# DOCUMENTATION

**EXPERIMENT:-**DIFFIE-HELLMANN KEY ESTABLISHMENT.

**LAB:-**CRYPTOGRAPHY LAB

## INTRODUCTION:-

Diffie-Hellman is a way of *generating* a shared secret between two people in such a way that the secret can't be seen by observing the communication. That's an important distinction: **You're not *sharing information* during the key exchange, you're *creating a key* together.**

## USE:-

**This is particularly useful because you can use this technique to create an encryption key with someone, and then start encrypting your traffic with that key. And even if the traffic is recorded and later analyzed, there's absolutely no way to figure out what the key was, even though the exchanges that created it may have been visible. This is where perfect forward secrecy comes from. Nobody analyzing the traffic at a later date can break in because the key was never saved, never transmitted, and never made visible anywhere.**

The way it works is reasonably simple. A lot of the math is the same as you see in public key crypto in that a trapdoor function is used. And while the discrete logarithm problem is traditionally used (the $x^y$ *mod* $p$ business), the general process can be modified to use elliptic curve cryptography as well.

But even though it uses the same underlying principles as public key cryptography, this is *not* asymmetric cryptography because nothing is ever encrypted or decrypted during the exchange. It is, however, an essential building-block, and was in fact the base upon which asymmetric crypto was later built.

## EXPERIMENT PROCEDURE:-

➔ come up with two prime numbers **g** and **p** and tell you what they are.

➔ You then pick a secret number (**a**), but you don't tell anyone. Instead you compute $\mathbf{g}^{\mathbf{a}}$ *mod* **p** and send that result back to me. (We'll call that **A** since it came from **a**).

➔ Do the same thing, but we'll call my secret number **b** and the computed number **B**. So I compute $\mathbf{g}^{\mathbf{b}}$ *mod* **p** and send you the result (called "**B**")

➔ Now, you take the number I sent you and do the exact same operation with *it*. So that's $\mathbf{B}^{\mathbf{a}}$ *mod* **p**.

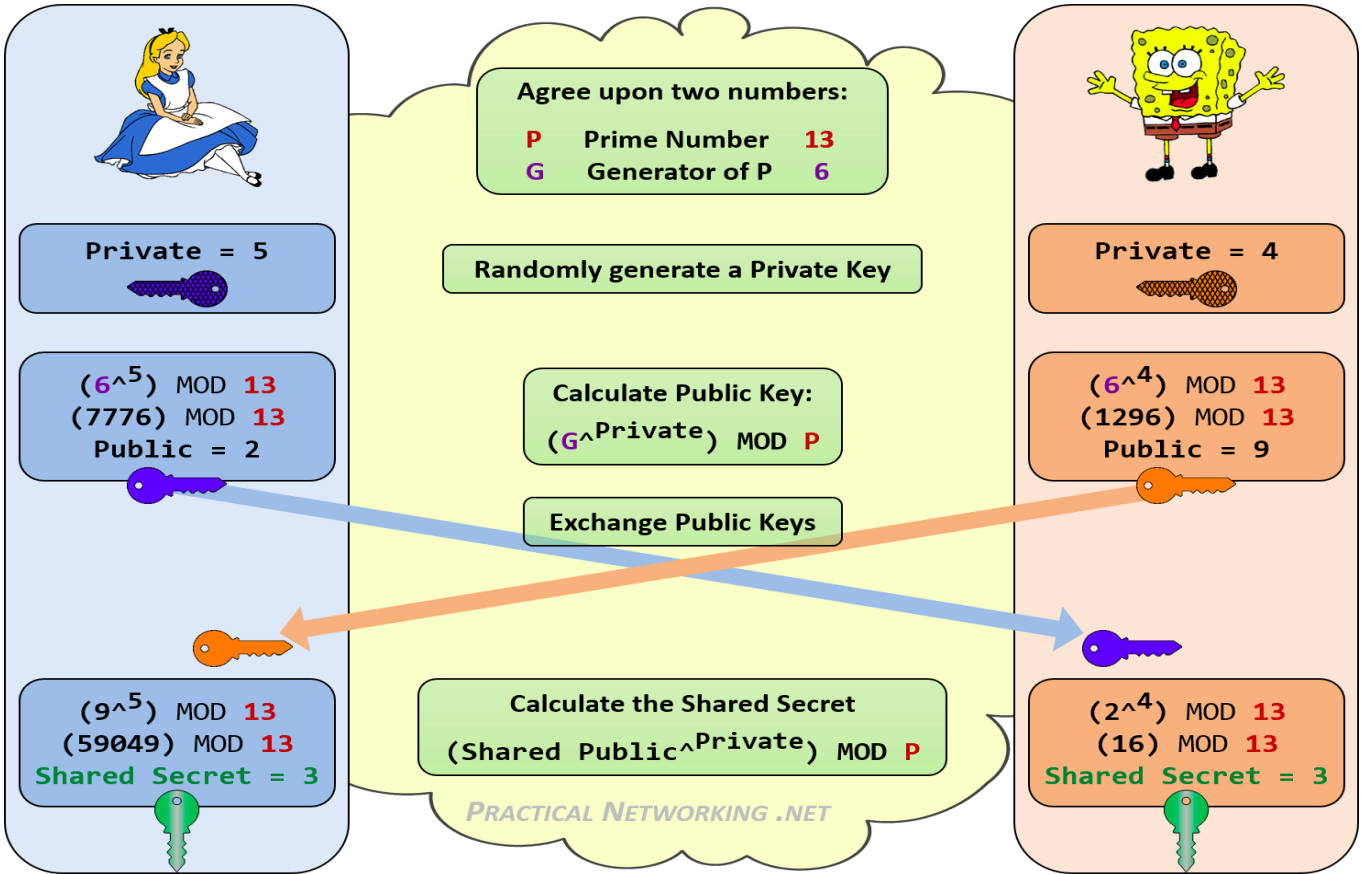
➔ Do the same operation with the result you sent me, so: $\mathbf{A}^{\mathbf{b}}$ *mod* **p**.

