5. Elementy kryptologii

Grzegorz Kosiorowski

Uniwersytet Ekonomiczny w Krakowie

Kryptologia

Kryptologia to dziedzina matematyki i informatyki zajmująca się zagadnieniami bezpieczeństwa informacji: w szczególności kodowaniem i dekodowaniem wiadomości. Dzieli się na dwie gałęzie: kryptografię (naukę o szyfrowaniu informacji, na niej się głównie skupimy) i kryptoanalizę (naukę o deszyfrowaniu wiadomości). Kryptologia zupełnie zmieniła swoje oblicze w XX wieku dzięki zastosowaniu algorytmów opartych na teorii liczb.

Kryptografia - wstęp

Rozważamy zagadnienie kryptograficzne polegające na przesłaniu wiadomości od nadawcy do odbiorcy w taki sposób, by żadna trzecia (przypadkowa) osoba nie była w stanie jej odczytać.

Tekst jawny

Tekstem jawnym będziemy nazywać ciąg symboli (znaków) w pewnym języku (np. w polskim), który jest dany w procesie kodowania lub jest rezultatem w procesie dekodowania.

Szyfrogram

Wiadomość zakodowana, czyli *szyfrogram*, to także ciąg symboli, którego elementami są znaki z tego samego alfabetu co elementy tekstu jawnego (przynajmniej w ramach naszego kursu). Jest rezultatem kodowania tekstu jawnego lub ciągiem danym w procesie dekodowania.

Technika szyfrowania

Oczywiście, można dodatkową funkcją zmienić treść szyfrogramu na symbole zupełnie inne niż wyjściowy alfabet, ale matematycznie nie wprowadza to żadnej dodatkowej komplikacji.
Przed zakodowaniem nadawca dzieli tekst jawny na tzw. jednostki. Jednostką tekstu jawnego może być pojedynczy symbol (a, b, c ...), lecz częściej dwójka (aa, ab, ac ...) lub więcej symboli. Jednostki tekstu jawnego poddaje się procesowi szyfrowania otrzymując jednostki szyfrogramu, niezależnie wysyłane od nadawcy do odbiorcy i tam deszyfrowane z powrotem na tekst jawny.

Funkcja kodująca

Funkcja kodująca

Niech J będzie zbiorem wszystkich możliwych jednostek tekstu jawnego i szyfrogramu (czyli *jednostek szyfrowania*). Wtedy *funkcją kodującą* nazywamy $f : J \to J$ - permutację tego zbioru (czyli przestawienie kolejności elementów, albo po prostu bijekcję zbioru w siebie).

Tej definicji będziemy używać w kursie: zdarzają się systemy kodowania, których funkcja kodująca nie jest permutacją, ale powoduje to dodatkowe problemy przy dekodowaniu informacji, gdyż wynik dekodowania nie jest jednoznaczny.

Podstawowe definicje

Funkcja kodująca

Niech J będzie zbiorem wszystkich możliwych jednostek tekstu jawnego i szyfrogramu (czyli *jednostek szyfrowania*). Wtedy *funkcją kodującą* nazywamy $f : J \longrightarrow J$ - permutację (czyli przestawienie kolejności elementów) tego zbioru.

Funkcja dekodująca

Dla funkcji kodującej f, funkcję do niej odwrotną $f^{-1} : J \longrightarrow J$ nazywamy *funkcją dekodującą*. Jest ona również permutacją.

Kryptosystem

Kryptosystemem nazywamy parę (J, f).

Analiza częstości

Szyfrowanie pojedynczych symboli za pomocą funkcji kodującej generalnie jest uważane za niebezpieczne, ze względu na łatwość ataku za pomocą tzw. analizy częstości. Mając dany długi szyfrogram, o którym wiemy, że odpowiadający mu tekst jawny jest zapisany np. w języku polskim, możemy prześledzić częstość występowania znaków. Np. jeśli litera h występuje wiele razy w szyfrogramie, to możemy przypuszczać, że powinniśmy ją zdekodować jako jedną z liter występujących w słowach dość często (a, i, e, o, n, z) raczej niż jedną z „rzadkich" (ź, ć, f, nie mówiąc o q, x, v). Ponadto, jeśli jednostką tekstu jest pojedyncza litera, to istnieje relatywnie mało funkcji kodujących, co naraża nasz szyfr na „brutalne" ataki polegające na sprawdzaniu wszystkich możliwości.

