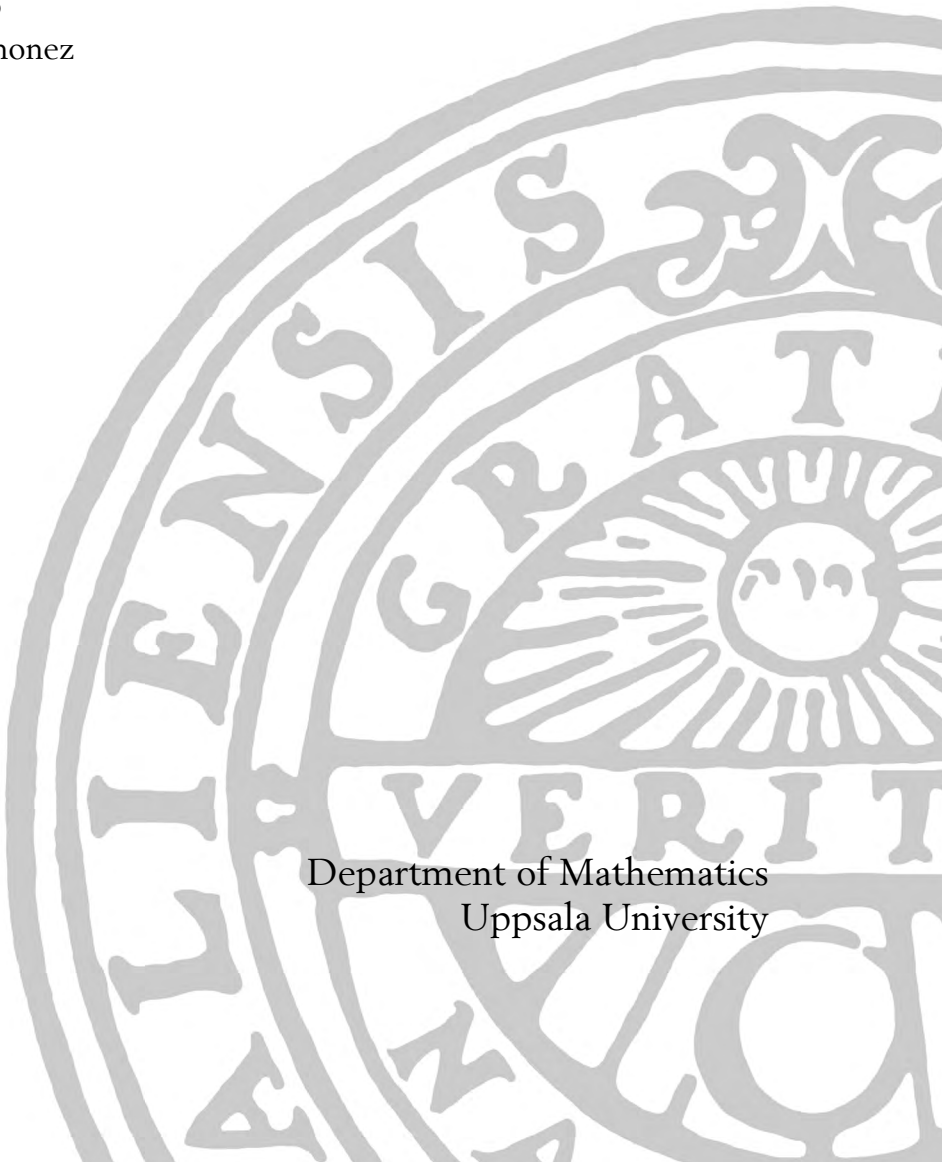
UPPSALA
UNIVERSITET

U.U.D.M. Project Report 2016:31

Primtal och kryptografi

Wilhelm Carlbaum

Examensarbete i matematik, 15 hp
Handledare: Veronica Crispin Quinonez
Examinator: Jörgen Östensson
Juni 2016



Department of Mathematics
Uppsala University

Primtal och kryptografi

Wilhelm Carlbaum

21 juni 2016

Innehåll

1 Bakgrund **2**
- 1.1 Definitioner och notationer . . . 2
- 1.2 Algebra . . . 3
  - 1.2.1 Grupp . . . 3
  - 1.2.2 Kropp . . . 4
  - 1.2.3 Ring . . . 5
  - 1.2.4 Euklides algoritm . . . 5
  - 1.2.5 Kongruenser . . . 7
- 1.3 Primtal . . . 8

2 Primtal och kryptografi **11**
- 2.1 Historisk översikt av enkla kryptosystem . . . 11
- 2.2 Primalitetstest . . . 14
  - 2.2.1 Regler för primtal och pseudoprimtal . . . 14
  - 2.2.2 Solovay-Strassens primalitetstest . . . 15
  - 2.2.3 Miller-Rabbin primalitetstest . . . 16
- 2.3 Komplexitet . . . 16
- 2.4 RSA . . . 17
  - 2.4.1 Algoritmer . . . 17
  - 2.4.2 Hashfunktion . . . 20
  - 2.4.3 Signatur . . . 21
  - 2.4.4 Attacker . . . 21
- 2.5 Diffie-Hellmans nyckelbyte . . . 25
  - 2.5.1 Algoritm . . . 25
  - 2.5.2 Säkerhet . . . 26

Kapitel 1

Bakgrund

1.1 Definitioner och notationer

De flesta definitioner och notationer som tas upp i detta stycke är desamma som används i *A Course in Number Theory and Cryptography* av Neal Koblitz [K], de kommer också att användas i denna uppsats.

Kryptografi är studien av metoderna för hur man skickar hemliga meddelanden så att endast den tilltänkta mottagaren kan återställa meddelandet och få fram innebörden. Meddelandet vi vill skicka kallas för *klartext* och det hemliga meddelandet kallas för *chiffertext*. Processen att konvertera en klartext till en chiffertext kallas för *chiffrering* eller *kryptering* och den omvända processen kallas *dechiffrering* eller *dekryptering*.

Chiffreringstransformationen är en funktion, vi kan kalla den f, som tar vilken klartext som helst och ger tillbaka en chiffertext. Med andra ord är f en avbildning från mängden av alla klartexter P till mängden av alla chiffertexter C. Vi kommer anta att f är ett "ett till ett"-förhållande, det vill säga att det för varje chiffertext bara finns en enda klartext. *Dechiffreringstransformationen* är avbildningen f^{-1} som går åt det motsatta hållet och ger tillbaka klartexten från chiffertexten. Vi kan representera situationen schematiskt med figuren

$$P \xrightarrow{f} C \xrightarrow{f^{-1}} P$$

En sådan uppställning kallas för *kryptosystem*.

Det finns också fall där man inte vill kryptera eller dekryptera ett meddelande utan bara läsa det. Det kallas att *knäcka* ett chiffer och vetenskapen som arbetar med det kallas *kryptoanalys*. För att knäcka ett chiffer behöver man två typer av information. Först måste man veta den generella *strukturen* för chiffret. Den typen av information kan vara att man vet vilket slags chiffer det är som används, vilket alfabetet som används i chiffret och om det överstätts till numeriska motsvarigheter. Den andra typen av information man behöver känna till är de speciella parametrarna som används i chiffret. I ett chiffer som byter plats på bokstäver enligt ett visst mönster skulle den andra typen av information motsvara parametern som visar exakt hur bokstäverna byter plats. Med de två typerna av information kan man chiffrera och dechiffrera en text enligt en funktion som tar $C \xrightarrow{f} P$ och $P \xrightarrow{f^{-1}} C$. Funktionen f använder parametern för

att omvandla P till C och f^{-1} använder parametern omvänt för att gå från C till P.

Parametern som gör att vi kan chiffrera och dechiffrera ett meddelande kallas för *nyckel* eller mer specifikt *krypteringsnyckel*. Mer avancerade chiffer har oftast flera parametrar.

Med *bit*, eller *bitar* menas förkortningen av det engelska ordet "binary digit".

Med *meddelande-enhet* menas hur stor sträng av bokstäver som krypteras. Om vi exempelvis krypterar en bokstav i taget blir en bokstav en enkel meddelande-enhet. I vissa kryptosystem kan en meddelande-enhet istället vara en sträng med k-antal bokstäver som krypteras med ett kodord som också är av längd k. Denna sista notation är densamma som i *Algebraic Aspects of Cryptography*, också av Neal Koblitz[N] och kommer att användas i uppsatsen.

1.2 Algebra

1.2.1 Grupp

Nedanstående framställning kommer från K. Erikssons och H. Gavels bok *Diskret matematik fördjupning* [E].

Definition 1. En *grupp* är en mängd G med ett räknesätt $*$ som uppfyller de fyra gruppaxiomen:

1. **Slutenhet**: Tar man två element i G så får man ett element i G.

   $$\forall x \forall y [x \in G \wedge y \in G \rightarrow (x * y) \in G]$$

   Då kallas det att G är sluten under operationen $*$.

2. **Associativitet**: Det spelar ingen roll hur vi grupperar en beräkning med tre objekt. Det vill säga att

   $$(x * y) * z = x * (y * z)$$

   och man inte behöver sätta ut parenteser utan kan skriva $x * y * z$.

3. **Identitetselement**: Det finns ett element I som inte påverkar de andra. Det betyder att identiteselementet, I, är ett element som vid operation ger samma tal tillbaka.

   $$I * x = x * I = x$$

   För addition är identitetselementet 0 och för multiplikation är det 1.

4. **Inverser**: Till varje element $x \in G$ kan vi hitta ett annat element $x^{-1} \in G$ som vid operation för oss till identitetselementet I. x^{-1} kallas då för inversen till x.

   $$x * x^{-1} = x^{-1} * x = I$$

Exempel 1. För addition är inversen till ett heltal a ett tal som adderat med a ger talet 0, nämligen $-a$. För multiplikation är inversen till ett tal a det tal som multiplicerat med a ger talet 1, nämligen $^1/a$.

Grupper brukar betecknas $\langle G, * \rangle$, eller bara G om ingen förvirring kan uppstå kring vilken operation som avses.

En *delgrupp* $\langle H, * \rangle$ är en grupp som består av en delmängd H till G som är en grupp under samma räknesätt som G.

En *ändlig grupp* är en grupp där mängden G är ändligt stor.

Om vi ur en grupp $\langle G, * \rangle$ tar ett element $g \in G$ och parar ihop det med sig själv får vi $g * g$. Det elementet kan vi beteckna g^2 med vanlig potensnotation. g^{-2} låter vi på samma sätt vara beteckningen för $g^{-1} * g^{-1}$, och g^0 får vara beteckningen för identitetelementet I. Vi kan nu räkna med de vanliga potenslagarna:

$g^m * g^n = g^{m+n}$ för alla heltal m, n.

För varje element $g \in G$ kan vi nu definiera mängden av alla dess potenser som $\langle g \rangle$:

$$\langle g \rangle = \{g^n \mid n \in Z\}$$

$\langle g \rangle$ bildar också en grupp, som kallas *gruppen som genereras av* g eftersom att alla element i gruppen är en potens av g. Elementet g kallas för gruppens *generator*.
En grupp som kan genereras av ett element kallas för en *cyklisk grupp*.

