# CSCI3381-Cryptography

Lecture 8: Diffie-Hellman and Related Methods

October 15, 2014

The relevant sections of the textbook are 3.7, 7.1, 7.4, and 7.5.

## 1 Prelude: Merkle Puzzles

The first public-key cryptographic system was a key-agreement protocol invented in 1974 by an undergraduate at Berkeley named Ralph Merkle. Merkle proposed it as a project for a Computer Security course, with the topic, 'Establishing secure communications between separate secure sites over insecure communication lines.'

Over the years, Merkle gave several different accounts of the scheme. The description here is closer to the one in his Ph.D. thesis than the version in his undergraduate proposal. Suppose Alice and Bob want to agree on a symmetric key, say a 56-bit DES key. Alice generates one million different messages that all have the format:

```
ID_TAG:03b59f26 KEY:892f3c490d1b92
```

where the key and ID fields have different values in each of the messages. She then *lightly* encrypts each message: for instance, Alice and Bob can publicly, over the insecure channel, agree on the high-order 32 bits of a DES key, then Alice can generate 24 additional key bits at random for each of the messages and encrypt each message with the resulting 56-bit key. She then sends all one million encrypted messages to Bob.

Bob, on his end, chooses one of the messages at random and proceeds to attack it by brute force. Since the size of the key space is $2^{24} \approx 16$ million, this attack should succeed after a few minutes, as soon as he guesses a key that reveals the characters `ID_TAG:` and `KEY:` in the appropriate positions. Let's suppose the decrypted message is the one shown above. Bob sends the ID field `03b59f26` back to Alice, who looks this up in the list of messages she generated. Now Alice and Bob share the key `892f3c490d1b92`.

Eve knows the scheme, can intercept all one million messages, and sees the ID field that Bob sends to Alice, but can only recover the key by trying to decrypt each of the messages and checking for a match with the ID field. She will have to check 500,000 messages on average. Assuming the brute-force attack requires 5 minutes to decrypt a single message, Eve will need five years to recover the key.

There are two security parameters in this scheme: the difficulty of decrypting a single message, and the number of messages. Alice's effort is proportional to the number of messages, Bob's effort is the difficulty of decryption, and Eve's to the product of the two. To simplify a bit, Eve's effort is proportional to the square of Alice's and Bob's effort. The problem is that this quadratic blowup in effort is not big enough to turn an easy problem into a hard one. If Alice and Bob are ordinary users and Eve is the NSA, Eve will easily bridge the gap. If Alice increases the security parameters significantly, then the time required by her and Bob to agree on a key, which is already pretty large, becomes impractical. What is needed is something more like an exponential blowup, rather than a quadratic increase in effort.

Still, this was a new way of thinking about cryptography. The professor in Merkle's course wrote that his proposal was 'muddled terribly' and rejected it, suggesting that he work on a different topic. Merkle dropped the course, and tried to publish his idea, but his initial version of the paper was also rejected. He moved from Berkeley to Stanford to do graduate study with Martin Hellman, who together with another grad student, Whitfield Diffie, had been working on very similar ideas. Diffie and Hellman developed a different key agreement protocol that seemed to meet the heightened security requirements. (The story is well known, but Merkle does not seem to have gotten his full share of the credit for the invention of public-key cryptography.)

The security of the Diffie-Hellman protocol is based on another presumably hard number-theoretic problem, the Discrete Log problem. This is a little more difficult to explain than factoring; the next section is devoted to the underlying math.

## 2 Primitive Elements Modulo a Prime

When we introduced Fermat's Theorem, we saw that if $p$ is prime, and $0 < a < p$, then the powers of $a$ mod $p$ decomposed into cycles whose length is a divisor of $p-1$. For example, with $p = 13$ and $a = 3$, we have the powers $a^k \pmod 13$ for $k = 1, \ldots, 12$ are

$$3, 9, 1, 3, 9, 1, 3, 9, 1, 3, 9, 1.$$

Thus $3^3 \equiv 1 \pmod{13}$. This of course implies $3^{12} \equiv 1 \pmod{13}$, which is what Fermat's Theorem tells us.

If we choose $a = 6$ instead, we get the following powers:

$$6, 10, 8, 9, 2, 12, 7, 3, 5, 4, 11, 1$$

All 12 nonzero elements mod 13 appear in this list, so the associated cycle has length 1. We call 6 a *primitive element* or a *primitive root* mod 13. Such primitive elements are always guaranteed to exist.

