DUOMENŲ STRUKTŪROS IR ALGORITMAI

MARIUS GŽEGOŽEVSKIS

Kriptografijos uždaviniai ir priemonės

Slaptumas (konfidencialumas, *confidentiality*)

- Šifravimas

Vientisumas (integralumas, *integrity*)

- Kriptografinės maišos funkcijos
- Skaitmeninis parašas

Autentiškumas (tapatumo nustatymas, *authenticity*)

- MAC
- Skaitmeninis parašas

Informacijos vientisumas

Vientisumas *(integrity)* – tai garantija, kad bus išsaugotos teisingos duomenų reikšmės. Tai užtikrinama draudžiant neautorizuotiems vartotojams kokiu nors būdu pakeisti, modifikuoti, sunaikinti, arba kurti duomenis.

Kriptografijos uždaviniai ir priemonės

Maiša (angl. hashing). Tai yra matematinė transformacija, atliekama tam tikro ilgio simbolių eilutę verčiant į fiksuoto ilgio – dažniausiai trumpesnę – reikšmę.

Ši reikšmė atstovauja originaliai eilutės reikšmei.

Dažniausiai maiša naudojama duomenų įrašų duomenų bazėje indeksuoti ar ištraukti, nes taip operacija yra atliekama greičiau – surasti trumpesnę reikšmę yra mažiau sąnaudų reikalaujanti operacija, negu surasti originalią simbolių eilutę. Maiša taip pat naudojama ir kodavimo algoritmuose.

Kriptografijos uždaviniai ir priemonės

Maišos funkcija (angl. hash function). Tai funkcija f = f (x), priskirianti argumentui x pseudoatsitiktinį skaičių, vadinamą maišos kodu (angl. hash code). Tam pačiam argumentui funkcija visada turi duoti tokį patį rezultatą, taigi ji nėra atsitiktinė. Dažniausiai funkcijos reikšmių sritis (t. y. f) yra, palyginti su apibrėžimo sritimi (t. y. x), nedidelė.

Duomenų santrauka (angl. message digest, cheksum). Kriptografinės maišos funkcijos rezultatas, transformavus tam tikro fiksuoto dydžio duomenis (Preneel 2003).

Maišos funkcijų savybės

Maišos funkcijos yra ***determinuotosios***, t. y. ne atsitiktinės:

- Skaičiuojant maišos reikšmę tai pačiai įvesčiai kelis kartus, visada bus gaunamas tas pats rezultatas.

Kadangi galimų įvesčių yra daugiau, negu galimų išvesčių, tai maišos funkcijos nėra *injektyvios*:

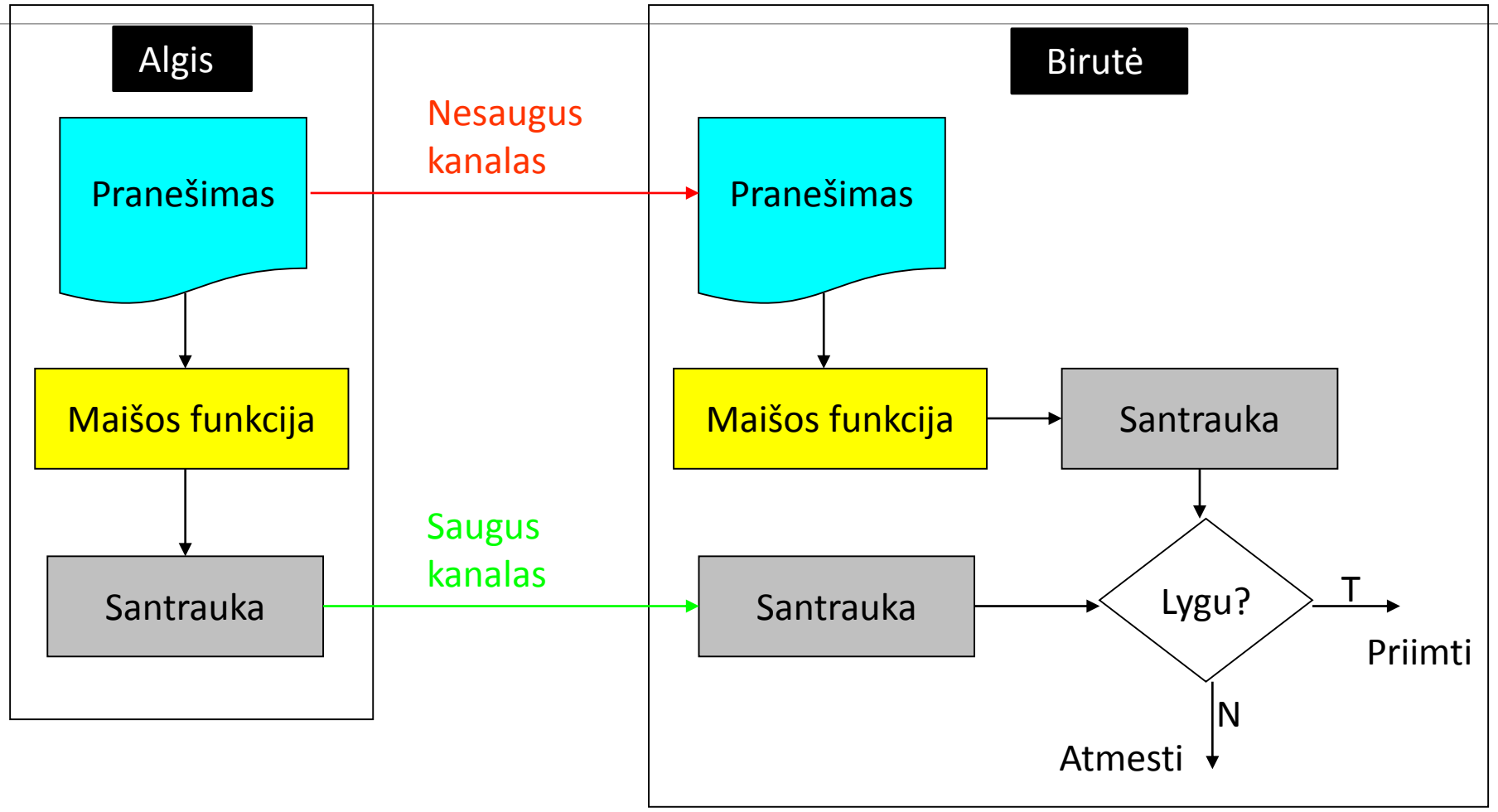
- Gali būti, kad skirtingoms įvestims maišos reikšmės bus vienodos. Tai vadinama **sutapimu** (kolizija, angl. ***collision***).

Pavyzdys

Algis rašo testamentą, norėdamas paskirstyti savo turtą po mirties.

- Testamentas ***neturi būti užšifruotas***, kad bet kas galėtų su juo susipažinti.
- Tačiau jo ***vientisumas turi būti išsaugotas*** (Algis nenori, kad jo testamentą pakeistų kas nors kitas).

užtikrinti: Schema



Santraukos naudojimas vientisumui užtikrinti

Kadangi maišos funkcijos yra žinomos viešai, tai bet kas gali apskaičiuoti jų reikšmes. Todėl santrauka turi būti perduodama ***saugiu kanalu***, nes kitaip atakuojantysis galės pakeisti ir pranešimą, ir santrauką, apskaičiavęs pakeisto pranešimo **maišos reikšmę**.

Santraukos naudojimas vientisumui užtikrinti

Be to, vientisumui užtikrinti bet kokios maišos funkcijos netinka. Pavyzdžiui, tarkime, kad maišos funkcija yra tokia, kad turint pranešimą ir jo santrauką, nesunkiai galima rasti kitą pranešimą su tokia pačia santrauka.

Tada atakuojantysis galės pakeisti pranešimą į kitą pranešimą, ir gavėjas to nepastebės. Vientisumui užtikrinti reikia naudoti ***kriptografines maišos funkcijas***.

Kriptografinės maišos funkcijos apibrėžimas

Kriptografine maišos funkcija vadiname maišos funkciją, kuri yra:

Vienakryptė (*preimage resistant*, *one-way*): pranešimo santrauką apskaičiuoti yra lengva, o turint santrauką rasti atitinkamą pranešimą yra skaičiavimų prasme neįmanoma (*computationally infeasible*),

Kriptografinės maišos funkcijos apibrėžimas

Atspari sutapimams (*2-nd preimage resistant*, *weakly collision resistant*): turint pranešimą, skaičiavimų prasme neįmanoma rasti dar vieną pranešimą su ta pačia santrauka,

Labai atspari sutapimams (*collision resistant*, *strongly collision resistant*): skaičiavimų prasme neįmanoma rasti sutapimą, t. y. du pranešimus su ta pačia santrauka.

