AES

Formalnie DES posiadał certyfikat bezpieczeństwa amerykańskiego biura standardów do roku 1997 (jego bezpieczeństwo było czterokrotnie certyfikowane, za każdym razem na 5 lat).[2mm]W roku 1997 NIST (*National Institute of Standards and Technology*) ogłosił konkurs na nowy algorytm szyfrujący mający zastąpić DES.[2mm]Wymagano między innymi, żeby algorytm:

- dopuszczał użycie klucza o długości 128, 192 i 256 bitów,
- był szyfrem blokowym o bloku długości 128 bitów,
- mógł być uruchomiony na sprzęcie różnego rodzaju, od kart chipowych po mikrokomputery,
- był odporny na wszystkie znane rodzaje ataków kryptograficznych,
- był odpowiednio szybki,
- był dostępny na całym świecie bezpłatnie (nie był chroniony prawami autorskimi).

AES

Do 15 czerwca 1998 wpłynęło 21 algorytmów. 15 z nich spełniało wymagane kryteria i zostało zaakceptowanych jako oficjalni kandydaci na następcę DES.[2mm]Do sierpnia 1999 algorytmy były oceniane przez NIST, zorganizowano w tym celu między innymi dwie międzynarodowe konferencje. W sierpniu wybrano 5 najlepszych algorytmów do dalszej oceny.[2mm]

- MARS twórcy: IBM
- RC6 twórcy: RSA Laboratories
- Rijndael twórcy: Joan Daemen, Vincent Rijmen
- Serpent twórcy: Ross Anderson, Eli Biham, Lars Knudsen
- Twofish twórcy: Bruce Schneier, John Kelsay, Doug Whiting, David Wagner, Chris Hall, Niels Ferguson

AES

Wszystkie algorytmy, które znalazły się w finale zostały uznane za bardzo bezpieczne. Zwycięzca został wybrany na podstawie innych kryteriów (złożoność obliczeniowa, szybkość działania w różnych implementacjach, prostota algorytmu). [2mm]W październiku 2000, po jeszcze jednej konferencji międzynarodowej, NIST wybrał oficjalnie algorytm Rijndael jako podstawę nowego standardu szyfrowania w USA.[2mm]Nowy standard został nazwany AES (Advanced Encryption Standard).[2mm]Należy podkreślić, że proces wyboru AES-a był jawny i międzynarodowy. Twórcy algorytmu Rijndael są Belgami.[2mm]

AES

Rijndael jest szyfrem iterowanym, ilość rund zależy od długości klucza:

- przy kluczu 128 bitowym – 10 rund,
- przy kluczu 192 bitowym – 12 rund,
- przy kluczu 256 bitowym – 14 rund,

Klucze poszczególnych rund są wyprowadzane z klucza wyjściowego. [2mm]Każda runda przekształca 128 bitowy ciąg w inny ciąg 128 bitowy.

AES

Poszczególne rundy AES-a składają się z wykonywanych w odpowiedniej kolejności następujących operacji, nazywanych warstwami (ang. *layers*):

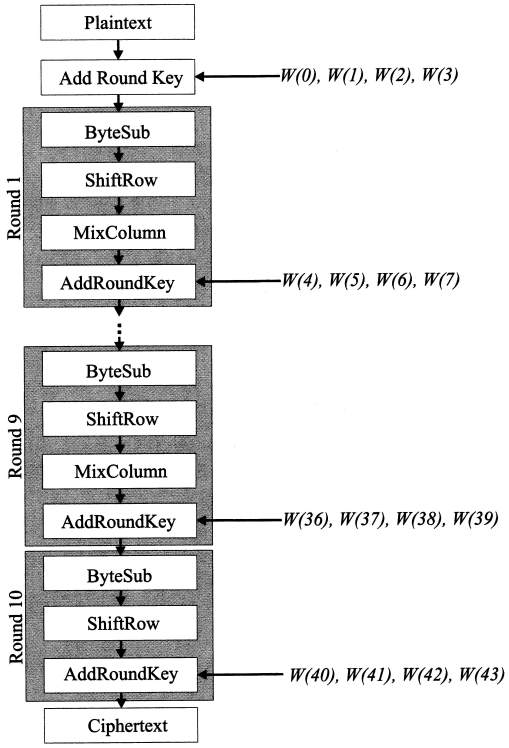
- Transformacja ByteSub (BS) – nieliniowa, z wykorzystaniem S-boxu,
- Transformacja ShiftRow (SR) – liniowa,
- Transformacja MixColumn (MC) – liniowa,
- Transformacja AddRoundKey (ARK) – klucz rundy jest dodawany modulo 2 do rezultatu poprzedniej transformacji.

Typowa runda ma zatem postać:[1mm]

$$\ldots \rightarrow \boxed{\text{ByteSub}} \rightarrow \boxed{\text{ShiftRow}} \rightarrow \boxed{\text{MixColumn}} \rightarrow \boxed{\text{AddRounKey}} \rightarrow \ldots$$

AES

Szyfrowanie w wersji 10-rundowej



Zauważmy, że:

- przed rozpoczęciem pierwszej rundy wykonujemy operację AddRoundKey,
- W ostatniej rundzie nie wykonujemy operacji MixColumn.