Om G är en ändlig grupp kommer delgruppen $\langle g \rangle$ också att vara ändlig, eftersom att den inte kan vara större än G. Antalet element i den cykliska delgruppen $\langle g \rangle$ kallas för *ordningen* av g.

1.2.2 Kropp

Följande definition kommer från *Algebraic Aspects of Cryptography* [N].

Definition 2. En *kropp* är en mängd F med *multiplikations-* och *additionsoperationer* $\times$ och $+$ som tillfredsställer följande fem regler

(i) Associativitet på samma sätt som för grupper, och man kan skriva $x * y * z$ för en operation med tre element.

(ii) Kommutativitet, det vill säga att

$$a * b = b * a$$

som gäller för alla $a, b \in F$ med avseende på både addition och multiplikation.

(iii) Den distributiva lagen

$$a * (b + c) = a * b + a * c$$

för alla $a, b, c \in F$.

(iv) Existensen av en additativ identitet som vi kallar 0 och en multiplikativ identitet som vi kallar 1.

(v) Additiva inverser och multiplikativa inverser för varje element i F utom 0.

1.2.3 Ring

Definition 3 är hämtad från *Algebraic Aspects of Cryptography*[N], och definition 4, 5, och 6 kommer från *Abstrakt Algebra* av Per-Anders Svensson[S].

Definition 3. En *ring* är en mängd R med en multiplikation- och en additionoperation som tillfredsställer reglerna för en kropp, förutom att nollsklida element inte alltid har en multiplikativ invers.

Definition 4. Ett element a i en ring kallas för en *nolldelare*, om $a \neq 0$ och om det finns ett $b \neq 0$ i R så att $ab = 0$ eller $ba = 0$. En kommutativ ring med en etta kallas för ett *integritetsområde* om den saknar nolldelare.

Ett klassiskt exempel på ett integritetsområde är heltalsringen $\mathbb{Z}$.

Definition 5. Låt R vara en ring med en etta 1. Ett element $a \in R$ som har en multiplikativ invers a^{-1} sägs vara en *enhet*.

Definition 6. Låt D vara ett integritetsområde. Ett element $p \neq 0$, som inte är en enhet, kallas för ett *irreducibelt element*, om det för varje faktorisering $p = ab$ gäller att antingen a eller b är en enhet.

Hädanefter kommer vi att arbeta med mängder av tal och de gängse operationerna + och ×.

1.2.4 Euklides algoritm

Euklides algoritm används som en metod för att hitta den största gemensamma delaren, SGD, för två tal a, b även om man inte vet primtalsfaktoriseringen för varken a eller b. Då Euklides algoritm är bekant sedan tidigare påminner vi om den här med ett exempel från *A Course in Number Theory and Cryptography* [K].

Exempel 2. Hitta $SGD(1547, 560)$.

$$1547 = 2 \times 560 + 427$$

$$560 = 1 \times 427 + 133$$

$$427 = 3 \times 133 + 28$$

$$133 = 4 \times 28 + 21$$

$$28 = 1 \times 21 + 7$$

Eftersom att $7|21$, är vi klara och $SGD(1547, 560) = 7$

Om $SGD(a, b) = 1$ för två tal a och b säger vi att a och b är *relativt prima* till varandra. Det betyder att de inte har någon gemensam delare större än 1.

Proposition 1. Låt $d = SGD(a, b)$, där $a > b$. Då finns det heltal u och v så att $d = ua + bv$. Med andra ord, SGD:n av två tal kan uttryckas som en linjär kombination av talen med heltalskoefficienter.

Bevis. : Idén är att man utför Euklides algoritm "baklänges", det vill säga att man använder uträkningarna nerifrån och upp. SGD:n låter man vara orörd eftersom att den är slutresultatet och den ska skrivas som en linjär kombination av talen a och b. I varje steg byter man ut termer mot tidigare och tidigare uträkningar och förenklar med hjälp av multiplikation och addition eller subtraktion. Till slut när man kommer upp till talen a och b igen är man klar. □

Exempel 3. För att uttrycka 7 som en linjär kombination av 1547 och 560, utför vi följande beräkningar som vi fick från exempel 2.

$$\begin{aligned}
7 &= 28 - 1 \times 21 = 28 - 1(133 - 4 \times 28) \\
&= 5 \times 28 - 1 \times 133 = 5(427 - 3 \times 133) - 1 \times 133 \\
&= 5 \times 427 - 16 \times 133 = 5 \times 427 - 16(560 - 1 \times 427) \\
&= 21 \times 427 - 16 \times 560 = 21(1547 - 2 \times 560) - 16 \times 560 \\
&= 21 \times 1547 - 58 \times 560.
\end{aligned}$$

Euklides algoritm kan också appliceras på de *Gaussiska heltalen*, det vill säga tal på formen $a + bi$, där $a, b \in \mathbb{R}$. Om α, β och γ är tre Gaussiska heltal säger vi att $\alpha \mid \beta$ om $\beta = \alpha\gamma$. $SGD(\alpha, \beta)$ definieras som ett Gaussiskt heltal δ som har det största absolutbelopp som delar både α och β. Men SGD:n är inte unik eftersom att δ kan multipliceras med ± 1 eller $\pm i$ och ändå behålla sitt absolutbelopp och därför fortfarande dela α och β.

Vi illustrerar algoritmen med följande exempel från *Abstrakt Algebra* [S].

1. Om vi har två Gaussiska heltal $\alpha = 3 - 11i$ och $\beta = 7 - i$ börjar vi med att bestämma en kvot κ och en rest ρ genom att dela α med β.

$$\frac{\alpha}{\beta} = \frac{3 - 11i}{7 - i} = \frac{16}{25} - \frac{37}{25}i$$

2. Det heltal som ligger närmast $\frac{16}{25}$ och $\frac{37}{25}$ är 1 respektive 1. Därför sätter vi $\kappa = 1 - i$ och

$$\rho = \alpha - \beta\kappa = (3 - 11i) - (7 - i)(1 - i) = -3 - 3i,$$

   då får vi $\alpha = \beta\kappa + \rho$, där $\kappa = 1 - i$ och $\rho = -3 - 3i$.

3. I nästa steg ser vi att

$$\frac{\beta}{\rho} = \frac{7 - i}{-3 - 3i} = -1 + \frac{4}{3}i,$$

   som ger $\beta = \rho\kappa_1 + \rho_1$, där $\kappa_1 = -1 + i$ och $\rho_1 = 1 - i$.

4. Sedan får vi

$$\frac{\rho}{\rho_1} = \frac{-3 - 3i}{1 - i} = -3i.$$

Här går divisionen jämnt ut, det vill säga $\rho = \rho_1\kappa_2 + \rho_2$, där $\kappa_2 = -3i$ och $\rho_2 = 0$. Då kan vi säga att en SGD till α och β ges av $\rho_1 = 1 - i$. Övriga SGD till α och β ges av $-\rho_1 = -1 + i, i\rho_1 = 1 + i$ och $-i\rho_1 = -1 - i$.

Så då vet vi att vi har en divisionsalgoritm för heltal och Gaussiska heltal. Euklides algoritm kan också användas för att beskriva en hel klass av ringar, enligt en definition från *Abstrakt Algebra* [S]:

Definition 7. Låt D vara ett integritetsområde. En *Euklidisk värdering* på D är en avbildning $\nu : D\backslash\{0\} \to \mathbb{N}$ sådan att det

(i) för alla $a, b \in D$ med $b \neq 0$ finns $q, r, \in D$ så att $a = bq + r$ där $r = 0$ eller $\nu(r) < \nu(b)$

(ii) för alla $a, b \in D\backslash\{0\}$ gäller $\nu(a) \leq \nu(ab)$.

Vi säger att D är en *Euklidisk ring* om det finns en Euklidisk värdering på D.

1.2.5 Kongruenser

Kongruenser och begreppet kvadratisk rest kommer att beskrivas så som det görs i *A Course in Number Theory and Cryptography* [K].

Givet tre heltal a, b och m, säger vi att "a är *kongruent* med b *modulo* m" och vi skriver $a \equiv b \bmod m$, om differensen $a - b$ är delbar med m. m kallas för *modulus* till kongruensen. Följande egenskaper följer ur definitionen:

1. (i) $a \equiv a \bmod m$; (ii) $a \equiv b \bmod m$ om och endast om $b \equiv a \bmod m$; (iii) om $a \equiv b \bmod m$ och $b \equiv c \bmod m$, då är $a \equiv c \bmod m$. För ett bestämt m, betyder (i)-(iii) att kongruens modulo m är en *ekvivalensrelation*.

2. För bestämda m, har varje *ekvivalensklass* med avseende på kongruens modulo m en och endast en representation mellan 0 och $m - 1$. Mängden av ekvivalensklasser, som kallas *restklasser*, betecknas $\mathbb{Z}/m\mathbb{Z}$. Varje mängd av representationer för restklasserna kallas en *fullständig mängd av rester modulo m*.

3. Om $a \equiv b \bmod m$ och $c \equiv d \bmod m$, så gäller $a \pm c \equiv b \pm d \bmod m$ och $ac \equiv bd \bmod m$. Ett annat sätt att säga det är att kongruenser, med samma modulus, kan adderas, subtraheras och multipliceras. Man säger att mängden av ekvivalensklasser $\mathbb{Z}/m\mathbb{Z}$ är en *kommutativ ring*, det vill säga restklasser kan adderas, subtraheras och multipliceras och resultatet beror inte på vilken representation av ekvivalensklassen som användes. Dessa operationer uppfyller de kända axiomen, associativitet, kommutativitet, additativ invers osv.

4. Om $a \equiv b \bmod m$, så gäller $a \equiv b \bmod d$ för vilken delare $d \mid m$ som helst.

5. Om $a \equiv b \bmod m$, $a \equiv b \bmod n$, där m och n är relativt prima, så gäller $a \equiv b \bmod mn$.

Alla heltal modulo ett primtal p kan betecknas som $\mathbb{Z}/p\mathbb{Z}$ och bildar en kropp. $(\mathbb{Z}/n\mathbb{Z})^*$ betecknar en grupp som utgörs av alla tal som har en multiplikativ invers modulo ett tal n.

Definition 8. (Kvadratisk rest) Ett tal a är en kvadratisk rest modulo ett heltal n om det finns ett heltal b som uppfyller $b^2 \equiv a \bmod n$.

Exempel 4. I den gruppen $(\mathbb{Z}/5\mathbb{Z})^*$ är de kvadratiska resterna 1 och 4 eftersom

(i) $1^2 \equiv 1 \bmod 5$

(ii) $2^2 \equiv 4 \bmod 5$

(iii) $3^2 \equiv 4 \bmod 5$

(iv) $4^2 \equiv 1 \bmod 5$

Talen 2 och 3 kallas för icke-rester modulo 5.

1.3 Primtal

Primtal kommer i denna uppsats att definieras som i *Prime numbers: a computational perspective* av Richard Crandall och Carl Pomerance [C].
Ett primtal är ett positivt heltal p som har endast två delare, nämligen 1 och talet p själv. Ett heltal n är sammansatt om $n > 1$ och n är inte ett primtal. (Talet 1 anses varken vara primtal eller sammansatt tal). Det betyder att ett heltal n är sammansatt om och endast om det har en icke-trivial faktorisering (det vill säga att det finns en faktorisering av n förutom $1 \times n$ eftersom att det alltid gäller) $n = ab$, där a, b är heltal, och talen är strikt mellan 1 och n. Primtal kan också "bygga upp" alla andra naturliga tal, något som kallas för aritmetikens fundamentalsats.

