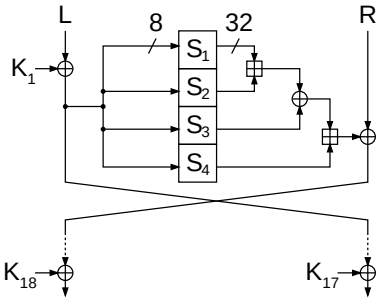
Blowfish
----

* nepatentovaný algoritmus, navržený pro implementaci na 32 bitových CPU

* 64-bitový blok, proměnná délka klíče od 32 do 448 bitů
* 16 iterací Feistelovské sítě
* v první fázi je provedena expanze klíče - vytvoří 18 podklíčů a obsah
  čtyř S-boxů (8 bitů vstup, 32 bitů výstup, S box jako pole)
 - pro vytvoření podklíčů a S-boxů se zapotřebí 521 iterací algoritmu Blowfish



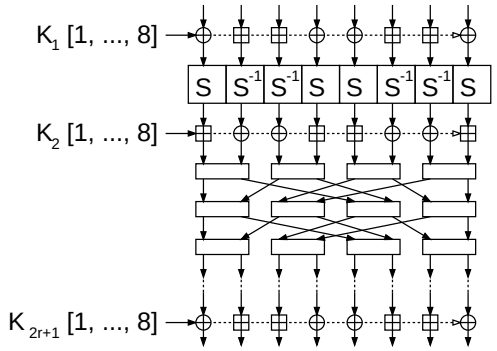
* dešifrování stejné jako šifrování, podklíče se použijí v opačném pořadí

Útoky:
* Vaudenay 1995 našel množinu slabých klíčů, klíč je slabý s p=2^{-14},
  ale nenašel se zatím žádný způsob jak toho využít pro útok

SAFER
-----

* dalším zajímavým volně použitelným algoritmem je SAFER (Secure and Fast
  Encryption Routine)
* navržen pro použití na čipových kartách apod. - 8 bitové mikroprocesory,
  omezená paměť
* několik variant: původní verze SAFER K64 měla 64 bitový klíč
 - Singapurské Ministerstvo vnitra navrhlo SAFER K128
 - varianty SAFER SK64 a SK128 odstraňují slabinu v tvorbě podklíčů -
   měly by se používat místo K64 a K128

* 64-bitový blok, varianta SK64 64-bitový klíč atd.
* není Feistelovská šifra, používá odlišné operace pro šifrování a
  dešifrování
* na začátku vstupuje 8 bytů, pak 8 iterací:
 - XOR-add s podklíči
 - substituce, obsah S-boxů S(x) = 45^x mod 257, fce je invertovatelná
 - add-XOR s podklíči, sčítání a XOR je zaměněno
 - třívrstvá pseudo-hamardova transformace: f(L, R)=(2L+R, L+R)
* na konci výstupní transformace - XOR-add s podklíči



* dešifrování inverze operací v opačném pořadí:

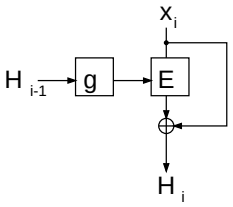
```
 - opak výstupní transformace - XOR-sub podklíčů K_{2r+1}
 - pak opak iterací (inverzní PHT: f(L, R)=(L-R, -L+2R), sub-XOR s podklíči,
   substituce - použita inverze S-boxů, XOR-sub s podklíči)

Útoky:
* útoky proti plné verzi SAFER 64SK a SAFER 128SK zatím nejsou známy


AES
---

* DES malá délka klíče, 3DES pomalý a jisté problémy, EES se neuchytil
* proto 1997 NIST konkurs na AES (Advanced Encryption Standard), několik délek
  klíčů a délek bloku (128, 192 a 256 bitů)
* v roce 2002 vybrán algoritmus Rijndael (Daemen, Rijmen)

* zde popis pro variantu s délkou bloku 128 a klíčem 128 bitů:
 - data i klíč chápány jako matice 4x4 byty
 - na začátku XOR vstupních dat a podklíče
 - pak 10 iterací:
  . ByteSub     - substituce 8 bitů -> 8 bitů všech dat
  . ShiftRow    - cyklický posun řádků dat o 0, 1, 2 a 3 pozice
  . MixColumn   - násobení sloupců konstantním polynomem
  . AddRoundKey - XOR dat a podklíče
 - pak závěrečná iterace - má vynecháno MixColumn

* AES může být rychlý v HW a SW, na 8 bitech i na >= 32 bitových CPU
* zatím ale nový, před praktickým používáním dobré počkat cca 1-2 roky


Poznámka pro zajímavost (algoritmus TEA).
.........................................
 * TEA = Tiny Encryption Algorithm, Wheeler a Needham 1995
 * blok 64 bitů, klíč 128 bitů
 * zatím nalezené nedostatky:
  - ke každému klíči k = (k_0, k_1, k_2, k_3) existují tři další ekvivalentní
    klíče:  (k_0 xor M, k_1 xor M, k_2, k_3)
            (k_0, k_1, k_2 xor M, k_3 xor M)
            (k_0 xor M, k_1 xor M, k_2 xor M, k_3 xor M), kde M=80000000H
    (každé iteraci vyruší dva nejvyšší bity klíče)
  - klíč by bylo možno nalézt, pokud by oponent mohl zvolit vztah mezi dvěma
    klíči a účastník komunikace by oběma klíči zašifroval 2^{32} oponentem
    zvolených datových bloků (tento útok je poněkud nepraktický)

        /* Šifrování: klíč = k[0]..k[3]
                      data = v[0] a v[1] */
        void encrypt(word32 v[2], word32 k[4])
        {
            word32 y = v[0], z = v[1], sum = 0,
                   delta = 0x9e3779b9, n = 32;

            while (n-- > 0)
            {
                sum += delta;
                y += (z << 4) + k[0]  ^  z + sum  ^  (z >> 5) + k[1];
                z += (y << 4) + k[2]  ^  y + sum  ^  (y >> 5) + k[3];
             }
             v[0] = y; v[1] = z;
        }

        /* Dešifrování: klíč = k[0]..k[3]
                        data = v[0] a v[1] */
        void decrypt(word32 v[2], word32 k[4])
        {
            word32 y = v[0], z = v[1],
                   delta = 0x9e3779b9, n = 32, sum = n*delta;

            while (n-- > 0)
            {
```

```
                z -= (y << 4) + k[2]  ^  y + sum  ^  (y >> 5) + k[3];
                y -= (z << 4) + k[0]  ^  z + sum  ^  (z >> 5) + k[1];
                sum -= delta;
            }
            v[0] = y; v[1] = z;
        }
[]
```

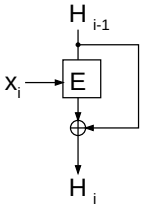
MDC založené na blokových šifrách

* každá zašifrovaná zpráva by měla obsahovat nějakou redundanci
* např. představme si firmu, pobočky posílají na ústředí počet objednaných kusů produktu; pokud číslo 64 bitů, ústředí neumí rozpoznat pravou zprávu od falešné
 - tj. aktivní oponent může způsobit potíže, i když nezná obsah zpráv
 - řeší se přidáním redundance ke zprávám
 - např. pokud každý blok má 1/2 nulovou, umíme rozpoznat správnou zprávu od falešné; na druhou stranu pomůžeme analytikovi - někdy predikovatelnější C
 - nejlepší by bylo, kdyby redundancí mohl být náhodně vypadající řetězec

