DSA - vnitřní kolize

Najděte dvě (smysluplné) zprávy m1 a m2, pro které existují veřejné parametry DSA (p, q, g) takové, že obě zprávy mají pro libovolnou hodnotu soukromého klíče zaměnitelnou hodnotu platného podpisu (r, s). Podpis zprávy m1 je tak zároveň platným podpisem zprávy m2 a obráceně. Parametry (p, q, g) nalezněte a kolizi podpisů obou zpráv demonstrujte pro náhodně zvolenou hodnotu soukromého klíče:

Nejprve zvolíme první zprávu, kterou zahashujeme. Jako hashovací funkce je zvolena funkce SHA-256, postup je však aplikovatelný pro libovolnou hashovací funkci:

In[1]:=
```
m1 = "Čas události nelze garantovat!";
"První zpráva: " <> m1
hm1 = Hash[m1, "SHA256"];
"Hash první zprávy: " <> ToString[hm1]
```

Out[2]=
```
První zpráva: Čas události nelze garantovat!
```

Out[4]=
```
Hash první zprávy:
  92062028454352618822104265121643404747010111249595380003604729003263243932539
```

Následně je třeba nalézt druhou zprávu, jejíž hash bude mít stejnou hodnotu jako hash první zprávy (modulo q). Pojďme se nyní podívat na pravděpodobnost nalezení N-bitového prvočísla q při náhodném generování zpráv. Pro tuto úlohu bylo zvoleno N=256, což je maximální délka výstupu hashovací funkce SHA-256. Pro zjednodušení uvažujme, že náhodně zvolená zpráva m2 s hashem hm2 způsobí že výraz |(hm1 - hm2)| bude náhodné číslo z intervalu <0 ; 2^256), a že počet prvočísel menších než x odpovídá přibližně výrazu x / ln(x) (Hadamard a Poussin 1896). Poté z výpočtu níže vidíme, že pravděpodobnost, že při náhodně zvolené druhé zprávě získáme 256-bitové prvočíslo q = |(hm1 - hm2)|, je přibližně 0.0028. Pokud však budeme opakovat generování druhé zprávy 3276x, získáme přibližně pravděpodobnost 0.9999, že alespoň jednou narazíme na 256-bitové prvočíslo q = |(hm1 - hm2)|. Nalezení druhé zprávy by tedy nemuselo být příliš obtížné:

In[5]:=
```
bitCountN = 256;
ApproximateCountOfPN[x_] := Floor[N[x / Log[x]]]
a = 2^(bitCountN - 1);
aa = 0;
b = 2^(bitCountN);
prob = (ApproximateCountOfPN[b] - ApproximateCountOfPN[a]) / ( b - aa )
"pst = " <> ToString[N[prob]]
"Procentuální zastoupení prvnočísel u čísel s " <>
 ToString[bitCountN] <> " bity: " <> ToString[Round[100 * prob, 0.01]] <> "%"
pstSolution = Solve[{(1 - Power[(1 - prob), attempt]) == 0.9999}, {attempt}];
attempts = Floor[attempt /. pstSolution[[1, 1]]] + 1;
"Pravděpodobnost nalezení prvočísla na intervalu <2^" <>
 ToString[bitCountN - 1] <> " ; 2^" <> ToString[bitCountN] <> ") během " <>
 ToString[Floor[attempts]] <> " pokusů je: " <> ToString[1 - ((1 - N[prob]) ^ attempts)]
```

Out[10]= $\frac{3\,235\,920\,579\,477\,307}{1\,152\,921\,504\,606\,846\,976}$

Out[11]= pst = 0.00280671

Out[12]= Procentuální zastoupení prvnočísel u čísel s 256 bity: 0.28%

Out[15]= Pravděpodobnost nalezení prvočísla na intervalu <2^255 ; 2^256) během 3277 pokusů je: 0.9999

Nyní je třeba nalézt druhou zprávu m2, která bude mít hodnotu hashovací funkce kongruentní s hm1 modulo N-bitové prvočíslo q. Aby byla zpráva smysluplná,

můžeme například generovat řetězec, který obsahuje náhodný datum a čas ve formátu “Čas události: D.M.Y, hHmMsS”. Náhodným generováním můžeme vytvořit až 580608000 různých řetězců, tudíž díky pravděpodobnosti v předchozím bodě by neměl být problém rychle nalézt řetězec, který by mohl představovat zprávu m2:

In[16]:=
```
q = 1;
hm2 = 0;
While[! PrimeQ[q] || BitLength[q] ≠ bitCountN,
  m2 = "Čas události: " <> IntegerString[RandomInteger[{1, 28}], 10] <>
    ". " <> IntegerString[RandomInteger[{1, 12}], 10] <> ". " <>
    IntegerString[RandomInteger[{2000, 2020}], 10] <> ", h" <>
    IntegerString[RandomInteger[{0, 23}], 10] <> "m" <> IntegerString[
     RandomInteger[{0, 59}], 10] <> "s" <> IntegerString[RandomInteger[{0, 59}], 10];
  hm2 = Hash[m2, "SHA256"];
  q = Abs[hm1 - hm2]];
"Druhá zpráva: " <> m2
"Hash druhé zprávy: " <> ToString[hm2]
"q = " <> ToString[q]
"Délka q: " <> ToString[BitLength[q]] <> " bitů"
"Rozdíl hashů obou zpráv modulo q: " <> ToString[Abs[Mod[hm1 - hm2, q]]]
```

Out[19]= Druhá zpráva: Čas události: 2. 12. 2004, h19m13s29

Out[20]= Hash druhé zprávy:
1311263279930790237628198262904859796838073453557346225404694104336790521612

Out[21]= q = 90750765174421828584476066858738544950172037796038033778200034898926453410927

Out[22]= Délka q: 256 bitů

Out[23]= Rozdíl hashů obou zpráv modulo q: 0

Nyní můžeme podobným způsobem nalézt L-bitové prvočíslo p, pro které by mělo platit N < L a dle specifikace FIPS 186-5 by pro N = 256 mělo být L = 2048 nebo L = 3072. Dále by výraz (p - 1) měl být násobkem q:

In[24]:=
```
bitCountL = 2048;
nmin = Ceiling[2^(bitCountL - 1) / q];
nmax = Floor[(2^bitCountL) / q];
p = 1;
While[! PrimeQ[p] || BitLength[p] ≠ bitCountL, p = 1 + RandomInteger[{nmin, nmax}] * q];
"p = " <> ToString[p]
"Délka p: " <> ToString[BitLength[p]] <> " bitů"
```

Out[29]=
```
p =
 2131294811780475074760917695901242876126207820645404839183007043189435238103810383⋱
  832501408899076098774811431282895591086326131515018721535749272567149981532589404⋱
  075041998922685023637321185280252042410479353855153914847666181963989950432299969⋱
  774916599490231602727815327923658658240595325447660914428389252166576387623230754⋱
  322043344862007022116332841730828882549993831970917376566703195626782715194599916⋱
  101620370752197578568487254887257281926015957241880692098889426917112884972368467⋱
  717487335241032450241646297983230215156558902090541901092740458173771806958824559⋱
  8579186859514796586607597380198514966231047550751
```

Out[30]=
```
Délka p: 2048 bitů
```

Dále zvolíme náhodně h na intervalu <2 ; p - 2> a snadno dopočteme g, čímž získáme veřejné parametry {p, q, g}:

In[31]:=
```
h = RandomInteger[{2, p - 2}];
"h = " <> ToString[h]
g = PowerMod[h, (p - 1) / q, p];
"g = " <> ToString[g]
"{p, q, g} = " <> ToString[{p, q, g}]
```

Out[32]=
```
h =
 1556300293920288226551421601500993362693067331083036308938703967439870959779456917⋱
  0550630126420850544017622893651586675604573266376379991747682076445077884213335512⋱
  9609252731966981856936886732741917370273152153635740412075662492378612752509667955⋱
  5745610018396705067167192352929050408315521839669664563241134394749746377549605577⋱
  7905976478276890065597358483883975152034103833808108188891869948551282855636969888⋱
  5558725812249804280173720045304073436123311832748256579972298679122146350588044466⋱
  2603994691738796953798587263550291258948649695272513573963324518194325204642721700⋱
  433242578537529035795159029762122152540622572 84
```

