



Denne forelesningsøkten vil bli tatt opp og lagt ut i emnet i etterkant.

Hvis du ikke vil være med på opptaket:

	La være å delta med webkameraet ditt.
	La være å delta med mikrofonen din.
To: Marianne Sundby (Privately) Type message here...	Still spørsmål i Chat i stedet for som lyd. Hvis du ønsker kan spørsmålet også sendes privat til foreleser.

Påminnelse - Regler for Zoom forelesninger

Jeg fikk takk fra studenter for at jeg hadde tydelige regler, det ble påpekt at det i andre fag har vært mye støy i chat, så jeg minner om:

- Skru av mikrofon
- KUN faglige spørsmål i chat
- Ikke kommenter andres spørsmål i chat, vent til foreleser svarer (Marcus Dahl vil noen dager være med på zoom som hjelpe-lærer; han vil svare spørsmål i chat, han er ikke student :-)
- Ikke snakk med hverandre i chat (det forstyrrer andre)
- I øvingstimene prøver vi å få en bedre dialog, da anbefaler jeg at spørsmål stilles med mikrofon



Høyskolen
Kristiania

TK2100: Informasjonsikkerhet

Andre forelesning:
KRYPTERING

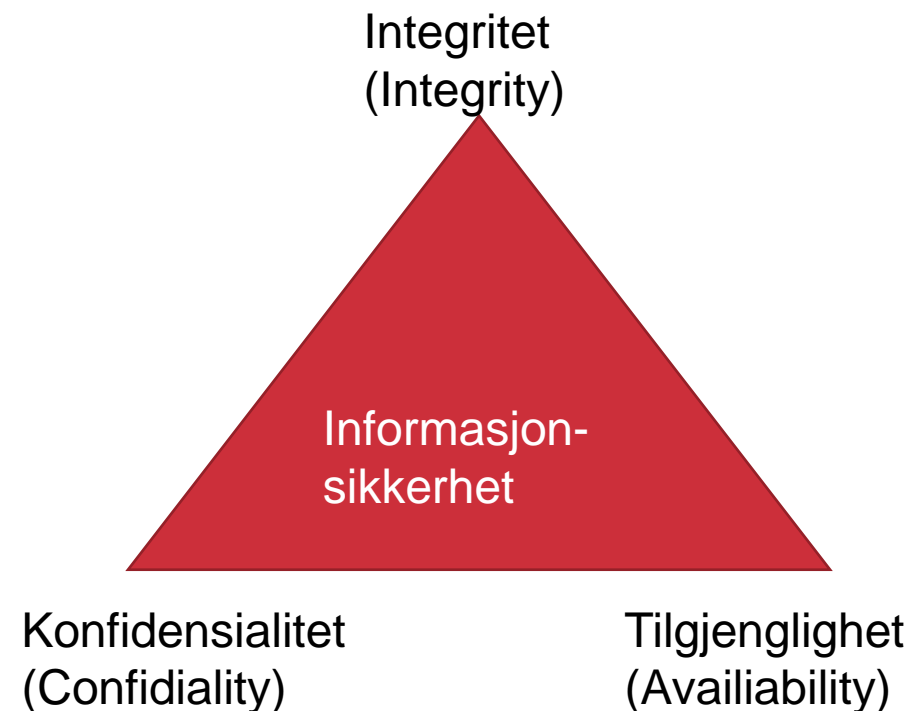
«Pensum»:

I 2011-utgaven s. 19-32 (repetisjon + utdyping), s 54-91 (Nytt stoff)

I 2014-utgaven: s. s. 25-38 (rep + utdyping), s. 387-438

NIVÅER for sikkerhet og tiltak

- Data
 - Access Control Lists, kryptering (av filsystem), ...
- Applikasjon
 - Patching, sertifikater, Anti-virus, ...
- Vertsmaskin
 - **OS**, autentisering og autorisering, brukeradm, group policies
- LAN (Internt nett)
 - IP-spoofing, IPSec, IDS, ...
- Grense (Perimeter)
 - IDS, Firewall, VPN, NAT, ...
- Fysisk sikring
 - Vakt, låser og overvåking
- Policy, prosedyrer, bevissthet
 - Brukeropplæring



CIA-modellen

Hvordan oppnå stabile **systemer**?

- Policy (retningslinjer) er grunnlaget
 - En beslutning om hva vi ønsker oss, hvordan det skal være, sett i sammenheng med hva vi har råd til.
- Forutsigbarhet
 - Det viktigste målet er et forutsigbart system (på alle nivåer). Det gir: pålitelighet, tillit og dermed sikkerhet.
- Skalerbarhet
 - Systemet skal vokse i tråd med policy, fungere forutsigbart, m.a.o. sikkert.



- Design bør være offentlig, ikke basér sikkerheten på at ingen vet hvordan et system er implementert (krypteringsalgoritme...)
- Standardinnstilling: Ingen aksess sørg for at alle som skal tilgang til objekter eksplisitt må gis tilgang
- Sjekk for nåværende tilgang
 - Filer kan være åpne i uker... LanManager
- Gi hver prosess færrest mulige privilegier
- Mekanisme burde være: Enkel, uniform, på lavest mulig nivå
- Psykologisk akseptabel
- KISS (**Keep It Simple, Stupid**) kompleksitet er sikkerhetens ende. Hvis det er vanskelig å forstå et system, er det også vanskelig å avgjøre om det er sikkert.

Beskyttelse og sikkerhet

- Både **brukere** og **prosesser** trenger å ha adgang til felles ressurser, samt behov for eksklusive rettigheter til private ressurser
- Må ha beskyttelses-mekanismer som kan stadfeste (authenticate) ressurs-tilgang
 - Filbeskyttelse, minne-beskyttelse
- En organisasjon bør/skal ha et sett med regler (**policy**) som definerer eierskap og adgang til ressursene

Kryptering

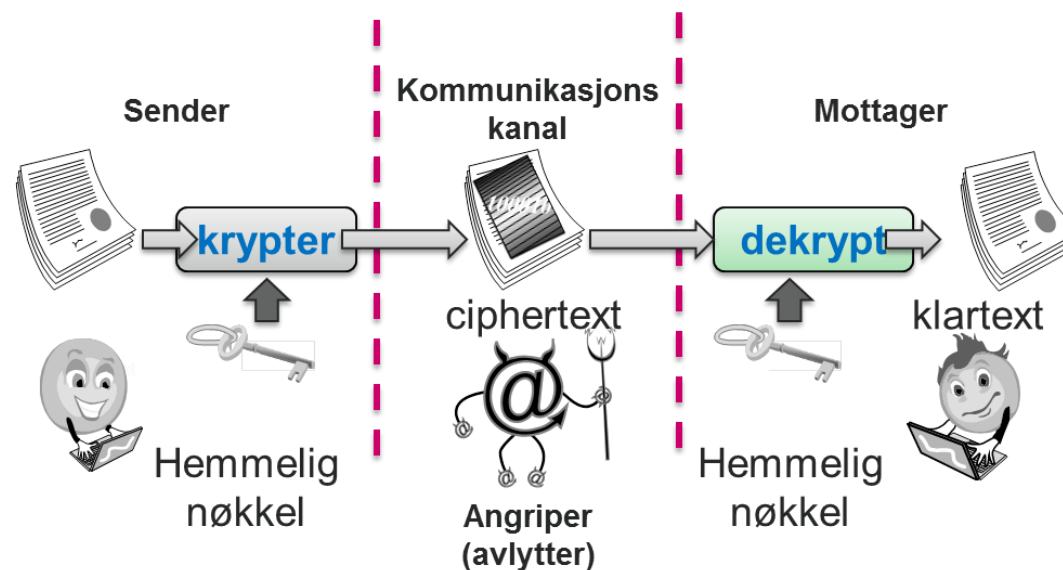
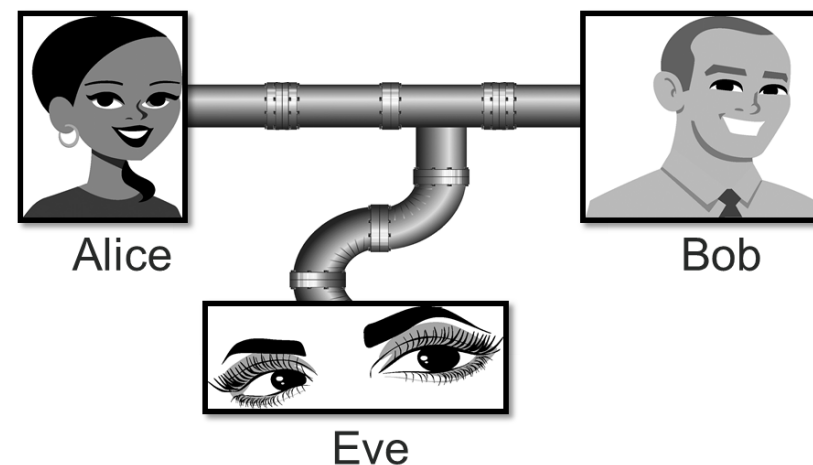
Kryptering er den viktigste teknikken for å gjennomføre retningslinjene for å sikre systemet

Dagens Mål

- Forstå de grunnleggende prinsippene for kryptering og kryptoanalyse
- Kjenne til og kunne vurdere vanlig brukte krypteringsalgoritmer opp mot hverandre
- Det meste av **matematikken** er der for «de interesserte»

Fra sist

- Hvordan fordeles nøkkel?
 - Symmetrisk vs Public Key (asymmetrisk)
- Brukes til?
 - Lagring av passord
 - Sikring av konfidensielle data under lagring og overføring
 - Signering
 - «Sjekksummer»
 - ...



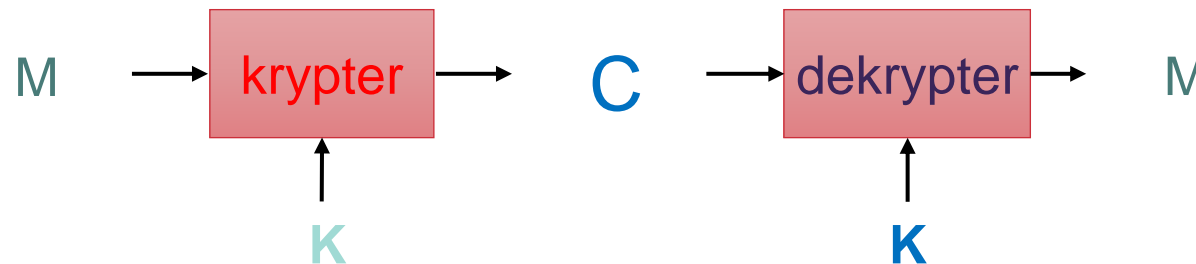
Symmetrisk Kryptosystem

- Scenario

- Alice vil sende en melding (klartekst **P** – eller melding **M**) til Bob.
- Kommunikasjonskanalen er usikret og kan være avlyttet
- Dersom Alice og Bob på forhånd har blitt enige om en symmetrisk krypteringsmetode og en hemmelig nøkkel **K**, så kan meldingen sendes kryptert (chifertekst/kryptogram **C**)

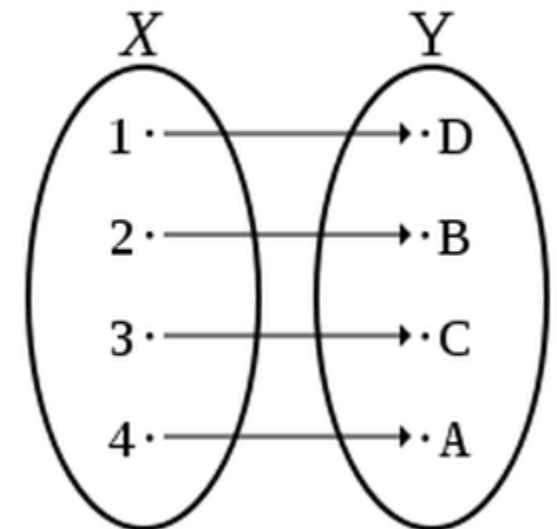
- Spørsmål

- Hva er en god krypteringsmetode (algoritme)?
- Hvor komplisert er det å kryptere/dekryptere?
- Hva er størrelsen på kryptogrammet vs klarteksten?



