblicza pierwiastki. Okazuje się, że znalezienie pierwiastków z zastosowaniem czynników jest łatwy, natomiast rozkład na czynniki pierwsze – trudny i w trudności tego algorytmu mieści się trudność znalezienia odwrotności funkcji RSA bez zanajomości klucza sekretnego.

Drogi na skróty?

Czy musimy dokonać rozkładu na czynniki pierwsze liczby N aby obliczyć pierwiastki e-ntego stopnia?

Aby wykazać, że nie istnieją „skróty” musimy pokazać redukcję:

- Istnienie efektywnego algorytmu wyliczające pierwiastki e-tego stopnia mod N

  $\Rightarrow$ efektywny algorytm faktoryzacji N.

- Najstarszy problem w kryptografii klucza publicznego.

Pewne dowody, że brak jest redukcji: (BV’98)

- “Algebraiczna” reduction $\Rightarrow$ faktoryzacja jest łatwa.

32

Powstaje pytanie, czy rzeczywiście musimy dokonywać faktoryzacji, aby rozwiązać problem? Jak moglibyśmy matematycznie wykazać taką właściwość? Wystarczyłoby dowieść, że istnienie efektywnego algorytmu obliczającego e-nte pierwiastki mod N ZAWSZE sprowadza się do rozwiązania problemu nie szybszego niż problem faktoryzacji. Gdybyśmy mieli taki dowód, to wiedzielibyśmy, że RSA nie da się szybciej złamać, jak w czasie na rozkład na czynniki pierwsze. Jest to najstarszy problem w kryptografii klucza publicznego.

Są wprawdzie pewne przesłanki, że redukcja nie istnieje, ale bardzo słabe, opublikowane w pracy w skazanej na slajdzie. Jak dotąd, RSA jest więc uznawana za funkcję jednokierunkową i trzeba użyć algorytmu o złożoność porównywalne z rozkładem na czynniki pierwsze wartości N, aby ją odwrócić bez klucza sekretnego.

Jak **nie** poprawiać wydajności RSA

Aby przyspieszyć odszyfrowywanie RSA można użyć małego klucza prywatnego d ($d \approx 2^{128}$)

$c^d = m \pmod N$

$N = 2^{2048}$
$d < 2^{512}$

Wiener'87: jeśli $d < N^{0.25}$ wtedy RSA nie jest bezpieczne.

BD'98: jeśli $d < N^{0.292}$ wtedy RSA nie jest bezpieczne

(problem otwarty dla: $d < N^{0.5}$)

$0.292 \approx 1 - \frac{1}{\sqrt{2}}$

Brak bezpieczeństwa: klucz prywatny d może być obliczony z (N,e)

33

Kolejnym problemem implementacyjnym algorytmu RSA jest dość długi czas wykonywania. Dlatego od lat prowadzone są badania, jak poszczególne etapy obliczeń przyspieszyć. Wiemy, że szyfrowanie i deszyfrowanie polega na podnoszeniu liczby do pewnej duże potęgi, co ma złożoność obliczeniową $O((\log x) \cdot n^2)$.

Rozważmy pomysł (niestety nieudany) polegający na skróceniu długości klucza deszyfrującego.

Niech d będzie miał wielkość ok. 2^{128}. Wartość ta jest już odporna na próbę znalezienia jej metodą brutalnego ataku. Trzeba pamiętać, że wciąż wartość x do d ma być dużą liczbą, rzędu 2000 bitów. Zastosowanie krótkiego klucza przyspieszyłoby wtedy odszyfrowywanie 20-krotnie.

Okazuje się jednak, że jest to złe podejście. Tak złe, że zastosowanie tak krótkiego klucza d pozwala na wyliczenie go na podstawie klucza publicznego! Opublikowane ataki pokazują, że jeśli klucz ma długość mniejszą niż trochę więcej niż $2^{1/3}$, to można go efektywnie wyliczyć na podstawie klucza publicznego.

RSA z małym wykładnikiem w kluczu publicznym

Aby przyspieszyć szyfrowanie RSA można użyć małego e:

$c = m^e \pmod N$

- Wartość minimalna: **e=3** (nwd(e, $\varphi(N)$) = 1)
- Rekomendowana wartość: **e=65537=2^{16}+1**

Szyfrowanie zajmuje 17 mnożeń

(RSA-CRT)?

Asymetryczność of RSA: szybkie szyfrowanie / wolne deszyfrowanie

- ElGamal:

  w przybliżeniu taki sam czas szyfrowania/deszyfrowania.

34

W odróżnieniu od wcześniejszego przypadku, nie ma problemu, jeśli zastosujemy jako klucz publiczny małej wartości. Wartość minimalna to 3. Rekomendowaną wartością jest 2^{16}-1. Prowadzi to zjawiska nazywanego asymetrycznością RSA, gdzie szyfrowanie jest szybkie, a odszyfrowywanie wolne (wymaga ok. 2000 mnożeń). Jest metoda na czterokrotne przyspieszenie deszyfrowania, nazywana RSA-CRT, ale wciąż różnica pomiędzy czasem szyfrowania i deszyfrowania jest rzędu 30 razy. Szybkie szyfrowanie i wolne deszyfrowanie jest specjalną własnością systemu RSA. Inne systemy szyfrowania z kluczem publicznym, np. ElGamal mają porównywalne czasy szyfrowania i deszyfrowania.

