Cryptology (6)

Public Key Encryption—Diffie-Hellman Key Exchange

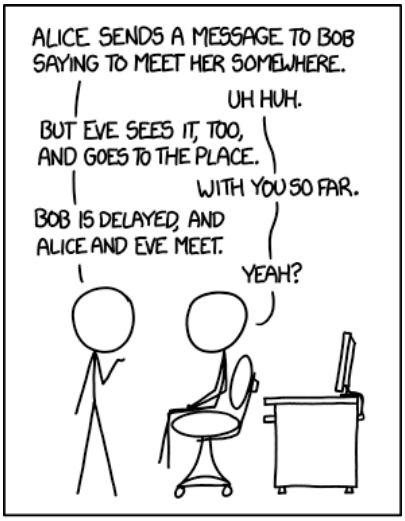
John York, Blue Ridge Community College

http://www.brcc.edu

Much of the information in this course came from “Understanding Cryptography” by Christoff Parr and Jan Pelzl, Springer-Verlag 2010
Much of the classical cryptography material and most Python scripts came from “Cracking Codes with Python” by Al Sweigart, NoStarch Press 2018
Also helpful, “Cryptography Engineering” by Ferguson, Schneier, and Kohno, Wiley Publishing, 2010

Obligatory XKCD Cartoon

https://imgs.xkcd.com/comics/protocol.png
"Changing the names would be easier, but if you're not comfortable lying, try only making friends with people named Alice, Bob, Carol, etc."



Diffie-Hellman Key Exchange (DHKE)

- Alice and Bob exchange info that allows them to create a key
  - Unlike RSA, they don't encrypt messages
  - Once key is determined, switch to another method (AES)
  - Elgamal extension of DHKE does encrypt messages
- Based on Discrete Logarithm Problem (DLP) over a finite field
- Widely used
  - SSH
  - TLS (HTTPS)
  - IPSec

Diffie-Hellman key exchange, and other methods that are based on the discrete logarithm problem, are becoming the predominant methods for public key encryption. This is because of the problems with RSA we've already discussed, plus something we will discuss in Cryptography 10 TLS and HTTPS called forward secrecy. Basically, a person who records an entire RSA key exchange and knows the private key can use the recording to decrypt the entire conversation. This does not work with DHKE.

Discrete Logarithm Problem (DLP)

- In the continuous world this is easy:
  - $5^x = 17$
  - $x = \log_5(17) = \ln(17)/\ln(5) = 1.76037$
- In the discrete world it is not easy:
  - $5^x = 17 \bmod 23$
  - $x = 7$
- There are methods other than brute force, but DLP is still infeasible for large numbers properly chosen

```
>>> for x in range(23):
        print(x, pow(5,x,23))
```

```
0 1
1 5
2 2
3 10
4 4
5 20
6 8
7 17
8 16
9 11
10 9
11 22
12 18
13 21
14 13
15 19
16 3
17 15
18 6
19 7
20 12
21 14
22 1
```

Brute Force Calculation

A major problem with the Caesar cipher was that the encrypted indexes followed an obvious pattern. If the shift was 3, the indexes were 3, 4, 5, …, 25, 0, 1, 2. The Affine cipher improved this by making the sequence less obvious. The DLP takes this another step further (other than for the indexes 0 and 1.)

The word "discrete" is what makes this problem hard. As the exponents get larger, the result is much bigger than the modulus and when we use the mod function the answer "wraps" many times.

There are several algorithms that can compute a discrete logarithm faster than the brute force method, but none are fast enough to solve the DLP for huge numbers.

The Python script in the slide is a simple For Loop that computes the power 5 ^ x, modulo n (23 in this case) for all values of n. The function, range(23), generates the list 0, 1, …, 23.

DHKE Setup

- Choose (or use trusted third party*) public parameters
- Large (2048 bit) prime, p
  - p should be a “safe” prime, such that (p-1)/2 is also prime
- Integer α, $2 < \alpha < p-2$
  - α must be carefully selected

*Selection of p and α is not simple, so some implementations use precomputed values. This was not thought to be a problem, but there are now attacks that allow much of the DLP problem to be solved ahead of time if the value of p is known. When p is 512 bits and known ahead of time, server class machines can solve the DLP. For known p that is 1024 bits long, nation-states can solve the DLP.