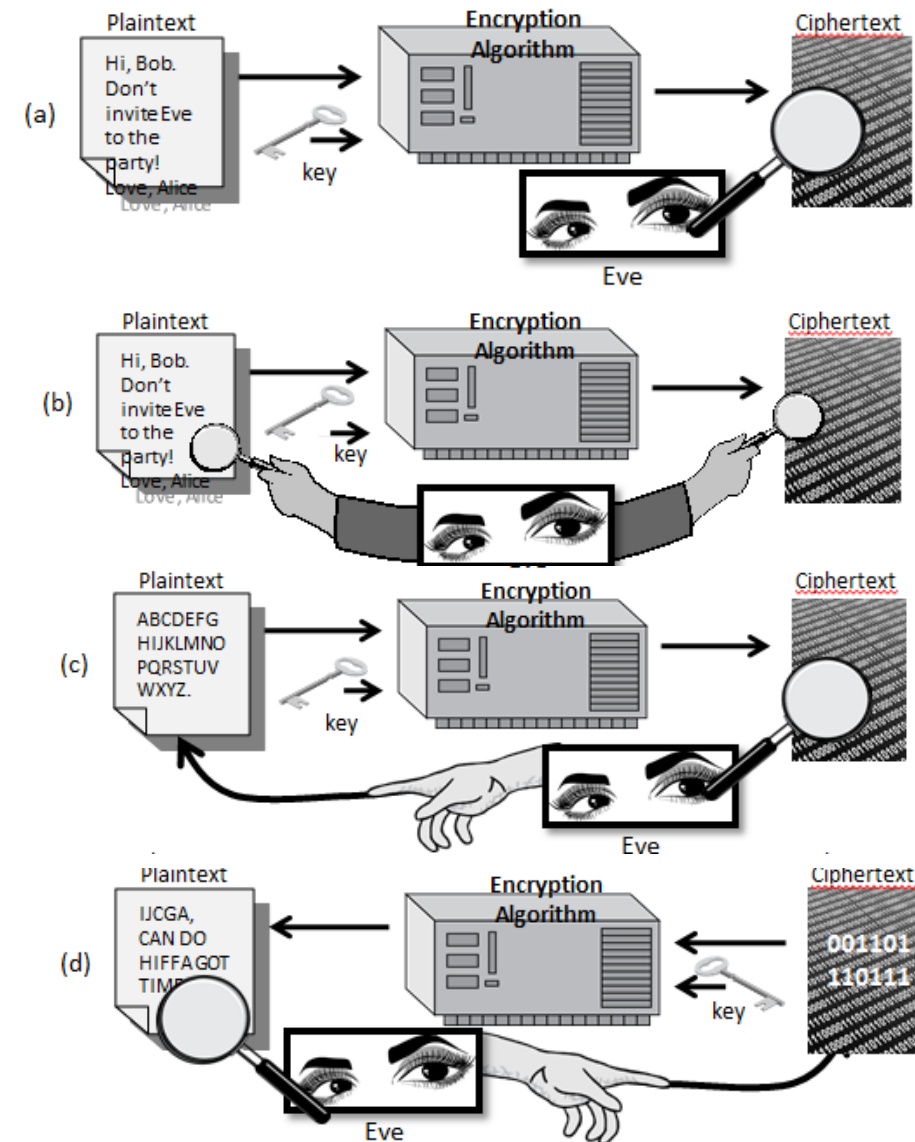
Grunnleggende

- Notasjon
 - Meldingen (plaintekst)
P
 - Hemmelig nøkkel
K
 - Krypteringsfunksjon
 $E_K(P)$
 - Dekrypteringsfunksjon
 $D_K(C)$
- Klartekst er typisk cirka **like lang** som kryptogrammet
- Kryptering og dekryptering er permuteringsfunksjoner på en mengde av alle N-bit arrays i klarteksten/kryptogrammet
- Effektivitet
 - Metodene **E_K** og **D_K** bør ha effektive algoritmer
 - Regne-tid og Minne-plass
- Konsistens
 - Dekryptering av kryptogrammet skal alltid gi klarteksten
 - **$D_K(E_K(P)) = P$**



4 mulige **angrep** (trusselbilder)

- Angriper kan ha
 - a) Samling av kryptogram
(**chiphertext only attack**)
 - b) Samling av klartekst/kryptogram par
(**known plaintext attack**)
 - c) samling av klartekst/kryptogram for klartekster valgt av angriper
(**chosen plaintext attack**)
 - d) Samling av klartekst/kryptogram par for kryptogram valgt av angriper
(**chosen ciphertext attack**)



Brute-Force (“Rå makt”) angrep

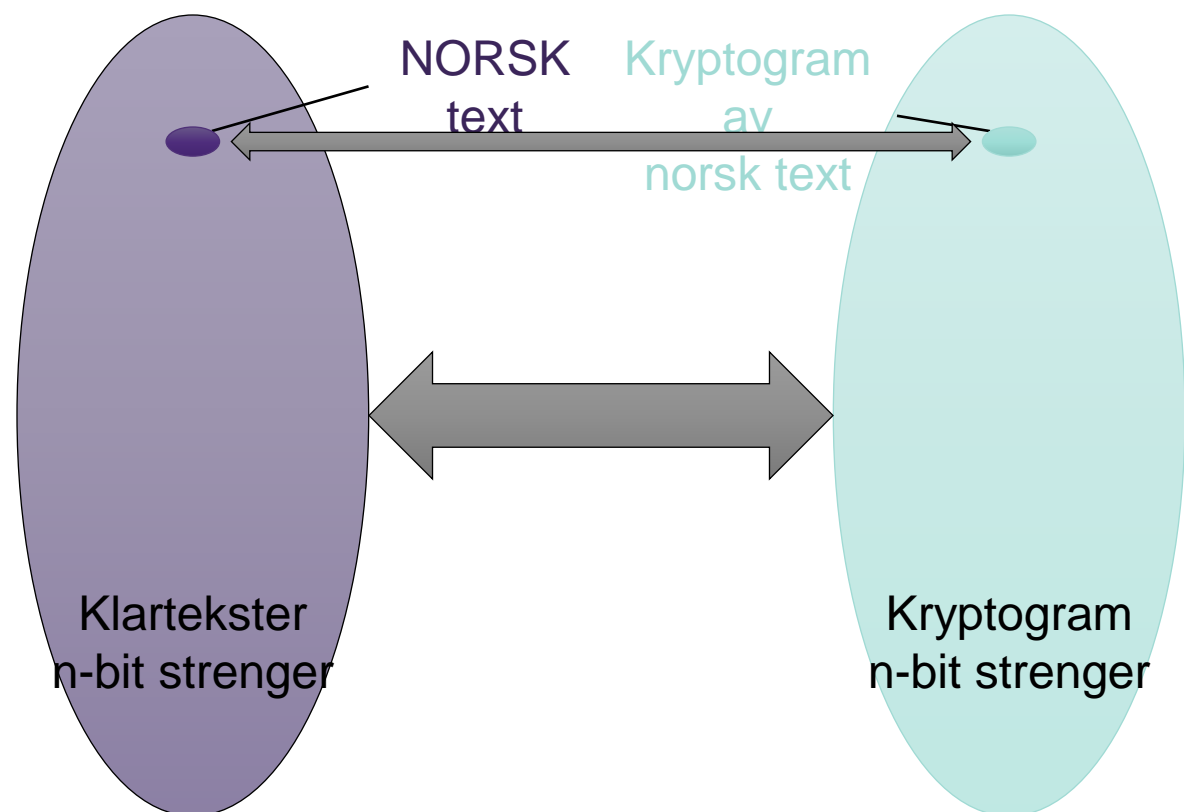
- Prøv alle mulige nøkler **K** og sjekk om **D_K(C)** ser ut som en (sannsynlig) klartext
 - Forutsetter at man vet noe om strukturen på klartexten (PDF?, email?, ..)
- Nøkkelen bør dermed være en tilstrekkelig lang og tilfeldig («random struktur») variabel til at det er ugjennomførbart å sjekke alle muligheter



Kryptere Norsk Tekst

- Norsk tekst er typisk representert som ISO 8859-1 (Windows 1252), UTF-8 eller UTF-16 koding
- En melding er dermed **t** tegn som svarer til en n-bit array, med ca. $n=8t$

- Redundans pga gjentatte ord og mønstre
 - F.ex., “og”, “ing”
- Norske klartekster er en **veldig liten** undermengde av alle mulige n-bit arrays





Entropi i Naturlige Språk

- Informasjonsinnhold (**entropi**) i Norsk: ca 1.3 bits pr bokstav
- t-bokstav array med Norsk tekst:
$$(2^{1.3})^t = 2^{1.3 t}$$
- Av alle mulige n-bit arrays (2^n) er det dermed en liten andel som faktisk er/kan være norsk tekst:
$$2^{1.3 n/8} \approx 2^{0.16 n}$$
- F.ex. Av alle mulige 128 Byte arrays ca $1,8 \cdot 10^{308}$, så er det bare ca $2,3 \cdot 10^{49}$ som er kandidater: ca 1 av 10^{250}

- Brute-force

- Forutsetter dermed mer enn bare rå forsøk
- F.eks. Ordbøker ([John the Ripper](#))

Substitusjonschiffer

Å bytte ut et tegn med ett annet
Den eldste og mest kjente metoden

Futurama AL1



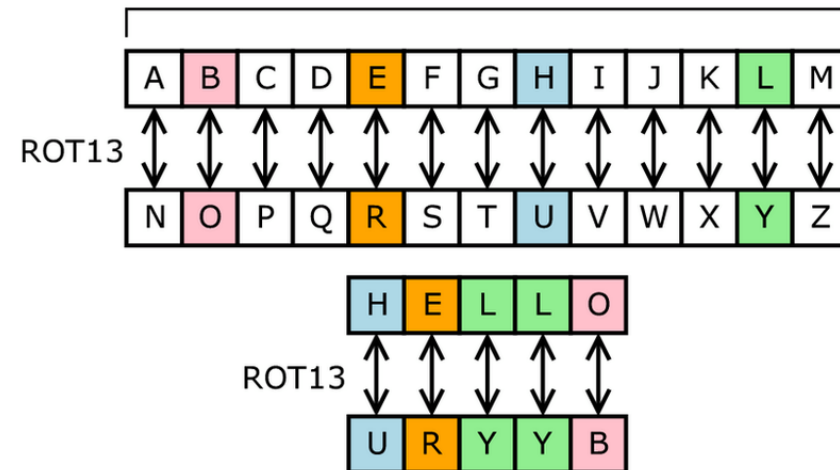
- Enkleste form for koding: bytte ut glyfer
- Egentlig ikke kryptering (ingen nøkkel eller algoritme)

A	B	C	D	E	F	G	H	I	J	K	L	M
⬇	⋈	⬆	⊠	⬇	⊠	⬆	⬇	⊠	⋈	⬆	⊠	⬇
N	O	P	Q	R	S	T	U	V	W	X	Y	Z
⊠	⬆	⬇	⋈	⬆	⬇	⬆	⬆	⬆	⬆	⬆	⬆	⬆
0	1	2	3	4	5	6	7	8	9	10	11	12
⊠	⬆	⬆	⬆	⬆	⬆	⬆	⬆	⬆	⬆	⬆	⬆	⬆
.	!	-	'	:	:							
⬆	⬆	⬆	⬆	⬆	⬆							

Substitusjons Chiffer

- Hver bokstav erstattes med en annen..
- På engelsk vil det være 26!
 $= 26 \times 25 \times 24 \times 23 \times \dots \times 3 \times 2 \times 1$
mulige slike koder.
- Det blir 4.03×10^{26} ulike.