Długości kluczy

Bezpieczeństwo systemów z kluczem publicznym powinno być porównywalne do bezpieczeństwa szyfrowania zastosowaniem klucza symetrycznego::

Rozmiar klucza	RSA Rozmiar modułu
80 bity	1024 bity
128 bity	3072 bity
256 bity (AES)	**15360** bitów

35

Przypomnijmy, że żeby zachować podobny poziom bezpieczeństwa na złamanie systemu powinniśmy stosować odpowiednio długie klucze RSA. Przykładowo, zachowanie poziomu bezpieczeństwa na poziomie 128 bitowego klucza symetrycznego wymaga klucza RSA długości 3072 bitów, w praktyce stosuje się 2048…

Ataki na implementacje

Czasowy attack: [Kocher et al. 1997] , [BB'04]
Wyliczenie czasu na obliczenie wartości c^d (mod N) może ujawnić d

Atak na zużycie mocy: [Kocher et al. 1999)
Zmierzenie mocy zużywanej przez kartę chip na wyliczenie c^d (mod N) może wyjawić d.

Atak na błędy w czasie deszyfrowania: [BDL'97]
Wykrycie jednego błędu komputera w czasie obliczania c^d (mod N) może ujawnić d.

Typowa obrona: sprawdź wyjście. 10% zwolnienie.

36

Opublikowano kilka ataków na implementacje RSA. Z punktu widzenia matematycznego były to prawidłowe konstrukcje, jedna z punktu widzenia technicznego realizacje algorytmów ujawniały informacje. Okazało się, że mając możliwość zmierzenia czasy wykonywania odszyfrowywania można było ujawnić d. Uwaga jest taka, że w poprawnej implementacji czas obliczeń powinien być niezależny od wartości parametrów. Drugi atak, w którym mierzono zużycie mocy podczas wykonywania odszyfrowywania, pozwalał na odczyt bitów podczas wykonywania przez kartę algorytmu podnoszenia do potęgi (por. wyk z teorii liczb i pokazany tam szybki algorytm potęgowania). Jeśli karta nie jest odporna na taki atak, jest możliwość uzyskania tajnego klucza takim sposobem. Ostatni atak analizuje wykryte błędy podczas wykonywania deszyfracji (wiadomość została odszyfrowana nieprawidłowo). Jeśli wystąpi choć jeden taki błąd, to można uzyskać wartość d. Zabezpieczeniem jest dodatkowe sprawdzenie rezultatu odszyfrowywania, zanim zostanie przesłany do zlecającego. Obliczany jest po stronie odbiorcy jeszcze raz szyfrogram z otrzymanej wiadomości i jeśli jest on taki sam, jak szyfrogram nadesłany, to proces odszyfrowywania przebiegł prawidłowo. Zwiększa to czas pracy algorytmu o ok. 10%, ale zmniejsza podatność systemu na ataki na błędy deszyfrowania.

Problem z generowaniem kluczy RSA (1) (Heninger et al./Lenstra et al.)

Generowanie kluczy RSA w OpenSSL (w skrócie):

```
prng.seed(seed)
p = prng.generate_random_prime()
prng.add_randomness(bits)
q = prng.generate_random_prime()
N = p*q
```

Załóżmy słabą entropię na początku:

- To samo p będzie generowane na różnych urzadzeniach, ale różne q
- N_1 , N_2 : klucze RSA z różnych urządzeń $\Rightarrow$ nwd(N_1,N_2) = p

37

Ostatni atak pochodzi z ostatnich obserwacji. Okazuje się, że algorytm generowania kluczy RSA może być problematyczny, jeśli na urządzeniu nie można uzyskać wysokiej entropii. Weźmy sposób generowania kluczy przez bibliotekę OpenSSL. Generowanie zaczyna się od wprowadzenia wartości na generator liczb pseudolosowych. Nastepnie generowana pierwsza liczba pierwsza p, następnie dodaje się jeszcze „losowości" do systemu losowania i generuje drugą liczbę pierwszą. Wartość N jest iloczynem wygenerowanych liczb pierwszych. Jeśli założymy, że taka generacja ma nastąpić zaraz po uruchomieniu danego urządzenia(np. FireWall) (nie zdążyło ono jeszcze „nazbierać" losowych zdarzeń, które „nakarmiłyby" jego generator liczb losowych), to liczba p będzie wylosowywana z niewielkiego zbioru, natomiast q będzie miała znacznie większy poziom losowości. W konsekwencji w zbiorze urządzeń (FireWall) z dużym prawdopodobieństwem pojawią się klucze złożone z tej samej p i różnych q. Wystarczy więc obliczyć nwd dla wartości N1 i N2 utworzonych na różnych urządzeniach ale w podobnych warunkach (rozruchu urządzenia), aby odkryć p.

Problem z generowaniem kluczy RSA (2)

Eksperyment: można było dokonać faktoryzacji 0.4% kluczy publicznych HTTPS!!

Lekcja:

- Należy się upewnić, że generator liczb losowych jest odpowiednio zainicjalizowany (zawiera już wysoką entropię), jeśli ma nastąpić generowanie kluczy.

38

Literatura uzupełniająca

- Why chosen ciphertext security matters, V. Shoup, 1998
- Twenty years of attacks on the RSA cryptosystem, D. Boneh, Notices of the AMS, 1999
- OAEP reconsidered, V. Shoup, Crypto 2001
- Key lengths, A. Lenstra, 2004