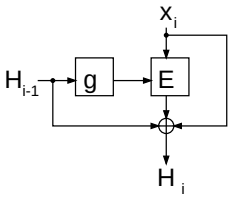
* k tomto účely byly navrženy MDC = manipulation detection code
 - předpokládejme, že máme k dispozici bezpečnou blokovou šifru
 - pak můžeme použít některou z následujících konstrukcí:



Matyas-Meyer-Oseas



Davies-Meyer



Miyaguchi-Preneel

 - vstupem zpráva rozdělená do n-bitových bloků (pro DES: n=56 pro Davies-Mayer, pro ostatní n=64)
 - poslední blok je jednoznačně doplněn na délku n bitů (např. 10000...0)
 - hodnota H0 je předdefinována
 - postupně vytváříme H1, H2, ...
 - výsledné H_t připojíme za poslední P_t, celé zašifrujeme
 - příjemce dešifruje, nezávisle spočte H_t a ověří

* MDC je tzv. jednosměrná hashovací fce; je snadné spočíst H = f(P), ale opačný výpočet je výpočetně neproveditelný
* uvedeno jako předběžný příklad; použití jednosměrných hashovacích fcí si uvedeme později

Poznámka (délka výstupu hashovací fce)

Pro většinu použití potřebujeme H s délkou > 80 bitů; takové fce existují, např. např. MD5 (Rivest 1992) se 128 bitů nebo SHA-1 (NIST 1993) se 160 bitů.

[]

Klíče vs. hesla

* pokud by klíč zadával uživatel, pak pro DES 7 osmibitových znaků
 - pokud pouze malá písmena: 26^7 ~ 2^33
 - pokud malá a velká: 52^7 ~ 2^40
 - všechny tisknutelné ASCII: 96^7 ~ 2^46

* to by ovšem platilo pouze pokud by všechny znaky byly stejně pravděpodobné
* v teorii informace je definována entropie jako minimální počet bitů potřebných pro zakódování všech možných hodnot zprávy
* pokud by všechna hesla stejně pravděpodobná, pak entropie = počet bitů

* pro angličtinu 1.3 bitu informace/znak (mění se podle délky textu, pro osmiznakové texty 2.3 bitu/znak, sníží se na 1.3 až 1.5 pro šestnáctiznakové)
* tj. běžné 8 znakové heslo cca 18.4 bitů entropie

* slabá hesla (s nízkou entropií) se dají snadno odhadnout
* oponent může prohledávat prostor klíčů podle klesající očekávané pravděpodobnosti hesla (slovníkové útoky)

* nejlepší používat náhodné klíče (vygenerované HW generátorem náhodných čísel)
* pokud klíče odvozené z uživatelem zadaného textu (passphrase) - použít jednosměrný hash

Problém distribuce klíčů

* pokud někomu zasílám tajnou zprávu, musí mít klíč aby jí rozšifroval
* klíč nutné zaslat - nejjednodušší mít bezpečný kanál (tradičně kurýr apod.)
* často není, nebo není praktický (banka se stovkami poboček nebo potřeba každodenní změny klíče)
* distribuce klíčů byl tradičně nejslabším článkem mnoha systémů

* bylo by mnohem vhodnější moci klíče distribuovat přímo příslušným komunikačním kanálem (sítí)

Metoda kryptogramů

* Merkle 1975 (publikováno 1978), ”Merkle puzzle scheme”

* Alice vytvoří např. 2^20 kryptogramů, každý zašifruje 20 bitovým klíčem
* v každém z nich zpráva typu ”Toto je zpráva číslo X, tajný klíč je Y”
* Bob náhodně jeden vybere, rozluští hrubou silou, zjistí X a Y
* Bob pošle Alici X (číslo kryptogramu) a tajnou zprávu zašifrovanou klíčem Y: m=(X, E_Y(p))

* Oskar potřebuje 2^40 (resp. n^2 oproti n) operací
* pokud mají Bob a Oskar stejný výpočetní výkon, pak např. 10 min oproti jednomu roku
* dnes vzhledem ke kryptografickým standardům málo, ale zajímavá myšlenka

