Jak vybrat velké prvočíslo?

* náhodně zvolíme číslo
* zjistíme, zda je prvočíslo; pravděpodobnost že p je prvočíslo bude 1/ln p, tj. pro nalezení 1024 bitového p by nám v průměru mělo stačit 1/2 ln 2^1000 = 346 pokusů

* vygenerujte náhodné n-bitové číslo p
* nastavte nejvyšší a nejnižší bit na 1
  (nejvyšší - bude mít požadovanou délku, nejnižší - bude liché)
* otestujte, zda není dělitelné malými prvočísly 3, 5, 7, ...
  (tím snížíme pravděpodobnost, že není prvočíslo)
* otestujte pomocí Rabin-Millerova testu - heuristická metoda
 - založen na myšlence, že je-li p prvočíslo, pak platí
   i^{p-1} mod p = 1, pro všechna i=2,3,...,n-1
 - např. pro p=5: 2^4 mod 5 = 16 mod 5 = 1
                  3^4 mod 5 = 81 mod 5 = 1
                  4^4 mod 5 = 256 mod 5 = 1
* inkrementální vyhledávání: pokud p neuspělo, zkusíme p:=p+2 a přejdeme na první krok (protože mezi kandidáty vztah, umožňuje urychlit pokusné testování dělitelnosti malými prvočísly)

Rabin-Millerův pravděpodobnostní test prvočíselnosti (alg. 4.24 v HAC):

1. spočti s, 2^s je největší mocnina 2 která dělí n-1
   r:=(n-1)/(2^s)
2. vyber náhodné a, 2 <= a <= n-2
3. spočti y=a^r mod n
4. pokud y<>1 a y<>n-1 pak
      j:=1
      dokud j<s a y<>n-1 opakuj:
            y:=y^2 mod n
            pokud y=1 pak n není prvočíslo
            j:=j+1
      pokud y<>n-1 pak n není prvočíslo
5. číslo n může být prvočíslo

Pravděpodobnostní test stačí; pokud generujeme 1000 bitové pravděpodobné prvočíslo, stačí 3 iterace Rabin-Millerova testu aby pravděpodobnost chyby byla menší než (1/2)^80, pro 2000 bitové stačí 2 iterace.

Existují i skutečné testy prvočíselnosti, ale trvají déle.

Implementace práce s velkými čísly:

* viz HAC kap. 14
* existují volně šířené knihovny, např. GNU MP (multiple precision arithmetic library)

Další algoritmy šifrování veřejným klíčem

* RSA je nejpoužívanější, ale není jediný

* vůbec první byl algoritmus založený na zavazadlovém problému
  (knapsack problem, Merkle a Hellman 1978)
* problém formulován takto:
 - máme předměty o hmotnosti m1, m2, ..., m_n
 - máme sbalit zavazadlo tak, aby mělo hmotnost M
 - tj. hledáme binární slovo x = x1, x2, ..., x_n takové aby platilo:
   M = x1*m1 + x2*m2 + ... x_n*m_n
 - vlastník zakóduje zprávu tak, že tajně vybere podmnožinu objektů a vloží ji do batohu
 - celkovou hmotnost M a seznam možných objektů m1, m2, ..., m_n zveřejní
* problém ve volbě hmotností m1, m2, ..., m_n
 - jestliže je problém lehký, mohl by dešifrovat kdokoli
 - jestliže je problém těžký, nemohl by dešifrovat nikdo
 - Merkle a Hellman navrhli způsob jak se snadného problému vytvořit těžký

* vytvoříme snadný zavazadlový problém, metodu jak z něj udělat těžký problém zveřejníme a metody jak z těžkého lehký utajíme; odesilatel zvolí M, vytvoří těžký problém (navrženo bylo M*V mod A), příjemce lehký problém a vyřeší

* Merkle coby autor algoritmu si byl natolik jistý, že nabídl $100 odměnu prvnímu, kdo by ho rozluštil; Shamir ('S' v RSA) to udělal
* Merkle zesílil algoritmus a nabídl 1000$ za rozluštění; Rivest ('R' v RSA) to udělal
  => algoritmy založené na zavazadlovém problému nejsou považovány za bezpečné

* další funkční algoritmy jsou založené na obtížnosti spočíst diskrétní logaritmus (Rabin 1979) - algoritmy na tomto principu jsou El Gamal (1985) a Schnorr (1991) atd.
 - problém spočíst y = a^x mod n je jednoduchý, ale nalézt x pokud máme y,a,n není jednoduché (některé diskrétní logaritmy nemají řešení)

Autentizační protokoly

* autentizace = technika, kterou proces ověří identitu druhé strany
* pokud může být oponent aktivní, je obtížné

* mimochodem, autentizace != autorizace
 - autentizace - ustanovení identity, např. uživatel je opravdu Alice
 - autorizace - co je procesu dovoleno, např. smí proces patřící Alici smazat soubor "kuchařka.doc"? (implementace - jednoduché vyhledání v tabulce)