AES

Wejściowy blok 128 bitów dzielimy na 16 grup o długości 8 bitów (1 bajt) każda.

$$\underbrace{b_0 b_1 b_2 b_3 b_4 b_5 b_6 b_7} \ldots \underbrace{b_{120} b_{121} b_{122} b_{123} b_{124} b_{125} b_{126} b_{127}}$$

16 bajtów zapisujemy w postaci macierzy 4×4

$$\begin{pmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{pmatrix}$$

gdzie $a_{00} = b_0b_1b_2b_3b_4b_5b_6b_7$, $a_{01} = b_8b_9b_{10}b_{11}b_{12}b_{13}b_{14}b_{15}$ itd.[2mm]Bajty możemy utożsamić z elementami ciała Galois $GF(2^8)$, możemy więc na nich wykonywać operacje dodawania, odejmowania, mnożenia i dzielenia.

AES
Transformacja ByteSub

S-Box

99	124	119	123	242	107	111	197	48	1	103	43	254	215	171	118
202	130	201	125	250	89	71	240	173	212	162	175	156	164	114	192
183	253	147	38	54	63	247	204	52	165	229	241	113	216	49	21
4	199	35	195	24	150	5	154	7	18	128	226	235	39	178	117
9	131	44	26	27	110	90	160	82	59	214	179	41	227	47	132
83	209	0	237	32	252	177	91	106	203	190	57	74	76	88	207
208	239	170	251	67	77	51	133	69	249	2	127	80	60	159	168
81	163	64	143	146	157	56	245	188	182	218	33	16	255	243	210
205	12	19	236	95	151	68	23	196	167	126	61	100	93	25	115
96	129	79	220	34	42	144	136	70	238	184	20	222	94	11	219
224	50	58	10	73	6	36	92	194	211	172	98	145	149	228	121
231	200	55	109	141	213	78	169	108	86	244	234	101	122	174	8
186	120	37	46	28	166	180	198	232	221	116	31	75	189	139	138
112	62	181	102	72	3	246	14	97	53	87	185	134	193	29	158
225	248	152	17	105	217	142	148	155	30	135	233	206	85	40	223
140	161	137	13	191	230	66	104	65	153	45	15	176	84	187	22

S-box wykorzystywany w algorytmie AES

AES
Transformacja ByteSub

Każdy bajt oddzielnie poddawany jest transformacji z wykorzystaniem S-boxu (16×16) przedstawionego na poprzedniej transparencji.[1mm]Transformacja wygląda następująco:

- bajt dzielimy na dwie połówki, każda o długości 4 bity,
- każda połówka 4-bitowa to binarna reprezentacja liczby z zakresu od 0 do 15,
- pierwsza połówka bajtu to numer wiersza w S-boxie, druga połówka to numer kolumny w S-boxie,
- liczba leżąca na przecięciu wskazanego wiersza i kolumny w zapisie binarnym jest bajtem będącym rezultatem działania S-boxu.

Przykład: S-box(01101001)=

AES
Transformacja ByteSub

Transformacja ma zatem postać:[1mm]$a_{00} \rightarrow \boxed{\text{S-box}} \rightarrow b_{00}a_{01} \rightarrow \boxed{\text{S-box}} \rightarrow b_{01} \vdots a_{33} \rightarrow \boxed{\text{S-box}} \rightarrow b_{33}$[3mm]Czyli ostatecznie

$$\begin{pmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{pmatrix} \rightarrow \boxed{\text{ByteSub}} \rightarrow \begin{pmatrix} b_{00} & b_{01} & b_{02} & b_{03} \\ b_{10} & b_{11} & b_{12} & b_{13} \\ b_{20} & b_{21} & b_{22} & b_{23} \\ b_{30} & b_{31} & b_{32} & b_{33} \end{pmatrix}$$

AES

Transformacja ShiftRow

Każdy z wierszy macierzy poddawany jest cyklicznej permutacji:[2mm]pierwszy wiersz o 0 pozycji,[1mm]drugi wiersz o 1 pozycję,[1mm]trzeci wiersz o 2 pozycje,[1mm]czwarty wiersz o 3 pozycje.[3mm]

$$\begin{pmatrix} b_{00} & b_{01} & b_{02} & b_{03} \\ b_{10} & b_{11} & b_{12} & b_{13} \\ b_{20} & b_{21} & b_{22} & b_{23} \\ b_{30} & b_{31} & b_{32} & b_{33} \end{pmatrix} \rightarrow \boxed{\text{ShiftRow}} \rightarrow \begin{pmatrix} b_{00} & b_{01} & b_{02} & b_{03} \\ b_{11} & b_{12} & b_{13} & b_{10} \\ b_{22} & b_{23} & b_{20} & b_{21} \\ b_{33} & b_{30} & b_{31} & b_{32} \end{pmatrix} \equiv [c_{ij}]$$