Ustalenia techniczne

Pamiętając o tym zastrzeżeniu, podczas tego kursu nie będziemy się zajmować konstruowaniem kodów jak najtrudniejszych do złamania, a jedynie przykładów umożliwiających zrozumienie algorytmów kodujących. Zatem dla prostoty, wbrew regułom bezpieczeństwa, będziemy używać jednostek szyfrowania będących pojedynczymi symbolami tekstu jawnego.
Zanim zabierzemy się do kodowania, wykonamy jeszcze jedno przejście - litery języka (na których trudno zapisywać wyrażenia funkcyjne) będziemy zapisywać jako liczby. Sposób zamiany używany w przykładach pojawi się w tabelce na następnym slajdzie. Ponieważ w przykładach będziemy szyfrować zdania z łaciny, tabelka nie zawiera polskich liter.

Tabelka do przykładów szyfrowania

$*$	$+$	$-$	A	B	C	D	E	F	G	H
0	1	2	3	4	5	6	7	8	9	10
I	J	K	L	M	N	O	P	Q	R	S
11	12	13	14	15	16	17	18	19	20	21
T	U	W	V	X	Y	Z		.	?	!
22	23	24	25	26	27	28	29	30	31	32

Tabelka zastępuje 33 symbole liczbami 0 do 32 (resztami z dzielenia przez 33). Ponieważ alfabet łaciński ma mniej niż 33 litery (wliczając spację), uzupełniliśmy tabelkę różnymi nieistotnymi znakami. Później istotne się okaże, że pierwsze kilka symboli to nie są litery, tylko „zapychacze”.

Szyfr Cezara

Przykładem bardzo prostego klasycznego systemu kryptograficznego jest szyft, którego, zdaniem wielu historyków, Juliusz Cezar używał do porozumiewania się ze swoimi podwładnymi.
Funkcja kodująca jest tu oparta na dodawaniu modulo rozmiar zbioru jednostek szyfrowania, który oznaczamy $|J|$:

$$f(P) = (P + b) \mod |J|,$$

dla pewnego b - parametru ze zbioru liczb naturalnych mniejszych od $|J|$ (Cezar miał używać tego szyfru z $b = 3$).

Klucze kodowania i dekodowania szyfru Cezara

b jest jedyną liczbą, której znajomość jest potrzebna do kodowania szyfrem Cezara. Dlatego mówimy, że jest *kluczem kodowania*.

Klucze

Dla danego systemu szyfrowania, *klucz kodowania* - K_K to parametr(y) funkcji kodującej. Znając je można wysyłać zakodowane wiadomości. *Klucz dekodowania* - K_D to parametr(y) funkcji dekodującej. Znając je można dekodować szyfrogramy.

b jest w wypadku szyfru Cezara też kluczem dekodowania:

$$f^{-1}(C) = (C - b) \mod |J|.$$

Zatem znając b, odbiorca z łatwością dekoduje szyfrogram Cezara.

Szyfr Cezara - przykład

Załóżmy, że Cezar chce zakodować pierwsze słowa swego dzieła „O wojnie galijskiej”: *Gallia est omnis divisa in partes tres* za pomocą klucza $b = 23$ i wcześniejszej tabelki ($|J| = 33$)

$*$	$+$	$-$	A	B	C	D	E	F	G	H
0	1	2	3	4	5	6	7	8	9	10
I	J	K	L	M	N	O	P	Q	R	S
11	12	13	14	15	16	17	18	19	20	21
T	U	W	V	X	Y	Z		.	?	!
22	23	24	25	26	27	28	29	30	31	32

Kodując pierwszą literę „G”, sprawdzamy jej numer w tabeli (9), następnie dodajemy 23 i wychodzi nam 32, czyli „!”.
Kodując drugą literę „a”, sprawdzamy jej numer w tabeli (3), następnie dodajemy 23 i wychodzi nam 26, czyli „x”.
Ponieważ wychodziły nam liczby mniejsze od 33, mogliśmy się nie przejmować sprawdzaniem reszty z dzielenia przez $|J|$.

Szyfr Cezara - przykład

Gallia est omnis divisa in partes tres, $b = 23$, $|J| = 33$.