Sats 1. (Aritmetikens fundamentalsats) För varje natuligt tal n, finns det en unik faktorisering

$$n = p_1^{a_1} p_2^{a_2} \dots p_k^{a_k}$$

där exponenterna a_i är positiva heltal och $p_1 < p_2 < \dots < p_k$ är primtal.

Bevis hänvisas till *Discrete Mathematics* av Norman L. Biggs [N.B] s.72.

Aritmetikens fundamentalsats är också grunden till det som kan kallas för "aritmetikens fundamental*problem*." Med det menas att givet ett heltal $n > 1$, hitta dess primtalsfaktorisering.

Det finns ett oändligt antal primtal, något som bevisades av Euklides år 300 f.Kr. när han var professor vid det stora universitetet i Alexandria.

Om element i integritetsområden kan beskrivas med faktoriseringar på liknande sätt som primtal får vi vad som kallas en faktoriell ring (eng. UFD, Unique Factorization Domain) som definieras enligt *Abstrakt Algebra* [S].

Definition 9. Ett integritetsområde D kallas för en *faktoriell ring*, om följande två villkor är uppfyllda.

(i) Varje element $a \neq 0$ i D, som inte är en enhet, kan skrivas som en produkt av ändligt många irreducibla element.

(ii) Om $p_1p_2 \dots p_s$ och $q_1q_2 \dots q_t$ är två faktoriseringar av samma element i irreducibla faktorer, så är $s = t$. Om vi omnumrerar, ifall det behövs, q_j:na gäller också att p_i och q_i är associerade element för alla i, d.v.s. $p_i = u_iq_i$ för någon enhet u_i.

Exempel 5. Mängden av alla heltal $\mathbb{Z}$ är en UFD, eftersom att varje tal som inte är en enhet och är skilt från noll, d.v.s. alla tal utom -1, 0 och 1, kan skrivas som en produkt av ±primtal.

Följande två definitioner kommer från *A Course in Number Theory and Cryptography* [K].

Definition 10. (Legendre-symbolen)
Legendre-symbolen betecknas som $(\frac{a}{p})$, där a är ett heltal och p ett prmtal > 2. Legendre-symbolen är ett sätt att visa om ett tal a är en kvadratisk rest modulo p enligt:

$$(\frac{a}{p}) = \begin{cases} 0 & \text{om } p \mid a \\ 1 & \text{om a är en kvadratisk rest modulo p} \\ -1 & \text{om a är en icke-rest modulo p} \end{cases}$$

Definition 11. (Jacobi-symbolen)
Jacobi-symbolen betecknas på samma sätt som Legendre-symbolen $(\frac{a}{n})$, där a är ett heltal och n i detta fall är vilket udda heltal som helst. Om vi låter $n = p_1^{e_1} \cdots p_r^{e_r}$ vara faktoriseringen av n definieras Jacobi-symbolen som produkten av Legendre-symboler för primtalsfaktorerna av n:

$$(\frac{a}{n}) = (\frac{a}{p_1})^{e_1} \cdots (\frac{a}{p_r})^{e_r}$$

Dock så är det inte alltid så att $(\frac{a}{n}) = 1$ betyder att a är en kvadratisk rest modulo n. Till exempel $(\frac{3}{35}) = (\frac{3}{5})(\frac{3}{7}) = (-1)(-1) = 1$ men 3 är inte en kvadratisk rest modulo 35.

Vissa primtal, p, kan skrivas på en speciell form, det vill säga att de kan beskrivas av ett algebraiskt samband. Här följer två exempel från *Prime numbers: a computational perspective* [C].

Exempel 6 (**Mersenne-primtal**)**.** Mersenne-talen beskrivs på formen

$$M_q = 2^q - 1$$

varav några av dessa bildar primtal.

Sats 2. Om $M_q = 2^q - 1$ är ett primtal, då är q ett primtal.

Bevis. Om $2^c - 1$, med c sammansatt, $c = ab$, kan man skriva det som $2^{ab} - 1$. Men $2^{ab}-1 = (2^a-1)(1+2^a+2^{2a}+\cdots+2^{(b-1)a})$ som betyder att $2^a-1 \mid 2^{ab}-1$ och $2^{ab} - 1 = 2^c - 1$ är inte ett primtal. □

Det betyder att om man letar efter Mersenne-primtal kan man begränsa sig till primtalsexponenter q. Viktigt att notera är att det omvända för satsen inte gäller. Till exempel $2^{11} - 1$ är inte ett primtal även om 11 är det.

Mersenne-primtalen förekommer även i det antika ämnet om de så kallade perfekta talen. Ett perfekt tal är ett positivt heltal som är lika med summan

av sina delare förutom talet själv. Till exempel $28 = 1 + 2 + 4 + 7 + 14$ är ett perfekt tal. Ett annat sätt att definiera "perfektion" är att låta $\sigma(n)$ beteckna summan av delarna till n, där n är perfekt om och endast om $\sigma(n) = 2n$, eftersom att summan alla delare utom n är lika med n och $n + n = 2n$. Följande sats beskriver de jämna perfekta talen.

Sats 3. (Euklides-Euler). Ett jämnt tal n är perfekt om och endast om det är på formen

$$n = 2^{q-1}M_q$$

där $M_q = 2^q - 1$ är ett primtal.

Bevis. Antag att $n = 2^a m$ är ett jämnt tal, där m är den största udda delaren till n. Delarna till n är på formen $2^j d$, där $0 \leq j \leq a$ och $d \mid m$. Summan av delarna blir alltså $2^0 \times d_0 + 2^0 \times d_1 \cdots + 2^0 \times m + 2^1 \times d_0 + \cdots + 2^a \times m = 2^0(d_0 + \cdots + m) + 2^1(d_0 + \cdots + m) + \cdots + 2^a(d_0 + \cdots + m) = (2^0 + 2^1 + \cdots + 2^a) \times (d_0 + d_1 + \cdots + m)$. Låt D vara summan av delarna till m exklusive m, och låt $M = 2^0 + 2^1 + \cdots + 2^a = 2^{a+1} - 1$, där det sista likhetstecknet följer av formeln för geometrisk summa. Summan av delarna till $n = 2^a m$ kan då skrivas som $M(D + m)$. Om M är ett primtal och $M = m$, då är $D = 1$ och summan av alla sådana delare till n är $M(1+m) = (2^{a+1}-1)(1+2^{a+1}-1) = (2^{a+1}-1)(2^{a+1}) = (2^{a+1}-1) \times 2^a \times 2 = 2n$. Så n är perfekt.
För att bevisa den andra halvan av satsen antar vi att $n = 2^a m$ är perfekt. Då är $M(D + m) = MD + Mm = 2n = 2^{a+1}m = (M + 1)m = Mm + m$. Subtraherar man Mm från ekvationen ser vi att

$$m = MD$$

Om $D > 1$ skule det innebära att D och 1 skulle vara delare till m mindre än m. Det motsäger definitionen av D som sa att D skulle vara en summa av delare och inte själv en delare. D måste alltså vara 1 som betyder att m är ett primtal och $m = M = 2^{a+1} - 1$. □

Exempel 7 (**Fermat-primtal**)**.** Fermat-talen beskrivs genom formen

$$F_n = 2^{2^n} + 1$$

där några av dessa ger primtal.

1637 påstod Fermat att talen F_n alltid är primtal. Det är sant för de första fem Fermat-talen, upp till och med $F_4 = 65537$ är primtal.

En intressant sats, som inte tas upp här, om Fermat-primtal bevisades av Gauss när han var i tonåren. Han visade att en regelbunden polygon med n sidor är konstruerbar med en passare och linjal om och endast om den största udda delaren till n är en produkt av distinkta Fermat-primtal. Om $F_0, \ldots, F_4$visar sig vara de *enda* Fermat-primtalen, då är de enda n-hörningarna som är konstruerbara de med $n = 2^a m$ med $m \mid 2^{32} - 1$ (eftersom att produkten av dessa fem Fermat-primtal är $2^{32} - 1$). Om m delar $2^{32} - 1 = F_0 \times F_1 \times F_2 \times F_3 \times F_4$ måste m dela något, eller några, eller alla primtalen, som ger att m är en produkt av Fermat-primtal.

Kapitel 2

Primtal och kryptografi

2.1 Historisk översikt av enkla kryptosystem

Två tidiga exempel på kryptosystem förklaras i *A Course in Number Theory and Cryptography* [K].

Det första vi ska ta upp är ett av de första historiska exemplen på kryptering, det klassiska förskjutnings-chiffret. Det fungerar genom att man räknar med modulär aritmetik och kan generaliseras på följande vis:

Antag att vi har ett alfabet med N bokstäver och deras numeriska motsvarigheter $0, 1, \ldots, N-1$. Låt b vara en konstant. Med en *förskjutning* menar vi krypteringsfunktionen f som definieras av formeln $C = f(P) \equiv P + b \bmod N$. För att dechiffera en sekvens C $\in \{0, 1, \ldots, N-1\}$, räknar man helt enkelt $P = f^{-1}(C) \equiv C - b \bmod N$ Detta chiffer användes i forntida Rom av Julius Caesar, och det sägs att det var han som uppfann det. Ett annat namn för detta chiffer är just Caesar-chiffer. Om vi i stället går över till det svenska språket kan ett exempel se ut så här.

Exempel 8. Antag att vi använder det svenska alfabetet A-Ö, med 29 bokstäver (W inräknat) och tilldelar varje bokstav ett tal mellan 0 och 28. I chiffret vill vi förskjuta bokstäverna med fem steg, så $b = 5$. Låt P $\in \{0, 1, ...28\}$ stå för varje bokstav som vi vill kryptera. Definiera en funktion f på mängden {0,1,...,28} till sig själv genom

$$f(P) = \begin{cases} P + 5 & \text{om } x < 24 \\ P - 24 & \text{om } x \geq 24 \end{cases}$$

Anledningen till att man drar bort 24 i andra raden är att om $x \geq 24$ börjar det om igen. Exempelvis Å har numerisk motsvarighet 26. Adderar man 5 till 26 modulo 29 blir det 2 då det blir $27, 28, 0, 1, 2$ vilket är samma sak som att subtrahera 24 från 26. Med ett lite enklare uttryck, f adderar 5 modulo 29:

$$C = f(P) = P + 5 \bmod 29$$

För att till exempel kryptera ordet "HÅL" översätter vi först bokstäverna till deras motsvarande nummer 7 26 11. Sedan adderar vi 5 modulo 29 och får 12 2 16 som vi sedan konverterar till deras motsvarande bokstäver "MCQ". För att dechiffrera en text subtraherar man 5 modulo 29. Till exempel chiffertexten "GOEWS" blir klartexten "BJÖRN".