AES

Transformacja MixColumn

Macierz $[c_{ij}]$ mnożymy przez stałą macierz 4×4 korzystając z reguł działań w ciele $GF(2^8)$.[2mm]

$$\begin{pmatrix} 00000010 & 00000011 & 00000001 & 00000001 \\ 00000001 & 00000010 & 00000011 & 00000001 \\ 00000001 & 00000001 & 00000010 & 00000011 \\ 00000011 & 00000001 & 00000001 & 00000010 \end{pmatrix} \begin{pmatrix} c_{00} & c_{01} & c_{02} & c_{03} \\ c_{10} & c_{11} & c_{12} & c_{13} \\ c_{20} & c_{21} & c_{22} & c_{23} \\ c_{30} & c_{31} & c_{32} & c_{33} \end{pmatrix}$$
$$= \begin{pmatrix} d_{00} & d_{01} & d_{02} & d_{03} \\ d_{10} & d_{11} & d_{12} & d_{13} \\ d_{20} & d_{21} & d_{22} & d_{23} \\ d_{30} & d_{31} & d_{32} & d_{33} \end{pmatrix}$$

AES

Transformacja AddRoundKey

- Klucz rundy, składający się ze 128 bitów, dzielimy na 16 bajtów.
- Bajty te przedstawiamy w postaci macierzy 4×4, analogicznie jak blok tekstu jawnego.
- Macierz klucza rundowego oznaczamy $[k_{ij}]$.
- Macierz klucza rundowego dodajemy do macierzy otrzymanej w rezultacie transformacji MixColumn (dodajemy element po elemencie, zgodnie z regułami z ciała $GF(2^8)$).

$$\begin{pmatrix} d_{00} & d_{01} & d_{02} & d_{03} \\ d_{10} & d_{11} & d_{12} & d_{13} \\ d_{20} & d_{21} & d_{22} & d_{23} \\ d_{30} & d_{31} & d_{32} & d_{33} \end{pmatrix} \oplus \begin{pmatrix} k_{00} & k_{01} & k_{02} & k_{03} \\ k_{10} & k_{11} & k_{12} & k_{13} \\ k_{20} & k_{21} & k_{22} & k_{23} \\ k_{30} & k_{31} & k_{32} & k_{33} \end{pmatrix} = \begin{pmatrix} e_{00} & e_{01} & e_{02} & e_{03} \\ e_{10} & e_{11} & e_{12} & e_{13} \\ e_{20} & e_{21} & e_{22} & e_{23} \\ e_{30} & e_{31} & e_{32} & e_{33} \end{pmatrix}$$

AES

Schemat generowania kluczy rund

Początkowy klucz składający się ze 128 bitów dzielimy na 16 bajtów, które zapisujemy w postaci macierzy 4×4. [2mm]Kolumny tej macierzy oznaczmy $W(0)$, $W(1)$, $W(2)$ i $W(3)$.[2mm]Do macierzy tej dodajemy następne 40 kolumn $W(i)$, $i \geq 4$, które tworzymy według następującego schematu:gdy i nie dzieli się przez 4 to

$$W(i) = W(i-4) \oplus W(i-1)$$

gdy i dzieli się przez 4 to

$$W(i) = W(i-4) \oplus T\big(W(i-1)\big).$$

Transformacja T jest zdefiniowana na kolumnie $W(i-1)$ równej

$$\begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix}$$

w sposób następujący:

AES
Schemat generowania kluczy rund

1. cyklicznie przestawiamy bajty
$$\begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} \to \begin{pmatrix} b \\ c \\ d \\ a \end{pmatrix}$$
2. każdy bajt przekształcamy za pomocą S-boxu
$$\begin{pmatrix} b \\ c \\ d \\ a \end{pmatrix} \to \boxed{\text{S-box}} \to \begin{pmatrix} e \\ f \\ g \\ h \end{pmatrix}$$
3. wyliczamy stałą rundy
$$r(i) = (00000010)^{\frac{i-4}{4}},$$
gdzie mnożenie odbywa się zgodnie z regułami z ciała $GF(2^8)$

AES
Schemat generowania kluczy rund

- ostatecznie przyjmujemy
$$T\colon \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} \mapsto \begin{pmatrix} e \oplus r(i) \\ f \\ g \\ h \end{pmatrix}$$

Klucz i-tej rundy ma wtedy postać macierzy 4×4

$$\Big(W(4i), W(4i+1), W(4i+2), W(4i+3)\Big)$$

AES
Konstrukcja S-boxu

S-box wykorzystywany w algorytmie AES został skonstruowany z wykorzystaniem reguł działań w ciele Galois $GF(2^8)$. Element leżący na przecięciu wiersza w i kolumny k S-boxu tworzymy następująco:[2mm]