$*$	$+$	$-$	A	B	C	D	E	F	G	H
0	1	2	3	4	5	6	7	8	9	10
I	J	K	L	M	N	O	P	Q	R	S
11	12	13	14	15	16	17	18	19	20	21
T	U	W	V	X	Y	Z		.	?	!
22	23	24	25	26	27	28	29	30	31	32

Kodując trzecią literę „l”, sprawdzamy jej numer w tabeli (14), następnie dodajemy 23 i wychodzi nam 37, za dużo do tabeli. Jednakże, zgodnie z zasadą szyfru Cezara obliczymy nie tyle samą sumę, co jej resztę z dzielenia przez $|J| = 33$, więc ostatecznie wychodzi $37 \mod 33 = 4$, czyli „b”.

Szyfr Cezara - przykład

Gallia est omnis divisa in partes tres, $b = 23$, $|J| = 33$.

$*$	$+$	$-$	A	B	C	D	E	F	G	H
0	1	2	3	4	5	6	7	8	9	10
I	J	K	L	M	N	O	P	Q	R	S
11	12	13	14	15	16	17	18	19	20	21
T	U	W	V	X	Y	Z		.	?	!
22	23	24	25	26	27	28	29	30	31	32

Analogicznie kodujemy kolejne litery otrzymując (jako ćwiczenie - sprawdzić):

!xbb+xq.ijqecd+iq +m+ixq+dqfxhj.iqjh.i

Szyfr Cezara - przykład

!xbb+xq.ijqecd+iq +m+ixq+dqfxhj.iqjh.i, $b = 23$, $|J| = 33$.

$*$	$+$	$-$	A	B	C	D	E	F	G	H
0	1	2	3	4	5	6	7	8	9	10
I	J	K	L	M	N	O	P	Q	R	S
11	12	13	14	15	16	17	18	19	20	21
T	U	W	V	X	Y	Z		.	?	!
22	23	24	25	26	27	28	29	30	31	32

Odczytując szyfr, postępujemy zgodnie z regułami dekodowania: Na przykład, zdekodujmy siódmy symbol szyfrogramu, czyli „q”. Sprawdzamy jego numer w tabeli (19), następnie **odejmujemy** 23 i wychodzi nam -4, za mało do tabeli. Jednakże, zgodnie z algorytmem liczymy nie tyle różnicę, co jej resztę z dzielenia przez $|J| = 33$, więc ostatecznie wychodzi $(-4) \mod 33 = 29$, czyli spacja: „ ”. Analogicznie możemy zdekodować resztę szyfrogramu.

Wady szyfru Cezara

Oczywiście, szyfr Cezara nie nadaje się współcześnie do zakodowania czegokolwiek istotnego:

- Szyfrogram łatwo rozkodować - dla przykładowej tabeli wystarczy „brutalną siłą” zbadać 33 możliwości (a nawet mniej, bo przesunięcie o 0 jest nonsensowne), a analiza częstości jeszcze sprawę upraszcza. Dlatego nawet kodowanie par lub trójek symboli metodą Cezara nie jest odporne na szybkie złamanie.
- Jeśli Cezar używał tego samego kryptosystemu do kontaktu ze wszystkimi swoimi generałami, wtedy mogło dochodzić do sytuacji niepożądanych np. gdy generał A wysłał Cezarowi zaszyfrowaną wiadomość, generał B przechwytując kuriera po drodze mógł ją odczytać. Przecież każdy z generałów musiał znać klucz kodujący, by móc wysyłać listy do Cezara, a klucz kodujący był równoważny z dekodującym!

Zasada Kerckhoffsa

Dlatego szyfr Cezara nie spełnia tzw. zasady Kerckhoffsa.

Zasada Kerckhoffsa

Kryptosystem powinien być odporny na złamanie, nawet gdy wszystkie zasady jego działania (poza kluczem dekodowania) są znane.

Popularne sformułowanie tej zasady (znane jako maksyma Shannona), którą powinny spełniać idealne kryptosystemy to: *Wróg zna system*.