Förskjutningschiffer är i allmänhet för enkelt för att användas. Säg att vi har chiffertexten "JVTLOLYL" och vi vet att det chiffrerades med ett förskjutningschiffer med det engelska alfabetet A-Z, med 26 bokstäver och W inkluderat som vanligt. Det enda vi behöver göra är att hitta b. Ett sätt att göra det är genom så kallad *frekvensanalys*. Antag att vi redan fått tag i en lång sekvens av chiffertext, säg flera hundra bokstäver. Om vi utgår från det engelska alfabetet och engelska ord vet vi att "E" är den oftast förekommande bokstaven. Då är det rimligt att anta att den mest frekventa bokstaven i chiffertexten är krypteringen av E. Ska man använda frekvensanalys behöver man först ha en lång sträng med chiffertext, för en kort text kan avvika från statistiken. Antag att vi ser att "L" är den mest förekommande bokstaven i chiffertexten. Det betyder att förskjutningen tar "E" $= 4$ till "L" $= 11$, det vill säga, $11 \equiv 4 + b \bmod 26$ så att $b = 7$. För att nu dechiffrera meddelandet behöver vi bara subtrahera 7 (modulo 26) från de numeriska motsvarigheterna till "JVTLOLYL":

$$\text{“}JVTLOLYL\text{”} = 9\ 21\ 19\ 11\ 14\ 11\ 24\ 11 \mapsto 2\ 14\ 12\ 4\ 7\ 4\ 17\ 4 = \text{“}COMEHERE\text{”}$$

I ett fall med förskjutningschiffer av enstaka boksäver i ett alfabet med 26 bokstäver behövs inte ens en lång sekvens med chiffertext för att hitta den mest frekventa bokstaven. Det finns trots allt bara 26 möjligheter för b och man kan helt enkelt pröva dem alla. En av dem kommer troligtvis att ge ett meddelande som är begripligt och det värdet på b är nyckeln.

Ett annat kryptosystem, som uppfanns på 1500-talet, var "Vigenre-chiffret" som var populärt under flera hundra år. Det kan beskrivas som följande. För något bestämt k, betrakta en sträng med k bokstäver som en vektor i $(\mathbb{Z}/N\mathbb{Z})^k$. Välj någon bestämd vektor $b \in (\mathbb{Z}/N\mathbb{Z})^k$($b$ var vanligtvis en vektor som motsvarade något enkelt "kodord"), och kryptera med vektor-transformationen $C = P + b$ (där chiffertexten C och klartexten P delas in i meddelande-enheter med k antal heltal modulo N).

Exempel 9. Säg att vi använder det svenska alfabetet med 29 bokstäver igen. Vårt kodord har vi valt till "RÄV". Vi vill nu kryptera meddelandet "NÄR MAN HAR ELIMINERAT DET OMÖJLIGA MÅSTE DET SOM FINNS KVAR VARA SANNINGEN". Kodordet omvandlar vi nu till en vektor med de tre numeriska motsvarigheterna *"RÄV"*= 17 27 21. Sedan delar vi in klartextmeddelandet P i grupper om tre och krypterar enligt $C = P + (17\ 27\ 21)$ modulo 29 enligt vektoraddition. Den första tripletten skulle då alltså bli *"NÄR"*$= 13\ 27\ 17 \mapsto 1\ 25\ 17 =$ *"BZJ"*. Hela chiffertexten skulle bli "BZJAÄFÄJVJAAGFVPVHBZHMEQHDZEVAYKHCYVRKCKÅZLFGINRPNRPVGÄFBGFXCF"

Detta system är dessvärre nästan lika lätt att knäcka som enbokstavs-transformation. Nämligen, om man vet (eller kan gissa) N och k, delar man helt enkelt in chiffertexten i strängar med k bokstäver och utför frekvensanalys på den första bokstaven i varje sträng för att bestämma första komponenten i b, sedan samma sak för den andra bokstaven i varje sträng, och så vidare.

En mer historisk bakgrund till kryptorafin vi har idag beskrivs i *Algebraic Aspects of Cryptography*[N].
Fram till sent 1970-tal, var all kryptografisk meddelande-sändning av vad man kan kalla *privat nyckel*. Det betyder att någon som har tillräcklig information

för att kryptera meddelanden automatiskt har tillräcklig information för att dekryptera meddelanden. Som ett resultat av det var två användare av kryptosystemet, som ville kommunicera i hemlighet, tvungna att byta nycklar på ett säkert sätt. Exempelvis genom en betrodd budbärare.

Kryptografins utseende ändrades dramatiskt när Diffie och Hellman uppfann ett helt ny typ av kryptografi, kallat *offentlig nyckel*. Kärnan i detta koncept är idén att använda en enkelriktad funktion för kryptering.

Definition 12. Informellt säger vi att en en-till-en funktion $f : X \to Y$ är "enkelriktad" (eng."one-way" function) om det är lätt att beräkna $f(x)$ för vilket $x \in X$ som helst men svårt att beräkna $f^{-1}(y)$ för de flesta y i bilden av (range of) f.

Funktionerna som används mest för kryptering tillhör en speciell klass av enkelriktade funktioner som fortsätter att vara enkelriktade endast om viss information ("dekrypteringsnyckeln") hålls hemlig. Informellt kan vi definiera en *offentlig nyckel-krypteringsfunktion* (även kallad "fallucke"-funktion) som en avbildning från klartextmeddelande-enheter till chiffertextmeddelande-enheter som enkelt kan beräknas av någon som har den så kallade "offentliga" nyckeln men vars omvända funktion, som dechiffrerar chiffertextmeddelande-enheterna, inte kan beräknas inom rimlig tid utan någon ytterligare information (den "hemliga" nyckeln).

Detta innebär att vem som helst kan skicka ett meddelande till en given användare genom att anända samma chiffreringsnyckel, som de helt enkelt hittar i ett offentligt register. Det finns inget behov för avsändaren att göra någon hemlig överenskommelse med mottagaren.

Det var uppfinnandet av kryptosystemen med offentlig nyckel som ledde till en dramatisk ökning av algebrans och talteorins roll i kryptografin. Anledningen är att denna typ av matematik verkar vara den bästa källan för enkelriktade funktioner.

En intressant historisk fråga är varför offentlig nyckel-kryptografi inte blev uppfunnet förrän 1976. Ingenting i idén med offentlig nyckel-kryptografi eller tidiga offentlig nyckel-kryptosystem krävde 1900-talets matematik. Det första offentlig nyckel-kryptosystemet som anändes i den verkliga världen − RSA-systemet − använder talteori som var välbekant for Euler. Varför hade inte han tänkt på att uppfinna RSA?

En möjlig anledning till den sena utvecklingen av konceptet med offentlig nyckel är att fram till 1970-talet var kryptografi mest använt för militära och diplomatiska syften, och där passade hemlig nyckel-kryptografi väldigt bra. Men med den ökande datoriseringen av ekonomin, växte nya behov för kryptografi fram. För att ta ett vardagligt exempel, när stora summor pengar skickas elektroniskt måste man kunna förhindra tjuvar som arbetar på kontoret och nyfikna hackare, eller konkurrenter, från att övervaka vad andra gör med sina pengar. Ett annat exempel är det relativt nya (år 1998) användningen för kryptografi när man ska skydda sekretessen för data (medicinska journaler, kredit värdighet, osv.). Till skillnad från den militära eller diplomatiska situationen − med strikta hierarkier, en långvarig lista med auktoriserade användare, och system av budbärare − stöter man på en mycket större och mer rörlig struktur av kryptografi-användare inom affärstransaktioner och dataintegritet. Därför var

kanske inte offentlig nyckel-kryptografi uppfunnet tidigare eftersom att det helt enkelt inte fanns något behov av det fram tills nyligen.

En annan anledning till varför det inte var troligt att RSA skulle upptäckas under Eulers tid är att då behövde alla beräkningar göras för hand. För att få en acceptabel nivå av säkerhet vid användning av RSA skulle det vara nödvändigt att arbeta med ganska stora heltal där beräkningarna hade blivit ohanterliga.

I praktiken är det stora värdet av offentlig nyckel-kryptografi tätt sammankopplat med den stora spridningen av kraftfull datorteknik.

2.2 Primalitetstest

Ett *primalitetstest* förklaras i *A Course in Number Theory and Cryptography*[K] som ett kriterium som avgör om ett tal n *inte* är ett primtal. Om n "klarar av" ett primalitetstest, så är talet *möjligen* ett primtal. Om n klarar av flera primalitetstest, är det mycket troligt att det är ett primtal. Å andra sidan, om n inte klarar av testet är det definitivt ett sammansatt tal.

2.2.1 Regler för primtal och pseudoprimtal

Pseudoprimtalen beskrivs som i *Prime numbers: a computational perspective* [C].
Antag att vi har en sats, *"Om n är ett primtal, då är S sant för n"*, där *"S"* är ett aritmetiskt påstående som är enkelt att undersöka. Om vi får ett stort tal n, och vi vill bestämma om det är ett primtal eller ett sammansatt tal, kan vi mycket väl testa det aritmetiska påståendet S och se om det verkligen håller för n. Om påståendet inte stämmer har vi visat att n är sammansatt. Om påståendet å andra sidan stämmer är det möjligt att n är ett primtal, men det är också möjligt att n är sammansatt. Så vi inför beteckningen S-pseudoprimtal, som är ett sammansatt heltal för vilket S gäller.

Sats 4 (**Fermats lilla sats**)**.** Om n är ett primtal, så gäller för varje heltal b att

$$b^n \equiv b \pmod{n}.$$

Om n är ett primtal så gäller för varje b sådant att $SGD(b, n) = 1$ att man har

$$b^{n-1} \equiv 1 \bmod n.$$

Om n *inte* är ett primtal är det fortfarande möjligt att kongruensen gäller, men det är inte speciellt troligt.

Bevis hänvisas till *A Course in Number Theory and Cryptography* [K] s.19. Där beskrivs också den första typ av pseudoprimtal som följer här.

Definition 13. Om n är ett udda sammansatt tal och b är ett heltal sådant att $SGD(n, b) = 1$ och Fermats lilla sats håller, kallas n för ett **Fermat***pseudoprimtal till basen b.*

Med andra ord, ett "**Fermat**pseudoprimtal" är ett tal som "låtsas" att vara ett primtal genom att klara Fermats lilla sats.

(Carmichael-tal): Om vi vill hitta en enkel och snabb metod för att avgöra

vilka tal som är primtal och vilka som är sammansatta, kan vi överväga att kombinera Fermats test för olika baser b. Till exempel, fast 341 är ett pseudoprimtal till basen 2, är det inte ett pseudoprimtal till basen 3. Talet 91 är ett pseudoprimtal till basen 3 men inte till basen 2.
Emellertid finns det tal som inte är så snälla. Talet $561 = 3 \times 11 \times 17$ är inte bara ett Fermat-pseudoprimtal till både basen 2 och 3, det är ett pseudoprimtal till alla baser b. Det kan verka förvirrande att sådana tal existerar men det gör de. De upptäcktes först av R. Carmichael år 1910, och det är efter honom vi har döpt dem.

Definition 14. Ett sammansatt heltal n, för vilket $a^n \equiv a \pmod{n}$ för varje heltal a, är ett Carmichael-tal.

Sats 5 (Korselts kriterium)**.** Ett heltal n är ett Carmichael-tal om och endast om n är positivt, sammansatt, är kvadratfritt, och för varje primtal p som delar n har vi att $p-1$ delar $n-1$.