* obecný model:
 - Alice chce vytvořit (bezpečné) spojení s Bobem (Alice a Bob jsou v angl. nazýváni "principals", tj. představitelé hlavních rolí)
 - Alice si vyměňuje zprávy s Bobem nebo důvěryhodnou třetí stranou (může mít různou roli, např. centrum pro distribuci klíčů, certifikační autorita apod.)
 - Oskar může zprávy modifikovat, vkládat nebo rušit
 - po dokončení protokolu Alice ví, že komunikuje s Bobem a Bob ví, že komunikuje s Alicí
 - ve většině protokolů se také ustanoví klíč pro šifrování jedné relace symetrickým šifrovacím algoritmem (relační klíč, angl. session key)

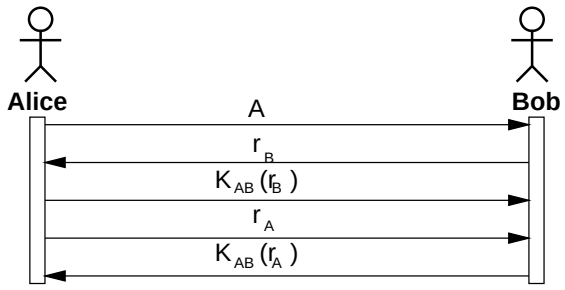
* typické předpoklady:
 - zprávy protokolu se přenášejí nechráněnou (otevřenou) sítí
 - oponent může může zaznamenávat, měnit nebo rušit zprávy a vkládat zprávy vlastní
  . pro zvýraznění je obvykle uváděn model, ve kterém legální účastníci komunikace získávají zprávy výhradně prostřednictvím oponenta, který může provést kteroukoli z uvedených akcí bez znatelného zpoždění
 - předpokládáme, že základní kryptografické mechanismy (jako jsou šifrovací algoritmy) jsou bezpečné
  . Oskar kryptografické mechanismy nemůže napadnout, ale snaží se napadnout způsob, jakým jsou základní mechanismy kombinovány, tj. napadá vlastní protokol

Autentizace založená na sdíleném tajném klíči

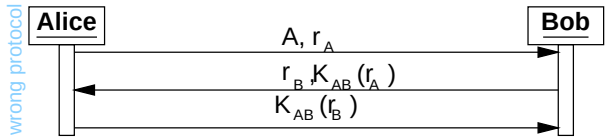
* předpokládáme, že Alice a Bob sdílejí tajný klíč K_AB, např. si ho předali osobně
* protokol založen na myšlence, že jedna strana pošle druhé výzvu (challenge) - náhodné číslo, které druhá strana transformuje a pošle odpověď
* takové protokoly se nazývají protokoly typu výzva-odpověď (challenge-response protocols)

Jednoduchý protokol tohoto typu (praktické použití viz závěrečná poznámka)
1. Alice pošle Bobovi informaci o identitě: (A)
2. Bob vybere velké náhodné číslo r_B (např. 128 bitů), pošle ho Alici:
   (r_B)
3. Alice zašifruje klíčem který sdílí s Bobem, pošle výsledek Bobovi:

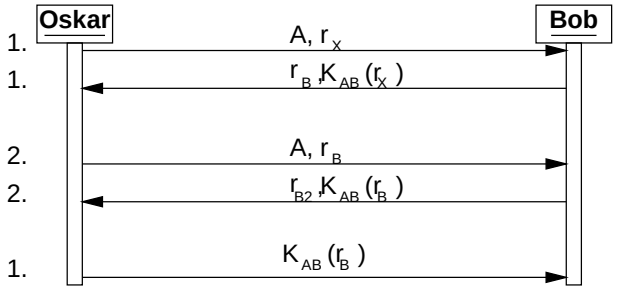
```
                                                ( K_AB(r_B) )
   Bob ví, že komunikuje s Alicí, ale Alice neví nic, Oskar mohl
   zachytit první zprávu a poslat zpět r_B.
4. Alice vybere náhodné r_A a pošle Bobovi: ( r_A )
5. Bob zašifruje a odpoví:                  ( K_AB(r_A) )
   Alice ví, že komunikuje s Bobem; teď může např. vybrat relační klíč,
   zašifrovat K_AB a poslat Bobovi.
```




```
* protokol má 5 zpráv, co kdybychom sdružili informace do 3 zpráv?
```




```
* tento kratší protokol chybný, umožňuje tzv. reflection attack
 - předpoklad - s Bobem je možné zahájit více relací současně (např. Bob je
   banka, má více zákazníků kteří chtějí provést transakci)
 - Oskar zahájí komunikaci, tvrdí že je Alice a pošle r_X
 - Bob odpoví zasláním vlastní výzvy r_B: (r_B, K_AB(r_X))
 - Oskar zahájí druhou relaci, pošle r_B
 - Bob zašifruje r_B a pošle zpět
 - Oskar má chybějící informaci, v první relaci odpoví K_AB(r_B)
```