Zasada Kerckhoffsa - uzasadnienie

- System kryptograficzny, który ma być używany w dużym zakresie (np. system bankowy) dla bezpieczeństwa wymagałby tajności czegoś tak dużego i skomplikowanego jak cały algorytm szyfrowania jest nierealny. „Element ludzki” zwykle jest słabym punktem takich zabezpieczeń: ogólne informacje o tak popularnym systemie można zdobyć za pomocą przekupstwa, kradzieży, szpiegostwa itp. Dużo łatwiej jest osiągnąć bezpieczeństwo systemu, jeśli wymagana jest tylko ochrona małych zbiorów danych, takich jak klucze.
- O ile, w razie zagrożenia bezpieczeństwa, klucze jest dość łatwo zmieniać, to zmiana całego kryptosystemu na potrzeby przekazywania każdej kolejnej wiadomości jest niepraktyczna. Sensownym jest używanie tego samego kryptosystemu przez długi czas i przez wielu użytkowników. A w tej sytuacji ryzyko przechwycenia ogólnej zasady działania algorytmu jest dość wysokie.

Zasada Kerckhoffsa - uzasadnienie

- Gdy wiadomości są przesyłane w zamkniętej grupie użytkowników, dla których prawdopodobieństwo ujawnienia systemu jest niewielkie, czasem używa się procedur szyfrowania, dla których częścią obrony przed złamaniem jest ich utajnienie (np. szyfry wojskowe). Niemniej takie postępowanie (zwane *security through obscurity*) jest generalnie uważane za złą praktykę, która osłabia bezpieczeństwo szyfru i utrudnia wykrycie problemów.
- Jawność systemu ma wiele dodatkowych zalet - na przykład ułatwia szacowanie bezpieczeństwa danego systemu komunikacji oraz poprawianie potencjalnych luk (afera *London Oyster Card*, system DVD-CSS).

Przypomnienie

Funkcja kodująca

Niech J będzie zbiorem wszystkich możliwych jednostek tekstu jawnego i szyfrogramu (czyli *jednostek szyfrowania*). Wtedy *funkcją kodującą* nazywamy $f : J \longrightarrow J$ - permutację tego zbioru.

Funkcja dekodująca

Dla funkcji kodującej f, funkcję do niej odwrotną $f^{-1} : J \longrightarrow J$ nazywamy *funkcją dekodującą*. Jest ona również permutacją.

Kryptosystem

Kryptosystemem nazywamy parę (J, f).

Odwracalność funkcji kodującej

Brak możliwości spełnienia zasady Kerckhoffsa do całkiem niedawna wydawała się „wbudowana" w każdy rozsądny kryptosystem. W końcu, znajomość funkcji f pozwala na zdefiniowanie funkcji odwrotnej (f^{-1}), zwłaszcza na zbiorze skończonym.
W 1976 Whitfield Diffie i Martin Hellman zaproponowali jednak dodatkowe, realistyczne założenie: jeśli założymy, że osoba próbująca odtworzyć f^{-1} nie ma do dyspozycji nieograniczonej mocy obliczeniowej, to wystarczy, by funkcji f nie dało się odwrócić *szybko* (tj. na tyle szybko, by złamanie szyfru miało poważne konsekwencje).

Odwracalność funkcji kodującej

Z teorii liczb wiemy, że istnieją w miarę proste procedury, których odwrócenie jest skomplikowane. W szczególności mnożenie liczb jest szybkie ($O(\log a \log b)$), zaś rozłożenie wyniku na liczby wyjściowe (czyli proces odwrotny do mnożenia) jest w niewykonalne w rozsądnym (tj. wielomianowym) czasie. Jak z tej obserwacji przejść do sensownego kryptosystemu i jak jego „sensowność" definiujemy?

Kryptosystem z kluczem publicznym

Kryptosystem z kluczem publicznym

Kryptosystem z kluczem publicznym to czwórka (J, f, K_K, K_D), w której:
a) K_K jest powszechnie dostępny; K_D jest zachowywany w tajemnicy,
b) znajomość K_K pozwala na szybkie kodowanie jednostki tekstu, czyli obliczanie $f(P)$ dla $P \in J$,
c) znajomość K_K nie pozwala (bez znajomości K_D) na szybkie dekodowanie jednostki szyfrogramu, czyli obliczanie $f^{-1}(C)$ dla $C \in J$,
d) znajomość K_D daje możliwość szybkiego odkodowania jednostki szyfrogramu.

