Ling Li, ling@cs.caltech.edu

9.1 Shrank's algorithm

$n = 1823$, $m = \lceil \sqrt{n} \rceil = 43$, $g = 5$, $y = 693$. The extended Euclidean algorithm gives $5 \times 1094 - 3 \times 1823 = 1$, i.e., $g^{-1} = 1094 \pmod{n}$. Compute ($g^{im} \bmod n$) and ($y \cdot g^{-i} \bmod n$) for $i = 0, 1, \ldots, m-1$, we find that*

$$g^{2m} = y \cdot g^{-12} = 702 \pmod{n},$$

i.e.,

$$g^{2m+12} = g^{98} = y \pmod{n},$$

or

$$\log_5 693 = 98 \pmod{1823}.$$

By the way, $3^{323} = 3^{1234} = 693 \pmod{1823}$ and $\mathrm{ord}(3) = 911$.

I list the commands I used in Mathematica to solve this problem here.

```
In[1]:=  n = 1823; g = 5; y = 693; m = Ceiling[Sqrt[n]];
In[2]:=  list1 = PowerMod[g, m*Range[0, m - 1], n]
Out[2]=  {1, 480, 702, 1528, 594, 732, 1344, 1601, 997, 934, 1685,
         1211, 1566, 604, 63, 1072, 474, 1468, 962, 541, 814, 598, 829,
         506, 421, 1550, 216, 1592, 323, 85, 694, 1334, 447, 1269, 238,
         1214, 1183, 887, 1001, 1031, 847, 31, 296}
In[3]:=  list2 = Mod[y*PowerMod[g, -Range[0, m - 1], n], n]
Out[3]=  {693, 1597, 684, 866, 1267, 618, 1582, 681, 1230, 246, 1143,
         1687, 702, 505, 101, 1114, 952, 555, 111, 1116, 1317, 628,
         1584, 1046, 1303, 1719, 1073, 1673, 1793, 1817, 728, 1604,
         1050, 210, 42, 373, 1533, 1765, 353, 1529, 1035, 207, 406}
In[4]:=  Sort[Flatten[{list1, list2}]]
Out[4]=  {1, 31, ..., 694, 702, 702, 728, ..., 1793, 1817}
In[5]:=  id = 702;
In[6]:=  First[First[Position[list1, id]]] - 1
Out[6]=  2
In[7]:=  First[First[Position[list2, id]]] - 1
Out[7]=  12
```

*My way to find this is: 1. generate two lists of $g^{im} \pmod{n}$ and $y \cdot g^{-i} \pmod{n}$; 2. sort these two lists; 3. merge the two sorted list. In the procedure of merging, two identical entries in those two lists will be found. The time complexity is $O(m \log m)$.

Ling Li, ling@cs.caltech.edu

9.2 Discrete log problem

The complexity of the discrete log problem does *not* depend on the choice of g.

To prove this, suppose for another primitive element $\hat{g} \in Z_p^*$ there exists an algorithm $\mathcal{A}$ with time complexity $\hat{f}(p)$ which computes the discrete log based on $\hat{g}$. Thus we can get $t = \log_{\hat{g}} g$ and $\hat{x} = \log_{\hat{g}} y$, i.e., $g = \hat{g}^t \bmod p$ and $y = \hat{g}^{\hat{x}} \bmod p$, for any $y \in Z_p^*$. Since both g and $\hat{g}$ are primitive elements, $\operatorname{ord}(g) = \operatorname{ord}(\hat{g}) = \phi(p)$. From $\operatorname{ord}(g) = \operatorname{ord}(\hat{g}) / \gcd(t, \operatorname{ord}(\hat{g}))$,[†] we also have $\gcd(t, \phi(p)) = 1$. Thus by the extended Euclidean algorithm, there is an integer a such that $ta = 1 \bmod \phi(p)$. So

$$g^{a\hat{x}} = \hat{g}^{ta\hat{x}} = \left(\hat{g}^{1 \bmod \phi(p)}\right)^{\hat{x}} = \hat{g}^{\hat{x}} = y \pmod{p}.$$

Thus by 2 runs of $\mathcal{A}$ and 1 run of the extended Euclidean algorithm, we get the discrete log of y based on g is $a\hat{x} \bmod \phi(p)$, i.e.,

$$\log_g y = a \log_{\hat{g}} y \bmod \phi(p). \tag{1}$$

The time complexity of (1) is just the same as $\hat{f}(p)$, since the complexity of discrete log problem is assumed to be higher than $\log p$, which is the complexity of the extended Euclidean algorithm on $\gcd(t, \phi(p))$. Thus, the choice of g does not affect the complexity of the discrete log problem.

[†] R. J. McEliece, *Finite Fields for Computer Scientists and Engineers*, Boston: Kluwer Academic Publishers, 1987.

Ling Li, ling@cs.caltech.edu

9.3 Professor Evenbetter's method (Collaborate with Xin Yu)

I want to say 3 points about Professor Evenbetter's method.