**Fact** If $p$ is a prime, then there will be at least one element $0 < a < p$ such that the powers of $a$ modulo $p$ consist of all $p-1$ elements $1, \ldots, p-1$.

### 2.1 How Many Primitive Elements?

Our calculations above show that 6 is a primitive elment mod 13, but 3 is not.

How many primitive elements mod 13 are there? Every element of $\{1, \ldots, 12\}$ has the form $6^k \bmod 13$ for some $k = 0, 1, ..., 11$. If $k$ has a factor in common with 12, then by Fermat's Theorem, $(6^k)^t \bmod 13 = 1$ for some $t$ smaller than 12, and thus $6^k$ cannot be a primitive element. But if $k$ has no factor in common with 12, such a $t$ cannot exist. Thus the number of primitive elements mod 13 is equal to the number of elements of $\{0, \ldots, 11\}$ that are relatively prime to 12.

Let's see how this plays out in our example. The numbers less than 12 that are relatively prime to 12 are 1,5,7,11. Thus the primitive roots are $6, 6^5 \bmod 13, 6^7 \bmod 13, 6^{11} \bmod 13$. In other words, the primitive elements are 6, 2, 7, and 11. The number of positive integers less than $n$ and relatively prime to $n$ is denoted $\phi(n)$. So the number of primitive elments mod $p$ is $\phi(p-1)$.

### 2.2 Discrete Logarithms

Suppose that we're given a large prime $p$, a large primitive element $a$ mod $p$, and a large integer $k$. "Large" for us means googol-sized: too big to count up to, but not too big to write down. We know we can compute $a^k \bmod p$ easily by the repeated squaring algorithm.

What about going in the opposite direction? That is, suppose we're given a nonzero element $b$ of $\{1, \ldots, p-1\}$ and we know a primitive element $a$. There is a unique $k$ such that $0 \leq k < p-1$ such that $a^k \bmod p = b$. (This value of $k$ is called the mod $p$ *discrete logarithm* of $b$ at base $a$. For instance, if $p = 13$, $a = 6$ and $b = 3$, the discrete logarithm is 8.) How do we find $k$? We could compute *all* the powers of $a$ mod $p$, and stop when we found $b$, but this is not feasible with such large values. Although there are better algorithms for finding discrete logs, there is currently none so good that it represents a practical alternative to brute-force search when the numbers get very large. The function $x \mapsto a^x \bmod p$ is another candidate for a one-way function: easy to compute, impossible in practice to invert, and it figures in a number of different cryptographic schemes. (The current record, achieved earlier this year, was the computation of a discrete log modulo a prime with 180 decimal digits.)

## 3 Diffie-Hellman Key Agreement

Like Merkle puzzles, Diffie-Hellman Key Agreement allows two parties who have never met to agree on a key to use for subsequent communication with a symmetric cipher. The difference is that the eavesdropper's computational effort to recover the key is, as far as we know, much larger than the effort of the communicating parties, so that key agreement can be very fast, but any attack is infeasible.

First, Alice and Bob agree on a large prime $p$ and $\alpha$, a primitive element mod $p$. (See below on finding primitive elements.) This agreement can be made over an insecure channel.

Alice chooses a random $x$ between 2 and $p-1$, computes $\alpha^x \bmod p$, using modular exponentiation, and sends this value to Bob, keeping $x$ secret.

Bob does likewise, picking a random $y$, sending $\alpha^y \bmod p$ to Alice, and keeping $y$ a secret.

Alice can now compute $(\alpha^y)^x \bmod p = \alpha^{xy} \bmod p$, and Bob can compute $(\alpha^x)^y \bmod p = \alpha^{xy} \bmod p$. This common value is the shared key.

### 3.1 Security against cryptanalysis.

Eve knows $p$ and $\alpha$, because these values were sent over an insecure channel. If he were able to compute discrete logs, he could intercept $\alpha^x \bmod p$ and recover $x$, then intercept $\alpha^y \bmod p$ and compute the shared key $\alpha^{xy} \bmod p$. Thus the scheme requires the assumption that computation of discrete logs of very large numbers is infeasible.

In fact, we have to assume something stronger than infeasibility of computing discrete logs: namely that one cannot recover $\alpha^{xy} \bmod p$ from $\alpha^x \bmod p$ and $\alpha^y \bmod p$. It's conceivable that there is an efficient method for doing this that does not entail an efficient method to compute discrete logs. Thus, as is the case with RSA, the security of Diffie-Hellman and related schemes depends upon a number of unproved assumptions about the computational complexity of number-theoretic problems. These assumptions have held up very well in practice: If anyone has found efficient algorithms for these problems, they're not talking about them.