## RESULT:-

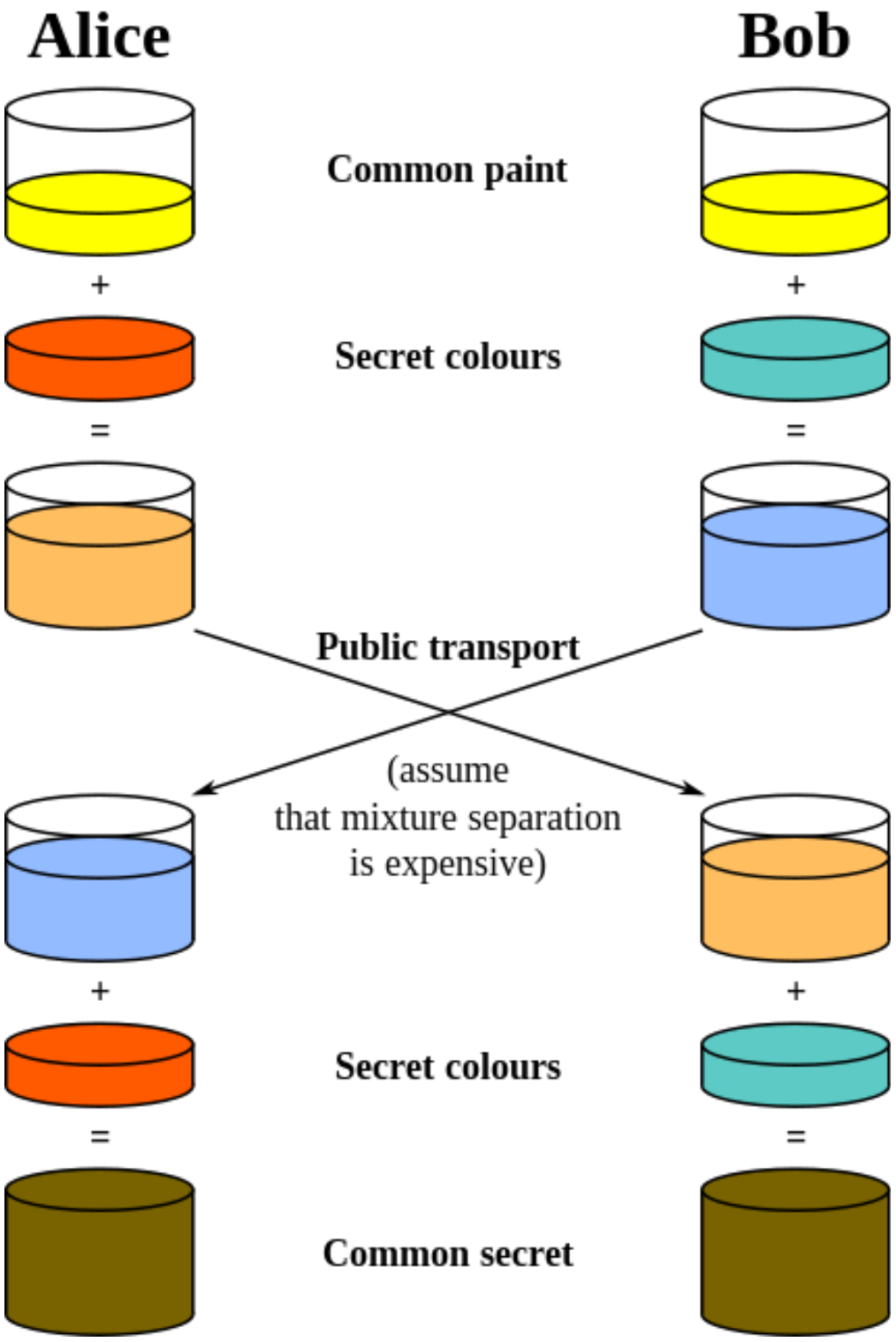
The "magic" here is that the answer I get at step 5 is *the same number* you got at step 4. Now it's not really magic, it's just math, and it comes down to a fancy property of modulo exponents.