Bevis. Antag först att n är ett Carmichael-tal. Då är n sammansatt. Låt p vara en primtalsfaktor till n. Vi har $p^n \equiv p \pmod{n}$, av att n är ett Carmichael-tal, och med hjälp av det kan vi bevisa att n är kvadratfritt. Vi antar motsatsen, att n inte är kvadratfritt. Då finns det ett tal, kalla det k, så att $k^2 \mid n$. Vi har fortfarande att $k^n \equiv k \bmod n$ vilket implicerar att $n \mid k^n - k$. Vi har då $k^2 \mid k^n$, eftersom att n är ett Carmichael-tal, sammansatt och därför större än 2, och $k^2 \mid n$, så $k^2 \mid k$. Men det är omöjligt och vi har visat att n är kvadratritt. Vi ska också visa att $p-1$ delar $n-1$. Låt a vara en primitiv rot modulo p. Eftersom att $a^n \equiv a \pmod{n}$, har vi $a^n \equiv a \bmod p$ av $(a^n - a) = s \times n$, där s är ett heltal, men $n = p \times g$ (g är ett heltal) vilket ger att $(a^n - a) = g \times s \times p$ som ger just att $a^n \equiv a \bmod p$ eftersom att $g \times s$ också ar ett heltal. Nu ser vi att $a^{n-1} \equiv 1 \bmod p$ men då ordningen av $a \bmod p$ är $(p-1)$ måste $(n-1)$ vara en multipel av $(p-1)$. Det betyder att $(p-1)$ delar $(n-1)$.
Omvänt ska vi anta att n är sammansatt, kvadratfritt, att för varje primtal p som delar n har vi att $p-1$ delar $n-1$ och bevisa att $a^n \equiv a \pmod{n}$ för varje heltal a som betyder att n är ett Carmichael-tal. Eftersom att n är kvadratfritt, vilket betyder att samma primtal inte förekommer mer än en gång i faktoriseringen av n, räcker det med att visa att $a^n \equiv a \pmod{p}$ för varje heltal a och för varje primtal p som delar n. Så antag att $p \mid n$ och att a är ett heltal. Om a och p är relativt prima, $SGD(a,p) = 1$, har vi att $a^{p-1} \equiv 1 \pmod{p}$ av Fermats lilla sats. Eftersom $p-1$ delar $n-1$, har vi $a^{n-1} \equiv 1 \pmod{p}$. Alltså $a^n \equiv a \pmod{p}$. Men denna kongruens håller även när a är delbart med p eftersom att den alltid gäller för primtal. Så den håller för alla a och n är ett Carmichael-tal. □

De två följande primalitetstesten är från *A Course in Number Theory and Cryptography* [K].

2.2.2 Solovay-Strassens primalitetstest

Antag att n är ett positivt udda heltal, och vi vill veta om det är ett primtal eller ett sammansatt tal. Välj k stycken heltal $0 < b < n$, där b är slumpmässigt. För varje b, räkna först ut bägge sidor av

$$b^{(n-1)/2} \equiv (\frac{b}{n}) \bmod n$$

där $(\frac{b}{n})$ står för Jacobi-symbolen. Om bägge sidor inte är kongruenta modulo n, då vet man att n är sammansatt, och testet är klart. Om det stämmer, gå vidare till nästa b. Om ekvationen håller för alla k slumpmässiga val av b är sannolikheten att n är ett sammansatt tal trots att det klarat av alla test som mest $1/2^k$. Därför är Solvay-Strassen testet en så kallad probabilistisk algoritm som leder till slutsatsen att n antingen är sammansatt eller "möjligen" ett primtal.

2.2.3 Miller-Rabbin primalitetstest

Definition 15. Låt n vara ett udda sammansatt tal och skriv $n-1 = 2^s t$ med t udda. Låt $b \in (\mathbb{Z}/n\mathbb{Z})^*$. Om n och b tillfredsställer något av villkoren

(i) $b^t \equiv 1 \bmod n$, eller

(ii) det finns $r, 0 \leq r < s$, så att $b^{2^r t} \equiv -1 \bmod n$ **(1)**

så kallas n för ett *starkt pseudoprimtal till basen b.*

Antag att vi vill bestämma om ett stort udda positivt heltal n är ett primtal eller ett sammansatt tal. Vi skriver $n-1 = 2^s t$ med t udda, och väljer ett slumpmässigt heltal $b, 0 < b < n$. Först räknar vi ut $b^t \bmod n$. Om vi får ± 1, drar vi slutsatsen att n klarar av testet (1) för vårt valda b och vi går vidare till ett annat slumpmässigt b. Annars tar vi kvadraten på $b^t \bmod n$, och sedan kvadrerar det modulo n, och så vidare till dess vi får kongruens med -1. Om vi får -1, klarar n testet. Däremot, om vi aldrig kommer fram till -1, det vill säga om vi får $b^{2^{r+1}} \equiv 1 \bmod n$ medan $b^{2^r} \not\equiv -1 \bmod n$ klarar n inte testet och vi vet att n är sammansatt. Om n klarar testet (1) för alla slumpmässiga val av b − antag att vi testar k olika baser b − vet vi att n har som störst 1 av 4^k sannolikhet att vara sammansatt. Det är för att om n är sammansatt är det som mest $1/4$ av baserna $0 < b < n$ som tillfredsställer (1).
Detta test kan också förklaras så här. Vi har $b^{n-1} \equiv 1 \bmod n$ av Fermats lilla sats som i sin tur ger att $b^{n-1} - 1 \equiv 0 \bmod n$ när n är ett primtal. Men $b^{n-1} - 1 = b^{2^s t} - 1$ av $n - 1 = 2^s t$. $b^{2^s t} - 1$ kan faktoriseras som $(b^{2^{s-1}t})^2 - 1 = (b^{2^{s-1}t} - 1)(b^{2^{s-1}t} + 1)$ av konjugatregeln. Detta kan vi utveckla som $((b^{2^{s-2}t})^2 - 1)(b^{2^{s-1}t} + 1) = (b^{2^{s-2}t} - 1)(b^{2^{s-2}t} + 1)(b^{2^{s-1}t} + 1) = \cdots = (b^t - 1)(b^t + 1)(b^{2t} + 1)(b^{4t} + 1) \ldots (b^{2^{s-1}t} + 1)$.
Om n är ett primtal ska faktoriseringen vara kongruent med 0. Det betyder att någon av faktorerna ovan måste vara 0 och då får vi just de kriterierna som är Miller-Rabin testet.

2.3 Komplexitet

För att beskriva hur komplex en algoritm, eller uträkning är brukar man använda "stora-O notation"(eng. "big-O notation"). Vi ska beskriva den som i *A Course in Number Theory and Cryptography* [K]. Antag att $f(n)$ och $g(n)$ är funktioner av det positiva heltalet n som tar *postitiva-* (men inte nödvändigtvis heltals-) värden för alla n. Vi säger att $f(n) = O(g(n))$ (eller helt enkelt $f = O(g)$) om det finns en konstant C så att $f(n)$ alltid är mindre än $C \times g(n)$. Till exempel, $2n^2 + 3n - 3 = O(n^2)$ (eftersom att det inte är svårt att bevisa att den vänstra sidan alltid är mindre än $3n^2$).

Eftersom att vi vill använda stora-O notationen i mer allmänna situationer, ska vi ge den en mer omfattande definition. Nämligen, vi ska låta f och g vara funktioner av flera variabler, och vi ska inte bry oss om relatoinen mellan f och g för små n. Precis som i studierna av gränserna då $n \to \infty$ i den matematiska analysen, ska vi även här bara bry oss om stora värden på n.

Definition 16. Låt $f(n_1, n_2, \ldots, n_r)$ och $g(n_1, n_2, \ldots, n_r)$ vara två funktioner vars definitionsmängd är alla r-tuppler av positiva heltal. Antag att den existerar konstanter B och C så att när alla n_j är större än B är de två funktionerna definierade och positiva och $f(n_1, n_2, \ldots, n_r) < Cg(n_1, n_2, \ldots, n_r)$. I det fallet säger vi att f är *begränsad* av g och vi skriver $f = O(g)$.

Notera att "=" i notationen $f = O(g)$ ska tolkas mer som ett "<" och att stora-O ska tolkas som "någon konstant multiplicerat".

Stora-O notationen kan även användas för att beskriva hur lång tid en beräkning tar, som de förklarar i *Prime numbers: a computational perspective* [C]. Ekvivalensen bygger, som man kan gissa, på att den fysiska tid det tar för en dator att utföra en beräkning är proportionell mot det totala antalet relevanta bit-operationer.

Definition 17. En algoritm som utför en beräkning som innefattar heltal $n_1, n_2, \ldots, n_r$ med respektive $k_1, k_2, \ldots, k_r$ bitar kallas för en *polynomisk tid-algoritm (eng. polynomial time algorithm)*, eller **P**-algoritm, om det finns heltal $d_1, d_2, \ldots, d_r$ så att antalet bit-operationer som är nödvändiga för att utföra algoritmen är $O(k_1^{d_1} k_2^{d_2} \cdots k_r^{d_r})$.

Sista förklaringen och definitionen kommer från *A Course in Number Theory and Cryptography* [K].

Exempel på polynomisk tid-algoritmer är de vanliga aritmetiska operationerna $+, -, \times, \div$, och även omvandling från en bas till en annan. Å andra sidan, beräkningen av $n!$ är inte det.

Definition 18. Med en bit-operation menar vi operationen där vi adderar två bitar med varandra enligt vissa regler, vars närmare beskrivning finns i *A Course in Number Theory and Cryptography* [K] på sida 3. Adderar vi två bitar med varandra krävs det en bit-operation.

2.4 RSA

En krypteringsmetod som bygger på användning av primtal är RSA-kryptosystemet, som är uppkallat efter sina upphovsmän Rivest, Shamir och Adleman. Säkerheten med RSA bygger på att för ett tal $n = pq$ är det svårt att med enbart information om n veta vilka tal p och q är, det vill säga det är svårt att faktorisera n om man väljer p och q på ett bra sätt. Algoritmerna för RSA som visas följer i stort det som finns i *Prime Numbers: a computational perspective* [C].

2.4.1 Algoritmer

1. I denna algoritm genererar vi en persons privata och tillhörande offentliga nyckel för RSA-kryptosystemet.

1. **Väljer primtal**
   Väljer två olika primtal efter rådande säkerhetskriterier.

2. **Genererar offentlig nyckel**
   $n = pq$
   $\varphi(n) = (p-1)(q-1)$
   Väljer slumpmässigt $e \in [1, \varphi(n)]$ relativt prima med $\varphi(n)$.
   Redovisar offentlig nyckel som (n, e).

3. **Genererar privat nyckel**
   $d = e^{-1} \bmod \varphi$
   Redovisar privat nyckel som d.

2. Vi antar att Alice har en privat nyckel d_A och en offentlig nyckel (n_A, e_A) från den förra algoritmen. Här visar vi hur en annan person, Bob, kan kryptera ett meddelande x (tänkt som ett heltal i $[0, n_A)$) till Alice, och hur Alice kan dekryptera meddelandet.

1. **Bob krypterar**
   $y = x^{e_A} \bmod n_A$;
   Bob skickar y till Alice.

2. **Alice dekrypterar**
   Alice får det krypterade meddelandet y;
   $x = y^{d_A} \bmod n_A$.