## 3.2 Vulnerability to an active adversary

As with public-key encryption, an active adversary Cruella can launch a man-in-the-middle attack: She can intercept the values sent between Alice and Bob, generate his own random values $x'$ and $y'$, send $\alpha^{y'} \bmod p$ to Alice, and $\alpha^{x'} \bmod p$ to Bob. Cruella now shares the key $\alpha^{xy'} \bmod p$ with Alice, and $\alpha^{x'y} \bmod p$ with Bob. If Alice then sends Bob a message encrypted with the key $\alpha^{xy'} \bmod p$ (which she thinks she shares with Bob), Cruella intercepts it, decrypts it, re-encrypts it with $\alpha^{x'y} \bmod p$, and sends the result on to Bob. Bob decrypts it with the key he thinks he shares with Alice, but in reality shares with Cruella. As long as Cruella can keep the masquerade up, Alice and Bob believe that they are communcicating securely, while Cruella reads everything they send to one another.

Fixing this problem requires some form of authentication of the parties, a subject we will take up later.

Diffie-Hellman key agreement can be adapted to sending a message securely as in the parable of the box with two latches. Let's suppose the plaintext message is $1 < \beta < p$. $\beta = \alpha^x \bmod p$ for some $x$. Alice chooses a secret random $y$ such that $1 < y < p - 1$, and such that $y$ is relatively prime to $p - 1$. This assures that $y^{-1} \bmod p - 1$ exists. She sends $\beta^y = \alpha^{xy} \bmod p$ to Bob. Bob similarly chooses a $z$ relatively prime to $p - 1$, raises Alice's message to the power $z$, and sends $\alpha^{xyz} \bmod p$ back to Alice. Alice raises to the power $y^{-1} \bmod p - 1$, giving $\alpha^{xz} \bmod p$, and sends this to Bob. Bob now applies $z^{-1} \bmod p - 1$, giving $\alpha^x \bmod p = \beta$. (This is the 'three-pass protocol' described in Section 3.6.1 of the textbook.)

## 3.3 Finding primitive elements mod $p$

As far as I know, there isn't a really good algorithm for doing this for arbitrary primes $p$. If $p-1$ has the form $2q$, where $q$ itself is prime, then there is a simple method for finding primitive elements. Thus a reasonable strategy is to repeatedly generate random primes $q$ and test to see if $p = 2q + 1$ is prime. For hundred-digit numbers, you may have to apply the primality test as many as one hundred thousand times to locate such a pair $(p, q)$. This is slow, but since this computation is just the preparatory phase of the algorithm, one can afford the extra time. Once $p$ is found, we guess a random $\alpha$ between 2 and $p-1$. How can we tell if $\alpha$ is a primitive element? We know there is at least one primitive element $\beta$, and we know that $\alpha = \beta^k \bmod p$ for some $k$. Furthermore, we know that $\alpha$ is a primitive root if and only if $k$ is relatively prime to $p-1 = 2q$, which means that $k$ is neither even, nor divisible by $q$. Of course we cannot compute $k$—that would mean solving the discrete log problem. But we can determine if $k$ is divisible by 2: If $k$ is even ($k = 2r$ for some $r$) then

$$\alpha^q = \beta^{kq} = (\beta^{2q})^r = (\beta^{p-1})^r \equiv 1^r = 1 \pmod{p},$$

and, by the same argument, if $k$ is divisible by $q$ then $\alpha^2 \equiv 1 \pmod{p}$. We thus compute both $\alpha^2$ and $\alpha^q$ mod $p$. If neither answer is 1, then $\alpha$ is a primitive element. In practice it will take very few trials to locate a primitive element.

## 3.4 Example with Small Numbers

Let $q = 5$ and $p = 2q + 1 = 11$. Suppose we were to guess that 5 is a primitive root mod 11. We note that (calculating mod 11) $5^2 \equiv 3$, $5^4 \equiv 9$, and so $5^5 \equiv 9 \cdot 5 = 45 \equiv 1$. So 5 flunks the test—it's not a primitive element. So let's guess 2. $2^5 = 32 \equiv 10$, and $2^2 = 4$, so 2 passes: It's a primitive element mod 11.