1. The decoding is almost always unique, though there do exist many ciphertexts that have 2 square roots with form $xr0\dots0 \bmod n$ and distinct x.
2. We can slightly modify the method to make the decoding always unique.
3. The proof that Rabin's cryptosystem is equivalent to factoring (I will call it "Rabin's proof" below) breaks down for this one.

Here are the details:

1. Suppose for ciphertext y there are two square roots $xr0\dots0$ and $x'r'0\dots0$ such that x, r, x' and r' are of (or, less than) $k/3$ bits and $x \neq x'$. Without loss of generality, assume $x > x'$. From
$$(xr0\dots0)^2 = (x'r'0\dots0)^2 = y \pmod{n},$$
we get (note that $2^{-1} = (n+1)/2 \bmod n$)
$$(xr)^2 = (xr0\dots0)^2 \cdot 2^{-2k/3} = (x'r'0\dots0)^2 \cdot 2^{-2k/3} = (x'r')^2 \pmod{n}.$$
Since $xr + x'r'$ is the sum of two $2k/3$ bits numbers, it is less than n which is k-bit. So $xr \neq -x'r' \bmod n$.[‡] From Rabin's proof, we know
$$\begin{cases} xr + x'r' = tp \\ xr - x'r' = t'q \end{cases} \text{or} \begin{cases} xr + x'r' = t'q \\ xr - x'r' = tp \end{cases}$$
for some positive integers t and t' (since $x > x'$). Thus
$$xr = \frac{tp + t'q}{2}, \quad x'r' = \frac{|tp - t'q|}{2}. \tag{2}$$
(Note that since p, q are odd and xr, $x'r'$ are integers, the parities of t and t' must be the same.) In other hand, if for some positive numbers t and t', the two numbers xr and $x'r'$ defined in (2) are of or less than $2k/3$ bits, we have
$$(xr)^2 - (x'r')^2 = tt'pq = tt'n = 0 \pmod{n},$$
i.e., $(xr)^2 = (x'r')^2 \bmod n$. Thus
$$(xr0\dots0)^2 = (xr)^2 \cdot 2^{2k/3} = (x'r')^2 \cdot 2^{2k/3} = (x'r'0\dots0)^2 \pmod{n},$$
which implies for ciphertext $y = (xr0\dots0)^2$ there are two distinct square roots $xr0\dots0$ and $x'r'0\dots0$.

   Usually, p and q are roughly $k/2$ bits long, thus leafs t and t' a space of within rough $2k/3 - k/2 = k/6$ bits. Then the combination of t and t' (with the same parity) is about $2^{k/3-2}$, or just say, $O(2^{k/3})$. This is the rough number of ciphertexts that have no unique decoding. Compared with the whole space of ciphertexts, whose volume is $O(2^{2k/3})$ since xr is of $2/3k$ bits long, it is very small, and the ratio tends to 0 when k becomes larger and larger.

[‡]Since $n = pq$ is an odd integer, $xr0\dots0 \neq -x'r'0\dots0 \bmod n$, for $0 < xr0\dots0 + x'r'0\dots0 < 2n$ and it is even. Thus we also know $xr \neq -xr \bmod n$.

Ling Li, ling@cs.caltech.edu

2. Define the ciphertext that has no unique encoding the *invalid ciphertext*. For a fixed message x, there are $2^{k/3}$ choices for r. The encoder can try several r's to ensure that $(xr0\dots0)^2$ is not an invalid ciphertext. Since the ratio of invalid ciphertexts to all ciphertexts is very small (see above discussion), we can always find a good r such that $(xr0\dots0)^2$ is not invalid with very small number of tries.

3. In Rabin's proof, x^2 is fed into the cracking algorithm and a root x' other than $\pm x$ is expected to be produced by the algorithm. Thus n can be factored by calculating $\gcd(n, x \pm x')$.

   However, if we feed $(xr0\dots0)^2$ into a cracking algorithm to Professor Evenbetter's method, the possibility that we would get a root $x'r'0\dots0$ other than $xr0\dots0$ is very small, roughly $2^{-k/3}$ (see discussion 1). And it is totally not possible if the modification in discussion 2 is adopted. Thus it is not feasible to continue the proof that it will be equivalent to the problem of factoring.

   One may argue that the input to the cracking algorithm may not be of the form $(xr0\dots0)^2$. Yes, we can feed any x^2 into the cracking algorithm and hope it will return some $x'r'0\dots0$ such that $(x'r'0\dots0)^2 = x^2 \bmod n$ and $x'r'0\dots0 \neq \pm x \bmod n$. However, as we have seen in discussion 1, the volume of the valid ciphertext space is about $2^{2k/3}$. Thus the probability that a random chosen x^2 will make the cracking algorithm return $x'r'0\dots0$ is roughly $2^{-k/3}$, still very small.

   Another thing I want to mention here is that a cracking algorithm to Professor Evenbetter's method need not provide the whole string of $xr0\dots0$. It is only required that it can calculate x from $(xr0\dots0)^2$.