Przykładem takiego kryptosystemu jest algorytm RSA (1977, od nazwisk pomysłodawców: Rivesta, Shamira i Adlemana, obecnie w domenie publicznej).

Algorytm RSA - część I

Algorytm RSA

- I. Wybieramy losowo dwie różne, bardzo duże liczby pierwsze p i q. Im większe, tym trudniej złamać kod, ale też dłużej trwa proces kodowania i dekodowania.
- II. Niech $n = pq$. Wtedy $\varphi(n) = \varphi(p) \cdot \varphi(q) = (p-1)(q-1)$. Znów losowo wybieramy liczbę e względnie pierwszą z $\varphi(n)$. Znajdujemy d takie, że $e \cdot d \equiv 1(\mod \varphi(n))$. (np. za pomocą rozszerzonego algorytmu Euklidesa).
- III. Klucz publiczny $K_K = (n, e)$ można opublikować. W sekrecie zachowujemy $K_D = (n, d)$.
- IV. Zbiór wszystkich jednostek wiadomości definiujemy jako $\mathbb{Z}_n = \{0, 1, \ldots, n-1\}$. Funkcją kodującą będzie $f(P) = P^e \mod n$.

Algorytm RSA - część II

Algorytm RSA

- IV. Zbiór wszystkich jednostek wiadomości definiujemy jako $\mathbb{Z}_n = \{0, 1, \ldots, n-1\}$. Funkcją kodującą będzie $f(P) = P^e \mod n$.
- V. Funkcją dekodującą będzie $f^{-1}(C) = C^d \mod n$.

Powstaje pytanie, czy ten algorytm jest poprawny, czyli czy funkcje f i f^{-1} faktycznie są odwrotne. Korzystając z dotychczasowej wiedzy z teorii liczb, potrafimy to udowodnić. Zanim to jednak zrobimy, spróbujmy ten algorytm zastosować.

Algorytm RSA - uwagi

- O ile rozkład dużej liczby na czynniki pierwsze jest niewykonalny w czasie wielomianowym, to istnieją testy sprawdzające, czy dana liczba n jest pierwsza w czasie $O(\log^3 n)$ - więc krok I nie jest trudny do zrealizowania.
- Wydaje się, że potęgowanie z resztą jest tym fragmentem algorytmu RSA, który może zająć bardzo dużo czasu. Jednakże, komputer potrafi je wykonać dość szybko, dzięki metodzie szybkiego potęgowania (tzw. potęgowania przez kwadraty, twierdzenie Eulera też bywa pomocne).
- Dla optymalnego bezpieczeństwa, liczby p i q nie powinny być zbyt bliskie, ale powinny być podobnego rozmiaru w zapisie bitowym.
- W niektórych nowszych implementacjach RSA zamiast tocjentu Eulera stosuje się tzw. tocjent Carmichaela, trochę łatwiejszy obliczeniowo, a matematycznie równoważny (lecz o bardziej skomplikowanej definicji).

Algorytm RSA - przykład

Spróbujmy algorytmem RSA zakodować zdanie: Alea iacta est! (powiedzmy, że Cezar chce je wysłać do swoich zwolenników w Senacie, przekraczając Rubikon). Załóżmy, że posługiwano się naszą tabelką:

$*$	$+$	$-$	A	B	C	D	E	F	G	H
0	1	2	3	4	5	6	7	8	9	10
I	J	K	L	M	N	O	P	Q	R	S
11	12	13	14	15	16	17	18	19	20	21
T	U	W	V	X	Y	Z		.	?	!
22	23	24	25	26	27	28	29	30	31	32

Jak widać, $n = 33$, zatem
$\varphi(n) = \varphi(3 \cdot 11) = \varphi(3) \cdot \varphi(11) = 2 \cdot 10 = 20$. Do klucza publicznego musimy wybrać e względnie pierwsze z $\varphi(n) = 20$. Niech $e = 7$ - czyli cały klucz publiczny to $(33, 7)$.

Algorytm RSA - przykład

Klucz publiczny $(33, 7)$. Tekst jawny: Alea iacta est!