Now suppose Alice and Bob want to use the pair $(11, 2)$ to agree on a key, using the Diffie-Hellman method. Alice generates at random $x = 8$ and computes

$$2^8 = 256 \equiv 3 \pmod{11},$$

and sends this to Bob. Bob generates, at random, 5, and computes

$$2^5 = 32 \equiv 10 \pmod{11},$$

and sends this to Alice. Alice now computes

$$10^8 \equiv (-1)^8 = 1 \pmod{11},$$

and Bob computes

$$3^5 = 243 \equiv 1 \pmod{11},$$

so the agreed value is 1. (This won't happen in real life with 100-digit numbers.)

## 4 ElGamal Public-Key Cryptosystem

Diffie-Hellman is, in a sense, a public-key system, since it enables users to securely agree on a secret key without having to meet. But it does not in itself supply a public-key encryption mechanism. We can use the same ideas, however, to create a public-key system. In essence, Bob publishes $\alpha^y \bmod p$ as his public key and saves $y$ as his private key. Alice, wishing to send a message $m$ to Bob, picks a random $x$ and "encrypts" $m$ with the shared key as $m \cdot \alpha^{xy} \bmod p$. Here are the details.

### 4.1 Key Generation

Bob picks a large prime $p$, a primitive element $\alpha$ mod $p$, and an integer $y$ between 2 and $p-1$. He saves $y$ as his private key, and publishes $(p, \alpha, \alpha^y \bmod p)$ as his public key.

### 4.2 Encryption

Alice breaks the plaintext message into blocks that can be encoded as integers less than $p$. To encrypt such an integer $m$, she generates a random $x$ between 2 and $p-1$, looks up Bob's public-key information, and computes $(\alpha^y)^x = \alpha^{xy} \bmod p$. She then sends Bob the pair

$$(\alpha^x \bmod p, (m \cdot \alpha^{xy}) \bmod p).$$

### 4.3 Decryption

Bob, upon receiving this, takes the first component of the pair and uses his private key $y$ to compute

$$(\alpha^x)^{p-1-y} \bmod p = (\alpha^{xy})^{-1} \bmod p.$$

(The above equation holds because

$$\alpha^{xy}(\alpha^x)^{p-1-y} = \alpha^{(p-1)x+xy-xy} = (\alpha^{p-1})^x \equiv 1^x = 1 \pmod{p}.$$

Alternatively, Bob could compute $(\alpha^x)^y \bmod p$ and then find the multiplicative inverse of this value, using the extended Euclid algorithm, but the method outlined above is faster.)

Bob now takes the second component of the pair to recover the plaintext:

$$(m \cdot \alpha^{xy}) \cdot (\alpha^{xy})^{-1} \bmod p = m.$$

## 4.4 Comments on the algorithm

*Security against active adversaries.* Like all public-key systems, this is vulnerable to an attack by an active adversary who attempts to pass off his own public key as someone else's. This difficulty can be addressed by the use of certified public keys.

*Public keys versus private keys.* Unlike RSA, private keys in this system look different from public keys, and are not interchangeable.

*Plaintext length versus ciphertext length.* The length of the encrypted message is twice the length of the plaintext message, since Alice must send a pair of values to Bob.

*Randomization* It's not necessary to salt the plaintext, since Alice generates a random value each time she transmits a block—this random value is incorporated into the key. It is essential that she do this, since the underlying symmetric encryption algorithm encrypts $m$ as $m \cdot K \bmod p$, where $K$ (in this case $\alpha^{xy} \bmod p$) is the key—it's just a multiplicative cipher. Although brute-force search through the key space is not feasible, if Alice reuses the key $K$ for all the blocks of the message, and one block of plaintext becomes known, then $K$ can be computed and the other blocks will be known as well.

One could avoid this last difficulty by using a different symmetric encryption algorithm. For instance, the high-order 56 bits of $\alpha^{xy} \bmod p$ could be used as a DES key to encrypt the message. This requires the assumption that it is computationally infeasible to determine the high order 56 bits of $\alpha^{xy} \bmod p$, given the values $\alpha^x \bmod p$ and $\alpha^y \bmod p$.

## 4.5 Example with small numbers

Bob's public key is $(7, 3, 3^4 \bmod 7 = 4)$, and his private key is 4. Alice wishes to encrypt the message block 5. She generates the random value $x = 2$ and computes

$$5 \cdot 4^2 \equiv 3,$$

and sends the pair

$$(3^2 \bmod 7 = 2, 3)$$

to Bob.

Bob computes

$$2^{6-4} = 4,$$

from the first component of the ciphertext, then multiplies this by the second component of the ciphertext to obtain

$$4 \cdot 3 \bmod 7 = 5,$$

which is the original plaintext message block.