- Liczby w i k zapisujemy w zapisie binarnym. Każda z nich jest z zakresu 0 – 15 czyli jest reprezentowana przez liczbę 4-bitową.
$$w = x_7x_6x_5x_4 \qquad k = x_3x_2x_1x_0.$$
- Zgodnie z regułami z $GF(2^8)$ znajdujemy bajt odwrotny względem mnożenia do bajtu $x_7x_6x_5x_4x_3x_2x_1x_0$
$$x_7x_6x_5x_4x_3x_2x_1x_0 \mapsto (x_7x_6x_5x_4x_3x_2x_1x_0)^{-1} = y_7y_6y_5y_4y_3y_2y_1y_0.$$
Jeżeli $x_7x_6x_5x_4x_3x_2x_1x_0 = 00000000$ to nie istnieje odwrotność tego bajtu. Przyjmujemy wtedy $y_7y_6y_5y_4y_3y_2y_1y_0 = 00000000$.

AES
Konstrukcja S-boxu

- Wykonujemy działanie (zgodnie z regułami z $GF(2^8)$)

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} z_0 \\ z_1 \\ z_2 \\ z_3 \\ z_4 \\ z_5 \\ z_6 \\ z_7 \end{pmatrix}$$

Bajt $z_7z_6z_5z_4z_3z_2z_1z_0$ jest elementem S-boxu leżącym na przecięciu wiersza w i kolumny k.[2mm]Wykorzystanie w konstrukcji S-boxu odwzorowania $x \mapsto x^{-1}$ gwarantuje nieliniowość.[2mm]Mnożenie przez macierz i dodawanie wektora wprowadzono w celu większej komplikacji konstrukcji S-boxu.

AES
Deszyfrowanie

Każda z transformacji: ByteSub, ShiftRow, MixColumn, AddRoundKey wykorzystywanych w algorytmie AES jest odwracalna.[2mm]

- Z konstrukcji S-boxu wynika, że ByteSub jest transformacją odwracalną. Transformację odwrotną do ByteSub oznaczmy IByteSub.
- Transformacja odwrotna do ShiftRow polega na przesunięciu bajtów w wierszach w prawo. Oznaczmy tę transformację IShiftRow.
- Transformacja odwrotna do MixColumn to po prostu mnożenie przez macierz odwrotną do używanej w MixColumn. Używana tam macierz została dobrana tak, żeby była odwracalna (w ciele $GF(2^8)$).

AES
Deszyfrowanie

Macierz realizująca transformację IMixColumn ma postać

$$\begin{pmatrix} 00001110 & 00001011 & 00001101 & 00001001 \\ 00001001 & 00001110 & 00001011 & 00001101 \\ 00001101 & 00001001 & 00001110 & 00001011 \\ 00001011 & 00001101 & 00001001 & 00001110 \end{pmatrix}$$

- Transformacja AddRoudKey jest dodawaniem modulo 2 bit po bicie zatem sama jest swoją odwrotnością.

Zatem deszyfrowanie polega na zastosowaniu transformacji AddRoundKey, IMixColumn, IShiftRow, IByteSub w odpowiedniej kolejności i odpowiednią ilość razy z właściwymi kluczami.

AES
Deszyfrowanie

Szyfrowanie:[2mm]ARK BS, SR, MC, ARK . . . BS, SR, MC, ARK BS, SR, ARK Deszyfrowanie:[2mm]ARK, ISR, IBS ARK, IMC, ISR, IBS . . . ARK, IMC, ISR, IBS ARK

AES
Bezpieczeństwo

- Nie jest znany publicznie żaden atak, który byłby szybszy niż wyczerpujące przeszukiwanie przestrzeni kluczy dla algorytmu AES złożonego z 7 lub więcej rund.
- Wyczerpujące przeszukanie przestrzeni kluczy przy długości klucza 128 bitów jest zadaniem beznadziejnym:

$$2^{56} \approx 7{,}2 \times 10^{16}$$

$$2^{128} \approx 3{,}4 \times 10^{38}$$

Kryptografia z kluczem publicznym

W omawianych dotąd metodach (DES, AES, RC4) do szyfrowania i deszyfrowania używany jest ten sam klucz. Metody takie nazywamy metodami symetrycznymi.

Poważnym problemem metod symetrycznych jest konieczność wcześniejszego uzgodnienia klucza służącego do szyfrowania i deszyfrowania.

Problem uzgodnienia kluczy dotyczy szczególnie sytuacji, w których zaszyfrowane wiadomość chcą wymienić osoby nieznajome (na przykład dwa komputery w Internecie).

Problem ten rozwiązuje kryptografia z kluczem publicznym (kryptografia asymetryczna).

Kryptografia z kluczem publicznym

Wykorzystujemy dwa klucze:

Klucz publiczny K_{pub}

Klucz prywatny K_{pr}

Klucz publiczny K_{pub} ujawniamy publicznie (np. na stronie internetowej, w prasie). Klucz prywatny K_{pr} jest tajny.[2mm]

Klucze K_{pub} i K_{pr} muszą spełniać warunki:

- Wiadomość można zaszyfrować kluczem publicznym, ale odszyfrować można ją tylko kluczem prywatnym.
- Klucza prywatnego nie da się wyznaczyć przy pomocy klucza publicznego

Gdy chcemy osobie NN przesłać zaszyfrowaną wiadomość:

- Znajdujemy klucz publiczny NN.
- Szyfrujemy wiadomość kluczem publicznym NN i wysyłamy szyfrogram.
- NN deszyfruje wiadomość swoim kluczem prywatnym.