Out[34]=
```
g =
 1806471054246539402340398514921250256066574251171607841380519703804313395839681604⋱
  2255751266626693390231081042229027980507546992286952327346039425388157013438407266⋱
  3142280427659307113562487408014039000094815538130701372712607039742388572373215266⋱
  6009078235917448374584081286233265180764885922374224026396451119275711806806399500⋱
  7984498800004057296481181336452568094141436525305716681064982923365533203004921133⋱
  5963121515620483699739008825866592692035260587887417784199810405163912296077962355⋱
  2114004851438192396464145873516730322298045857233594257579180521149713285150545611⋱
  138572995631900684406222020234739102063296057872 6
```

```
Out[35]= {p, q, g} =
  {2131294811780475074760917695901242876126207820645404839183007043189435238103810 38⋱
   38325014088990760987748114312828955910863261315150187215357492725671499815325894 0⋱
   40750419989226850236373211852802520424104793538551539148476661819639899504322999 6⋱
   97749165994902316027278153279236586582405953254476609144283892521665763876232307 5⋱
   43220433448620070221163328417308288825499938319709173765667031956267827151945999 1⋱
   61016203707521975785684872548872572819260159572418806920988894269171128849723684 6⋱
   77174873352410324502416462979832302151565589020905419010927404581737718069588245 5⋱
   98579186859514796586607597380198514966231047550751,
  9075076517442182858447606685873854495017203779603803377820003489892645341 0927,
  18064710542465394023403985149212502560665742511716078413805197038043133958396816 04⋱
   22557512666266933902310810422290279805075469922869523273460394253881570134384072 6⋱
   31422804276593071135624874080140390000948155381307013727126070397423885723732152 6⋱
   60090782359174483745840812862332651807648859223742240263964511192757118068063995 0⋱
   79844988000040572964811813364525680941414365253057166810649829233655332030049211 3⋱
   59631215156204836997390088258665926920352605878874177841998104051639122960779623 5⋱
   21140048514381923964641458735167303222980458572335942575791805211497132851505456 1⋱
   138572995631900684406222020234739102063296057872 6}
```

Nyní pojďme demonstrovat kolizi podpisů obou zpráv pro náhodně zvolený soukromý klíč. Nejdříve tedy náhodně zvolíme soukormý klíč x, dále k němu můžeme vypočítat veřejný klíč y, který však v této úloze nebudeme potřebovat:

```
In[36]:= x = RandomInteger[{1, q - 1}];
         "x = " <> ToString[x]
         y = PowerMod[g, x, p];
         "y = " <> ToString[y]

Out[37]= x = 47054574105312591774165120398272541989248222841950589867006128188377064633129

Out[39]= y =
  3611743502880506451693685600276771139238739108056522164353896808311441637923718688⋱
   514874176123338127131973694709441504557412552706839927271919803275546675703747571⋱
   456289775265768045510186070392058566919743667508146706837099957677790713564080488⋱
   074123165979049323785676487087541455022295986757928533828683959837405946300700711⋱
   670016508738568971614835294368216751995331486151672283243909163064115544036032707⋱
   299103128423391013826562667047640598376631014687064895331432296823343374292525590⋱
   147842051933485598687493651254601759439978274729864987682168874165236383201933913⋱
   78340931543885662211773058822696969300069415336
```

Vytvoříme podpis pro zprávu m1 a poté ho verifikujeme:

In[40]:=
```
(*Podepisování*)
k = RandomInteger[{1, q - 1}];
"k = " <> ToString[k]
r1 = Mod[PowerMod[g, k, p], q];
"r1 = " <> ToString[r1]
s1 = Mod[PowerMod[k, -1, q] * (hm1 + x * r1), q];
"s1 = " <> ToString[s1]
"podpis: (" <> ToString[r1] <> ", " <> ToString[s1] <> ")"
(*Verifikace*)
w1 = PowerMod[s1, -1, q];
"w1 = " <> ToString[w1]
u11 = Mod[hm1 * w1, q];
"u11 = " <> ToString[u11]
u21 = Mod[r1 * w1, q];
"u21 = " <> ToString[u21]
v1 = Mod[Mod[PowerMod[g, u11, p] * PowerMod[y, u21, p], p], q];
"v1 = " <> ToString[v1]
If[v1 == r1, "Podpis pro zprávu m1 je validní!", "Podpis pro zprávu m1 není validní!"]
```

Out[41]= k = 27809959494992964913577113606638099563945414618448236311481918249954133729916

Out[43]= r1 = 82386407433737027387700854716850686039136205943296157964559173770753503716224

Out[45]= s1 = 86583580120947764166674225783084497279324440110118267059085733112067027524980

Out[46]= podpis:
(82386407433737027387700854716850686039136205943296157964559173770753503716224,
86583580120947764166674225783084497279324440110118267059085733112067027524980)

Out[48]= w1 = 73189634832090322780168736893231445411902627895515638889190601881343595532008

Out[50]= u11 = 39645536497714221868250947035311789641741781017038694370377929184485387341086

Out[52]= u21 = 47449724339964526881469969636215417299230241317735969279412009072918450605545

Out[54]= v1 = 82386407433737027387700854716850686039136205943296157964559173770753503716224

Out[55]= Podpis pro zprávu m1 je validní!