We will get to it in a few slides, but you can’t pick just any random numbers for p and α. There are combinations of p and α that would appear to give a huge number of possibilities for α^n, but in fact just give a few possibilities. If those combinations are used, DHKE is easy to crack.

The research that showed that the combination of small (512 or 1024 bit) keys and using same p is not secure is here:
https://weakdh.org/imperfect-forward-secrecy-ccs15.pdf

The preselected values in the DHKE RFC are in the appendix of this document
https://tools.ietf.org/html/rfc7919

Likewise, NIST specifies some values here in appendix D
https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-56Ar3.pdf

Key Exchange

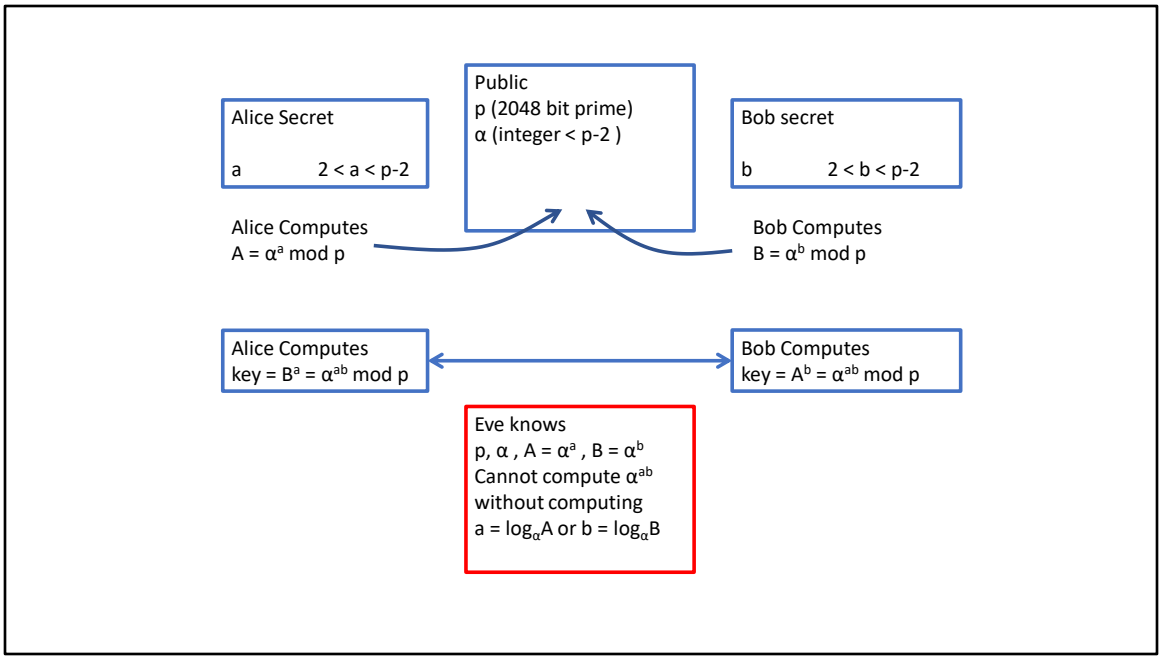
- Alice chooses secret a, gives Bob (public) $A = \alpha^a \bmod p$
- Bob chooses secret b, gives Alice (public) $B = \alpha^b \bmod p$
- Alice computes key $B^a = (\alpha^b)^a = \alpha^{ab} \bmod p$
- Bob computes key $A^b = (\alpha^a)^b = \alpha^{ab} \bmod p$, the same as Alice computed
- $\alpha^{ab} \bmod p$ is the shared key
- Eve knows A, B, α, and p
  - To compute key, must solve $a = \log_\alpha(A) \bmod p$ or $b = \log_\alpha(B) \bmod p$
  - No **known** ways to do this quickly (for 2048-bit p, p and q selected properly)
  - No one has proven that a fast method does not exist, though…

Another way of looking at this is that Eve sees A and B, but not a and b. Alice knows a and Bob knows b. To get the shared key, you have to know either a or b, so Eve is out of luck.