Kryptografia z kluczem publicznym

Idea kryptografi z kluczem publicznym została zaproponowana przez Diffiego i Hellmana w roku 1976. Autorzy nie znali jednak praktycznej metody realizacji swojej idei.
W. Diffie, M. Hellman, „New directions in cryptography”, *IEEE Trans. in Information Theory*, **22** (1976), 644–654.

W roku 1997 brytyjska agencja wywiadowcza GCHQ (Government Communications Headquaters) ujawniła, że idea kryptografii asymetrycznej została wynaleziona w roku 1970 przez pracownika GCHQ Jamesa Ellisa.

Algorytm RSA

Metoda praktycznej realizacji kryptografii z kluczem publicznym została zaproponowana w roku 1978:
Ronald L. Rivest, Adi Shamir, Leonard M. Adleman, „A Method for Obtaining Digital Signatures and Public-Key Cryptosystems", *Commun. ACM*, **21** (1978), 120.

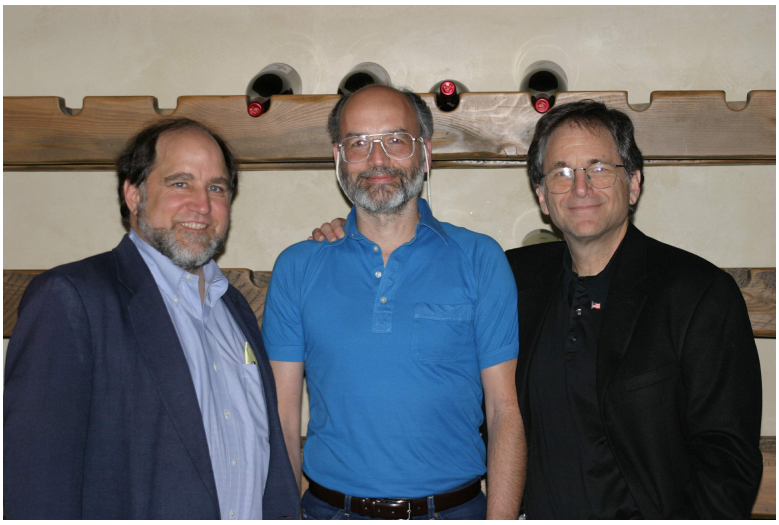
Metoda ta nosi obecnie nazwę algorytmu RSA.

Według danych odtajnionych w roku 1997 przez brytyjski wywiad, algorytm RSA został odkryty w roku 1973 przez C. Cocksa, pracownika CESG (The Communications-Electronic Security Group, instytucja wchodząca w sklad GCHQ)
Clifford C. Cocks „A Note on "non-secret" encryption", 20 Nov 1973, http://www.cesg.gov.uk/site/publications/media/notense.pdf

Algorytm RSA

Lata 70-te XX wieku, od lewej: A. Shamir, R. Rivest, L. Adleman

Algorytm RSA



Rok 2003, od lewej: R. Rivest, A. Shamir, L. Adleman

Algorytm RSA
Elementy teori liczb

Liczbę naturalną p nazywamy pierwszą, gdy p dzieli się tylko przez 1 i samą siebie.[2mm]Liczby pierwsze: 2, 3, 5, 7, 11, 13, 17, 19, 23, . . . [2mm]

Twierdzenie (Euklides)
Istnieje nieskończenie wiele liczb pierwszych.

Dowód: . . . [3mm]Liczby naturalne m i n nazywamy względnie pierwszymi, gdy jedynym ich wspólnym dzielnikiem jest 1.[2mm]Pary liczb względnie pierwszych: 4 i 9, 6 i 25, 8 i 15, . . . Pary liczb, które nie są względnie pierwsze: 6 i 8, 9 i 12, . . . [2mm]

Twierdzenie
Każda liczba naturalna posiada jednoznaczny rozkład na czynniki pierwsze.

Algorytm RSA
Elementy teorii liczb

Małe twierdzenie Fermata
Jeżeli p jest liczbą pierwszą i p nie jest dzielnikiem a, to

$$a^{p-1} = 1 \mod p.$$

Dowód:Niech

$$S = \{1, 2, \ldots, p-1\}.$$

Definiujemy odwzorowanie:

$$\psi \colon S \to S, \qquad \psi(x) = ax \mod p.$$

Odwzorowanie ψ jest dobrze określone, tzn. $\psi(x) \neq 0$ dla $x \in S$.

$$\psi(x) = \psi(y) \Rightarrow x = y \mod p.$$

Stąd $\psi(1), \psi(2), \ldots, \psi(p-1)$ są różnymi elementami S.Czyli $S = \{\psi(1), \psi(2), \ldots, \psi(p-1)\}$.