Lavinos efektas

Pageidautina, kad kriptografinė maišos funkcija tenkintų savybę, vadinama **lavinos efektu** (angl. *avalanche effect*).

Taip kriptografijoje vadinama kriptografinių algoritmų (paprastai blokinių kriptosistemų ir kriptografinių maišos funkcijų) savybė, kai nežymiai pakeitus įvestį (pavyzdžiui, pakeitus vos vieną bitą), išvestis pasikeičia žymiai (pavyzdžiui, pusė visų išvesties bitų pasikeičia).

Lavinos efektas

Pavyzdys. Žodžiai *Taisyklė* ir *taisyklė* skiriasi tik vienu bitu, o jų MD5 reikšmės labai skiriasi:

- T – 0x54 – 0101 0100
- t – 0x74 – 0111 0100
- ***Taisyklė***:
  - MD5: f26ad7627a11f62e559a9a6516dd2392
- ***taisyklė***:
  - MD5: f0517727eaba12da46385b6b9418ff30

Maišos funkcijos

Pagrindinės plačiausiai naudojamos maišos funkcijos:

1. **MD5** (Message-Digest algorithm 5). Naudojama tokiuose plačiai naudojamuose protokoluose ir programose, kaip TLS ir SSL, SSH, PGP, S/MIME, IPsec.
2. **SHA** (Secure Hash Algorithm).

Maišos (angl. **hash**) formavimo funkcijos yra vienakryptės funkcijos, kurios bet kokio ilgio pranešimui suformuoja fiksuoto ilgio maišos reikšmę (santrauką).

Pavyzdžiui pritaikius MD5 maišos funkciją:

- **Pradinis tekstas :** Automobilių spūstys vargina viso pasaulio vairuotojus.
- **Santrauka (MD5):** E537F1B48B26FB88C2586E5F5F7E32BF

Maišos funkcijos

Maišos funkcijos naudojamos informacijos vientisumo patikrinimui, slaptažodžių saugojimui, paieškos raktų formavimui duomenų bazėse ir panašiai.

Prieš siunčiant pranešimą, jam suformuojama maišos reikšmė, kuri perduodama kartu su pranešimu.

Vartotojas, gavęs pranešimą, vėl suformuoja jo maišos reikšmę ir sulygina ją su gautąja.

MD5 santraukos formavimo algoritmo esmė:

Pradinis tekstas suskirstomas į N blokų po 512 bitų (64 baitus). Jei paskutiniame M_N bloke trūksta duomenų iki 512 bitų, bloko gale pridedamas 1 ir tiek nulių, kad būtų užpildyta likusi bloko dalis.