$*$	$+$	$-$	A	B	C	D	E	F	G	H
0	1	2	3	4	5	6	7	8	9	10
I	J	K	L	M	N	O	P	Q	R	S
11	12	13	14	15	16	17	18	19	20	21
T	U	W	V	X	Y	Z		.	?	!
22	23	24	25	26	27	28	29	30	31	32

Pierwszą literą, którą mamy zakodować, jest A. Odpowiada jej liczba 3. $3^7 = 2187$. Np. korzystając z kalkulatora obliczamy, że $2187 : 33 \approx 66,27$ (w drugim przykładzie przedstawimy wygodniejszy sposób obliczania reszt z dzielenia dużych potęg) , a $2187 - 66 \cdot 33 = 2187 - 2178 = 9$, zatem $f(3) = 2187 \mod 33 = 9$, a liczbie 9 odpowiada w naszej tabelce G, więc litera A zostanie zakodowana jako G.

Algorytm RSA - przykład

Klucz publiczny $(33, 7)$. Tekst jawny: Alea iacta est!
Drugą literą, którą mamy zakodować, jest L. Odpowiada jej liczba 14. Podnoszenie liczb do siódmej potęgi (nie mówiąc o większych) 14^7 może się okazać kłopotliwe, jeśli posługujemy się prostym kalkulatorem. Można rzecz sobie ułatwić „potęgując przez kwadraty" (komputery też tak robią) i sprytnie wykorzystując własności kongruencji:

$$14^7 = (14^2)^3 \cdot 14 \equiv_{33} 31^3 \cdot 14 = 31^2 \cdot (31 \cdot 14) \equiv_{33}$$

$$\equiv_{33} (-2)^2 \cdot ((-2) \cdot 14) = 4 \cdot (-28) \equiv_{33} 4 \cdot 5 = 20.$$

Liczbie 20 przyporządkowana jest litera R, więc L będzie zakodowane jako R.

Algorytm RSA - przykład

Klucz publiczny $(33, 7)$. Tekst jawny: Alea iacta est!

$*$	$+$	$-$	A	B	C	D	E	F	G	H
0	1	2	3	4	5	6	7	8	9	10
I	J	K	L	M	N	O	P	Q	R	S
11	12	13	14	15	16	17	18	19	20	21
T	U	W	V	X	Y	Z		.	?	!
22	23	24	25	26	27	28	29	30	31	32

Analogicznie kodujemy kolejne litery otrzymując (jako ćwiczenie - sprawdzić):

grzgoigltgozst!

Warto zauważyć, że niektóre znaki (np. „i” „s”, „t”, „!”) kodują same siebie - jest to raczej nieuniknione przy tak małych liczbach tworzących klucz publiczny (co jest kolejnym słabym punktem wyboru małych liczb w algorytmie RSA).

Algorytm RSA - dekodowanie

Klucz publiczny $(33, 7)$. Szyfrogram: grzgoigltgozst!

$*$	$+$	$-$	A	B	C	D	E	F	G	H
0	1	2	3	4	5	6	7	8	9	10
I	J	K	L	M	N	O	P	Q	R	S
11	12	13	14	15	16	17	18	19	20	21
T	U	W	V	X	Y	Z		.	?	!
22	23	24	25	26	27	28	29	30	31	32

Jeśli chcemy rozkodować szyfrogram, potrzebujemy klucza prywatnego. Zazwyczaj odtworzenie d jest bardzo trudne. W tym przypadku $n = 33$ jest bardzo małe, więc łatwo szyfr złamać. Obliczamy $\varphi(33) = 20$. Następnie, szukamy takiej liczby d, że $7d \equiv 1(\mod 20)$. Oczywiście, pasuje $d = 3$, bo $7 \cdot 3 = 1 \cdot 20 + 1$. Zatem klucz prywatny, to $(33, 3)$, a funkcją dekodującą jest $f^{-1}(C) = C^3 \mod n$.

Algorytm RSA - dekodowanie

Klucz prywatny $(33, 3)$. Szyfrogram: grzgoigltgozst!

$*$	$+$	$-$	A	B	C	D	E	F	G	H
0	1	2	3	4	5	6	7	8	9	10
I	J	K	L	M	N	O	P	Q	R	S
11	12	13	14	15	16	17	18	19	20	21
T	U	W	V	X	Y	Z		.	?	!
22	23	24	25	26	27	28	29	30	31	32

Powiedzmy, że chcemy odkodować „z”, występujące w szyfrogramie. Liczba odpowiadająca „z” to 28. Tak jak poprzednio, obliczamy

$$28^3 \equiv_{33} (-5)^3 \equiv_{33} 25 \cdot (-5) \equiv_{33} (-8) \cdot (-5) \equiv_{33} 40 \equiv_{33} 7.$$

7 odpowiada literze „e”, więc wiemy, że „z” możemy odkodować jako „e”. Resztę wiadomości dekodujemy analogicznie.