För att denna algoritm ska fungera måste vi ha

$$x^{de} \equiv x \pmod{n}.$$

Detta följer från konstruktionen av d så att $de = 1 + k\varphi$ och denna proposition från *A Course in Number Theory and Cryptography* [K].

Proposition 2. *Om $SGD(a, m) = 1$, så gäller $a^{\varphi(m)} \equiv 1 \bmod m$*

Bevis. Låt $M = \{x_0, x_1, x_2, \ldots, x_k\}$ vara mängden av alla enheter, det vill säga alla tal i $\mathbb{Z}/m\mathbb{Z}$ som har en multiplikativ invers modulo m, där $k = \varphi(m)$. ($\varphi(m)$ ger just antalet enheter modulo m). Mängden $aM = \{ax_0, ax_1, ax_2, \ldots, ax_k\}$ kommer också att utgöra en mängd av alla enheter modulo m. För att visa det kan vi tänka så här. Om ax_i och ax_j tillhör samma restklass gäller att $ax_i \equiv ax_j \bmod m$. Det betyder också att $m \mid (x_i - x_j)a$ och eftersom att $m \nmid a$ måste $m|(x_i - x_j)$. Men x_i och x_j är mindre än m, så m kan inte dela $(x_i - x_j)$. Såvida inte $x_i = x_j$.
Då har vi att produkten av alla element i M är kongruent med produkten av alla element i aM modulo m

$$(x_0 \times x_1 \times \cdots \times x_k) \equiv (ax_0 \times ax_1 \times \cdots \times ax_k) \bmod m.$$

Men eftersom att vi kan faktorisera ut $k = \varphi(m)$ stycken a ur det högra ledet, kan kongruensen skrivas som

$$(x_0 \times x_1 \times \cdots \times x_k) \equiv (a^{\varphi(m)})(x_0 \times x_1 \times \cdots \times x_k) \bmod m.$$

Då $x_0, x_1, \ldots, x_k$ är enheter har alla en invers modulo m, vilket betyder att det finns en invers till $(x_0 \times x_1 \times \cdots \times x_k) \bmod m$, nämligen $(x_k^{-1} \times \cdots x_1^{-1} \times \times x_0^{-1}) \bmod m$. Multiplicerar vi kongruensen med den inversen får vi

$$1 \equiv a^{\varphi(m)} \bmod m$$

eller

$$a^{\varphi(m)} \equiv 1 \bmod m.$$

□

Ovanstående bevis kommer från *Discrete Mathematics* [N.B]. Det finns ett annat bevis i *A Course in Number Theory and Cryptography* [K], s.21 som bygger på induktion men det lämnar vi åt läsaren att undersöka.

Detta tillsammans ger $x^{de} = x^{1+k\varphi} = x(x^{\varphi})^k \equiv x \times 1^k \equiv x \pmod{n}$, när $SGD(x, n) = 1$.

Några kommentarer till denna algoritm kommer från *A Course in Number Theory and Cryptography* [K].

Anmärkning 1. : När vi säger "slumpmässigt"menar vi att talet valdes med hjälp av en slumptalsgenerator (eller "pseudo-slump"talsgenerator), d.v.s ett datorprogram som genererar ett följd av siffror på ett sätt som ingen kan kopiera eller förutse, och som sannolikt har alla statistiska kriterier för en verkligt slumpmässig följd. I RSA-kryptosystemet behöver vi en slumptalsgenerator inte bara för att välja e utan även för att välja de stora primtalen p och q (så att ingen kan lista ut våra val genom att titta på listor över speciella primtal, exempelvis Marsenne-primtal eller faktorer av $b^k \pm 1$ för små b och relativt små k). Vad betyder ett "slumpmässigt valt" primtal? Jo, välj först ett stort slumpmässigt heltal m. Om m är jämnt, ersätt m med $m + 1$. Utför sedan lämpligt *primalitetstest* för att se om det udda talet är ett primtal. Om m inte är ett primtal, pröva med $m + 2$, sedan $m + 4$, och så vidare tills dess att du hittar det första primtalet $\geq m$, som är det tal du väljer som ditt "slumpmässiga" primtal.

På liknande sätt kan det slumpmässiga talet e relativt prima till $\varphi(n)$ väljas genom att först generera ett slumpässigt (udda) tal med ett lämpligt antal bitar och sedan succesivt öka talet till man hittar ett e med $SGD(e, \varphi(n)) = 1$. (Alternativt, man kan utföra primalitetstest tills man hittar ett primtal e, säg mellan $max(p, q)$ och $\varphi(n)$; ett sådant primtal kommer att tillfredsställa $SGD(e, \varphi(n)) = 1$, eftersom att e är ett primtal.)

Chiffreringstransformationen är avbildningen från $\mathbb{Z}/n_a\mathbb{Z}$ till sig själv givet av $f(P) \equiv P^{e_A} \bmod n_A$. Dechiffreringstransformationen är avbildningen från $\mathbb{Z}/n_A\mathbb{Z}$ till sig själv givet av $f^{-1}(C) \equiv C^{d_A} \bmod n_A$. Vi kan se att de två avbildningarna är inverser till varandra på grund av vårt val av d_A. Nämligen, att utföra f följt av f^{-1} eller f^{-1} följt av f innebär att höja upp med exponenten $d_A e_A$. Men, eftersom $d_A e_A$ ger resten 1 vid division med $\varphi(n_A)$, är det samma sak som att höja upp med exponenten 1 som vi såg tidigare.

Från beskrivningen i den senaste paragrafen ser det ut som att vi arbetar med mängder $P = C$ av klartext-, och chiffertextmeddelande-enheter som varierar från en användare till en annan. I praktiken skulle vi förmodligen

vilja välja enhetliga värden på P och C för hela systemet. Till exempel, antag att vi arbetar med ett N-bokstavsalfabet. Låt sedan $k < l$ vara lämpligt valda positiva heltal så att, exempelvis, N^k och N^l har ungefär 200 decimalsiffror (eng. decimal digits). Vi tar som våra klartextmeddelande-enheter alla strängar med k bokstäver, som vi ser som k-vektor i bas N, det vill säga vi ger dem numeriska motsvarigheter mellan 0 och N^k. Vi tar på liknande sätt chifertextmeddelande-enheterna att vara strängar med l bokstäver i vårt alfabet med N bokstäver. Sedan måste varje användare välja hans/hennes stora primtal p_A och q_A så att $n_A = p_A q_A$ tillfredsställer $N^k < n_A < N^l$. Sedan kommer varje klartextmeddelande-enhet, det vill säga heltal mindre än N^k, att motsvara ett element i $\mathbb{Z}/n_A\mathbb{Z}$ (för vilken användare n_A som helst) och, eftersom $n_A < N^l$, kan bilden $f(P) \equiv \mathbb{Z}/n_A\mathbb{Z}$ skrivas som en unik sträng med l bokstäver. Det ska noteras att alla l-bokstavssträngar kan inte dyka upp, bara de som motsvarar heltal mindre än n_A för den särskilda användarens n_A.

Exempel 10. I detta exempel ska vi välja små heltal för att illustrera processen, i verkligheten används mycket större tal. Välj $N = 26, k = 3, l = 4$. Det betyder att klartexten består av tripletter och chiffertexten består av strängar med fyra bokstäver i det vanliga engelska alfabetet med 26 bokstäver. För att skicka meddelandet "YES" till en användare A med chiffreringsnyckeln $(n_A, e_A) = (46927, 39423)$, hittar vi först den numeriska motsvarigheten till "YES", nämligen $24 \times 26^2 + 4 \times 26 + 18 = 16346$, och beräknar sedan $16346^{39423} \bmod 46927$, som är $21166 = 1 \times 26^3 + 5 \times 26^2 + 8 \times 26 + 2 =$"BFIC". Mottagaren A vet dechiffreringsnyckeln $(n_A, d_A) = (46927, 26767)$, och så beräknar $21166^{26767} \bmod 46927 = 16346 =$"YES". Hur genererade användare A sina nycklar? Först multiplicerade hon primtalen $p_A = 281$ och $q_A = 167$ för att få n_A; sedan valde hon e_A slumpmässigt (men under villkoret att $SGD(e_A, 280) = SGD(e_A, 166) = 1$). Sedan hittade hon $d_A = e_A^{-1} \bmod 280{\times}166$. Talen p_A, q_A, d_A hölls hemliga.

RSA kan även användas till att signera meddelanden och verifiera att det var rätt person som skickade det, något som berörs i *Algebraic Aspects of Cryptography* [N].

2.4.2 Hashfunktion

Antag att vi skickar ett meddelande som innehåller l antal symboler, och vi skulle vilja att vår signatur var mycket kortare − säg, ungefär k antal symboler. Här följer en informell definition av "hash".

Definition 19. En funktion $H(x)$ från mängden l till mängden k kallas för en *hashfunktion* om $H(x)$ är enkel att beräkna för vilket x som helst, men

1. Ingen kan möjligen hitta två olika värden på x som ger samma $H(x)$ ("kollisionsresistant"), och

2. givet y i bilden av H, kan ingen möjligen hitta ett x så att $H(x) = y$ ("urbildsresistant").

Ett av de huvudsakliga användningsområdena för hashfunktioner är i digitala signaturer. Antag att Bob sänder ett långt meddelande x till Alice bestående av l symboler. Både Alice och Bob använder samma hashfunktion och det finns

inget behov för dem att hålla den hemlig för deras motståndare Catherine. Efter att Bob sänder Alice meddelandet x, bifogar han hashvärdet $H(x)$ som han får genom att applicera hashfunktionen på det meddelande han skickar. Alice skulle vilja vara säker på att det verkligen var Bob som skickade meddelandet x och att Catherine inte ändrade meddelandet innan Alice fick det. Vi antar här att hon på något sätt åtminstone kan vara säker på att det bifogade $H(x)$ kom från Bob. I det fallet är allt hon behöver göra att applicera hashfunktionen på meddelandet hon fick. Om det överensstämmer med $H(x)$, är hon nöjd. Hon vet att Catherine omöjligt kunde ändrat på x så att hon fick fram det förvrängda meddelandet x' så att $H(x') = H(x)$.

2.4.3 Signatur

Hur Alice kan vara säker på att $H(x)$ verkligen kom från Bob kan också lösas genom RSA. För enkelhetens skull, välj k så att meddelanden av längd k är små nog för att utgöra en meddelande-enhet: om det engelska alfabetet med 26 bokstäver används, har vi $0 \leq m < N$, där m är antalet meddelande-enheter och $N = 26^k$. Efter att ha skickat meddelandet x, räknar Bob ut hashvärdet $H = H(x)$. Han skickar inte bara H till Alice utan först upphöjer han det till sin *dechiffreringsexponent*, d_{Bob} modulo n_{Bob}. Sedan skickar Bob hela meddelandet x med $H' = H^{d_{Bob}} \pmod{n}_{Bob}$ bifogat, där han använder Alice krypteringsexponent e_{Alice} och hennes modulus n_{Alice}, till Alice. Vilket betyder att han skickar

$$(H^{d_{Bob}} \pmod{n}_{Bob})^{e_{Alice}} \pmod{n}_{Alice}$$

tillsammans med meddelandet och där notationen $a \pmod{n}$ betecknar den minsta icke-negativa resten av a modulo n. Efter att Alice dechiffrerat meddelandet, tittar hon på den sista meddelande-enheten som hon får, vilket kommer motsvaras av

$$((H^{d_{Bob}} \pmod{n}_{Bob})^{e_{Alice}})^{d_Alice} \pmod{n}_{Alice} = H^{d_{Bob}} \pmod{n}_{Bob}.$$

Vad hon sedan gör är att höja upp denna meddelande-enhet, som ser ut som rappakalja för henne, med Bobs offentliga *krypteringsexponent* e_{Bob} modulo n_{Bob} som ger henne hashvärdet H tillbaka

$$(H^{d_{Bob}} \pmod{n}_{Bob})^{e_{Bob}} = H.$$

Det enda hon behöver göra sen är att använda hashfunktionen på meddelandet hon har fått och se om de överensstämmer. Om de gör det vet hon att meddelandet kom från Bob eftersom att *endast Bob* vet den exponent som är inversen till att höja upp med e_{Bob} modulo n_{Bob}. Då vet hon att H kom från Bob och att meddelandet inte snappades upp och ändrades på vägen.