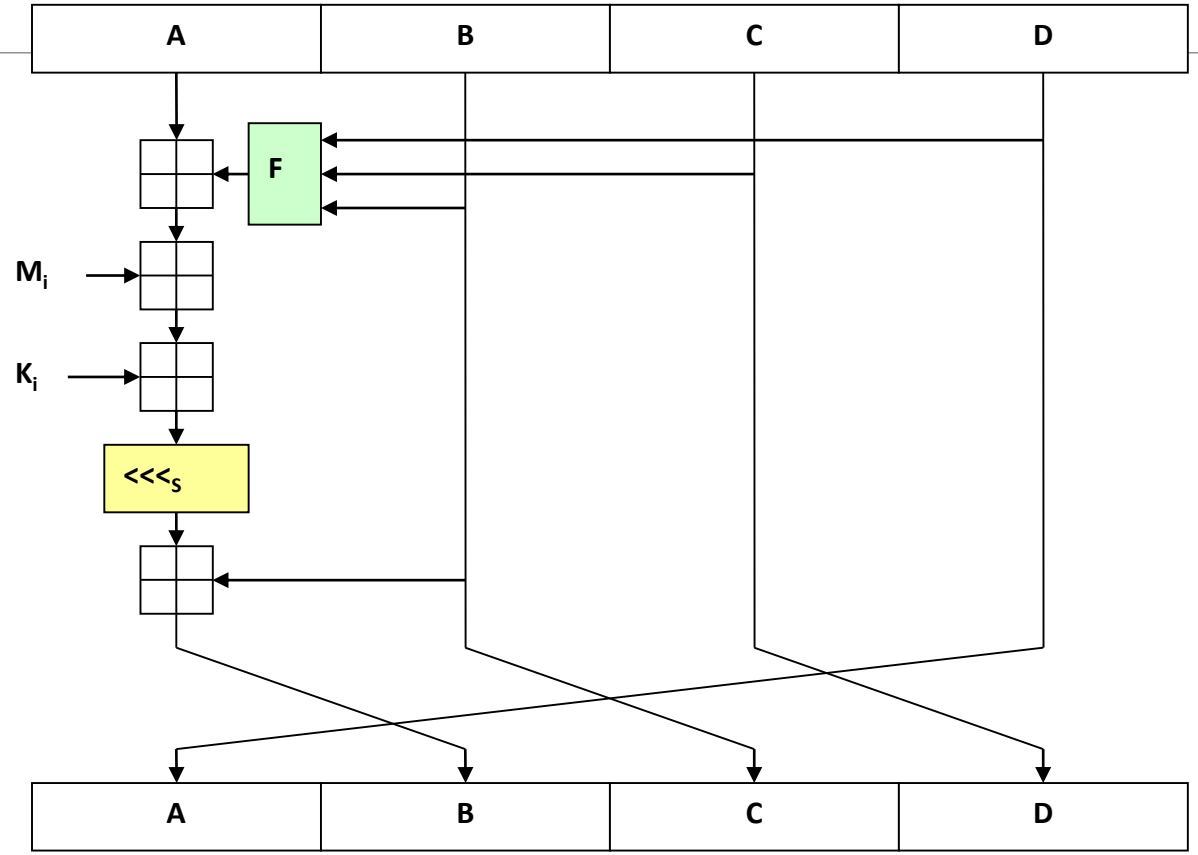
Pagrindinis MD5 algoritmas dirba su 128 bitų rezultato eilute, padalinta į keturis 32 bitų žodžius, pažymėtus A, B, C ir D. Šie žodžiai užpildomi pradinėmis reikšmėmis.

MD5 santraukos formavimo algoritmo esmė:

Po to algoritmas ima pradinio teksto **512** bitų ilgio blokus ir modifikuoja rezultato eilutę.

Pranešimo bloko apdorojimas susideda iš **4** panašių ciklų, kurių kiekvienas susideda iš 16 operacijų pagrįstų netiesine funkcija F, sudėties moduliu ir postūmio į kairę operacijomis.

MD5 vieno ciklo veikimo schema:



<<<$_s$ – ciklinis postūmis kairėn per *s* bitų, *s* kinta kiekvienai operacijai;
- suma moduliu 2^{32}. K_i – konstanta, skirtinga kiekvienai iteracijai.

MD5 santraukos formavimo algoritmas

Pradžioje būsenos eilutė užpildoma pradinėmis reikšmėmis:
A: 01 23 45 67
B: 89 ab cd ef
C: fe dc ba 98
D: 76 54 32 10
Naudojamos keturios funkcijos F, kurių argumentai yra trys 32-bitų žodžiai, o rezultatas vienas 32-bitų žodis:

$$F(X,Y,Z) = (X \wedge Y) \vee (\neg X \wedge Z)$$

$$G(X,Y,Z) = (X \wedge Z) \vee (Y \wedge \neg Z)$$

$$H(X,Y,Z) = X \oplus Y \oplus Z$$

$$I(X,Y,Z) = Y \oplus (X \vee \neg Z)$$

$\oplus$ - suma moduliu 2, $\wedge$ - griežta disjunkcija (xor), $\vee$ - arba, $\neg$ - ne.

MD5 santraukos formavimo algoritmas

1. Kiekviename etape žodžiai M_i perbėgami vis kita eilės tvarka.
2. Konstantos K_i kiekvienoje iteracijoje yra skirtingos.
3. Atlikus šiuos veiksmus visiems blokams gautos A, B, C, D reikšmės sujungiamos. Tai ir yra maišos funkcijos reikšmė.