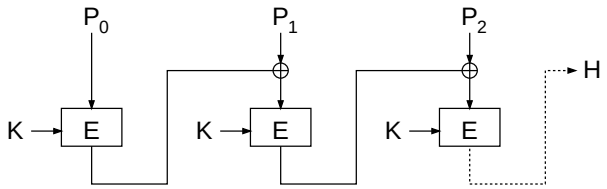

```
* poučení: navržení správného autentizačního protokolu je obtížnější než to
  vypadá na první pohled
* v praxi je třeba používat pouze ověřené publikované metody, např. z [HAC]
* pro zamezení reflexivního útoku se může jako odpověď na výzvu r_X posílat
  místo K_AB(r_X) např. K_AB(r_X, X) kde X je identifikátor vyzyvatele
  (tato modifikace je součástí standardu ISO 9798-2)


Protokol AKEP2
..............

* AKEP2 = Authenticated Key Exchange Protocol 2
* navrhli Bellare a Rogaway 1993, jeho vlastnosti dokázali
* pomocí třech zpráv vzájemná autentizace a dohodnutí sdíleného relačního
  klíče

* předpoklady:
 - Alice a Bob  sdílejí dlouhodobé tajné klíče K_1 a K_2
 - k dispozici mají klíčovanou jednosměrnou hashovací fci
```

(MAC = message authentication code, většinou modifikace jednosměrné hashovací fce)
. např. CBC-MAC - základní verze uvedena na obrázku, před první blok OT je třeba přidat délku hashovaného vstupu



CBC-MAC (basic)

* Alice vybere náhodné číslo r_A a pošle ho Bobovi
* Bob vybere náhodné číslo r_B a pošle Alici zprávu T1 = (A, B, r_A, r_B) spolu s MAC_K1(T1)
* Alice při příjmu ověří identity (tj. A a B), porovná přijaté r_A s odeslaným a ověří MAC => Alice ví, že komunikuje s Bobem
* Alice pošle Bobovi zprávu T_2=(A, r_B) spolu s MAC_K1(T_2)
* Bob ověří r_B a MAC => Bob ví, že komunikuje s Alicí
* oba vypočtou relační klíč W = MAC_K2(r_B)

* výhoda vzhledem k restrikcím v některých státech - v protokolu nemusí být použita žádná šifrovací funkce
* nevýhoda - získá-li oponent klíč K_2, může dešifrovat obsah již proběhlé komunikace

Autentizace s účastí třetí strany

* předchozí algoritmy založeny na předpokladu, že obě strany sdílejí tajný klíč
* pokud bychom chtěli komunikovat s N stranami, potřebujeme N (dvojic) tajných klíčů => problémy s distribucí a správou klíčů

* proto myšlenka zavést důvěryhodnou třetí stranu, která má roli centra pro distribuci klíčů (key distribution server, KDC)
* uživatel má pouze jeden klíč, který sdílí s KDC (Alice sdílí K_A a Bob K_B)
* autentizace a vytváření relačního klíče proběhne s využitím KDC

* nejjednodušší protokol (má závažný nedostatek, který chceme ilustrovat)
 (1) Alice vytvoří relační klíč a pošle KDC zprávu: (A, K_A(B, K))
 (2) KDC dešifruje, vytvoří novou zprávu K_B(A, K) a pošle Bobovi
 - KDC ví, že zprávu 1 musela vytvořit Alice, zprávu 2 umí dešifrovat pouze Bob => autentizace stran vedlejším efektem
* má nedostatek:
 - Oskar bude potřebovat peníze, nabídne Alici své služby
 - po dokončení práce Alice zaplatí Oskarovi bankovním převodem:
  . Alice vytvoří relační klíč se svým bankéřem Bobem
  . Alice pošle Bobovi požadavek K(X) na převod peněz Oskarovi
 - Oskar odposlechne zprávu (2) a K(X), pošle Bobovi později ještě několikrát
 - nazývá se replay attack (česky by bylo ”přehrávka”)

* několik možností obrany proti uvedenému útoku
 - do každé zprávy vložit časové razítko t_A (timestamp): (A, K_A(t_A, B, K))
  . pokud kdokoli obdrží zastaralou zprávu, může jí zahodit
  . problém - hodiny na síti nebudou nikdy přesně synchronizované, časová značka musí mít nějakou dobu platnosti - Oskar může přehrát během doby platnosti
 - druhá možnost - do zprávy vložit Nonce, např. jedinečné číslo zprávy
  . obě strany si musejí zapamatovat všechny předchozí Nonce a odmítat všechny zprávy obsahující již použité Nonce
  . problém - všechny použité Nonce je třeba si pamatovat pořád - co kdybychom o seznam obsahující Nonce přišli (např. při havárii HD)?
 - nejlepší adaptovat challenge-response protokol pro více stran

❇