Algorytm RSA
Elementy teorii liczb

Zatem

$$\begin{aligned} 1 \cdot 2 \cdot 3 \cdot \cdots \cdot (p-1) &= \psi(1) \cdot \psi(2) \cdot \cdots \cdot \psi(p-1) \\ &= a^{p-1}\big(1 \cdot 2 \cdot 3 \cdot \cdots \cdot (p-1)\big) \mod p \end{aligned}$$

Dla każdego $x \in S$ mamy NWD$(x, p) = 1$, czyli dzielimy ostatnie równanie przez $1, 2, \ldots, p-1$ i dostajemy:

$$1 = a^{p-1} \mod p.$$

Algorytm RSA
Elementy teorii liczb

Definicja
Funkcję Eulera ϕ definiujemy jako odwzorowanie $\phi \colon \mathbb{N} \to \mathbb{N}$. Wartość funkcji ϕ dla liczby n określamy jako ilość liczb ze zbioru $1, 2, \ldots, n$ względnie pierwszych z n.

$\phi(2) = 1$, $\phi(3) = 2$, $\phi(4) = (2)$, $\phi(5) = 4$, $\phi(6) = 2$, . . . [1mm]Spostrzeżenie: jeżeli p jest liczbą pierwszą, to $\phi(p) = p - 1$. [1mm]Stąd mamy: jeżeli p_1, p_2 są różnymi liczbami pierwszymi oraz $n = p_1 p_2$, to $\phi(n) = (p_1 - 1)(p_2 - 1) = p_1 p_2 - p_1 - p_2 + 1$. W zbiorze $\{1, \ldots, p_1 p_2\}$ mamy $p_1 p_2$ elementów.Z liczbą $p_1 p_2$ względnie pierwsze nie są tylko te liczby, które dzielą się przez p_1 lub przez p_2 (ponieważ p_1 i p_2 są pierwsze).Zatem od $p_1 p_2$ odejmujemy p_1 liczb podzielnych przez p_2 i p_2 liczb podzielnych przez p_1; liczbę $p_1 p_2$ odjęliśmy dwukrotnie, czyli dodajemy 1. [1mm]Ogólniej: jeżeli $p_1, p_2, \ldots, p_i$ są różnymi liczbami pierwszymi oraz $n = p_1 p_2 \ldots p_i$, to $\phi(n) = (p_1 - 1)(p_2 - 1) \ldots (p_i - 1)$.

Algorytm RSA
Elementy teorii liczb

Twierdzenie Eulera
Jeżeli liczby a i n są względnie pierwsze, to

$$a^{\phi(n)} = 1 \mod n.$$

Dowód: analogicznie do dowodu Małego Twierdzenia Fermata.[2mm]

Twierdzenie
Niech a, n, x, y będą liczbami naturalnymi, $n \geq 1$, NWD$(a, n) = 1$. Wtedy, jeżeli

$$x = y \mod \phi(n)$$

to

$$a^x = a^y \mod n.$$

Dowód: Jeżeli $x = y \mod \phi(n)$ to znaczy, że $x = y + \phi(n)k$, $k \in \mathbb{N}$. [2mm]Zatem

$$a^x = a^{y+\phi(n)k} = a^y(a^{\phi(n)})^k = a^y 1^k = a^y \mod n.$$

Algorytm RSA
Bob wybiera dwie liczby pierwsze p i q i wyznacza

$$n = pq.$$

Bob wybiera następnie liczbę e taką, która jest względnie pierwsza z liczbą $(p-1)(q-1)$.[2mm]Bob wyznacza następnie liczbę d taką, że

$$de = 1 \mod (p-1)(q-1).$$

Kluczem publicznym jest para liczb (n, e). Kluczem prywatnym jest para liczb p i q (oraz liczba d, którą wyznaczamy znając p i q).[2mm]Tekst jawny: liczba $m < n$. Szyfrowanie:

$$k = m^e \mod n.$$

Deszyfrowanie:

$$m = k^d \mod n$$

Algorytm RSA
Uwagi:

- $n = pq$ i p oraz q są liczbami pierwszymi, zatem

  $$\phi(n) = \phi(pq) = (p-1)(q-1).$$

- Na liczbę e nakładamy warunek NWD$(e, (p-1)(q-1)) = 1$ ponieważ tylko wtedy równanie

  $$de = 1 \mod (p-1)(q-1)$$

  posiada rozwiązanie ze względu na d.