Note that by themselves, Alice and Bob cannot control what the final result, the key, will be. That is why this is key exchange and not encryption. It allows Alice and Bob to agree on a secret key without eavesdroppers intercepting the key. It does not allow Alice and Bob to encrypt a message like RSA does. Once Alice and Bob agree on a secret key, they can switch to AES to encrypt messages.

There are encryption schemes that use the DLP and can encrypt messages; Elgamal is one of them.

A nice thing about DHKE is that the key is computed on the fly, and is different for each session. In RSA, if you record a session and learn the private key somehow (even years later) you can decrypt the session. In DHKE, you can only decrypt the session if either Alice or Bob recorded the keys as they were generated and gave them to you.



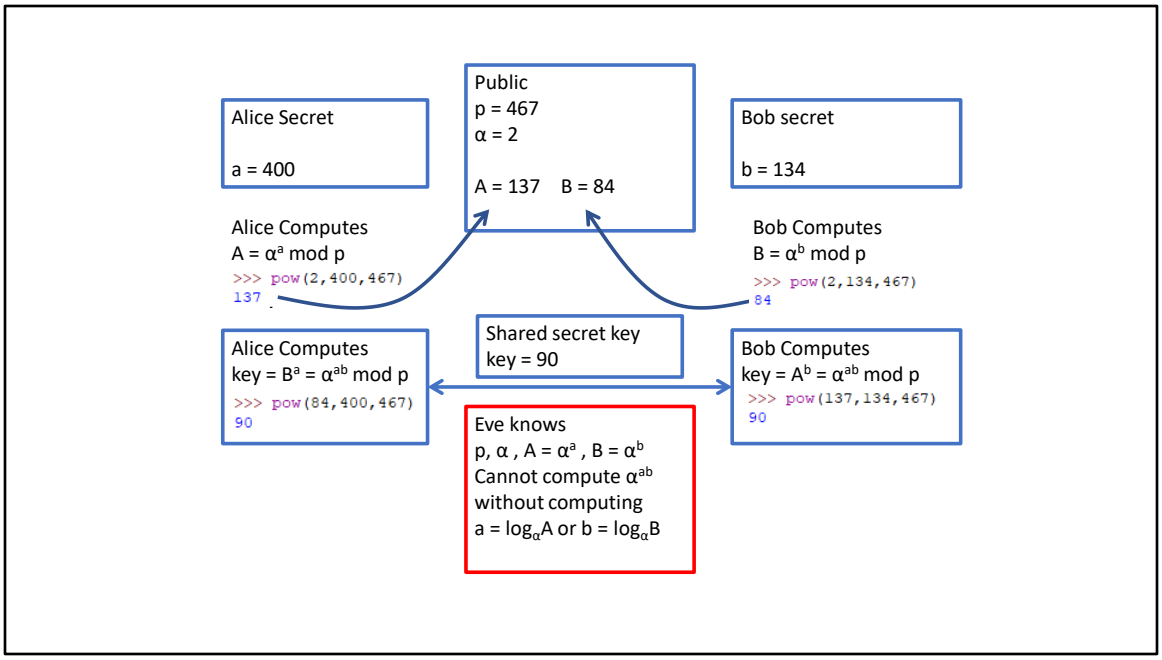
Here are the steps to follow for DHKE
p and α are known ahead of time, or can be included in the setup messages in an unencrypted form.

p _____
α _____

Alice
Select a _____
Compute A _____ $A = \alpha^a \bmod p$
Give A to Bob

Bob
Select b _____
Compute B _____ $B = \alpha^b \bmod p$
Give B to Alice

Alice computes key = $B^a \bmod p$ (she picked a, Bob gave her B) _____
Bob computes key = $A^b \bmod p$ (he picked b, Alice gave him A) _____



In this example, Alice and Bob use:

p	467
α	2

Alice selects a = 400, and gives A to Bob

Select a	400	
Compute A	137	$A = 2^{400} \bmod 467$
Give 137 to Bob		

Bob

Select b	134	
Compute B	84	$B = 2^{134} \bmod 467$
Give 84 to Alice		

Alice computes key = $84^{400} \bmod p$	90
Bob computes key = $137^{134} \bmod p$	90

Alice and Bob have the same shared key = 90

Cryptography in Practice

Public-key cryptography...