$$(g^a \bmod p)^b \bmod p = g^{ab} \bmod p$$
$$(g^b \bmod p)^a \bmod p = g^{ba} \bmod p$$

## EXAMPLE-1:-



**EXAMPLE-2:-**


Alice
Bob
Common paint
+
+
Secret colours
=
=
Public transport
(assume
that mixture separation
is expensive)
+
+
Secret colours
=
=
Common secret

## PSEUDO CODE:-

```
from cryptography.hazmat.backends import default_backend
from cryptography.hazmat.primitives.asymmetric import dh

parameters = dh.generate_parameters(generator=2,
key_size=512, backend=default_backend())

a_private_key = parameters.generate_private_key()
a_peer_public_key = a_private_key.public_key()

b_private_key = parameters.generate_private_key()
b_peer_public_key = b_private_key.public_key()

a_shared_key = a_private_key.exchange(b_peer_public_key)
b_shared_key = b_private_key.exchange(a_peer_public_key)

print 'a_secret: '+a_shared_key
print 'b_secret: '+b_shared_key
```

## TEST CASES:-

| | P | G | a | b |
|---|---|---|---|---|
| 1. | 28201 | 12263 | 5738334 | 4149870 |
| 2. | 64341839 | 6418473687326541851 | 73897938 | 439321413 |
| 3. | 36854637539731995079217806081 | 13611869 | 752657557 | 176540905 |
| 4. | 27320486073538567382001436308694 1906687 | 1474066112874910397783025776 35687158961 | 8675676518 | 9909156 |

## FUNCTIONS USED:-

### TRY:

The `try` block lets you test a block of code for errors.

### EXCEPT:

The `except` block lets you handle the error.

## getrandbits:

You can generate these just random required-bit strings using the random module's getrandbits function that accepts a number of bits as an argument.

## randrange:

Used to generate the pseudo-**random number** between the given **range** of values.

## FLASK:

Flask is a popular Python web framework, meaning it is a third-party Python library used for developing web applications.

## Database:

The database is a collection of organized information that can easily be used, managed,update, and they are classified according to their organizational approach (SQL is a database) .

## flask_sqlalchemy:

Flask-SQLAlchemy is an extension for Flask that adds support for SQLAlchemy to your application.