Stejným způsobem vytvoříme podpis pro zprávu m2:

In[56]:=
```
(*Podepisování*)
r2 = Mod[PowerMod[g, k, p], q];
"r2 = " <> ToString[r2]
s2 = Mod[PowerMod[k, -1, q] * (hm2 + x * r2), q];
"s2 = " <> ToString[s2]
"podpis: (" <> ToString[r2] <> ", " <> ToString[s2] <> ")"
(*Verifikace*)
w2 = PowerMod[s2, -1, q];
"w2 = " <> ToString[w2]
u12 = Mod[hm2 * w2, q];
"u12 = " <> ToString[u12]
u22 = Mod[r2 * w2, q];
"u22 = " <> ToString[u22]
v2 = Mod[Mod[PowerMod[g, u12, p] * PowerMod[y, u22, p], p], q];
"v2 = " <> ToString[v2]
If[v2 == r2, "Podpis pro zprávu m2 je validní!", "Podpis pro zprávu m2 není validní!"]
```

Out[57]= r2 = 82386407433737027387700854716850686039136205943296157964559173770753503716224

Out[59]= s2 = 86583580120947764166674225783084497279324440110118267059085733112067027524980

Out[60]= podpis:
(82386407433737027387700854716850686039136205943296157964559173770753503716224,
86583580120947764166674225783084497279324440110118267059085733112067027524980)

Out[62]= w2 = 73189634832090322780168736893231445411902627895515638889190601881343595532008

Out[64]= u12 = 39645536497714221868250947035311789641741781017038694370377929184485387341086

Out[66]= u22 = 47449724339964526881469969636215417299230241317735969279412009072918450605545

Out[68]= v2 = 82386407433737027387700854716850686039136205943296157964559173770753503716224

Out[69]= Podpis pro zprávu m2 je validní!

Můžeme ověřit, že podpisy pro obě zprávy jsou shodné:

In[70]:=
```
If[s1 == s2 && r1 == r2, "Podpisy (r1, s1) a (r2, s2) jsou shodné!",
 "Podpisy (r1, s1) a (r2, s2) nejsou shodné!"]
```

Out[70]= Podpisy (r1, s1) a (r2, s2) jsou shodné!

Nyní se pokusíme dokázat, že pro libovolný soukromý klíč budou podpisy pro zpávy m1 a m2 vždy shodné. Na začátek pojďme označit

hm = (hm1 mod q) = (hm2 mod q)

Víme, že jednotlivé složky podpisů (r1, s1) a (r2, s2) vypočítáme následovně:

r1 = (g^k mod p) mod q
s1 = (k^-1 * (hm1 + x * r1)) mod q

a analogicky při záměně m2 za m1:

r2 = (g^k mod p) mod q
s2 = (k^-1 * (hm2 + x * r2)) mod q

Vidíme že r1 = r2, označme tedy oboje jako r. Dále je třeba ukázat rovnost s1 = s2:

s1 = (k^-1 * (hm1 + x * r)) mod q
s2 = (k^-1 * (hm2 + x * r)) mod q

s1 = ((k^-1 * hm1) + (k^-1 * x * r)) mod q
s2 = ((k^-1 * hm2) + (k^-1 * x * r)) mod q

s1 = ((k^-1 mod q) * (hm1 mod q) + (k^-1 * x * r)) mod q
s2 = ((k^-1 mod q) * (hm2 mod q) + (k^-1 * x * r)) mod q

s1 = ((k^-1 mod q) * hm + (k^-1 * x * r)) mod q
s2 = ((k^-1 mod q) * hm + (k^-1 * x * r)) mod q

s1 = s2

Klíče (r1, s1) a (r2, s2) tedy budou vždy shodné nezávisle na hodnotě soukromého klíče x.

Na závěr je nutno zmínit, že abychom podstatně ztížili nalezení této kolize, je třeba bezpečněji volit parametry p, q, g. Při generování je nutné použít tzv. SEED. Přesný postup je uveden ve specifikaci FIPS 186-5.