- En populær versjon på Internett pleide å være ROT13.
- $C = (P + 13) \% 26$
- $P = (C + 13) \% 26$



Frekvensanalyse

- Bokstavene i naturlige språk, som norsk og engelsk, brukes ikke like ofte (er ikke “uniformt distribuert”)
- Kjennskap til hvor ofte enkeltbokstaver, parr og tripler vanligvis opptrer brukes ofte i angrep på substitusjonsmetoder.

a: 8.05%	b: 1.67%	c: 2.23%	d: 5.10%
e: 12.22%	f: 2.14%	g: 2.30%	h: 6.62%
i: 6.28%	j: 0.19%	k: 0.95%	l: 4.08%
m: 2.33%	n: 6.95%	o: 7.63%	p: 1.66%
q: 0.06%	r: 5.29%	s: 6.02%	t: 9.67%
u: 2.92%	v: 0.82%	w: 2.60%	x: 0.11%
y: 2.04%	z: 0.06%		

Mark Twain: Tom Sawyer(Engelsk)

Skinne-chiffer



- Alternativ til substitusjon
- Basert på å endre rekkefølge («bake om»)...
- Enkelt skinne-chiffer (2):

P: HEI!ALLE

H	I	A	L	
	E	!	L	E

**Kryptering/
Dekryptering**

C: HIALE!LE

S-Boxer

- Substitusjon kan gjøres med/på binærtall.
- Her erstattes ved oppslag nibble for nibble av P.
- Kan unngå å erstatte hele “bokstaver”
- Gjør frekvensanalyse litt vanskeligere?

	00	01	10	11		0	1	2	3
00	0011	0100	1111	0001	0	3	8	15	1
01	1010	0110	0101	1011	1	10	6	5	11
10	1110	1101	0100	0010	2	14	13	4	2
11	0111	0000	1001	1100	3	7	0	9	12
(a)					(b)				

Figure 8.3: A 4-bit S-box (a) An S-box in binary. (b) The same S-box in decimal. This particular S-box is used in the Serpent cryptosystem, which

Vigenére chiffer



Vigenére (1523-1596)

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
k →	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
o →	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
p →	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
r →	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	S
	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
t →	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
y →	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y



Nøkkel: krypto
 Klartekst: OLAOGKARI
 Chiffertekst: ycydzykig

- Polygrafisk – blokker
- Bruker nøkkelen til å bestemme blokk lengde
- Kan knekkes

One-Time Pads (Engangs blokk)

- Fullstendig **umulig** å knekke (i prinsippet).
 - Oppfunnet i 1917 av Joseph Mauborgne og Gilbert Vernam
 - Ble funnet en eldre kilde høsten 2013... (1865)
 - Vi benytter en tabell med shift-nøkler, (k_1, k_2, \dots, k_n) , til å kryptere en rentekst, M , med lengde n , der hver shift-nøkkel er valgt fullstendig tilfeldig (“uniformt tilfeldig”)
- Siden hvert shift er fullstendig tilfeldig så er ethvert kryptogram like sannsynlig for enhver klartekst!

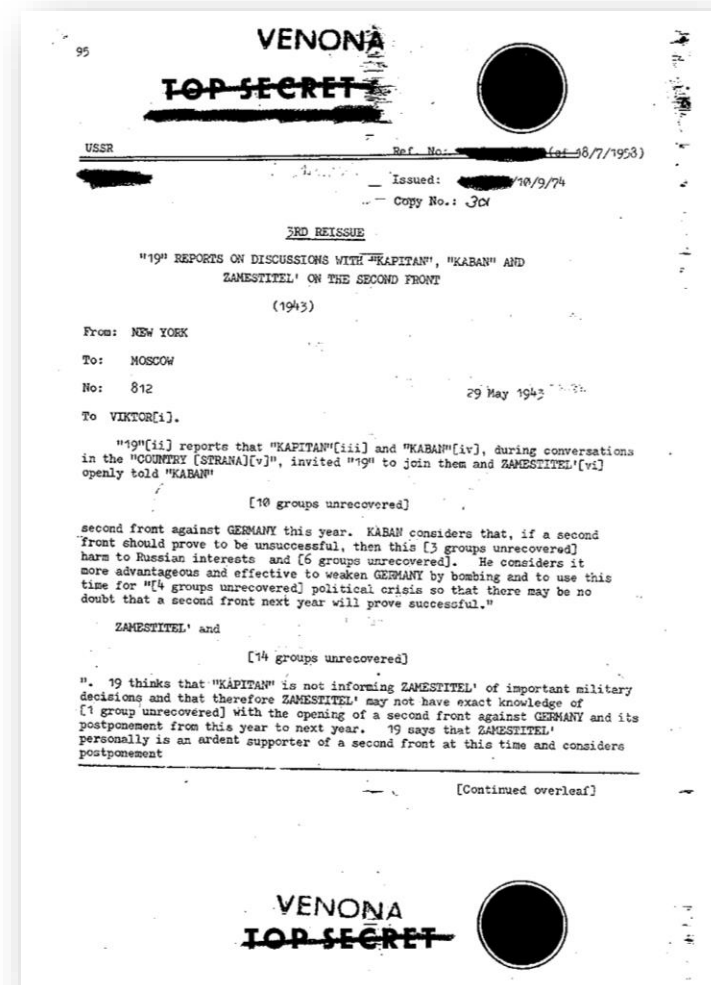
Engangs-blokk

10	15	24	16	6	0	24	1	23	17	21
21	4	7	5	9	4	1	17	3	14	8
24	18	21	12	25	5	13	23	23	3	23
2	11	2	21	24	25	18	23	24	1	0
8	23	13	13	25	8	14	19	20	10	6
2	21	17	23	0	2	25	22	3	3	18
0	15	21	7	25	0	0	5	6	21	5
5	4	16	1	16	12	2	12	10	11	24
8	5	5	9	10	0	9	13	6	23	4
2	15	23	0	1	12	10	12	24	19	14

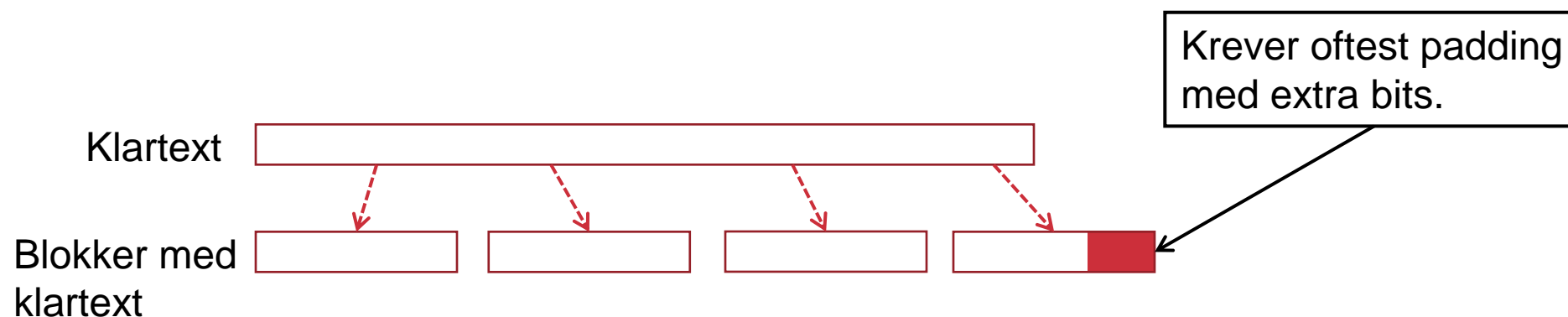
- Både Bob og Alice må ha kopier, som ødelegges etter bruk.
- Hver bokstav forskyves (mod 26) i samsvar med sin indeks-verdi i blokken.
- Eller: Kjør XOR mellom klartekst og nøkkel...

Svakheter ved engangs-blokk

- Nøkkelen må være like lang som klarteksten
- Nøkler må aldri gjenbrukes!!!
 - USA klarte å knekke noen Sovjetiske kryptogrammer statistisk pga gjenbruk av kodeblokk



- Når man bruker **blokk chiffer**:
 - Klartekst og kryptogram har fast lengde b (f.ex., 128 bits)
 - En klartekst med lengde n deles opp i en sekvens av m **blokker**, $P[0], \dots, P[m-1]$, hvor $n \leq bm < n + b$
- Hver melding krypteres og dekrypteres blokkvis



Padding

- Blokkchiffre krever at lengden **n** på klartekst må være et mutippel av blokkstørrelsen **b**
- Padding i siste blokk kan ikke være flertydig (kan ikke bare legge inn 0'er)
- Når blokk-størrelsen og klarteksten er multipler av 8 så er en vanlig padding-metode **PKCS5**: der legges det inn en sekvens med identiske bytes, som hver er lengden på paddingen
- Eksempel for $b = 128$ (16 bytes)
 - Klartekst: "beostby" (7 bytes)
 - "Paddet" klartekst: "beostby99999999" (16 bytes), der 9 er tallet, ikke tegnet
- Vi vil alltid måtte padde siste blokk, som kan bestå nærmest utelukkende av padding

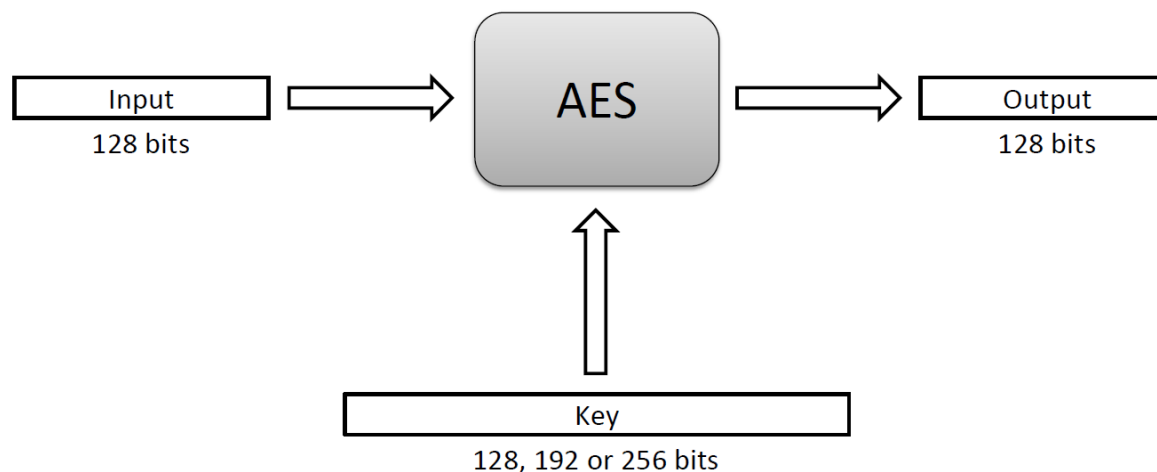
Blokk-chiffer i praksis

- **Data Encryption Standard (DES)**
 - Utviklet av IBM og tatt i bruk av amerikanske standardsinstituttet (NIST) i 1977
 - 64-bit blokk og 56-bit nøkler
 - Såpass få nøkler ("lite nøkkel-rom») at brute-force har vært gjennomførbart siden ca 2000
- **Triple DES (3DES)**
 - Nøstet DES med tre forskjellige nøkler K_A , K_B , and K_C
 - Effektiv nøkkellengde er 168 bit, som gjør brute-force mye vanskeligere
 - $C = E_{K_C}(D_{K_B}(E_{K_A}(P)))$; $P = D_{K_A}(E_{K_B}(D_{K_C}(C)))$
 - Lik DES når $K_A=K_B=K_C$ (backward compatible)
 - Ansett som mulig å knekke for «stater» eller andre med tilstrekkelig regnekraft – brukes ikke lenger
- **Advanced Encryption Standard (AES)**
 - Valgt av NIST i 2001 etter åpen konkurranse og debatt
 - 128-bit blokker og flere mulige nøkkellengder: 128, 192 og 256 bits
 - Ikke knekkbar med dagens teknologi
 - **AES-256 er i dag den foretrukne symmetriske krypteringsmetoden**