Asimetrinio šifravimo algoritmai

1. Diffte-Hellman.
2. Elipsinės kreivės.
3. **RSA** (Rivest-Shamir-Adleman).

RSA sistema

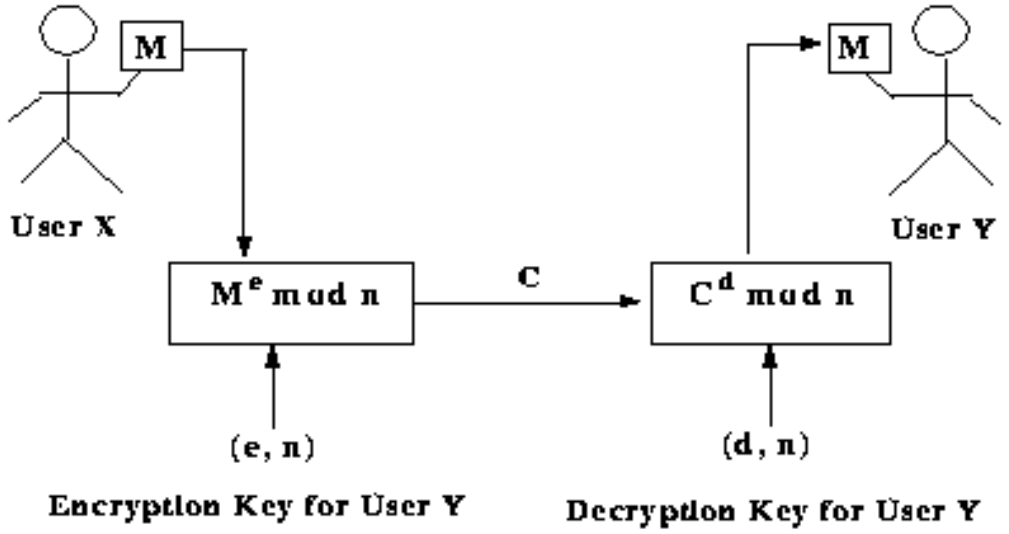
RSA (Rivest-Shamir-Adleman, kintamo kodo ilgio) populiariausias viešojo rakto algoritmas.

Šio metodo patikimumas pagrįstas didelių skaičių faktorizacijos sudėtingumu.

RSA šifravimo raktai veikia abejomis kryptimis, t.y. galima užšifruoti žinutę **privačiuoju raktu**, o iššifruoti **viešuoju siuntėjo raktu**, arba galima užšifruoti žinutę **viešuoju raktu**, o **iššifruoti privačiuoju raktu**.

RSA sistema

Tai naudojama skaitmeniniam parašui, kadangi tik parašo autorius yra vienintelis asmeninio rakto savininkas. Tai garantuoja dokumento autoriaus ar siuntėjo autorystę ir duomenų pirmapradį originalumą.



RSA algoritmo esmė

1. Imami 2 pirminiai skaičiai *p* ir *q* (paprastai didesni už 10^{100}).
2. Apskaičiuojamos sandaugos ***s = p * q*** **ir** ***t = (p - 1) * (q - 1)***.
3. Randamas skaičius *a*, neturintis bendro vardiklio su *t*.
4. Randamas skaičius *b* toks, kad ***b * a = 1*** pagal modulį ***t***.

RSA algoritmo esmė

1. Išeities tekstas *T* suskaidomas blokais taip, kad $0 < T < s$. Tai galima padaryti grupuojant tekstą blokais po *k* bitų, kur *k* — didžiausias sveikas skaičius, kuriam $2^k < s$.
2. Informacijos užkodavimui reikia apskaičiuoti $C = T * b$ pagal modulį s, o atkodavimui - $T = C * a$ pagal modulį s. Tokiu būdu informacijos užšifravimui reikia žinoti skaičių porą *(b, s)*, o iššifravimui — skaičių porą *(a, s)*. Pirmoji pora yra viešasis raktas, antroji pora – asmeninis raktas.

RSA algoritmo pavyzdys (raktų formavimas) 1 iš 3

Realiose realizacijose naudojami labai dideli skaičiai, iki 100+ skaitmenų. Pavyzdyje paprastumo dėlei paimsime mažus skaičius.

1. Tegul ***p = 3, q = 11***. Dalybos moduliu operaciją žymėsime ***mod*** ženklu. Raktų porą gaunama taip:
   - ***s*** *= 3 *11 = 33;*
   - ***t*** *= (3 – 1) * (11 – 1) = 20.*
2. Randame skaičių *a*, neturintį bendro vardiklio su ***20***. Tai bus ***a = 7***.
3. Apskaičiuojame skaičių *b* tokį, kad ***(b * 7) mod t = 1***. Tai bus ***b = 3***.
4. Taigi raktų pora: ***(3, 33)*** **– viešasis raktas**; ***(7, 33)*** **– asmeninis raktas**.

RSA algoritmo pavyzdys (raktų formavimas) 2 iš 3

Tarkime, mums reikia užšifruoti keturioliktą abėcėlės raidę, kurios kodas yra **14**.

