Lecture 8: Public-key Crypto I
Diffie-Hellman Key Exchange

William Stein

Math 124 HARVARD UNIVERSITY **Fall 2001**

Key Ideas

- Public-key cryptography
- The Diffie-Hellman key exchange

1 Public-key Cryptography

Nikita must communicate vital information to Michael, who is a thousand kilometers away. Their communications are being monitored by The Collective, which must not discover the message. If Nikita and Michael could somehow agree on a secret encoding key, they could encrypt their message. Fortunately, Nikita knows about an algorithm developed by Diffie and Hellman in 1976.

2 The Diffie-Hellman Key Exchange Protocol

Nikita and Michael agree on a prime number p and an integer g that has order $p-1$ modulo p. (So $g^{p-1} \equiv 1 \pmod{p}$, but $g^n \not\equiv 1 \pmod{p}$ for any positive $n < p-1$.) Nikita chooses a random number $n < p$, and Michael chooses a random number $m < p$. Nikita sends $g^n \pmod{p}$ to Michael, and Michael sends $g^m \pmod{p}$ to Nikita. Nikita can now compute the secret key:

$$s = g^{mn} = (g^m)^n \pmod{p}.$$

Likewise, Michael computes the secret key:

$$s = g^{mn} = (g^n)^m \pmod{p}.$$

Now Nikita uses the secret key s to send Michael an encrypted version of her critical message. Michael, who also knows s, is able to decode the message.

Meanwhile, hackers in The Collective see both $g^n \pmod p$ and $g^m \pmod p$, but they aren't able to use this information to deduce either m, n, or $g^{mn} \pmod p$ quickly enough to stop Michael from thwarting their plans. Yeah!

The Diffie-Hellman key exchange is the first public-key cryptosystem every published (1976). The system was discovered by GCHQ (British intelligence) a few years before Diffie and Hellman found it, but they couldn't tell anyone about their work; perhaps it was discovered by others before. That this system was discovered independently more than once shouldn't surprise you, given how simple it is!

2.1 Some Quotes

A review of Diffie and Hellman's groundbreaking article is amusing, because the reviewer, J.S. Joel, says "They propose a couple of techniques for implementing the system, but the reviewer was unconvinced."

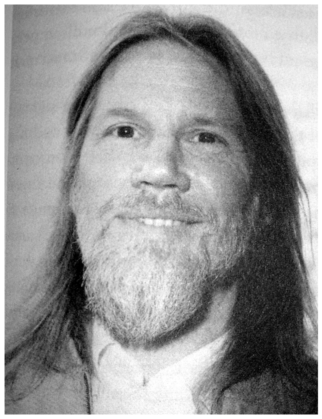
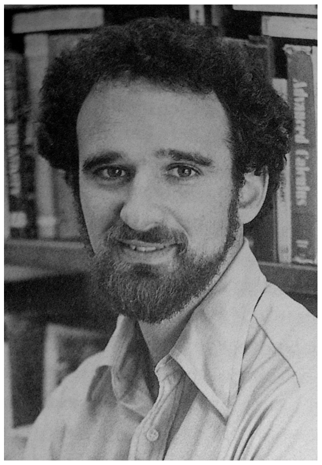
```
Diffie, Whitfield; Hellman, Martin E.
 New directions in cryptography.
 IEEE Trans. Information Theory IT-22 (1976), no. 6, 644--654.
```

> The authors discuss some of the recent results in communications theory that have arisen out of the need for security in the key distribution channels. They concentrate on the use of ciphers to restrict the extraction of information from a communication over an insecure [channel]. As is well known, the transmission and distribution is then likely to become a problem, in efficiency if not in security. The authors suggest various possible approaches to avoid these further problems that arise. The first they call a "public key distribution system", which has the feature that an unauthorized "eavesdropper" will find it computationally infeasible to decipher the message since the enciphering and deciphering are governed by distinct keys. They propose a couple of techniques for implementing the system, but the reviewer was unconvinced.

Somebody named Alan Westrope wrote in 1998 about political implications:

> The 1976 publication of "New Directions in Cryptography", by Whitfield Diffie and Martin Hellman, was epochal in cryptographic history. Many regard it as the beginning of public-key cryptography, analogous to a first shot in what has become an ongoing battle over privacy, civil liberties, and the meaning of sovereignty in cyberspace.

Here is what Diffie and Hellman look like, respectively:

3 Let's try it!

To make finding g easier, let's choose a prime p such that $(p-1)/2 = q$ is prime (so $p-1 = 2q$, with q prime). Since for any g with $\gcd(g,p) = 1$,

$$g^{2q} \equiv 1 \pmod{p},$$

the order of g is 1, 2, q, or $2q = p-1$, so the order of g is easy to compute.

For our first example, let $p = 23$. Then $g = 5$ has order $p - 1 = 22$. (I found $g = 5$ using the function `znprimroot` in PARI. You can also just compute the order of 2, 3, etc., until you find a number with order $p-1$.)