Algorytm RSA - dodatkowe uwagi

W praktyce algorytm taki jak RSA jest tylko podstawą systemu szyfrowania: na przykład elementy, którym przypisano liczby 0 i 1, nigdy nie zmienią swojej postaci podczas kodowania tym algorytmem, gdyż nie zmieniają swojej wartości podniesione do dowolnej potęgi (dlatego w tabelce są im przypisane nieistotne symbole). Między innymi dlatego wiadomość przed zakodowaniem powinna zostać poddana wstępnej obróbce (tzw. *padding*): dobranie jednostkom szyfrowania odpowiednich liczb, dodanie pewnego „szumu informacyjnego", dodanie bezsensownych sekwencji znaków na początku i końcu (by nie wskazywać położenia charakterystycznych zwrotów typu: Raport, albo Szanowny Panie, czy też podpis nadawcy).Te techniki nie wchodzą w zakres materiału wykładu.

Algorytm RSA - dodatkowe uwagi

System RSA pozostaje bezpieczny, o ile klucze są odpowiednio długie - ich długość musi się stawać coraz większa. Obecnie rekomendowane są klucze 2048-bitowe. Te ostatnie nie powinny być możliwe do złamania w rozsądnej przyszłości (chyba, że w wyniku użycia zupełnie nowych narzędzi np. komputerów kwantowych). Największy złamany klucz (tj. klucz dla którego przedstawiono efektywny algorytm znajdowania rozkładu) na 15 X 2024 miał rozmiar 829 bitów (250 cyfr zapisu dziesiętnego), acz też w olbrzymim czasie. Klucze o rozmiarze 300 bitów lub mniejszym da się złamać przy pomocy zwyczajnych komputerów i darmowego oprogramowania w kilka godzin.

Inne algorytmy kryptograficzne

Algorytm RSA jest dobrym przykładem ilustrującym działanie kryptosystemów z kluczem publicznym, ale nie jedynym. Najbardziej znane z innych to:

- Protokół Rabina - oparty na wyznaczaniu pierwiastków w $\mathbb{Z}_n$. Funkcja kodująca nie jest różnowartościowa!
- Protokół Diffiego-Hellmana i algorytm ElGamal - oparte na wyznaczaniu tzw. logarytmów dyskretnych w $\mathbb{Z}_n$..
- Kryptografia krzywych eliptycznych (Koblitz, Miller 1985) - odporna na komputery kwantowe, ale o niejasnym statusie prawnym.

Inne algorytmy kryptograficzne

Inne algorytmy:

- Protokół Rabina.
- Protokół Diffiego-Hellmana i algorytm ElGamal.
- Kryptografia krzywych eliptycznych (Koblitz, Miller 1985)
- Ponadto, gdy obie strony komunikacji mają możliwość wcześniejszego uzgodnienia wspólnego klucza używanego wyłącznie do kontaktów między nimi, używane są algorytmy *kryptografii symetrycznej* np. AES (Rijndael), Twofish, Serpent, dawniej DES. Matematycznie, zazwyczaj takie algorytmy są oparte na tzw. sieci substytucji-permutacji.

Algorytm RSA - część I - przypomnienie

Algorytm RSA

- I. Wybieramy losowo dwie różne, bardzo duże liczby pierwsze p i q. Im większe, tym trudniej złamać kod, ale też dłużej trwa proces kodowania i dekodowania.
- II. Niech $n = pq$. Wtedy $\varphi(n) = \varphi(p) \cdot \varphi(q) = (p-1)(q-1)$. Znów losowo wybieramy liczbę e względnie pierwszą z $\varphi(n)$. Znajdujemy d takie, że $e \cdot d \equiv 1(\mod \varphi(n))$. (np. za pomocą rozszerzonego algorytmu Euklidesa).
- III. Klucz publiczny $K_K = (n, e)$ można opublikować. W sekrecie zachowujemy $K_D = (n, d)$.
- IV. Zbiór wszystkich jednostek wiadomości definiujemy jako $\mathbb{Z}_n = \{0, 1, \ldots, n-1\}$. Funkcją kodującą będzie $f(P) = P^e \mod n$.