2.4.4 Attacker

Under de år som RSA använts har det ägnats mycket tid åt att analysera kryptosystemet för svagheter. En sammanställning av attacker mot RSA gjordes av Dan Boneh [B] och här beskrivs några av dem och vad de bygger på.

Elementära attacker: Dessa attacker bygger på en uppenbar felanvändning av RSA. Två exempel på elementära attacker är:

1. *Gemensam modulus.* Om man vill undvika att skapa flera modulus $n = pq$ för flera användare av RSA kan man vilja bestämma n en gång för alla. Någon central enhet tilldelar varje användare i ett unikt par e_i, d_i från vilken användare i skapar en offentlig nyckel n, e_i och en privat nyckel n, d_i.
   Det kan verka bra till en början: en chiffertext $C = M^{e_A} \bmod n$ skickat till Alice kan inte dekrypteras av Bob eftersom att Bob inte känner till d_A. Emellertid så stämmer det inte helt och systemet är osäkert. Bob kan använda sina egna exponenter e_B, d_B för att faktorisera talet n. När n är faktoriserat kan Bob få fram Alice privata nyckel d_A från hennes offentliga nyckel e_A. Denna observation visar att RSA modulus *aldrig* ska användas av mer än en person.

2. *Bländning(eng.blinding).* Låt (n, d) vara Bobs privata nyckel och (n, e) den motsvarande offentliga nyckeln. Antag att en motståndare Catherine vill ha Bobs signatur på ett meddelande $M \in \mathbb{Z}_n^*$. Bob vägrar att signera M. Catherine kan då pröva följande: hon väljer ett slumpmässigt $r \in \mathbb{Z}_n^*$ och sätter $M' = r^e M \bmod n$. Sedan ber hon Bob signera det ändrade meddelandet M'. Bob kan gå med på att signera det oskyldiga M' med sin signatur S'. Men kom ihåg att $S' = (M')^d \bmod n$. Catherine beräknar nu helt enkelt $S = S'/r \bmod n$ och får Bobs signatur S på det ursprungliga M.

   $$S = (S')/r = (M')^d/r = r^{ed}M^d/r \equiv rM^d/r = M^d \pmod{n}.$$

   Så Bob har ovetandes signerat meddelandet M. Mottagaren som sedan får meddelandet gör helt enkelt som vanligt när han eller hon ska verifiera att meddelandet kom från den rätta personen.

   $$S^e = (S')^e/r^e = (M')^{ed}/r^e \equiv M'/r^e = M \pmod{n}.$$

   Det ser ut som att Bob har signerat meddelandet även om det inte var han som gjorde det. Denna teknik kallas bländning, och låter Catherine att få en giltig signatur på ett eget meddelande genom att be Bob signera ett slumpmässigt "bländat" meddelande. Bob har ingen aning om vilket meddelande han egentligen signerar. Eftersom att de flesta signaturscheman använder en "enkelriktad hash" på M innan signering är denna attack egentligen inget stort problem.

Liten privat exponent: För att minska dekrypteringstiden (eller signaturgenereringstiden), kan man vilja använda ett litet värde på d snarare än ett slumpmässigt d. Ett litet d kan förbättra utförandet med minst en faktor 10 (för en 1024 bitars moduls). Oturligt nog finns det en attack som visar ett ett litet d resulterar i ett totalt avbrott i kryptosystemet

Sats 6. : Låt $n = pq$ med $q < p < 2q$. Låt $d < \frac{1}{3}n^{1/4}$. GIvet (n, e), med $ed = 1 \bmod \varphi(n)$, kan Catherine enkelt få fram d.

Bevis. Hänvisas till artikeln i fråga, bevis av "Theorem 2 (M.Weiner)" s.4 i [B]. □

Eftersom att n brukar vara 1024 bitar, följer att d måste vara minst 256 bitar för att undvika denna attack. Det finns dock en del tekniker som tillåter snabb dekryptering och som inte är sårbar för den ovanstående attacken:

I *Stora* e: Antag att istället för att reducera e modulo $\varphi(n)$, använder man (n, e') som den offentliga nyckeln, med $e' = e + t \times \varphi(n)$ för något stort t. e' kan användas istället för e i kryptering av meddelanden genom

$$x^{de'} = x^{d(e+t\times\varphi(n))} = x^{de+d\times t\times\varphi(n)} = x^{de} \times x^{d\times t\times\varphi(n)} \equiv x \times (x^{\varphi(n)})^{d\times t} \bmod n$$

$$\equiv x \times 1^{d\times t} \bmod n \equiv x \bmod n.$$

En enkel uträkning visar ett om $e' > n^{1.5}$ så kan attacken inte inledas. Olyckligtvis resulterar stora e i en ökad krypteringstid.

II *Använda kinesiska restsatsen(KRS)*: Ett alternativt tillvägagångssätt är att använda KRS. Vi ska först notera att om $y \equiv x \bmod n (= pq)$ så är $y \equiv x \bmod p$ och $y \equiv x \bmod q$ generellt. Antag att man väljer d så att både $d_p = d \bmod (p-1)$ och $d_q = d \bmod (q-1)$ är små, till exempel 128 bitar var. Då kan snabb dekryptering av en chifertext C uppnås som följande: beräkna först $M_p = C^{d_p} \bmod p$ och $M_q = C^{d_q} \bmod q$. Det vill säga vi har $M \equiv C^d \bmod n$ som vi delar upp i $M \equiv C^d \bmod p$ och $M \equiv C^d \bmod q$ och sedan reducerar vi C^d med de valda exponenterna d_p och d_q så att vi får M_p och M_q. Reduceringen av exponenten d till d_p eller d_q fungerar om vi tänker på följande vis. Vi skriver om d som en multipel av $(p-1)$ plus en rest $d = k \times (p-1) + d \bmod (p-1)$. Det ger oss att $C^d = C^{k\times(p-1)+d \bmod (p-1)} = (C^{(p-1)})^k \times C^{d \bmod (p-1)}$. Och eftersom att $C^{(p-1)} \equiv 1 \bmod p$ får vi att $(C^{(p-1)})^k \times C^{d \bmod (p-1)} \equiv 1^k \times C^{d \bmod (p-1)} \bmod p \equiv C^{d \bmod (p-1)} \bmod p$. Använd sedan KRS för att beräkna det unika värdet $M \in \mathbb{Z}_n$ som tillfredsställer $M = M_p \bmod p$ och $M = M_q \bmod q$ och är det meddelande M som vi vill återställa. Det M som vi får fram med KRS tillfredsställer $M = C^d \bmod n$ som är tanken med att använda KRS. Poängen är att även som d_p och d_q är små kan värdet på $d \bmod \varphi(n)$ vara stort, d.v.s. på ordningen av $\varphi(n)$. Som ett resultat kan inte attacken utföras.

Vi vet inte om någon av desa metoder är säkra. Allt vi vet är att den beskrivna attacken som bygger på sats 6 inte är effektiv mot dem. Det har visats att så länge som $d < n^{0.292}$, kan en motståndare effektivt få fram d från (n, e).

Liten offentlig exponent: För att minska krypterings- och signaturverifieringstid, är det vanligt att använda en liten offentligt exponent e. Det minsta möjliga värdet för e är 3, men för att stå emot vissa attacker är värdet $e = 2^{16} + 1 = 65537$ rekommenderat. När värdet $2^{16} + 1$ används, kräver singnaturverifikation 17 multiplikationer jämfört med drygt 1000 när ett slumpmässigt $e \leq \varphi(n)$ används. Till skillnad från attacken i föregående stycke orsakar attacker som appliceras när e är litet inte ett totalt avbrott.
Hastads spridnigsattack: Denna attack utvecklades av J. Hastad. Antag att Bob vill skicka ett krypterat meddelande M till ett antal grupper $P_1, P_2, \ldots, P_k$. Varje grupp har sin egan RSA-nyckel (n_i, e_i). Vi antar att M är mindre än alla n_i. För att skicka M krypterar Bob naivt det med var och en av de offentliga nycklarna och skickar ut den i:te chiffertexten till P_i. En angripare Catherine kan tjuvlyssna på förbindelsen utan att Bob ser det och samla de skickade chiffertexterna. För enkelhetens skull, antag att alla offentliga exponenter e_i är lika med 3. Ett enkelt argument visar att Catherine kan få fram M om $k \geq 3$. Catherine

får C_1, C_2, C_3, där

$$C_1 = M^3 \bmod n_1, \quad C_2 = M^3 \bmod n_2, \quad C_3 = M^3 \bmod n_3.$$

Vi kan anta att $SGD(n_i, n_j) = 1$ för alla $i \neq j$ eftersom att annars kan Catherine faktorisera några av n_i. Catherine kan sedan använda kinesiska restsatsen på C_1, C_2, C_3 och få fram ett $C' \in \mathbb{Z}_{n_1 n_2 n_3}$ som kommer att tillfredsställa $C' = M^3 \bmod n_1 n_2 n_3$ eftersom att vi letade efter ett värde på C' som uppfyllde att

$$C' \equiv C_1 \bmod n_1$$

$$C' \equiv C_2 \bmod n_2$$

$$C' \equiv C_3 \bmod n_3$$

Vilket är gör att M^3 är detsamma som C' eftersom att vi har

$$M^3 \equiv C_i \bmod n_i, i = 1, 2, 3.$$

Och eftersom att M är mindre än alla n_i har vi att $M^3 < n_1 n_2 n_3$. $C' = M^3$ håller därför för alla heltal. Catherine kan på detta sätt få fram M när $k \geq e$ och i detta fall behöver Catherine bara ta tredjeroten ur C' för att få fram M. Attacken är bara genomförbar när ett litet e används.
Attacken går att skydda sig mot genom att förvanska meddelandet M innan han krypterar det. Men det fungerar bara om han använder det rätt, exempelvis linjär förvanskning har visat sig vara osäkert.
Partial Key Exposure Attack: Det har visat sig att om en motståndare Catherine får tag i en del av den privata nyckeln d, säg en fjärdeldel av den, kan hon återskapa hela d om den motsvarande offentliga nyckeln är liten. Det har visats att så länge $e < \sqrt{n}$ är det möjligt att återskapa hela d från enbart ett fragment av dess bitar.

