

Diffie-Hellman

A very clever and very simple way that two people or programs communicating with each other can make a secure and secret encryption key, even when someone else can see everything either of them transmits. One of those things that sounds like a logical impossibility until you see how its done. It also makes number theory useful: another apparent impossibility.

The method:

1.

Person A chooses three numbers, a Base (g), a Modulus (n), and a Secret (x).

Person A computes $c = g^x \text{ modulo } n$.

Person A transmits g , n , and c to person B, not caring about security.

Person A keeps x absolutely secret for ever. It is never transmitted, so is perfectly safe.

2.

Person B chooses one number, a Secret (y).

Person B computes $d = g^y \text{ modulo } n$.

Person B transmits d back to person A, not caring about security.

Person B keeps y absolutely secret for ever. It is never transmitted, so is perfectly safe.

3.

Person A computes $K = d^x \text{ modulo } n$.

4.

Person B computes $K = c^y \text{ modulo } n$.

As if by magic, the value of K worked out independently by both people will be the same:

A's computation is

$$\begin{aligned} \text{given } & d = g^y \text{ modulo } n \\ \text{and } & K = d^x \text{ modulo } n \\ \text{so } & K = (g^y \text{ modulo } n)^x \text{ modulo } n \\ &= ((g^y)^x) \text{ modulo } n \quad (\text{by the rules of number theory}) \\ &= (g^{yx}) \text{ modulo } n \\ &= (g^{xy}) \text{ modulo } n \end{aligned}$$

B's computation is

$$\begin{aligned} \text{given } & c = g^x \text{ modulo } n \\ \text{and } & K = c^y \text{ modulo } n \\ \text{so } & K = (g^x \text{ modulo } n)^y \text{ modulo } n \\ &= ((g^x)^y) \text{ modulo } n \quad (\text{by the rules of number theory}) \\ &= (g^{xy}) \text{ modulo } n \end{aligned}$$

This works because

$$(A^B) \% N = ((A \% N)^B) \% N$$

It is not true that

$$(A^B) \% N = ((A \% N)^{(B \% N)}) \% N$$

and a spy listening to the communications has no access to either of the secret numbers (x or y) un-ruined by modular exponentiation.

So it is possible for two people to select a completely secret number without any prior set-up, even when somebody else is listening to everything they say.

Of course, all the numbers have to be Very Big Integers, otherwise a spy can solve the problem just by trying out all possible numbers.

```
$ cat diffiehellman.cpp
#include <stdio.h>
#include <stdlib.h>
#include "bint.h"                                // "bint.h" is my Big Integer library
                                                 // of course, the type bint stands for big int.
void main(void)
{ printf("Person A:\n");
  bint g=read(" choose base g: ");           // read prints a prompt and reads a bint
  bint n=read(" choose modulus n: ");
  bint x=read(" choose secret x: ");
  printf(" computing      c= (g^x)%%n\n");
  bint c=g;
  c.powmod(x, n);                            // a.powmod(b,c) ≡ a=(a^b)%c
  printf(" send g, n, c to person B\n");
  printf("\n");
  printf("Person B:\n");
  printf("      received g,\n");
  printf("      received n,\n");
  bint y=read(" choose secret y: ");
  printf(" computing      d= (g^y)%%n\n");
  bint d=g;
  d.powmod(y, n);
  printf(" send d back to person A\n");
  printf("\n");
  printf("Person A:\n");
  printf(" knows g = %s\n", g.tostring());
  printf("      n = %s\n", n.tostring());
  printf("      x = %s\n", x.tostring());
  printf("      d = %s\n", d.tostring());
  printf(" computes key k = (d^x)%%n\n");
  bint ka=d;
  ka.powmod(x, n);
  printf("      = %s\n", ka.tostring());
  printf("\n");
  printf("Person B:\n");
  printf(" knows g = %s\n", g.tostring());
  printf("      n = %s\n", n.tostring());
  printf("      y = %s\n", y.tostring());
  printf("      c = %s\n", c.tostring());
  printf(" computes key k = (c^y)%%n\n");
  bint kb=c;
  kb.powmod(y, n);
  printf("      = %s\n", kb.tostring());
  printf("\n");
  printf("Spy:\n");
  printf(" could have heard g, n, c, d, but x, y were never transmitted\n");
  bint try1=c;
  try1.powmod(d, n);
  printf(" can compute      (c^d)%%n = %s\n", try1.tostring());
  bint try2=d;
  try2.powmod(c, n);
  printf("      or (d^c)%%n = %s\n", try2.tostring());
  bint try3=g;
  try3.powmod(mul(c, d), n);
  printf("      or (g^(c*d))%%n = %s\n", try3.tostring());
  printf(" But nothing gives the key K, unless g, n too small\n");
  printf("\n"); }
```

\$ diffiehellman

Person A:

```
choose base g: ?30      // entering ?30 means "make up a 30-digit random number
choose modulus n: ?31
choose secret x: ?30
computing      c= (g^x)%n
send g, n, c to person B
```

Person B:

```
recevied g,
recevied n,
choose secret y: ?30
computing      d= (g^y)%n
send d back to person A
```

Person A:

```
knows g = 267535629127093606261879202375
n = 6228973612931947845036106320615
x = 147656937452547443078688431492
d = 1504794103763526892201770746515
computes key k = (d^x)%n
= 2214603010965799720745746640185
```