In theory:

- ElGamal, NTRU, Ring-LWE, Paillier, MQ, LWE, McEliece, Hash-based signatures, ...

In practice:

(Cipher preference November 2016; censys.io and custom scans)

		Key exchange			Signatures		
	Hosts	RSA	DH	ECDH	RSA	DSA	ECDSA
HTTPS	39M	39%	10%	51%	99%	≈ 0	1%
SSH	17M	≈ 0	52%	48%	93%	7%	0.3%
IKEv1	1.1M	-	97%	3%	-	-	-
IKEv2	1.2M	-	98%	2%	-	-	-

(* Preferences depend on client ordering.)

Nadia Heninger, "The Reality of Cryptographic Deployments on the Internet"

ASIACRYPT 2016

https://www.youtube.com/watch?v=hamXcmaF0ts

Note to self: good info here, need to decide just where it should go, tho…

As of three years ago, RSA was the dominant algorithm by far for computing digital signatures. However, it was used in only 39% of HTTPS key exchanges. This number should be even smaller now, since TLS v1.3 forbids the use of RSA. For the foreseeable future, expect HTTPS to be mostly the elliptic curve variant of DHKE.

Exponents and Subgroups

- Public key is [p, α], where p is the modulus and α the number (base) we will take to a power
- Ideally, α^1, α^2, α^3, …, α^p (all are mod p) give us different numbers for every power < p, and the results cover all the elements 1, …, p-1
  - In practice, that depends on α
- Example with p = 17
  - For α = 3, 5, 6, 7, 10, 11, 12, 14: α^x gives results in 1, …, p-1 as x changes
  - For α = 2, 8, 9, 15: α^x only gives results in 1, 2, 4, 8, 9, 13, 15, 16
    - No matter what we put in for x, we only get 8 of the elements before it repeats
  - For α = 4, 13: α^x only gives results 1, 4, 13, 16. It repeats after 4 elements
  - For α = 16: α^x only gives results 1, 16. It repeats after only 2 elements

This discussion of subgroups is not necessary to understand that the Diffie-Hellman scheme works. It does help to explain why careful selection of p and α is important.

We would prefer that whenever we take α to a power, we get a different answer whenever the power is different. That's not always what happens though. Sometimes, a specific value of α may get trapped in a small loop. For that α, even though we use all possible exponents {1, 2, …, p-1} we may get only a few answers.

In the example with p=17, if we have the bad luck to choose α=4 there are only four possible answers. If we chose α=3, there would be 16 possible answers.

Even worse, if we were foolish enough to choose α=16 or (p-1) there would only be two possible answers. If the attacker knew we were rattling about in this tiny loop, they wouldn't have much trouble in breaking our system.

See the spreadsheet Z-17subgroups.xlsx, which shows the subgroups in Z_{17} that result from different selections of α.

Python Example, p = 17, α = 3, 2, and 16

- Try these examples that compute α^x mod 17. α^x repeats as soon as the result hits 1

Good

```
>>> p = 17
>>> alpha = 3
>>> for x in range(1, p):
        print(pow(alpha, x, p))


3
9
10
13
5
15
11
16
14
8
7
4
12
2
6
1
>>>
```

OK

```
>>> p = 17
>>> alpha = 2
>>> for x in range(1, p):
        print(pow(alpha, x, p))


2
4
8
16
15
13
9
1
2
4
8
16
15
13
9
1
>>>
```

Awful

```
>>> p = 17
>>> alpha = 16
>>> for x in range(1, p):
        print(pow(alpha, x, p))


16
1
16
1
16
1
16
1
16
1
16
1
16
1
16
1
```

This is just a simple Python For Loop, with α values of 3, 2, and 16. You can see that as soon as α^x hits 1, the answers start to repeat. The file Z-17subgroups.xlsx is a simple spreadsheet that shows the exponents for all values of α.