Algorytm RSA - część II - przypomnienie

Algorytm RSA

- IV. Zbiór wszystkich jednostek wiadomości definiujemy jako $\mathbb{Z}_n = \{0, 1, \ldots, n-1\}$. Funkcją kodującą będzie $f(P) = P^e \mod n$.
- V. Funkcją dekodującą będzie $f^{-1}(C) = C^d \mod n$.

Wszystkie operacje algorytmu nie budzą wątpliwości, aż do punktu V. Ale czy faktycznie funkcja dekodująca jest odwrotnością kodującej? Sprawdzenie na kilku przykładach nie wystarczy...

Twierdzenie o poprawności algorytmu RSA

Twierdzenie o poprawności algorytmu RSA

Jeśli $p, q \in \mathcal{P}$, $n = pq$, $e \perp \varphi(n)$ i $e \cdot d \equiv 1(\mod \varphi(n))$, to funkcja $g(x) = x^d \mod n$ $(g : \mathbb{Z}_n \to \mathbb{Z}_n)$ jest odwrotna do funkcji $f(x) = x^e \mod n$ $(f : \mathbb{Z}_n \to \mathbb{Z}_n)$, czyli

$$\forall_{x \in \mathbb{Z}_n} : x^{ed} \equiv_n x.$$

Dowód: Wiemy, że $e \cdot d \equiv_{\varphi(n)} 1$, więc $ed = a\varphi(n) + 1$ dla pewnego a. Rozważymy dwie (jedyne) możliwości: albo $x \perp n$, albo nie.
Jeśli $x \perp n$, to z Twierdzenia Eulera dostajemy $x^{\varphi(n)} \equiv_n 1$, a zatem

$$x^{ed} = x^{a\varphi(n)+1} = \left(x^{\varphi(n)}\right)^a \cdot x \equiv_n 1 \cdot x = x.$$

Twierdzenie o poprawności algorytmu RSA - dowód

Twierdzenie o poprawności algorytmu RSA

Jeśli $p, q \in \mathcal{P}$, $n = pq$, $e \perp \varphi(n)$ i $e \cdot d \equiv 1(\mod \varphi(n))$, to (...)

$$\forall_{x \in \mathbb{Z}_n} : x^{ed} \equiv_n x.$$

Dowód: Teraz niech $x \not\perp n$. Skoro n ma tylko dwa dzielniki pierwsze, p i q, więc $p|x$ lub $q|x$.Załóżmy, bez straty ogólności, że $q|x$ (jeśli $p|x$ dowód jest taki sam, tylko zamieniamy literki p i q w dalszym rozumowaniu), czyli $x = bq$ dla pewnego $0 < b < p$ (bo $bq = x < n = pq$). Na mocy Małego Twierdzenia Fermata mamy:

$$b^{p-1} \equiv_p 1; \qquad q^{p-1} \equiv_p 1.$$

Zatem

$$b^{a(p-1)(q-1)} \equiv_p 1; \qquad q^{a(p-1)(q-1)} \equiv_p 1.$$

Twierdzenie o poprawności algorytmu RSA - dowód

Twierdzenie o poprawności algorytmu RSA

Jeśli $p, q \in \mathcal{P}$, $n = pq$, $e \perp \varphi(n)$ i $e \cdot d \equiv 1(\mod \varphi(n))$, to (...)

$$\forall_{x \in \mathbb{Z}_n} : x^{ed} \equiv_n x.$$

Dowód: Wiemy, że $x = bq$, $ed = a\varphi(n) + 1$ i $\varphi(n) = (p-1)(q-1)$ oraz

$$b^{a(p-1)(q-1)} \equiv_p 1; \qquad q^{a(p-1)(q-1)} \equiv_p 1,$$

więc

$$(bq)^{a(p-1)(q-1)+1} = bq \cdot b^{a(p-1)(q-1)} \cdot q^{a(p-1)(q-1)} \equiv_p bq.$$

I ostatecznie:

$$x^{ed} = (bq)^{a(p-1)(q-1)+1} \equiv_n bq = x. \quad \square$$