Implementeringsattacker: Dessa attacker attackerar implementeringen av RSA istället för den underliggande strukturen av RSA.

1. *Timingattacker*. Denna attack går ut på att man mäter tiden det tar för ett smartkort (ex. ett bankkort) att utföra en dekryptering (eller signatur), och med hjälp av informationen räkna ut den privata exponenten d. Attacken använder sig av den upprepade kvadrerings metoden, för närmare förklaring se artikeln (s.12, 5.1 “Timing attacks” i [B]). Catherine ber kortet att generera signaturer på ett stort antal tal med slumpmässiga meddelanden $M_1, \ldots, M_k \in \mathbb{Z}_n^*$ och mäter tiden det tar för kortet att generera var och en av signaturerna. Genom att räkna på tiden kan Catherine få fram bitarna till d en i taget. Det finns två sätt att skydda sig mot attacken. Den enklaste är att lägga till en lämplig fördröjning så att beräkningarna tar en bestämd tid. Det andra sättet är baserat på bländning. Innan dekrypteringen tar smartkortet ett slumpmässigt $r \in \mathbb{Z}_n^*$ och beräknar $M' = M \times r^e \bmod n$. Sedan applicerar den d på M' och får $S' = (M')^d \bmod n$. Slutligen sätter smartkortet $S = S'/r \bmod n$. Genom detta applicera smartkortet d på ett slumpmässigt meddelande M' som Catherine inte känner till. Som ett resultat kan Catherine inte påbörja attacken, men det ursprungliga meddelandet kan fås tillbaka med liknande resonemang som uträkningen i bländningen.

2. *Slumpvisa fel.* Implementeringar av RSA kryptering och signaturer använder ofta kinesiska restsatsen för att snabba på beräkningen av $M^d \bmod n$. Istället för att arbeta modulo n, räknar signeraren Bob ut signaturerna modulo p och q och kombinerar resultatet med KRS. Men det finns en risk med att använda den metoden. Antag att medan Bobs dator genererar en signatur får en bugg datorn att felaktigt beräkna en enda instruktion. Givet en ogiltig signatur kan en motståndare Catherine enkelt faktorisera Bobs modulus n. För att attacken ska fungera måste Catherine veta meddelandet M. Nämligen vi antar att Bob inte förvanskar meddelandet på något sätt. Slumpmässig förvanskning skyddar mot denna attack. Ett enklare försvar är att Bob kontrollerar den genererade signaturen innan han skickar ut den till världen.

2.5 Diffie-Hellmans nyckelbyte

I en banbrytande uppsats som publicerades år 1976 observerade författarna W.Diffie och M.E.Hellman följande "enkelriktad funktion"-beteende hos vissa gruppoperationer, som här beskrivs som i *Prime numbers: a computational perspective* [C]. För ett givet heltal $x \geq 0$ och ett element g i $\mathbf{F}_p^*$, är beräkningen av

$$h = g^x$$

i kroppen (så det involverar upprepade (mod p) reduktioner) generellt av låg komplexitet, $O(ln\ x)$-kropp operationer. Å andra sidan, att lösa denna ekvation för x, med g, h, p givet, är bevisligen mycket svårare. Då x är en exponent, och eftersom att vi tar något som liknar en logaritm i detta senare problem, är utbrytandet av x känt som det diskreta logaritm(DL)-problemet. Även om ekvationen åt höger (höjer upp g med exponenten x) är av polynomisk tid-komplexitet, finns det ingen känd generell metod för att lösa den diskreta logaritmen med någon liknande effektivitet.

Ordet "diskret" utmärker att det är en situation med ändliga grupper till skillnad från den klassiska (kontinuerliga) situationen, något som förklaras i *A Course in Number Theory and Cryptography* [K].

Definition 20. Om G är en ändlig grupp, b ett element i G, och y är ett element i G som är en potens av b, då är den *diskreta logaritmen* för y till basen b vilket heltal x som helst, så att $b^x = y$.

Exempel 11. : Om vi tar $G = \mathbf{F}_{19}^* = (\mathbb{Z}/19\mathbb{Z})^*$ och låter b vara generatorn 2, då är den *diskreta logaritmen* för 7 till basen 2 talet 6. (dvs. $2^6 \equiv 7 \bmod 19$)

2.5.1 Algoritm

Två personer, Alice och Bob, kommer överens om ett primtal p och en generator $g \in \mathbf{F}_p^*$. Denna algoritm låter Alice och Bob bestämma en gemensam nyckel (mod p), utan att någon av dem kan lista ut den andres hemliga nyckel.

Vi kommer återigen hämta algoritmen från *Prime numbers: a comutational perspective* [C] med några kommentarer och ett exempel från *A Course in Number Theory and Cryptography* [K].

1. **Alice genererar en offentlig nyckel**
   Alice väljer slumpmässigt $a \in [2, p-2]$;
   $x = g^a \bmod p$;

2. **Bob genererar en offentlig nyckel**
   Bob väljer slumpmässigt $b \in [2, p-2]$;
   $y = g^b \bmod p$;

3. **Varje person skapar samma gemensamma nyckel**
   Bob beräknar $k = x^b \bmod p$;
   Alice beräknar $k = y^a \bmod p$;

Denna gemensamma nyckelkonstruktion fungerar på grund av att

$$(g^a)^b = (g^b)^a = g^{ab}$$

och allt detta går igenom de vanliga reduktionerna (mod p). Notera att Alice och Bob inte hade behövt välja ett slumpmässigt tal, de kunde valt en minnesvärd fras, slogan, vad som helst, och omvandlat det till deras hemliga värde a, b. Notera också att den offentliga nyckeln $g^a, g^b \bmod p$ kan göras offentlig i det avseendet att − på grund av DL-problemet − det är bokstavligt talat säkert att offentligt publicera sådana värden.

Exempel 12. Antag att vi använder ett förskjutningschiffer på enbokstavs-meddelande-enheter i det engelska alfabetet med 26 bokstäver: $C \equiv P + B \bmod 26$. För att välja B, ta den minsta positiva resten modulo 26 av ett slumpmässigt tal i F_{53}. Låt $g = 2$ (som är en generator till F_{53}). Antag att Alice slumpvis valde ut $a = 29$ och letade fram Bobs offentliga 2^b, som är, säg $12 \in \mathbf{F}_{53}$. Hon vet då att krypteringsnyckeln är $12^{29} = 21 \in \mathbf{F}_{53}$, det vill säga $B = 21$. Under tiden har hon offentliggjort $2^{29} = 45$, och så kan Bob också hitta nyckeln $B = 21$ genom att höja upp 45 med exponenten b (hans hemliga nyckel är $b = 19$). När man jobbar med så små kroppar finns det ingen, eller en väldigt svag, säkerhet; en utomstående kan lätt hitta den diskreta logaritmen till basen 2 för 12 eller 45 modulo 53. Och det finns ändå ingen säkerhet i användningen av ett förskutningschiffer med enbokstavs-meddelande-enheter. Men detta exempel illustrerar tekniken bakom Diffie-Hellmans nyckelbytessystem.

2.5.2 Säkerhet

I en artikel som publicerades i oktober 2015, *Imperfcet Forward Secrecy: How Diffie-Hellman Fails in Practice* [Aea], hade en grupp forskare undersökt hur Diffie-Hellman används av webbläsare, servrar och andra användare av kryptografi på internet. De visade att genom att använda Number Field Sieve (NFS)-algoritmen kunde de utföra vissa förberäkningar på stora kroppar med primtal p, och med hjälp av det sedan lista ut den diskreta logaritmen. På så sätt kunde de knäcka Diffie-Hellman. Hade de en gång förberäknat primtalskroppen kunde de knäcka *alla* Diffie-Hellman system med det primtalet. De visar också på en ny attack som de kallar för "Logjam attack" där de tvingar anändare av ett visst internetprotokoll (TLS) att använda svagare Diffie-Hellman, där de sedan kunde läsa datan och själva införa egen data i kommunikationen. Den svagare typen av Diffie-Hellman använde primtal av storlek 512 bitar. De visade också att även om det skulle ta lång tid att försöka med samma sak för primtal av

storleken 1024 bitar skulle det vara möjligt om man hade statliga resurser. Forskarna varnar för följande saker om man använder Diffie-Hellman.
Felaktigt genererade grupper: Om grupperna som används för Diffie-Hellman inte är nog säkra är de sårbara för flera olika attacker.
Standardiserade primtal: Många implementationer av Diffie-Hellman använde samma parametrar. Om primtalen är säkra nog finns det ingen risk med att återanvända dem men om många använder samma parametrar kan även en motståndare med dåliga resurser ändå attackera många mål.

För att öka säkerheten föreslår författarna följande:
Byte till elliptiska kurvor: Byte till Diffie-Hellman med elliptiska kurvor (eng. Elliptic Curve Diffie-Hellman, ECDH) med lämpliga parametrar undviker alla typer av rimliga attacker. Nuvarande diskreta logaritm-algoritmer för elliptiska kurvor har inte så mycket fördel av förberäkningar (precomputation). Dessutom, ECDH nycklar är kortare än "mod p" Diffie-Hellman och beräkningarna för den delade hemligheten går fortare.
Öka minsta nyckel styrkan: Server-operatorer borde öka till att använda 2048-bitars primtal eller större i sina system. Webbläsare och andra klienter borde öka den minst acceptabla storleken på Diffie-Hellman grupper till minst 1024 bitar föra att undvika attacker som gör att de använder ett svagare Diffie-Hellman system när de kommunicerar med servrar som använder mindre grupper. Primtal under 1024 bitar borde inte anses säkra ens mot en motståndare med måttliga resurser.
Undivik 1024-bitars grupper med bestämda primtal: För de implementeringar som måste fortsätta att använda 1024-bitars grupper skulle generering av nya grupper lindra skadan som förberäkningar med NFS gjort med vanliga bestämda grupper. Det är möjligt att skapa fallucke-primtal (trapdoor primes) som beräkningsmässigt är svåra att upptäcka. Som minsta åtgärd borde klienter se efter så att servrar använder säkra primtal eller verifierbar genereringsprocess.

Litteraturförteckning

[Aea] Adrian D. et al., *Imprtfect Forward Secrecy: How Diffie-Hellman Fails in Practice.* CCS '15: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, ACM Conference Proceedings, 5-17.

[N.B] Biggs, N.L., *Discrete Mathematics, 2ed.* Oxford Science Publications. The Clarendon Press, Oxford University Press, New York, 1989.

[B] Boneh, D., *Twenty years of attacks on the RSA cryptosystem.* Notices Amer. Math. Soc. **46** (1999), no. 2, 203213.

[C] Crandall, R. Pomerance, C., *Prime numbers: a computational perspective.* Springer Verlag, New York, 2001.

[E] Eriksson, K. Gavel, H., *Diskret matematik fördjupning*, Studentlitteratur, Lund, 2003.

[K] Koblitz, N., *A Course in Number Theory and Cryptography.* Graduate Texts in Mathematics, **114**. Springer-Verlag, New York, 1987.

[N] Koblitz, N., *Algebraic Aspects of Cryptography.* Algorithms and Computation in Mathematics, **3**. Springer Verlag, Berlin, 1998.

[S] Svensson, P-A., *Abstrakt algebra*, Studentlitteratur, Lund, 2001.