- Poprawność deszyfrowania:Zachodzi:

  $$de = s(p-1)(q-1) + 1.$$

  Zatem z twierdzenia Eulera:

  $$m^{s(p-1)(q-1)} = \left[m^{(p-1)(q-1)}\right]^s = 1^s = 1 \mod pq.$$

  Czyli:

  $$m^{de} = m^{s(p-1)(q-1)+1} = m^{s(p-1)(q-1)} \cdot m = m \mod n.$$

Algorytm RSA

Bezpieczeństwo RSA:[1mm]Klucz prywatny d można wyznaczyć z klucza publicznego znajdując p i q czyli rozkładając liczbę n na czynniki pierwsze.[1mm]Gdy liczba n jest odpowiednio duża, to jej faktoryzacja jest zagadnieniem o dużej złożoności obliczeniowej.[1mm]W najlepszym znanym algorytmie faktoryzacji czas faktoryzacji zależy wykładniczo od długości liczby n (liczby cyfr w zapisie n).[1mm]W prakyce wykorzystuje się bardzo duże liczby n (co najmniej 1024 bity (309 cyfr w zapisie dziesiętnym)) co czyni faktoryzację zadaniem bardzo trudnym.

Algorytm RSA

Znając kryptogram k możemy wyznaczyć tekst jawny m nie faktoryzując klucza publicznego n ale rozwiązując równanie

$$k^x = m \mod n. \tag{1}$$

Jest to problem wyznaczenia tzw. logarytmu dyskretnego z liczby m przy podstawie k (nazwa bierze się stąd, że gdyby w (1) nie było "mod n" to rozwiązaniem byłaby liczba $x = \log_k m$).[1mm]Udowodniono jednak, że problem znalezienia logarytmu dyskretnego ma tę samą złożoność obliczeniową co problem faktoryzacji.

Algorytm RSA

Generowanie bezpiecznych kluczy RSA wymaga stosowania bardzo dużych liczb pierwszych.[2mm]Powstaje zatem pytanie, czy odpowiednio dużych liczb pierwszych jest wystarczająco wiele.[2mm]

Twierdzenie

Oznaczmy przez $\pi(x)$ liczbę liczb pierwszych mniejszych od x. Wtedy dla dużych x

$$\pi(x) \approx \frac{x}{\ln x},$$

(ściśle: $\lim_{x\to\infty} \pi(x)/(x/\ln x) = 1$).

Na przykład ilość liczb pierwszych o długości 100 cyfr wynosi około:

$$\pi(10^{100}) - \pi(10^{99}) \approx \frac{10^{100}}{\ln 10^{100}} - \frac{10^{99}}{\ln 10^{99}} \approx 3{,}9 \times 10^{97}$$

Algorytm RSA

Generowanie kluczy RSA wymaga też praktycznych sposobów znajdowania dużych liczb pierwszych (liczby p i q których iloczynem jest klucz publiczny). [2mm]Sposoby te powinny być bardziej efektywne niż rozkład na czynniki pierwsze, który jest zadaniem obliczeniowo złożonym.[2mm]Na przykład, żeby sprawdzić, czy 200 cyfrowa liczba x jest pierwsza należałoby sukcesywnie dzielić ją przez wszystkie liczby pierwsze mniejsze od $\sqrt{x}$ czyli wszystkie liczby pierwsze mniejsze od około 10^{100}. Ale takich liczb pierwszych jest

$$\pi(10^{100}) \approx \frac{10^{100}}{\ln 10^{100}} \approx 4 \times 10^{97}$$

Istnieją efektywne metody badania, czy dana liczba jest liczbą pierwszą czy złożoną bez znajdowania rozkladu na czynniki pierwsze. Są to tak zwane testy pierwszości.

Algorytm RSA

Testy pierwszości

Test pierwszości Fermata

Niech $n > 1$ będzie liczbą całkowitą. Wybierzmy losowo liczbę całkowitą a spełniającą warunek $1 < a < n - 1$. Wtedy, jeżeli

$$a^{n-1} \neq 1 \mod n$$

to n jest liczbą złożoną. Jeżeli natomiast $a^{n-1} = 1 \mod n$ to n jest *prawdopodobnie* liczbą pierwszą.

Dowód wynika z Małego Twierdzenia Fermata.[2mm]Uwagi:

- test Fermata jako test pierwszości jest testem probabilistycznym, z pewnością stwierdzamy tylko złożoność liczby n,
- większość stosowanych w praktyce testów to testy probabilistyczne.

Algorytm RSA
Testy pierwszości

Test pierwszości Millera–Rabina
Niech $n > 1$ będzie nieparzystą liczbą całkowitą. Zapiszmy

$$n - 1 = 2^k m,$$

gdzie m jest liczbą nieparzystą.Wybierzmy losowo liczbę całkowitą a, $1 < a < n - 1$.Policzmy $b_0 = a^m \mod n$.Jeżeli $b_0 = \pm 1 \mod n$ to kończymy obliczenia i stwierdzamy, że n jest prawdopodobnie pierwsze.W przeciwnym wypadku liczymy $b_1 = b_0^2 \mod n$.Jeżeli $b_1 = 1 \mod n$ wtedy n jest liczbą złożoną.Jeżeli $b_1 = -1 \mod n$ to kończymy obliczenia i stwierdzamy, że n jest prawdopodobnie liczbą pierwszą.W innym wypadku liczymy $b_2 = b_1^2 \mod n$.Jeżeli $b_2 = 1 \mod n$ wtedy n jest liczbą złożoną.Jeżeli $b_2 = -1 \mod n$ to kończymy obliczenia i stwierdzamy, że n jest prawdopodobnie liczbą pierwszą.Kontynuujemy analogiczne rachunki aż do osiągnięcia b_{k-1}. Jeżeli $b_{k-1} \neq -1 \mod n$ to n jest liczbą złożoną.

