Demonstration of Bellcore-Attack on RSA-CRT

RSA-Setup

Let's generate some random 512 bit primes p and q. Note, these primes are not secure (insecure RNG within Maple)!

```
> p:=nextprime(RandomTools[Generate](integer(range=1..2^512)));
```

$$p := 55925372697156185307724265604960359073704881621728270142156790234510991246860941712200311781928344781633201654893536779561483032468034084328832217711538873$$

```
> q:=nextprime(RandomTools[Generate](integer(range=1..2^512)));
```

$$q := 14590294965656671597342026288158484680735731545786085311465846164579525245053188459731555856080826048934307589428678272578554731421637804062122463901434 43$$

Did we really get prime numbers? What bit length?

```
> isprime(p); ilog2(p);
```

$$true$$

$$510$$

```
> isprime(q); ilog2(q);
```

$$true$$

$$508$$

RSA setup of modulus and public exponent. Don't use e=3 anymore!

```
> n:=p*q; ilog2(n); e:=2^16+1;
```

$$n := 81596768371579096413488602526631522900210655247009393887651567532685936888042558293704999383223395641614467788150081934670959463623268999247950184848825192131434159562918236621427101970397873484844383086188302008863819920493310346212494503920306235842158810517304149093126910552528288367473216592880950047391019$$

$$e := 65537$$

Compute Euler's totient function.

```
> phi:=(p-1)*(q-1);
```

$$\varphi := 81596768371579096413488602526631522900210655247009393887651567532685936888042558293704999383223395641614467788150081934670959463623268999247950184848825121615766496750061331555135208851554119044231215571832848386227420829976818432082322572052668226671328243008059826878074770514764398695584825638199337074241459029496565667159734202628815848468073573154578608531146584616457952524505318845973155585608082604893430758942867827257855473142163780406212246390143443$$

Compute private exponent.

```
> d:=e^(-1) mod phi; ilog2(d);
```

$$d := 35075517014268677299247767068682778631074577408467715101437660566257659246073743873693596634331286174459660749313580241139360514048472378149949686552040852131763335923870910059527734009276937328746842319448174386023145698187389243764944863205025699708327498390574201486322572515402637289653412696329580604171018$$

Sample message to be signed:

```
> m:=123456789;
```

$$m := 123456789$$

Classical RSA signature generation without CRT

Signature generation without CRT; easy to understand, but very slow.

```
> s:=m &^d mod n;
```

$$s := 116166782852885836495208217392778244868735245399077496528971942047374370039551032\backslash$$
$$7\backslash$$
$$98734076160418951460601253000139914579380123210632469449641110762641937360220136\backslash$$
$$46374247735126665051061532956319753926192619846003281388778126883266584093266043\backslash$$
$$43804427083432449358393550793404643684325271243133561363508606 5$$

Signature verification.

```
> s &^e mod n;
```

$$123456789$$

Setup for RSA-CRT with Garner's Algorithm

Setup for CRT parameters.

```
> dp:=d mod (p-1); ilog2(dp);
```

$$dp := 528780939706117182473784555667328930277735736236326653284234503207609149580082261\backslash$$
$$03395097729205972393284449604759645697580097617985476907235918903866719 69$$

$$510$$

```
> dq:=d mod (q-1); ilog2(dq);
```

$$dq := 102243600804545264473431786519060464691314708786147024431142782085612596915377990\backslash$$
$$02115315535443612266709751901248172457488788082995409277039794868198686 89$$

$$508$$

Setup for Garner's CRT algorithm.

```
> Qp:=q^(-1) mod p;
```

$$Qp := 247366743959648001610868033714999664462359274183373223355072333734762672399377 62\backslash$$
$$08405592347694300199699056544088177358948301671649919159656488738604397127$$

RSA-CRT with Garner's Algorithm

First small exponentiation.

```
> Sp:=m &^dp mod p;
```

$$Sp := 224597415515736958008228638637492994549233939714176188153021894006532920623901299\backslash$$
$$65181292090469481149909120381790834175380726900910874080440713651409159 63$$

Second small exponentiation.

```
> Sq:=m &^dq mod q;
```

$$Sq := 141733618979313292013549734284644655336511145413051177220548055433155737697738934\backslash$$
$$15495027150671990757731881217130053399887830074872873119647899287389177 23$$

Final combination step of Garner's RSA-CRT.

```
> S:=((Sp-Sq)*Qp mod p)*q+Sq;
```

$$S := 116166782852885836495208217392778244868735245399077496528971942047374370039551032\backslash$$
$$7\backslash$$
$$98734076160418951460601253000139914579380123210632469449641110762641937360220136\backslash$$
$$46374247735126665051061532956319753926192619846003281388778126883266584093266043\backslash$$