AES – mest brukte BLOKKCHIFFER

Advanced Encryption Standard(AES)

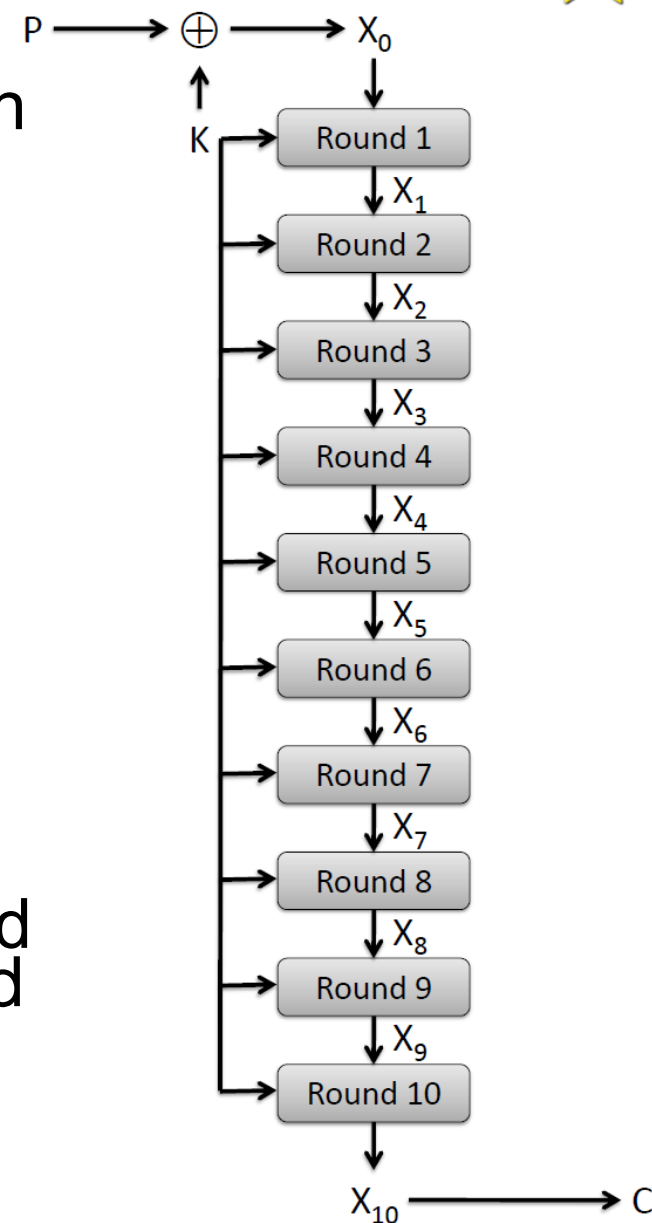
- Opprinnelig kalt Rijndael
- Blokkchiffer
- 128 bit blokker
- Versjoner AES-128, AES-192 og AES-256 ut fra nøkkel-lengde
- Veldig kompleks, men designet for datamaskiner!





AES Runde Strukturen

- AES “baker om” bitsene i meldingen P i flere runder
 - 128 10 runder
 - 192 12 runder
 - 256 14 runder
- Hver runde utfører en inverterbar transformasjon på en 128 bit array, som kalles tilstanden («state»)
- Initial-tilstanden X_0 er XOR av klarteksten P med nøkkelen K :
$$X_0 = P \text{ XOR } K.$$
- Runde i ($i = 1, \dots, 10$) mottar tilstand X_{i-1} som input og produserer tilstand X_i .
- Kryptogrammet C er output etter siste runde: $C = X_{10}$.

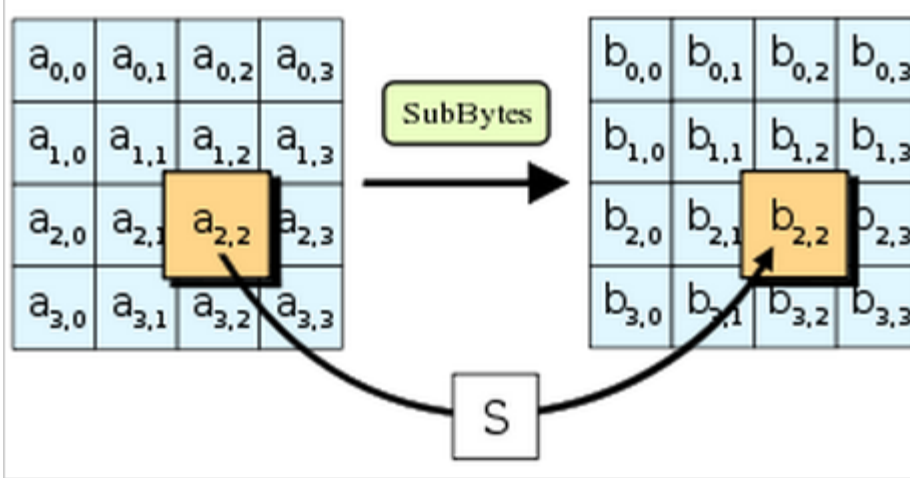


AES Rundene

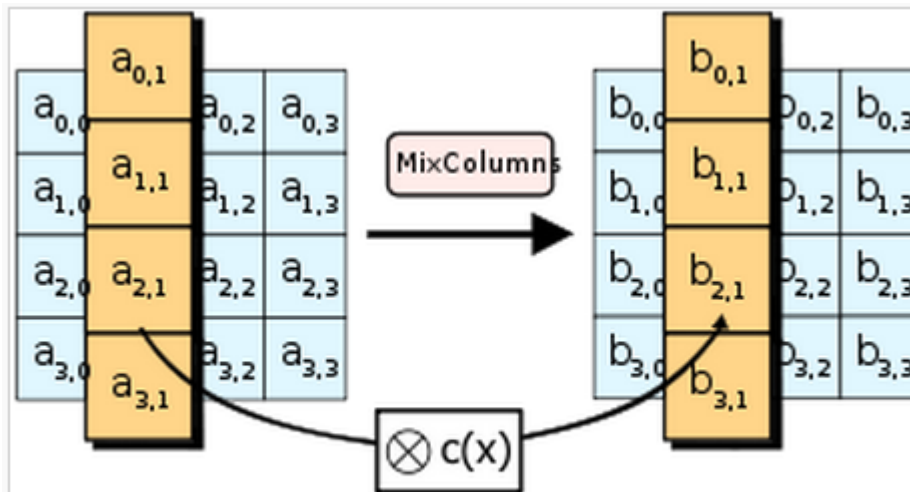


- Hver runde er bygd opp av fire trinn:
 1. **SubBytes step**: et S-box substitueringsstrinn
 2. **ShiftRows step**: a permuteringstrinn
 3. **MixColumns step**: en matrisemultiplikasjon
 4. **AddRoundKey step**: XOR med en egen rundenøkkel (“round key”) som er basert på krypteringsnøkkelen **K**

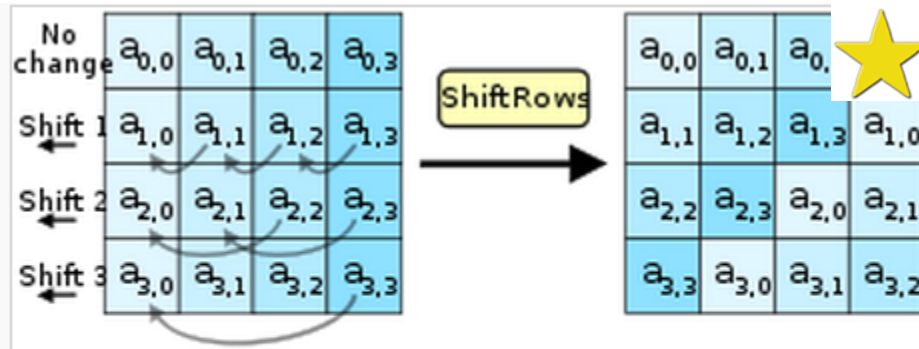
Trinnene



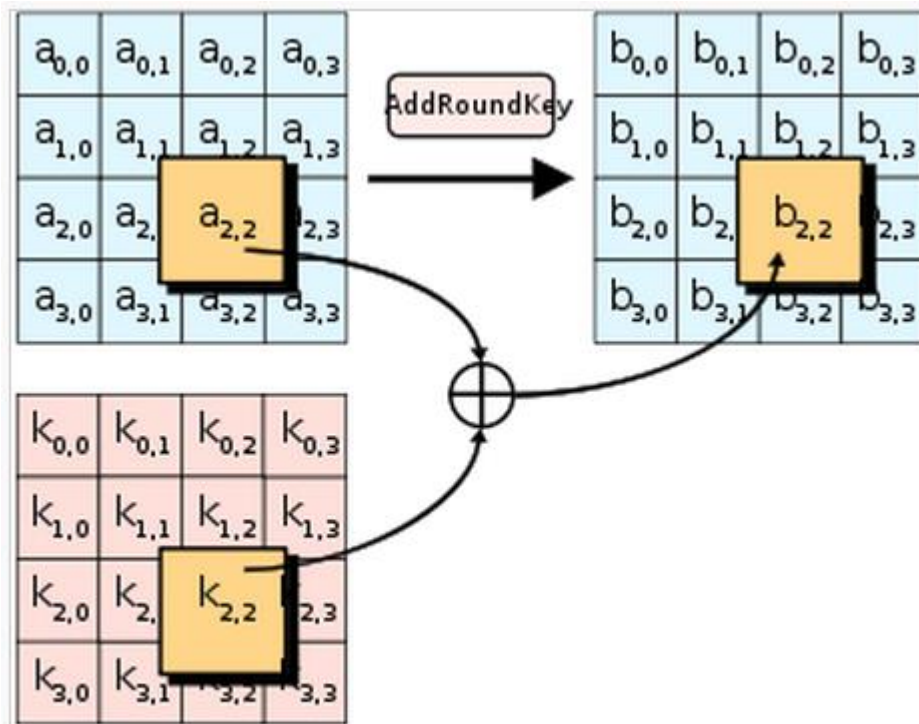
In the **SubBytes** step, each byte in the state is replaced with its entry in a fixed 8-bit lookup table, S ; $b_{ij} = S(a_{ij})$.



In the **MixColumns** step, each column of the state is multiplied with a fixed polynomial $c(x)$.



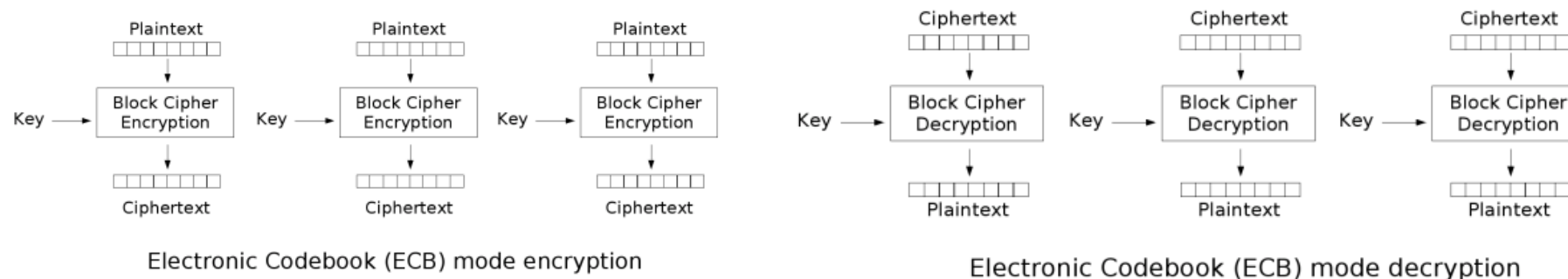
In the **ShiftRows** step, bytes in each row of the state are shifted cyclically to the left. The number of places each byte is shifted differs for each row.