1. Pakeliame laipsniu 3 skaičių 14. $\boldsymbol{14^3 = 2744}$;
2. Daliname rezultatą moduliu 33: $\boldsymbol{2744 \bmod 33 = 5}$;

Taip mūsų ***keturioliktoji*** abėcėlės raidė tampa *penktąja*.

RSA algoritmo pavyzdys (raktų formavimas) 3 iš 3

1. Taigi raktų pora: ***(3, 33)*** **– viešasis raktas**; ***(7, 33)*** **– asmeninis raktas**.
2. Raidei atstovaujantį skaičių 5 pakeliame laipsniu **7**: $\mathbf{5^7 = 78125}$;
3. Daliname rezultatą moduliu **33**: 78125 **mod** 33 = 14;
4. Taip ***penktoji*** abėcėlės raidė tampa ***keturioliktąja***.

Užkodavimas yra abipusis, naudojant viešąjį ar privatųjį raktą.

- It is sometimes misunderstood that encryption can only happen with the ***public key*** and decryption with the ***private key*.** This statement is not true.
- In RSA, both the ***public*** and the ***private*** keys will encrypt a message. If that message is to be decrypted, then the opposite key that was used to encrypt it must be used to decrypt it.

RSA Realaus panaudojimo pavyzdys

PLAČIAU APIE RSA: http://doctrina.org/How-RSA-Works-With-Examples.html

Now for a real world example, lets encrypt the message "attack at dawn". The first thing that must be done is to convert the message into a numeric format. Each letter is represented by an ascii character, therefore it can be accomplished quite easily.

I am not going to dive into converting strings to numbers or vice-versa, but just to note that it can be done very easily. How I will do it here is to convert the string to a bit array, and then the bit array to a large number. This can very easily be reversed to get back the original string given the large number. Using this method, "attack at dawn" becomes 19766202164023008896244827187751 50 (for those interested, here is the code that I used to make this conversion).

Rakto generavimas (angl. Key Generation)

Now to pick two large primes, p and q. These numbers must be random and not too close to each other. Here are the numbers that I generated: using **Rabin-Miller primality tests**:

p

12131072439211271897323671531612440428472427633701410925634
54931230196437304208561932419736532241686654101705736136521
41717117137979742993348710628298035 41

q

12027524255478748885956220793734512128733387803682075433
65389998395517985098879789986914690080913161115334681705
08320960221601463663463918124709871054152 33

RSA

With these two large numbers, we can calculate n and **ϕ(n)**

n

14590676800758332301869393490706352924018723753571643995818710198734387990053589383695714026701498021218180862924674228281570229220767469065434012248896724724079269699871005812901031993178587536637108623576565105078837142971156373427889114635351027120327651665184117268598379886721118372050855263466187400 53

ϕ(n)

14590676800758332301869393490706352924018723753571643995818710198734387990053589383695714026701498021218180862924674228281570229220767469065434012248896483138112322799663173013977778523653015478482734788712972220585874571528916064592697181192689711635550708026439995295496441168119475165139381842966835212 80

RSA

e - the public key

65537 has a gcd of 1 with $\phi(n)$, so lets use it as the public key. To calculate the private key, use extended euclidean algorithm to find the multiplicative inverse with respect to $\phi(n)$.

d - the private key

894894250092744443682285459217730939196695860658842574454978544564876748396298183909349419732628796167979706089172836798754993315741611138540888132754881105882471930775825272784379065040156806234235500672400424666656542323835029222154936232894721388664458187891279461234078077257026266440910365023725451397 13

Encryption/Decryption

Encryption: $19766202164023008896244827187751 50^{e} \bmod n$

350521113386730266902124239370533285118807608115799816206428023466858106231098502359430490809733862411137840407947041939782153784997654130836464387847409523069325349451950801838615742252262188798272324539128205968864403775360824656817500744174591514854074458625110234722355608230534977915189288202722577877 86

Decryption:

$$350521113386730266902124239370533285118807608115799816206428023466858106231098502359430490809733862411137840407947041939782153784997654130836464387847409523069325349451950801838615742252262188798272324539128205968864403775360824656817500744174591514854074458625110234722355608230534977915189288202722577877866^{d} \bmod n$$

19766202164023008896244827187751 50 (which is our plaintext "attack at dawn")

This real world example shows how large the numbers are that is used in the real world.