If you like math, p = 17 gives a cyclic group called Z_{17}^*, {1, 2, …, 16} (note that 0 is excluded because it does not have an inverse.) The group's order, $|Z_{17}^*|$, is p-1 = 16 which means the group has 16 members that are relatively prime to 17. The group's order (16) has factors 1, 2, 4, 8, and 16. Therefore, there will be one subgroup each with 1 member, 2 members, 4 members, and 8 members in addition to the main group of 16 members.

In this case, we want to choose α from the numbers that cause α^x to stay in the main group. Those good values for α are 3, 5, 6, 7, 10, 11, 12, and 14. Notice that there are 8 values, the same as Φ(16) = 8.

For a mathematical treatment of this, see "Understanding Cryptography," Paar, Springer 2010, pp. 208 - 216

Subgroups in our example for p = 17

- The main group is {1, 2, 3, …, 16}
  - If we use any of {3, 5, 6, 7, 10, 11, 12, 14} for α, α^x gives us the main group
  - {3, 5, 6, 7, 10, 11, 12, 14} are called generators or primitive elements for the main group
- The subgroup with 8 elements is { 1, 2, 4, 8, 9, 13,15, 16}
  - If α is in {2, 8, 9, 15} α^x is in this subgroup. {2, 8, 9, 15} are primitive elements
- The subgroup with 4 elements is {1, 4, 13, 16}
  - If α is in {4, 13} α^x is in this subgroup. {4, 13} are primitive elements
- The subgroup with 2 elements is {1, 16}
  - If α is in {16} α^x is in this subgroup. 16 is the primitive element

The best way for me to see this was by using Z-17subgroups.xlsx. Each of the factors of p-1 = 17-1 = 16, (1, 2, 4, 8), has a corresponding subgroup with that many elements.

I skipped the trivial subgroup with one element, {1} with the primitive element 1, to save space.

Ramifications of Subgroups

- In Diffie-Hellman key exchange, if we pick an α that is a primitive element of a subgroup, α^a will always be an element of that subgroup.
- Possible values of α^a will be reduced
  - Instead of p-1 possibilities, they are restricted to the size of the subgroup
  - Security is lessened
- Two ways of dealing with this
  - Pick a "safe" number for p, where (p - 1)/2 is also prime
    - Smallest usable subgroup is large, size is (p - 1)/2 – 1
    - (we can easily avoid the subgroups with 1 or 2 elements)
  - Pick p and α so that α generates a subgroup of at least 224 members

In summary, we can't just choose a large random number for p, and simply choose another number for α. It is important to be sure that our choice of α does not generate a very small subgroup.

"Safe" primes are discussed in https://weakdh.org/imperfect-forward-secrecy-ccs15.pdf

The comment that p and α should create a subgroup of at least 224 members comes from NIST. See the chart in the next slide.

Minimum subgroup size (NIST)

This is the NIST diagram from our lesson Cryptology4-Public-Key-Intro-RSA.

D-H (Diffie-Hellman) is circled in the FFC column (Finite Field Cryptography).

It specifies a minimum subgroup size of 224 for a key length (p) of 2048 bits

Calculating the subgroup size when p is this large takes work (our example with p = 17 was trivial.) That's why implementations often use p and α that have been approved by someone else.

Security Strength	Symmetric key algorithms	FFC (e.g., DSA, D-H)	IFC (e.g., RSA)	ECC (e.g., ECDSA)
≤ 80	2TDEA[21]	$L = 1024$ $N = 160$	$k = 1024$	$f = 160\text{-}223$
112	3TDEA	$L = 2048$ $N = 224$	$k = 2048$	$f = 224\text{-}255$
128	AES-128	$L = 3072$ $N = 256$	$k = 3072$	$f = 256\text{-}383$
192	AES-192	$L = 7680$ $N = 384$	$k = 7680$	$f = 384\text{-}511$
256	AES-256	$L = 15360$ $N = 512$	$k = 15360$	$f = 512+$

This chart is from https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt1r4.pdf

If you use Diffie-Hellman with a key size of 2048 bits, and p and α selected so that the subgroup size to be 224 bits, NIST considers the security to be equivalent to a 112-bit symmetric cipher. If you want security equivalent to AES 128, you need a key size of 3072 and a subgroup size of 256.