In the **AddRoundKey** step, each byte of the state is combined with a byte of the round subkey using the **XOR** operation (\oplus).

BlokkChiffer Moduser

- AES og andre blokk-ciffre kan kjøres i ulike **moduser**
- Modusen handler om **måten** og **rekkefølgen** kryptering og dekryptering foregår på/i meldings-blokkene
- **Electronic Code Book (ECB) Mode** (er den enkleste):
 - Blokk $P[i]$ krypteres til kryptogramblokk $C[i] = E_K(P[i])$
 - Block $C[i]$ dekrypteres til klartext blokk $M[i] = D_K(C[i])$
- M.a.o. Dette kan foregå i **parallel**

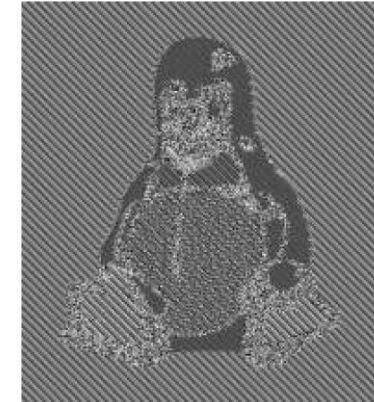


ECB: Styrker og svakheter

- Styrker:
 - Enkelt å gjennomføre
 - Tillater parallell kryptering/dekryptering av blokker
 - Taps-tollerant fordi om en blokk går tapt er de andre intakte
- Svakheter:
 - Noen dokumentformater og bilder egner seg ikke for ECB kryptering fordi mønstre i “klarteksten” kan reflekteres på “avstand” selv om hver blokk er “godt kryptert”



(a)

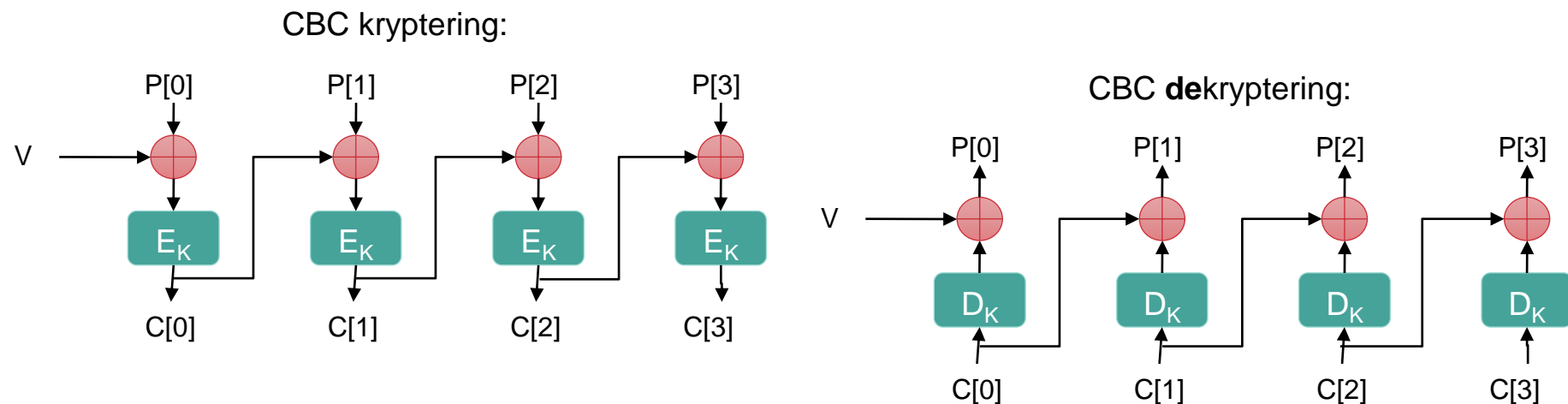


(b)

Figure 8.6: How ECB mode can leave identifiable patterns in a sequence of blocks: (a) An image of Tux the penguin, the Linux mascot. (b) An encryption of the Tux image using ECB mode. (The image in (a) is by Larry Ewing, lewing@isc.tamu.edu, using The Gimp; the image in (b) is by Dr. Juzam. Both are used with permission via attribution.)

Chiffer Blokk Lenking (CBC)

- I Cipher Block Chaining (CBC) Modus
 - kombineres forrige chiffer-text blokk med neste klartext-blokk $C[i]$
 $= E_K (C[i - 1] \oplus P[i])$
 - Initialiserer med $C[-1] = V$, en random blokk som er separat kryptert
 - Dekryptering: $P[i] = C[i - 1] \oplus D_K (C[i])$



CBC: Styrker og svakheter

- Styrker:
 - Avslører ikke mønstre i klarteksten
 - Er den vanligst brukte
 - Relativt rask og enkel
- Svakheter:
 - CBC krever pålitelig overføring av alle blokker, i riktig rekkefølge
 - CBC egner seg ikke for anvendelser som tillater pakketap (f.ex. Musikk- og video-streaming, telekonferanse, ...)

Java AES Kryptering eksempel



- Kilde

<http://java.sun.com/javase/6/docs/technotes/guides/security/crypto/CryptoSpec.html>

- Lage en AES nøkkel

```
KeyGenerator keygen = KeyGenerator.getInstance("AES");  
SecretKey aesKey = keygen.generateKey();
```

- Lage et kryptogram objekt med AES i ECB modus og PKCS5 padding

```
Cipher aesCipher;  
aesCipher = Cipher.getInstance("AES/ECB/PKCS5Padding");
```

- Krypter

```
aesCipher.init(Cipher.ENCRYPT_MODE, aesKey);  
byte[] plaintext = "My secret message".getBytes();  
byte[] ciphertext = aesCipher.doFinal(plaintext);
```

- Dekrypter

```
aesCipher.init(Cipher.DECRYPT_MODE, aesKey);  
byte[] plaintext1 = aesCipher.doFinal(ciphertext);
```


Hvor trygt er egentlig AES?

- Da det ikke er noen kjente svakheter i AES, så må man knekke det med brute force angrep
- For å kontrollere utfall av brute force må man lagre det, så la oss se på det:
 - 128-bit block cipher har 2^{64} known plaintexts
 - $(128 / 8) * 2 * 2^{64}$ bytes = 512 exabytes
 - 512 exabytes = 536.870.912 terrabytes!
- Det var 2^{64} , hver eksponent doubler dette!

Hvor trygt er egentlig AES #2

- AES 256 har 2^{100} plaintexts
- For å forsøke brute force må man som et minimum utføre dekryptering, så la oss se på det:
 - Hva kreves for å utføre 2^{100} kalkuleringer?
 - Ta den kjappeste, mest effektive bit-coin miner, gjør den så en million ganger MER effektiv (bare for å gjøre regnestykket litt mer morsomt)
 - $1957 \text{ (MHash/J)} * 2 \text{ (double hash)} / 1.000.000 \text{ (million times better)} * 1000 \text{ (kilohash)} * 1000 \text{ (hash)} * 64 \text{ cycles/hash} / 5 \text{ cycles/AES} * 2^{100} / 3600 \text{ (watt-hours)} / 1000 \text{ (kilowatt)} / 1000 \text{ (megawatt)} / 1000 \text{ (gigawatt)}$
- $1.76 * 10^{22}$ Gigawatt-timer
- Verdens elektriske produksjon: $2,09 * 10^7$ GWh
- All elektrisitet på jorda vil bli brukt i $8,42 * 10^{14}$ år for å brute force knekke AES 256!
 - Merk at universet kun har eksistert $1,4 * 10^{10}$ år...
- Med andre ord er den **ENESTE** måten å "knekke" AES på å få tak i nøkkelen, det vil si at brukerens passord er det eneste svakeste leddet.

Windows Encrypted File System (EFS)

- Innebygget kryptering i Windows 2000 ->
- Tilgjengelig gjennom Properties for en fil eller mappe
- Nøkkel lagret i user.dat (men serifikater kan eksporteres for backup)
- Bruker AES som default (kan bruke 3DES og DESX)
- Beskytter ikke hvis et program er logget inn med rettigheter til å lese
- Benny Pinkas mener å ha knekt Windows sin implementasjon av dette gjennom å angripe en svakhet i implementasjonen av PRNG.
- Microsoft benekter at dette er en svakhet
- Dette er i stor grad korrekt da svakheten har to forutsetninger, 1) du må ha tilgang til maskinens admin konto (og da kan du lese alle filer i EFS uansett) og 2) maskinen må ikke være skrudd av siden filen ble kryptert



STREAM-CIPHER

(Hvorfor en liten stjerne oppe til høyre? Ingen praktisk grunn for å bruke stream cipher, bruk block cipher istede...)

Strøm-chiffer (Stream Cipher)



- Nøkkel-strøm
 - Pseudo-random sekvens av bits $S = S[0], S[1], S[2], \dots$
 - Kan genereres online en bit (byte) om gangen
 - Kan benytte en envegsfunksjon som både sender og mottager kjenner/bruker
- Strøm chiffer
 - XOR klartexten med nøkkel-strømmen $C[i] = S[i] \oplus P[i]$
 - Egnet for klartext med uforutsigbar lengde, f.ex. Media stream, telefonsamtale, telekonferanse,...
- Synkron strøm chiffer
 - Nøkkelstrømmen får man fra en felles hemmelig nøkkel K
 - Fungerer i upålitelige kanaler forutsatt sekvensnummer i pakkene
 - Kan også baseres på tidligere pakker...

Nøkkel strøm generatorer



- **RC4**
 - Designet i 1987 av Ron Rivest for RSA Security
 - Nøkler på inntil 2,048 bits
 - Enkel algoritme for generering av pseudo-tilfeldige tall
 - Ikke lenger regnet som sikker siden analyse av oppstart kan avsløre nøkkelen (les Fluhrer, Mantin, Shamir hvis du er interessert)
 - Basis for WEP (derfor bruker man helst WPA2 I 802.11i, som baserer seg på AES)

Angrep på Strøm Chiffer



- **Repetisjonsangrep**
 - Dersom en nøkkel-strøm gjenbrukes kan nøkkelen lett finnes ved sammenligning (f.eks. Har da XOR av to klartexter...)
- **Insertion angrep** [Bayer Metzger, TODS 1976]
 - Sende om igjen en valgt klartext med
 - En valgt byte lurt inn av angriperen
 - Bruker samme nøkkelstrøm
 - Kan da regne seg frem til nøkkel ut fra registrerte forskjeller

Original

P	P[i]	P[i+1]	P[i+2]	P[i+3]
S	S[i]	S[i+1]	S[i+2]	S[i+3]
C	C[i]	C[i+1]	C[i+2]	C[i+3]

Omsending

P	P[i]	X	P[i+1]	P[i+2]
S	S[i]	S[i+1]	S[i+2]	S[i+3]
C	C[i]	C'[i+1]	C'[i+2]	C'[i+3]