Šifrování s veřejným klíčem

* distribuce klíčů mnoho let považováno za inherentní problém
* až v roce 1976 Diffie a Hellman myšlenka - šifrovací a dešifrovací klíč různý, dešifrovací klíč nemůže být odvozen z šifrovacího
* šifrovací zveřejním (”veřejný klíč”), dešifrovací utajím (”tajný klíč”)

Alice chce komunikovat s Bobem:
* Bob má tajný klíč DB, veřejný EB; tajný utají, veřejný zveřejní např. na síti
* Alice chce poslat zprávu p Bobovi, použije c = EB(p)
* Bob dešifruje p = DB(c)
* Oskar nic nerozluští, protože neumí odvodit DB z EB

* název asymetrická kryptografie, šifrování s veřejným klíčem (tradiční kryptografie = symetrická kryptografie, šifrování s tajným klíčem)
* koncepce považována za revoluci, ”před 1977 a po”

Algoritmus RSA

* Rivest, Shamir, Adleman 1977, nejznámější a nejpoužívanější

Algoritmus:	Minipříklad:
* Alice vybere 2 velká prvočísla P a Q	P=7, Q=17
* spočte N=P*Q	N=119
* spočte X = (P-1)(Q-1)	X=96
* zvolí E a D, aby E*D = 1 (mod X)	E=5 D=77
(E a D jsou ”multiplikativní inverze”,	

```
  E^{-1} == D (mod X), rozšířený Euklidův alg.)
* zveřejní N a E                                (119, 5)

Bob šifruje:
* c = m^E (mod N)              c = 19^5 (mod 119) = 66
Alice dešifruje:
* m = c^D (mod N)              m = 66^77 (mod 119) = 19

[Důkaz že funguje je v HAC, str. 286.]

Bezpečnost metody závisí na obtížnosti rozkladu velkých čísel na prvočísla:
* Oskar zná E a N a ví, že N=P*Q
* kdyby zjistil P a Q, spočte X=(P-1)(Q-1), najde D
 - problém zjistit D je dokazatelně výpočetně stejně složitý jako rozklad N
* P a Q nemůže zjistit jinak než rozložením N na P*Q
 - RSAP (RSA Problem): není znám efektivní algoritmus rozkladu na
   prvočinitele, ale není dokázáno že neexistuje
 - čas zhruba e^sqrt(ln n ln ln n)  [protože P, Q < sqrt(N)]
* pokud je N 100 ciferné - bude to trvat cca týden (336 bitů)
            150           1000 let                (500 bitů)
            200           mil. let                (665 bitů)
* ve skutečnosti záleží na množství investovaných prostředků
* pro běžné použití (investice do HW 25M$, doba faktorizace 25 let) by mělo
  stačit 1024 bitů

Algoritmus má různé potíže:
* výrazně pomalejší než symetrické šifry (1000bit RSA asi 4000x), pro stejnou
  bezpečnost vyžaduje podstatně větší délku klíče (17x delší)
* hodnoty nelze zvolit libovolně, např.:
 - P a Q musejí být zvoleny tak, aby byl rozklad N výpočetně neproveditelný
 - každá entita vlastní modulus N, protože prozrazení D, E dovolí rozklad
   modulu
 - z důvodů efektivity volen často malý šifrovací exponent, např. E=3
 - pokud je malý exponent E a posílám šifrovanou zprávu více entitám,
   provedu c_i = p^E mod n_i, z toho se ale dá zjistit p^E, pak stačí
   E-tou odmocninu
 - nejlepší aby p byla náhodná data atd.

Správnou volbou parametrů se zabývá [HAC], kap. 4.1 až 4.4.

* v praxi se většinou používá pro distribuci klíčů pro jednu relaci (klíč
  jsou náhodná data), relace pak proběhne symetrickým algoritmem
```

❇