Ling Li, ling@cs.caltech.edu

9.4 El Gamal with the same k

By using the same random number k, the El Gamal cryptosystem puts

$$x_1 \to (c^k \bmod p, x_1 d^k \bmod p) \quad \text{and} \quad x_2 \to (c^k \bmod p, x_2 d^k \bmod p).$$

Since the adversary knows both ciphertexts and x_1, he/she can calculate

$$\left[(x_2 d^k \bmod p)(x_1 d^k \bmod p)^{-1} x_1\right] \bmod p$$

to get $x_2 \bmod p$, since

$$\left[(x_2 d^k \bmod p)(x_1 d^k \bmod p)^{-1} x_1\right] \bmod p = (x_2 d^k d^{-k} x_1^{-1} x_1) \bmod p = x_2 \bmod p.$$

From $x_1 \neq 0 \pmod p$ and $d = c^a \bmod p$ and $c \in Z_p^*$, we have $x_1 d^k \in Z_p^*$, so $(x_1 d^k \bmod p)^{-1}$ exists and can be calculated by the extended Euclidean algorithm.

9.5 Man in the middle & CA

By introducing the Certificate Authority (CA) into the system, every one has his own private key S and public key P. Then Alice can send Bob her ID (issued by CA) and the g^x, both signed with her private key S_A. Here, 'signed with private key' means to send two copies of messages, with one encrypted with the private key and the other not. Bob can verify the ID with the public key of CA and knows the validity of the ID and the owner of the ID. Bob can also verify the ID and g^x with the public key P enclosed in the ID to know whether the sender owns the private key S which is paired with P. That is, Bob can know whether the sender is the owner of the ID. After these two verifications, Bob verifies the identity of Alice and gets g^x from Alice.

Then Bob sends his ID and g^y signed with his private key S_B to Alice. By the same procedures, Alice can verify the identity of Bob and get g^y. Thus they can share the same key g^{xy}.

In this case, Moe can get g^x and g^y, and both Alice's and Bob's ID. However, he can not change g^x to $g^{x'}$ or g^y to $g^{y'}$ since he has no private key S_A or S_B; he can not change the transmitted public key (thus he can use his own private key to sign a new $g^{x'}$) since he can not produce an ID with a new public key but with the identity as Alice or Bob. Thus Moe can not know the conversation contents between Alice and Bob, provided that get g^{xy} from g^x and g^y is fairly hard. So, the introducing of CA does defeat Moe as a man in the middle.

Ling Li, ling@cs.caltech.edu

9.6 El Gamal and Diffie-Hellman

Here, $\mathcal{E}$ is used to denote the algorithm that can solve $x \bmod p$ by given prime number p, primitive element $c \in Z_p^*$, $d = c^a \bmod p$ and the encrypted message $(c^k \bmod p, xd^k \bmod p)$. The random numbers a and k are unknown to $\mathcal{E}$. That is, $\mathcal{E}$ can crack the El Gamal cryptosystem. $\mathcal{D}$ is used to represent the algorithm that can get $g^{ab} \bmod p$ by given prime number p, primitive element $g \in Z_p^*$, $g^a \bmod p$ and $g^b \bmod p$. However, a and b are unknown to $\mathcal{D}$. So $\mathcal{D}$ is the algorithm to crack the Diffie-Hellman key exchange protocol. Let $\mathcal{E}$ and $\mathcal{D}$ share one p.

1. $\mathcal{E} \Rightarrow \mathcal{D}$. Now we have $\mathcal{E}$ and want to design $\mathcal{D}$. Let $c = g$, $d = g^a \bmod p$, $x = g^{ab}$, and $k = -b$, though the last two values are unknown. Thus the encrypted message is $(c^k \bmod p, xd^k \bmod p) = (g^{-b} \bmod p, 1)$. Since from the extended Euclidean algorithm we can get $g^{-b} = (g^b)^{-1}$ with modulus p. Now we have p, c, d, and the encrypted message $(g^{-b} \bmod p, 1)$, the algorithm $\mathcal{E}$ can get the message $x \bmod p$, which is $g^{ab} \bmod p$.

2. $\mathcal{D} \Rightarrow \mathcal{E}$. When p, c, d and $(c^k \bmod p, xd^k \bmod p)$ as described above are given, we can feed c as g, d as g^a, $(c^{-k} \bmod p)$ as g^b to algorithm $\mathcal{D}$, and get

   $$g^{ab} \bmod p = d^{-k} \bmod p.$$

   Then we can get $x \bmod p$ by

   $$(xd^k \bmod p) \cdot (g^{ab} \bmod p) = xd^k d^{-k} \bmod p = x \bmod p.$$

   Since c is a primitive element of Z_p^*, we know $c^{-k} \bmod p$ exists and it can be calculated by the extended Euclidean algorithm on $\gcd(c^k \bmod p, p)$.

The transformations between $\mathcal{D}$ and $\mathcal{E}$ only need 1 run of the extended Euclidean algorithm and a constant number of overload (multiplication, mod), thus the complexity is far below polynomial-time. So $\mathcal{D}$ and $\mathcal{E}$ are computationally equivalent, i.e., cracking the El Gamal cryptosystem and the Diffie-Hellman protocol are computationally equivalent.