Public Key kryptering

Kalles også ASYMMETRISK

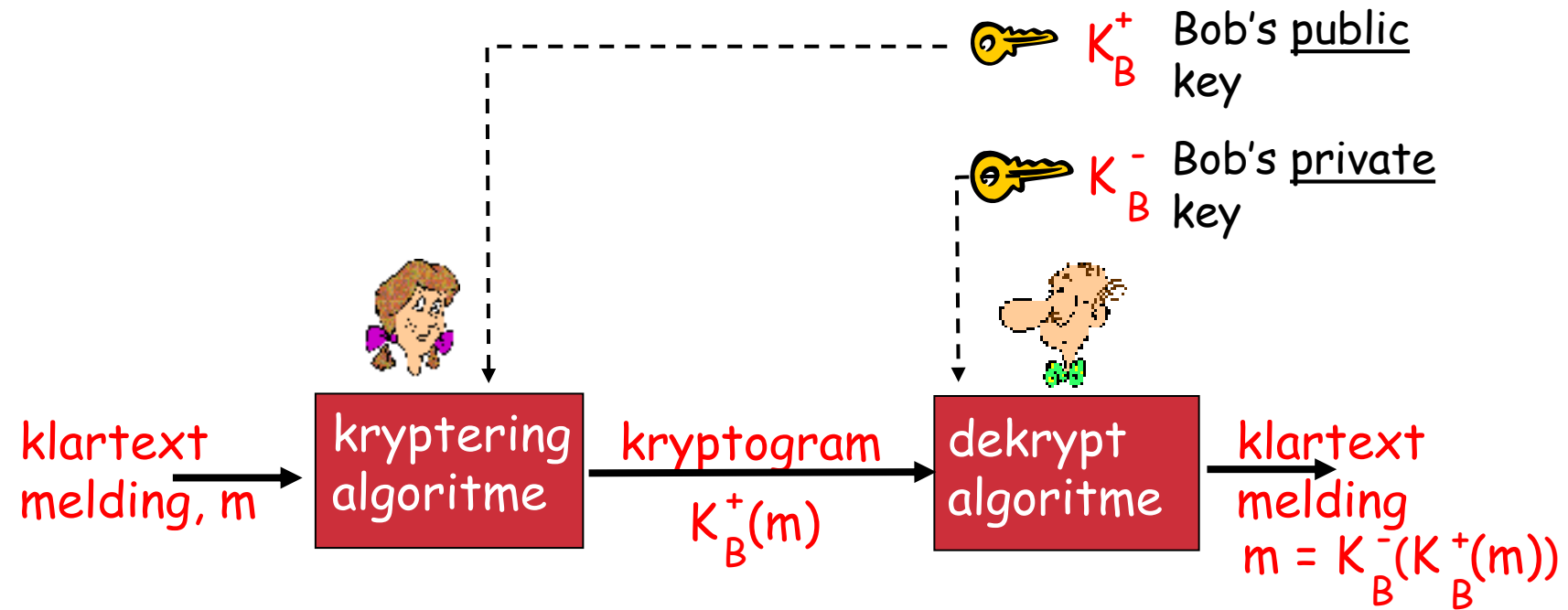
- Funksjoner/metoder som er slik at gitt $f(x)$
 - “Enkelt” å beregne $y = f(x)$
- Men gitt y
 - Beregningsmessig (nærmest) praktisk umulig å finne x
- Eksempel: $f(x) = (x^{173}) \bmod 65537$ (?)
 - $65^{173} =$

430537217011657199463939167152843644

- , mens $(65^{173}) \% 65537 = 12115$
- Hvor lett er det å finne tilbake til 65 når du får oppgitt 12115?

- Når vi først ganger tallet med seg selv mange ganger
- ..og så tar modulus (restdivisjon)
- så *fjerner vi informasjon* som trengs for å finne tilbake til opphavstallet...
- Dette er også kjent som **diskret logaritme problemet**, og vi *tror* (pr 2021) at det er ekstremt vanskelig å løse

Public key kryptografi

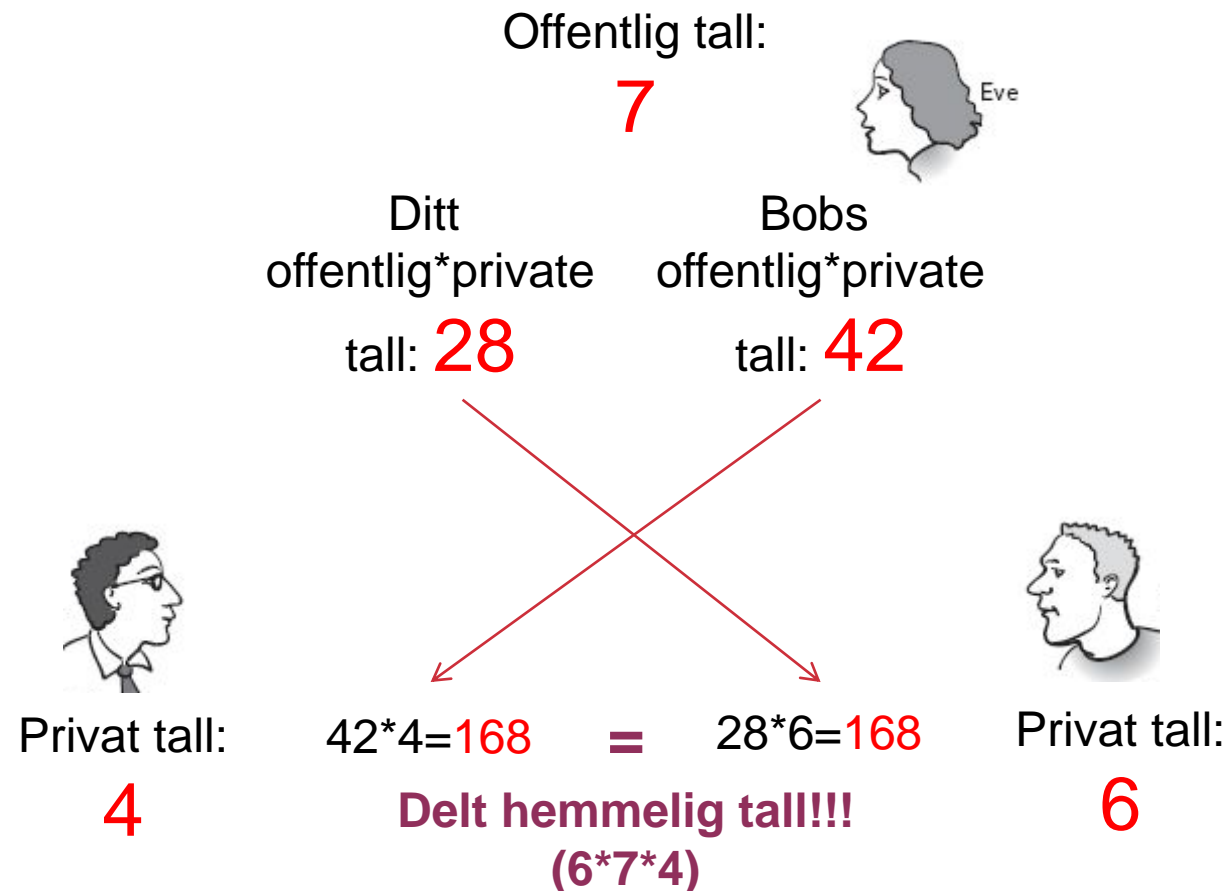


Primtall

- Public Key kryptering bygger på noen egenskaper ved primtall
 - Den felles kunnskapen/nøkkelen er i bunn og grunn tallteori (primtall), og det er vanskelig/kostbart og tidkrevende å faktorisere store tall
 - Det finnes ca 4 M primtall under 10 G, og de blir stadig sjeldnere for større tall (så langt...)
 - Primtallteoremet
Antall primtall mindre enn N er ca $N/\ln(N)$
 - Det er mange av dem...

Eksempel: Basis for D-H nøkkelutv.

- Tenk deg at multiplikasjon er en enveisfunksjon!
 - Lett å multiplisere («gange»)
 - Veldig, veldig vanskelig å dividere («dele»)
- Du vil dele en felles krypteringsnøkkel med Bob, men ikke Eve



Diffie-Hellman

- Divisjon er ikke vanskelig nok
- Vi *tror* at diskrete logaritmer er tilstrekkelig vanskelig.
 - Å finne verdien til x når du vet at f.eks. $2^x \% 11 = 3$ krever masse testing, og blir vanskelig når tallene er store!
 - Og se gangetabellen for $\%11$ øverst til venstre...
- Starter med å velge en basis (f.eks. 2) og en modulus (f.eks. 11)
 - Modulus må være et primtall:

	1	2	3	4	5	6	7	8	9	10
1	1	2	3	4	5	6	7	8	9	10
2	2	4	6	8	10	1	3	5	7	9
3	3	6	9	1	4	7	10	2	5	8
4	4	8	1	5	9	2	6	10	3	7
5	5	10	4	9	3	8	2	7	1	6
6	6	1	7	2	8	3	9	4	10	5
7	7	3	10	6	2	9	5	1	8	4
8	8	5	2	10	7	4	1	9	6	3
9	9	7	5	3	1	10	8	6	4	2
10	10	9	8	7	6	5	4	3	2	1

n	2^n	3^n	6^n
1	2	3	6
2	4	9	3
3	8	5	7
4	5	4	9
5	10	1	10
6	9	3	5
7	7	9	8
8	3	5	4
9	6	4	2
10	1	1	1

Offentlige tall:

$\% = 11, b = 2$



Ditt offentlig*private
tall: $(2^8)\%11 = 3$

Bobs offentlig*private
tall: $(2^9)\%11 = 6$



Privat tall:

8

$(6^8)\%11 = 4 = (3^9)\%11 = 4$

Delt hemmelig tall!!!



Privat tall:

9

Se film...



- Se gjennom <http://www.youtube.com/watch?v=3QnD2c4Xovk>
- Diffie-Hellman nøkkel-bytte.

Hva med kryptering?

- Diffie-Hellman utveksler (symmetriske) nøkler med private/public nøkler – men det er ikke noe kryptering
- Whitfield Diffie og Martin Hellman publiserte metoden i 1976
- Det hadde vært bra med samme "teori" for kryptering, det jobbet på det tidspunktet mange med...
- I 1977 kom Ron Rivest, Adi Shamir og Leonard Adleman opp med en metode:

RSA: Nøkkelvalg

1. Velg to store primtall p, q .
(f.eks. 1024 bit, dvs min. 100 desimale siffer hver)
2. Beregn $n = pq$, og $z = (p-1)(q-1)$
3. Velg krypteringsnøkkelen e (der $e < n$) slik at den har ingen felles faktorer med z . (e, z er "relative primtall").
4. Velg dekrypteringsnøkkelen d slik at $ed-1$ er "exakt delbar" på z . (m.a.o.: $ed \bmod z = 1$).
5. Public key er $\underbrace{(n, e)}_{K_B^+}$. Private key er $\underbrace{(n, d)}_{K_B^-}$.

RSA: Kryptering, dekryptering

0. Gitt off. krypteringsnøkkel (n,e) og privat (n,d) dekrypteringsnøkkel

1. For å **kryptere** bit mønsteret, m , beregn

$$c = m^{e \bmod n}$$

2. For å **dekryptere** mottatt bit mønster, c , beregn

$$m = c^{d \bmod n}$$

Magi?!
$$m = \underbrace{(m^{e \bmod n})}_c^{d \bmod n}$$

RSA (leke-)eksempel:

- Gitt at dere har en public key hvor $n = 3233$ og $e = 17$, og deres privat key er $n = 3233$ og $d = 2753$. Dekrypter min melding til dere:
- 3000 28 2726 2726 1307 1992 641 2726 2790 2680 2680

$$M = C^d \bmod n = 3000^{2753} \bmod 3233 = 72 = \text{bokstaven 'H'}$$

72	69	76	76	79	32	67	76	65	83	83
H	E	L	L	O		C	L	A	S	S

Hvorfor kan ikke EVE dekryptere?

- Dersom Eve avlytter, hvorfor kan hun ikke dekryptere?
- Eve kjenner
 - Bob's offentlige nøkkel **(n,e)**, som Alice brukte til å kryptere, og har avlyttet meldingen
 - For å kunne dekryptere må hun kunne faktorisere **n** til $p \cdot q$ slik at hun har grunnlag for å gjette dekrypteringsnøkkelen **d**
 - Å faktorisere store tall tar (vanligvis) flere år enn Eve kan forvente å leve!!!
 - Hun kommer aldri til å finne **p** og **q**, kan dermed heller ikke finne noen **d**
 - Hvis hun da ikke utvikler en fungerende fullskala **kvante-computer!**

Hvem vet hva?