438044270834324493583935507934046436843252712431335613635086065

> Signature verification.

```
> S &^e mod n;
```

123456789

> Let's check if we get the same signatures

```
> S-s;
```

0

> Bellcore and Lenstra's Attack on RSA-CRT

> First small exponentiation, this time with errors.

```
> Spfault:=Sp + 0815;
```

Spfault := 22459741551573695800822863863749299454923393971417618815302189400653292062390\
129965181292090469481149909120381790834175380726900910874080440714365140916778

> Final combination step with faulty temporary result.

```
> Sfault:=((Spfault-Sq)*Qp mod p)*q+Sq;
```

Sfault := 39882689396859372059796756712227811148644761668613422120904001630294235462423 0\
298837805284707139319427384827799250406614194009435134538043230609082887872089 24\
060380977116234232454983425575927970521502427447176813409091515748777046210322 58\
109786865859491019920461826551240348926683594317683271949834110312740

> Lenstra's method, i.e., one faulty signature, original message ==> RSA modulus is factored!

```
> qguess_Lenstra:=gcd((m-Sfault &^e mod n),n);
```

qguess_Lenstra := 14590294965656671597342026288158484680735731545786085311465846164579 52\
524505318845973155585608082604893430758942867827257855473142163780406212246390 14\
3443

```
> q-qguess_Lenstra;
```

0

> Bellcore attack, i.e., one faulty signature, correct signature ==> RSA modulus is factored!

```
> qguess_BDL:=gcd(S-Sfault,n);
```

qguess_BDL := 14590294965656671597342026288158484680735731545786085311465846164579 5252\
450531884597315558560808260489343075894286782725785547314216378040621224639014 34\
43

> RSA-CRT with Shamir's Countermeasure

```
> r:=nextprime(RandomTools[Generate](integer(range=1..2^31)));
```

r := 15263771

```
> dpr:=d mod ((p-1)*(r-1));
```

dpr := 46103476069939403718588131112279157783919415855604244602835690216235084659054699\
971279523220013126506986482265500809248759915593505421115711403966562635887480 97

```
> dqr:=d mod ((q-1)*(r-1));
```

dqr := 15050771533326790461075424308941146911258270440810800339800077167299941749723134\
262575967863052632895730320581649273255149688699046005561354372377690636070737557

```
> Spprime:=m &^dpr mod (p*r);
```

Spprime := 42565295549059462769509226607759039317086579497595054755734665557459319200003\
80413524557211780985532296832231426260720851313224151347965513700243386951331298\
59787

```
> Sqprime:=m &^dqr mod (q*r);
```

Sqprime := 82030154141075463252769435233309587375885192977094901586653460995151133329321\
60805973455809131901411735117923347070926127545344101806380038334704934448809890\
043

```
> Sp:=Spprime mod p; Sq:=Sqprime mod q;
```

Sp := 22459741551573695800822863863749299454923393971417618815302189400653292062390129\
9651812920904694811499091203817908341753807269009108740804407136514091596 3

Sq := 14173361897931329201354973428464465533651114541305117722054805543315573769773893\
4154950271506719907577318812171300533998878300748728731196478992873891772 3

```
> S:=((Sp-Sq)*Qp mod p)*q+Sq;
```

S := 11616678285288583649520821739277824486873524539907749652897194204737437003955103279\
87340761604189514606012530001399145793801232106324694496411107626419373602201364\
63742477351266650510615329563197539261926198460032813887781268832665840932660433\
43804427083432449358393550793404643684325271243133561363508606 5

```
> Spprime-Sqprime mod r;
```

0

> Now with fault induction in first exponentiation.

```
> Spprimefault:=Spprime+0815;
```

Spprimefault := 42565295549059462769509226607759039317086579497595054755734665557459319200\
03804135245572117809855322968322314262607208513132241513479655137002433869513312\
98 59860602

```
> Sp:=Spprimefault mod p; Sq:=Sqprime mod q;
```

Sp := 22459741551573695800822863863749299454923393971417618815302189400653292062390129\
9651812920904694811499091203817908341753807269009108740804407136514091677 8

Sq := 14173361897931329201354973428464465533651114541305117722054805543315573769773893\
4154950271506719907577318812171300533998878300748728731196478992873891772 3

```
> S:=((Sp-Sq)*Qp mod p)*q+Sq;
```

S := 39882689396859372059796756712227811148644761668613422120904001630294235462423029 88\
37805284707139319427384827799250406614194009435134538043230609082887872089240603\
80977116234232454983425575927970521502427447176813409091515748777046210322581097\
86865859491019920461826551240348926683594317683271949834110312740

> If the following is not zero then an error has occured. Do not output result S!

```
> Spprimefault-Sqprime mod r;
```

815