Algorytm RSA
Testy pierwszości

Można udowodnić, że prawdopodobieństwo, że test Millera–Rabina nie wyłapie liczby złożonej n przy losowym wyborze a jest równe co najwyżej $1/4$. W większości wypadków prawdopodobieństwo to jest dużo mniejsze.[2mm]Jeżeli zastosujemy test 10 razy do jednej liczby n z losowo wybranymi wartościami a to prawdopodobieństwo, że liczbę złożoną uznamy za pierwszą wynosi około:

$$(0{,}25)^{10} \approx 10^{-6}.$$

Isnieją również testy, które pozwalają dowieść pierwszości liczby całkowitej ale działają wolniej niż testy probabilistyczne.

Algorytm RSA
Łamanie RSA

W roku 1977 autorzy algorytmu RSA ogłosili następującą zagadkę (tzw. RSA Challenge):

Klucz publiczny ma postać:

$n =$
114381625757888867669235779976146612010218296721242362
562561842935706935245733897830597123563958705058989075
147599290026879543541

e=9007

Kryptogram ma postać:

$c =$
968696137546220614771409222543558829057599911245743198
746951209308162982251457083569314766228839896280133919
90551829945157815154.

Znaleźć tekst jawny.

Powyższa liczba n w zapisie binarnym na długość 428 bitów.

Algorytm RSA
Łamanie RSA

Przy pomocy metod faktoryzacji i sprzętu dostępnych w roku 1977 przewidywany czas faktoryzacji liczby n wynosił około 4×10^{16} lat.[2mm]Problem został rozwiązany w roku 1994 przez zespół matematyków pod kierunkiem Lenstry.[1mm]Wykorzystane zasoby:

- 600 użytkowników,
- 1 600 komputerów,
- oblicznia trwały niecałe 8 miesięcy.

Algorytm RSA
Łamanie RSA

W 1999 dokonano rozkładu liczby o długości 512 bitów (155 cyfr w zapisie dziesiętnym).[1mm]Wykorzystane zasoby:

- 300 komputerów,
- obliczenia trwały 7,4 miesiąca.

W roku 2005 rozwiązany został RSA Challenge 640. Dokonano faktoryzacji liczby o długości 640 bitów (193 cyfry w zapisie dziesiętnym).[1mm]Wykorzystane zasoby:

- moc obliczeniowa równoważna około 30 lat pracy pojedynczego procesora klasy Opteron-2,2GHz,
- obliczenia wykonano w czasie 5 miesięcy.

Algorytm RSA
Łamanie RSA

W grudniu 2009 rozwiązany został RSA Challenge 768. Dokonano faktoryzacji liczby o długości 768 bitów (232 cyfry w zapisie dziesiętnym).[2mm]Wykorzystane zasoby:

- faktoryzacji dokonał zespół pod kierunkiem Thorstena Kleinjunga składający się z 13 matematyków,
- moc obliczeniowa równoważna około 1 500 lat pracy pojedynczego procesora klasy Opteron-2,2GHz z 2 GB RAM,
- obliczenia trwały prawie 2 lata, wykorzystano kilkaset komputerów.

W grudniu 2019 dokonano faktoryzacji liczby o długości 795 bitów (240 cyfr w zapisie dziesiętnym).

- faktoryzacji dokonał szescioosobowy zespół (5 osób z Francji, 1 z USA),
- moc obliczeniowa równowazna 4000 lat ciągłej pracy procesora Intel Xeon Gold 6130 (2.1GHz).

Algorytm RSA
Łamanie RSA

W marcu 2020 dokonano faktoryzacji liczby o długości 829 bitów (250 cyfr w zapisie dziesiętnym).

- faktoryzacji dokonał zespół francusko-amerykański,
- obliczenia trwały kilka miesięcy, wykorzystano dziesiątki tysięcy komputerów.

Następujące RSA Challenges nie zostały dotąd rozwiązane: RSA-896 (270 cyfr w zapisie dziesiętnym), RSA-1024 (309 cyfr w zapisie dziesiętnym), RSA-1536 (463 cyfr w zapisie dziesiętnym), RSA-2048 (617 cyfr w zapisie dziesiętnym).

Algorytm RSA

Wykorzystanie RSA:

- algorytm RSA jest wbudowany w większość protokołów bezpiecznej komunikacji internetowej (SSL, S/MIME, S/WAN)),
- w bezpiecznej łączności telefonicznej,
- wbudowany w karty sieciowe,
- wykorzystywany w celach kryptograficznych przez wiele korporacji i instytucji rządowych.