Alice

- Skal sende og trenger bare å kjenne offentlig krypteringsnøkkel $K^+ = (e, n)$ (samt exponering e og mod n)

Eve

- Kan avlytte
- Kjenner kun Bobs offentlige **krypterings-**nøkkel $K^+ = (e, n)$
- Må kunne faktorisere n for å ha en sjanse til å finne p og q , og så gjette d

Bob

- Fant p og q
- Beregnet og publiserte n
- Valgte og publiserte e
- Valgte d , og holdt den hemmelig
- Eneste som kan dekryptere innen rimelig tid.

RSA: Hvorfor er

$$m = (m^e \bmod n)^d \bmod n$$



Tallteoretisk teorem: Hvis p, q primtall og $n = pq$, så:

$$x^y \bmod n = x^{y \bmod (p-1)(q-1)} \bmod n$$

$$(m^e \bmod n)^d \bmod n = m^{ed} \bmod n$$

$$= m^{ed \bmod (p-1)(q-1)} \bmod n$$

(i henhold til teoremet)

$$= m^1 \bmod n$$

(siden vi **valgte** ed slik at den er delbar på $(p-1)(q-1)$ med rest 1)

$$= m$$

RSA: en annen viktig egenskap



Følgende resultat er *svært* nyttig:

$$\underbrace{K_B^-(K_B^+(m))}_{\text{bruker public key først, fulgt av private key}} = m = \underbrace{K_B^+(K_B^-(m))}_{\text{bruker private key først, fulgt av public key}}$$

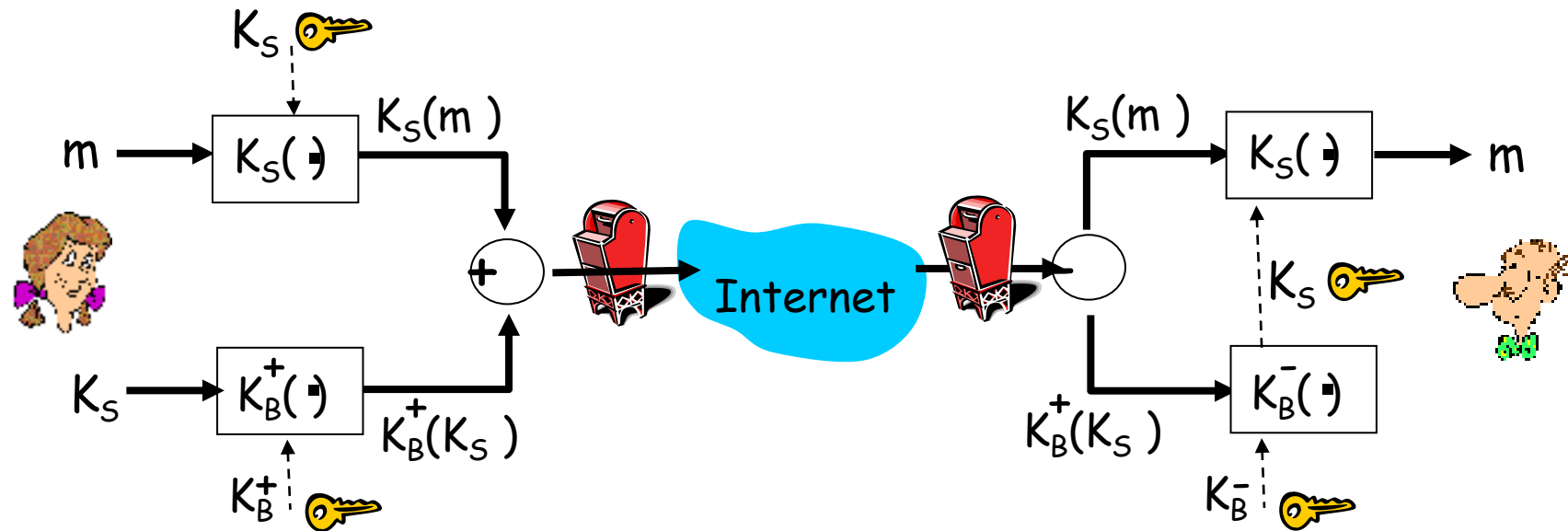
bruker public
key først, fulgt
av private key

bruker private
key først, fulgt
av public key

Samme resultat!
(svarer til $a^{xy} = a^{yx}$)

Sikker epost: kryptering

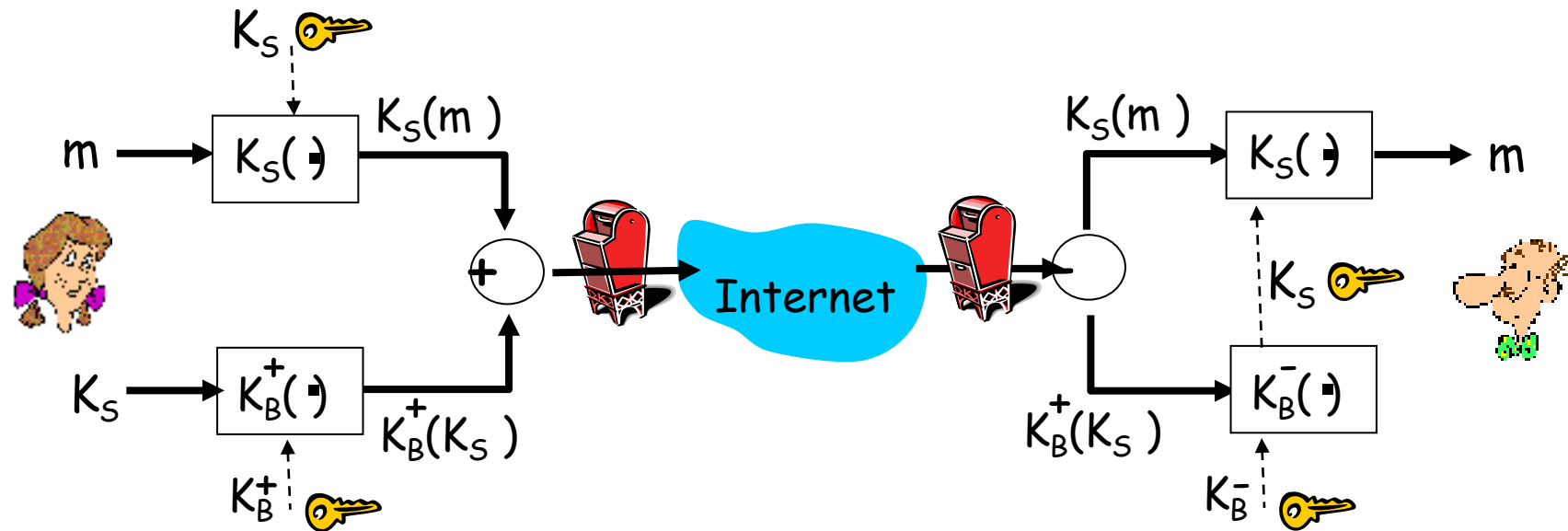
□ Alice sende hemmelig e-mail, m , til Bob.



Alice:

- generer random *symmetrisk* privat nøkkel, K_S .
- krypterer meldingen med K_S (effektivitet)
- Krypterer K_S med Bobs offentlige nøkkel
- sender både $K_S(m)$ og $K_B(K_S)$ til Bob.

Sikker epost: dekryptering

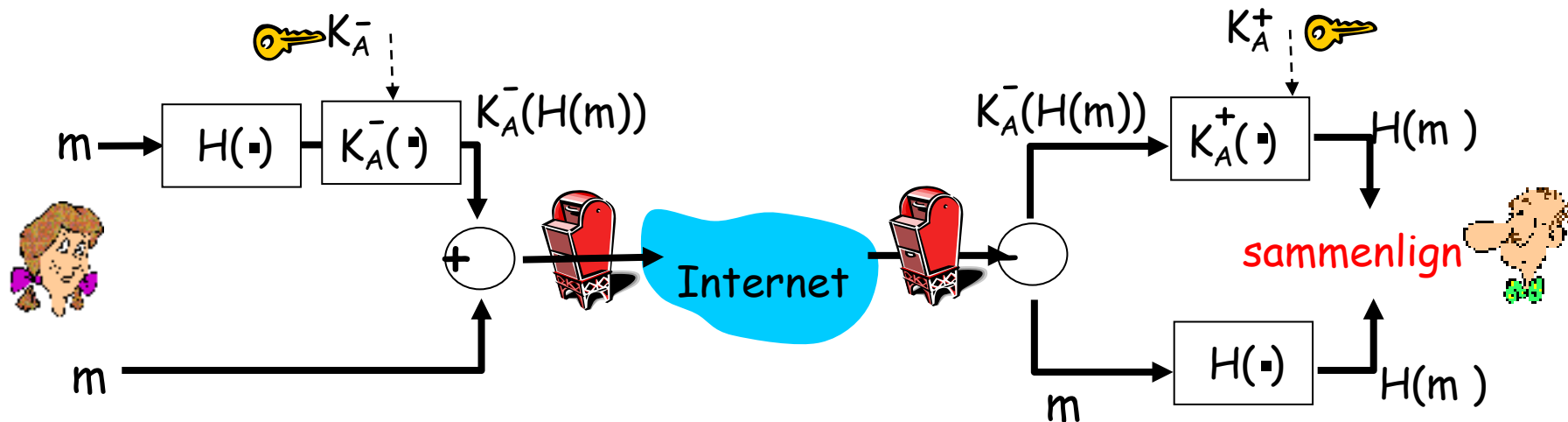


Bob:

- ☐ benytter sin private nøkkel til å dekryptere K_S
- ☐ benytter så nøkkelen hun fikk (K_S) til å dekryptere $K_S(m)$ for å finne m

Sikker epost

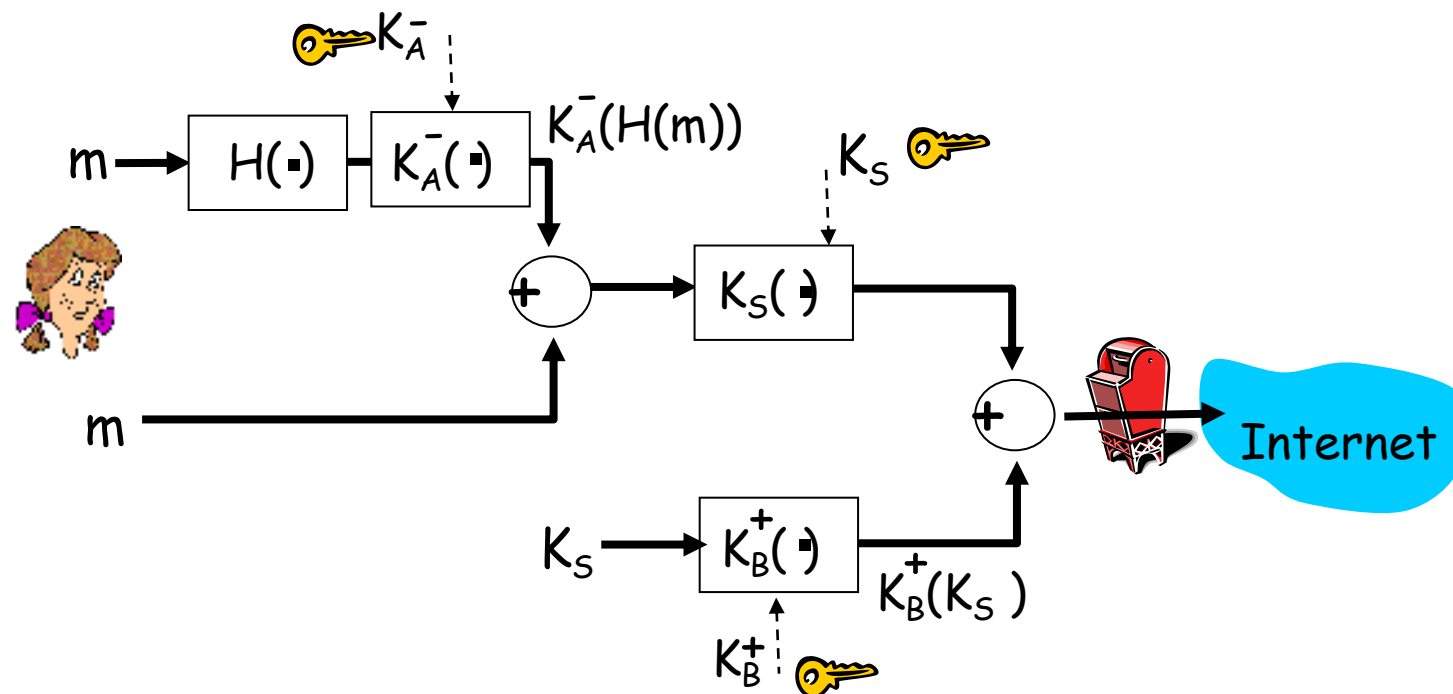
- Alice vil sikre at det kun er hun som kan være avsender og at ingen har fiklet med meldingen



- Alice **signerer** meldingen digitalt.
- sender både meldingen (i klartekst) og digital signatur.