Nikita: Chooses secret $n = 12$; sends $g^{12} = 5^{12} \equiv \mathbf{18} \pmod{23}$.

Michael: Chooses secret $n = 5$; sends $g^5 = 5^5 \equiv \mathbf{20} \pmod{23}$.

Compute Shared Secret:
Nikita: $20^{12} \equiv \mathbf{3} \pmod{23}$
Michael: $18^5 \equiv \mathbf{3} \pmod{23}$.

4 The Discrete Logarithm Problem

Let a, b, n be positive real numbers. Recall that

$$\log_b(a) = n \text{ if and only if } a = b^n.$$

Thus the $\log_b$ function solves the following problem: Given a base b and a power a of b, find an exponent n such that

$$a = b^n.$$

That is, given b^n and b, find n.

Example 4.1. $a = 19683$, $b = 3$. A calculator quickly gives that

$$n = \log(19683)/\log(3) = 9.$$

The discrete log problem is the analogue of this problem modulo p:

Discrete Log Problem: Given $b \pmod{p}$ and $b^n \pmod{p}$, find n. Put another way, compute $\log_b(a)$, when $a, b \in \mathbb{Z}/p\mathbb{Z}$.

As far as we know, this problem is **VERY HARD** to solve quickly. Nobody has admitted publicly to having proved that the discrete log can't be solved quickly, but many very smart people have tried hard and not succeeded. It's easy to write a *slow* program to solve the discrete log problem. (There are better methods but we won't discuss them in this class.)

```
? dislog(x,g, s) = s=g; for(n=1,znorder(g),if(x==s, return(n), s=s*g)); 0;
? dislog(18,Mod(5,23))
%6 = 12
? dislog(20,Mod(5,23))
%7 = 5
```

So the example above was far too simple. Let's try a slightly larger prime:

```
? p=nextprime(9584)
%8 = 9587
? isprime((p-1)\2)
%9 = 1
? znorder(Mod(2,p))
%10 = 9586
? g=Mod(2,p)
%11 = Mod(2, 9587)
? a = g^389
%15 = Mod(7320, 9587)
? dislog(a,g)
%16 = 389
```

This is still very easy to "crack". Let's try an even bigger one.

```
? p = 9048610007
%1 = 9048610007
?  g = Mod(5,p)
%2 = Mod(5, 9048610007)
? a = g^948603
%3 = Mod(3668993056, 9048610007)
? dislog(a,g)              \\ this take a while
%4 = 948603
? znlog(a,g)        \\ builtin super-optimized version takes about 1/2 second
%31 = 948603
```

Computing the discrete log gets slow quickly, the larger we make the p. Doubling the number of digits of the modulus makes the discrete log much much harder.

4.1 The State of the Art

```
Discrete logarithms in GF(2^n)
From: Reynald LERCIER <lercier@club-internet.fr>
 To: NMBRTHRY@LISTSERV.NODAK.EDU
 Date: Tue, 25 Sep 2001 13:37:18 -0400

We are pleased to announce a new record for the discrete logarithm
problem. We were able to compute discrete logarithms in
GF(2^521). This was done in one month on a unique 525MHz
quadri-processors Digital Alpha Server 8400 computer. The approach
that we followed is a careful implementation of the general Function
Field Sieve as described from a theoretical point of view by Adleman
[Ad94].

As far as we know, the largest such computation previously done was
performed in GF(2^401) [GoMc92] using an algorithm due to Coppersmith
[Co84].

[...]
So, as a conclusion, time that we need for computing discrete
logarithms in GF(2^521) on a 525 MHz quadri-processor alpha server
8400 computer is approximatively 12 hours for each, once the sieving
step (21 days) and the linear algebra step (10 days) is performed.

Antoine JOUX    (DCSSI, Issy les Moulineaux, France, Antoine.Joux@ens.fr),
Reynald LERCIER (CELAR, Rennes, France, lercier@celar.fr).
```

5 Realistic Example

```
? p=nextprime(93450983094850938450983409583)
%17 = 93450983094850938450983409611
? isprime((p-1)\2)
%18 = 0
? nextgoodprime(p) = while(!isprime((p-1)\2), p=nextprime(p+1)); p
? nextgoodprime(p)
%19 = 93450983094850938450983409623
? g=2
%21 = 2
? znorder(Mod(g,p))
%22 = 93450983094850938450983409610
? ?random
random({N=2^31}): random integer between 0 and N-1.
? nikita = random(p)
%23 = 18319922375531859171613379181
```

```
? michael = random(p)
%24 = 82335836243866695680141440300
? nikita_say = Mod(g,p)^nikita
%26 = Mod(17037287637415625385373411504, 93450983094850938450983409611)
? michael_say=Mod(g,p)^michael
%27 = Mod(2201425894324369970772940547, 93450983094850938450983409611)
? secret = nikita_say^michael
%28 = Mod(25591938014843312529239952955, 93450983094850938450983409611)
? secret = michael_say^nikita
%29 = Mod(25591938014843312529239952955, 93450983094850938450983409611)
```