Person B:

```
knows g = 267535629127093606261879202375
n = 6228973612931947845036106320615
y = 768926649504871727226106159490
c = 5282670959606845305558566767510
computes key k = (c^y)%n
= 2214603010965799720745746640185
```

)Spy:

```
could have heard g, n, c, d, but x, y were never transmitted
can compute      (c^d)%n = 2937782371781770516552541795125
                  or (d^c)%n = 502593708026866298533037500735
                  or (g^(c*d))%n = 818905403502340251401460929710
But nothing gives the key K, unless g, n too small
```

But if you are careless, things can go wrong: Here, the numbers are too small:

Person A:

```
knows g = 2
n = 100
x = 9
d = 56
computes key k = (d^x)%n
= 96          // The secret key is 96
```

Person B:

```
knows g = 2
n = 100
y = 8
c = 12
computes key k = (c^y)%n
= 96          // B gets 96 too
```

Spy:

```
could have heard g, n, c, d, but x, y were never transmitted
can compute      (c^d)%n = 16
                  or (d^c)%n = 36
                  or (g^(c*d))%n = 96          // But the spy also gets 96
// Just by chance because 2 and 100 were too small
// to survive an unfortunate relationship they had...
```

Base? 2
Modulus? 100

b^1%m=2	b^2%m=4	b^3%m=8	b^4%m=16	b^5%m=32	b^6%m=64
b^7%m=28	b^8%m=56	b^9%m=12	b^10%m=24	b^11%m=48	b^12%m=96
b^13%m=92	b^14%m=84	b^15%m=68	b^16%m=36	b^17%m=72	b^18%m=44
b^19%m=88	b^20%m=76	b^21%m=52	b^22%m=4=b^2%m		

Only 21 of the 99 possible values can be generated by 2 modulo 100

If the spy heard that $b=2$ and $m=100$, he/she/it could have worked out that there were only 21 possible values the key could come out to be. Some values for b and m are even worse:

Base? 5
Modulus? 100

b^1%m=5	b^2%m=25	b^3%m=25=b^2%m
---------	----------	----------------

Only 2 of the 99 possible values can be generated by 5 modulo 100

Some values for b and m work out perfectly:

\$ a.out
Base? 13
Modulus? 97

b^1%m=13	b^2%m=72	b^3%m=63	b^4%m=43	b^5%m=74	b^6%m=89
b^7%m=90	b^8%m=6	b^9%m=78	b^10%m=44	b^11%m=87	b^12%m=64
b^13%m=56	b^14%m=49	b^15%m=55	b^16%m=36	b^17%m=80	b^18%m=70
b^19%m=37	b^20%m=93	b^21%m=45	b^22%m=3	b^23%m=39	b^24%m=22
b^25%m=92	b^26%m=32	b^27%m=28	b^28%m=73	b^29%m=76	b^30%m=18
b^31%m=40	b^32%m=35	b^33%m=67	b^34%m=95	b^35%m=71	b^36%m=50
b^37%m=68	b^38%m=11	b^39%m=46	b^40%m=16	b^41%m=14	b^42%m=85
b^43%m=38	b^44%m=9	b^45%m=20	b^46%m=66	b^47%m=82	b^48%m=96
b^49%m=84	b^50%m=25	b^51%m=34	b^52%m=54	b^53%m=23	b^54%m=8
b^55%m=7	b^56%m=91	b^57%m=19	b^58%m=53	b^59%m=10	b^60%m=33
b^61%m=41	b^62%m=48	b^63%m=42	b^64%m=61	b^65%m=17	b^66%m=27
b^67%m=60	b^68%m=4	b^69%m=52	b^70%m=94	b^71%m=58	b^72%m=75
b^73%m=5	b^74%m=65	b^75%m=69	b^76%m=24	b^77%m=21	b^78%m=79
b^79%m=57	b^80%m=62	b^81%m=30	b^82%m=2	b^83%m=26	b^84%m=47
b^85%m=29	b^86%m=86	b^87%m=51	b^88%m=81	b^89%m=83	b^90%m=12
b^91%m=59	b^92%m=88	b^93%m=77	b^94%m=31	b^95%m=15	

All 96 possible values can be generated by 13 modulo 97

In number-theory-ese, they say that 13 is primitive modulo 97. It is a good idea to make sure that your choice of b and m at least generate a good proportion of the possible numbers before using them for Diffie-Hellman. The test program is quite simple:

```
void main(void)
{ int b, m, b0;
printf(" Base? ");
scanf("%d", &b);
printf("Modulus? ");
scanf("%d", &m);
int * made=new int[m];
for (int i=1; i<m; i+=1)
    made[i]=0;
int numtomake=m-1, numonline=0;
b0=b;
for (int i=1; i<=m; i+=1)
{ char s[20], t[20];
    b%=m;
    sprintf(s, "b^%d%m=", i);
    sprintf(t, "%8s%d", s, b);
    numonline+=1;
    if (!made[b])
    { made[b]=i;
        numtomake-=1;
        if (numtomake==0) break;
    }
    ....
    if (numonline==6)
    { printf("%s\n", t);
        numonline=0; }
    else
        printf("%-11s ", t); }
else
{ printf("%s=b^%d%m\n", t, made[b]);
    numonline=0;
    break; }
b*=b0; }
if (numonline!=0) printf("\n");
if (numtomake==0)
    printf("All ");
else
    printf("Only %d of the ", m-1-numtomake);
printf("%d possible values can be generated by"
" %d modulo %d\n", m-1, b0, m); }
```