Sikker e-mail

- Alice vil sende en hemmelig melding, som er autentisert og "fikle-sikker"



Alice bruker tre nøkler: sin egen private, Bob's offentlige, og en nylaget symmetrisk (AES?)

Hvor trygt er egentlig RSA?

- RSA-768
- Et semi-primtall (et tall som er ett produkt av to nesten jamstore primtall)
- 768 bit – 232 desimalsiffer (skriver du et tall i sekundet bruker du ca 4 minutter på å punche det...)
- Faktoriseringen ville tatt ca 2000 år på en 2,2 GHz prosessor; men ble gjennomført som et samarbeid mellom flere forskningsinstitutt over to år.
- Kan også få indikasjoner ved å måle Cache-bruk, tid det tar å beregne ut fra offentlig nøkkel mm
- INGEN VET SIKKERT HVOR VANSKELIG DETTE EGENTLIG ER!
- Merk: Vi bruker ikke så små nøkler som 768 bits, vi bruker minimum 8196 bits 😊

Hvor trygt er egentlig RSA? #2

- RSA-768 – ansett som knekket
- RSA-1024 – ansett som at den KAN knekkes av nation state attacker
- RSA-4096 – standard i dag, kan ikke knekkes
- RSA-8196 – «Future proof»

https://en.wikipedia.org/wiki/RSA_Factoring_Challenge

Vær obs på at kvantemaskiner vil kunne faktorisere tall ekstremt mye fortere enn dette (AES er «quantum-proof», RSA er ikke...) – men det er anslått å kreves en 4000+ qubits maskin (dagens kraftigste er på 65 qubits)

SSL / TLS

SSL/TLS



- Bruker key-exchange algoritme for å utveksle krypteringsnøkkel
 - RSA, Diffie-Hellman, ECC (eller SRP, PSK)
- Bruker asymmetrisk algoritme for å autentisere
 - RSA, ECC (eller DSA)
- Kryptering av data med symmetrisk nøkkel
 - AES (eller RC4, 3DES, Camellia, RC2, IDEA)
- Mest grunnleggende angrep er "downgrade" attacks, hvor MITM lurder hver side til å bruke en svak nøkkel (for eksempel eNULL cipher ;-)

<https://www.openssl.org/docs/man1.0.2/apps/ciphers.html>

SSL/TLS ved OpenSSL

- Ved implementasjon av TLS og/eller krypto systemer (i C/C++) anbefales det å bruke OpenSSL biblioteket
- Hvis du har kontroll på både server og klient kan du begrense cipher suites til de sikreste
- Eksempel på "cipher suite" format i TLS 1.2
 TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
 - Key exchange: ECC (ECDHE)
 - (Server) authentication: RSA
 - Encryption: AES 128, GCM mode
 - Message Authentication Code: SHA256

Videre lesning



- TLS (og OpenSSL) er dype emner (kunne vært et 10 studiepoengs fag ved skolen i seg selv), anbefaler interesserte å lese videre som selvstudie

https://en.wikipedia.org/wiki/Transport_Layer_Security

https://en.wikipedia.org/wiki/Cipher_suite

<https://en.wikipedia.org/wiki/OpenSSL>

<https://www.openssl.org/>

<https://www.openssl.org/docs/man1.0.2/ssl/>

<https://www.openssl.org/docs/man1.0.2/crypto/>

SIGNATURER, SJEKKSUMMER OG HASHFUNKSJONER

Ting som legges ved for å sikre autensitet og assurance på data
MD5, SHA o.l.

- En **hash funksjon** h transformerer en klartext P til en verdi $x = h(P)$ med en forhåndsbestemt lengde, som kalles hashverdien eller forkortingen (“digest”) av P
 - En **kollisjon** er når et par klartekster P og Q får samme hash verdi, $h(P) = h(Q)$
 - Kollisjoner er uunngåelige
 - Beregningstiden for x bør være proporsjonal med lengden på input P
 - Hash-funksjonen bør sørge for at klartextene spres uniformt ut over hash-verdiene

- Kryptografiske hash-funksjoner er
 - Enveis funksjoner
 - ***Kollisjonsresistente***
 - Det er vanskelig å finne to ulike klartexter som gir samme hashverdi
- Hash-verdier anbefales å være minimum 256 bit for å beskytte mot brute-force generering

Bruk av krypto-hash

- Typisk bruk av kryptografisk hash er å sikre at en fil/melding ikke har blitt endret (f.eks. injisert med et virus)
 - Signering av drivere
 - Signering av installasjonsfiler

Message-Digest Algorithm 5 (MD5)



- Utviklet av Ron Rivest i 1991
- Bruker 128-bit hash verdier
- Fremdeles mye brukt til tross for at den regnes som usikker
- Mange svakheter er funnet
- Chosen-prefix collisions attacks funnet av Marc Stevens, Arjen Lenstra & Benne de Weger
 - Start med to tilfeldige klartexter P og Q
 - Man kan beregne verdier/strenger S1 og S2 slik at dersom de konkatineres (føyes til på slutten) av P og Q, så vil P||S1 og Q||S2 kollidere på MD-5 verdi ved ca 250 hash-beregninger
 - Med denne tilnærmingen kan man lage to forskjellige eksekverbare filer, PDF e.l. samme MD5 hash

Secure Hash Algorithm (SHA)

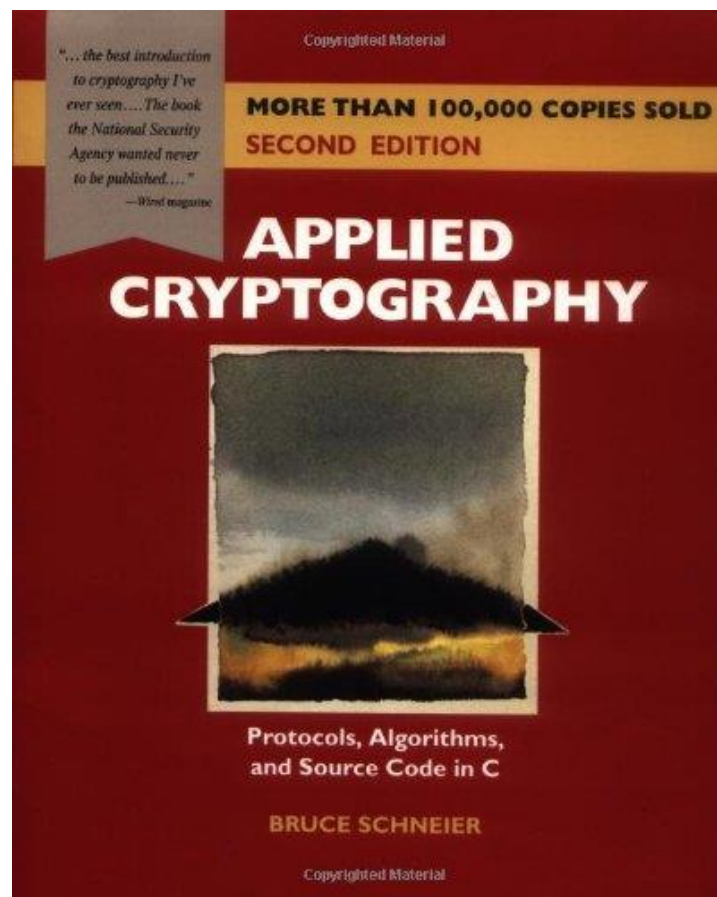
- Utviklet ved NSA og godkjent av NIST
- SHA-0 og SHA-1 (1993)
 - 160-bit
 - Regnes som usikker
 - Fremdeles i bruk
 - Mindre sårbar enn MD5
- SHA-2 (2002)
 - 256 bits (SHA-256) eller 512 bits (SHA-512)
 - Finnes publiserte angrepsteknikker, men regnes fremdeles som sikre
- Offentlig konkurranse om SHA-3 startet i 2007
 - Keccak algoritmen ble valgt som SHA-3, men det er mange kontroverser rundt interne endringer som ble tvunget inn av NIST, som gjør at mange i dag ikke stoler på den
 - Brukes ikke, av frykt for at den er bevisst svekket (av NSA?)

Avslutning

Vil du lære mer?



- Den mest essensielle boken om kryptering som alle som vil jobbe med det må lese 😊



Hva skal vi kunne?

- Forklare hva kryptering er og brukes til.
 - Ikke bare konfidensialitet, men også integritet!
- Beskrive hva en kryptografisk nøkkel er og hvordan den anvendes
- Forklare viktigste forskjeller på symmetrisk og asymmetrisk (public key) kryptering
- Forklare prinsippet bak et substitusjon-chiffer og eksemplifisere ulike typer.
- Beskrive hva som kjennetegner AES og de ulike modusene denne algoritmen kan benyttes i.
- Forklare i hvilke tilfeller man vil foretrekke strøm- fremfor blokk-chiffer (et lurespørsmål...)
- Beskrive Diffie-Hellman og RSA og beskrive hvorfor disse betraktes som sikre pr i dag (med hvilke forutsetninger).
- Beskrive anvendelsesområdene for RSA
- Forklare hva det er, samt hvor og når det er aktuelt å bruke, hashfunksjoner og digitale signaturer

- Øvingsoppgavesett 'TK2100_F01_øvingsoppgaver.pdf' i Canvas

Praktisk oppgave:

- Installer OPENSSL
 - Som source kode: <https://www.openssl.org/source/>
 - Som binaries: <https://wiki.openssl.org/index.php/Binaries>
- Dette er som nevnt i forelesning det mest populære biblioteket for å implementere SSL og TLS kryptering, og brukes også til all annen kryptering
- Biblioteket er skrevet i C
- Har også kommandolinje verktøy – som dere skal sette dere litt inn i
- > OPENSSL HELP

Praktisk oppgave:

1. Test AES kryptering

- `Openssl enc -help`
- `Openssl enc -aes-256-cbc -salt -in file.txt -out file.txt.enc -k "password"`
- `Openssl enc -aes-256-cbc -d -in file.txt.enc -out file.d.txt -k "password"`

2. Lag en SHA256 hash

- `Openssl dgst -help`
- `Openssl dgst -sha256 -out file.txt.hash file.txt`
- Endre EN byte i filen, og se hvor forskjellig SHA hashen er...

3. Dekrypter fil du ikke kjenner passordet til

- Sammen med oppgavesettet på Canvas ligger en fil `TK2100_F01_kryptert.dta`
- Passordet er en PIN kode på 4 tall, dvs koden er mellom 0000 og 9999, dere må knekke dette passordet med brute-force – filen er kryptert med kommandoen i oppgave 1
- Dette er en problemløsningsoppgave, hvordan kan dere lage en loop i shell?

Praktisk oppgave:

4. Generer tilfeldige tall

- openssl rand -hex 10

5. Test RSA kryptering

- Her får dere få instruksjoner og må finne ut selv
- Først må dere lage et nøkkelpar med 'openssl genpkey'
- Så krypterer dere med 'openssl rsautl -encrypt'
- Bruk hjelpefunksjonene '-help', og sitter dere fast bruk Google