Imperial College London

DIFFIE–HELLMAN KEY EXCHANGE: FROM 2 PARTIES TO n PARTIES

Jiaru (Eric) Li[1]

[1]**Department of Mathematics, Imperial College London**

INTRODUCTION

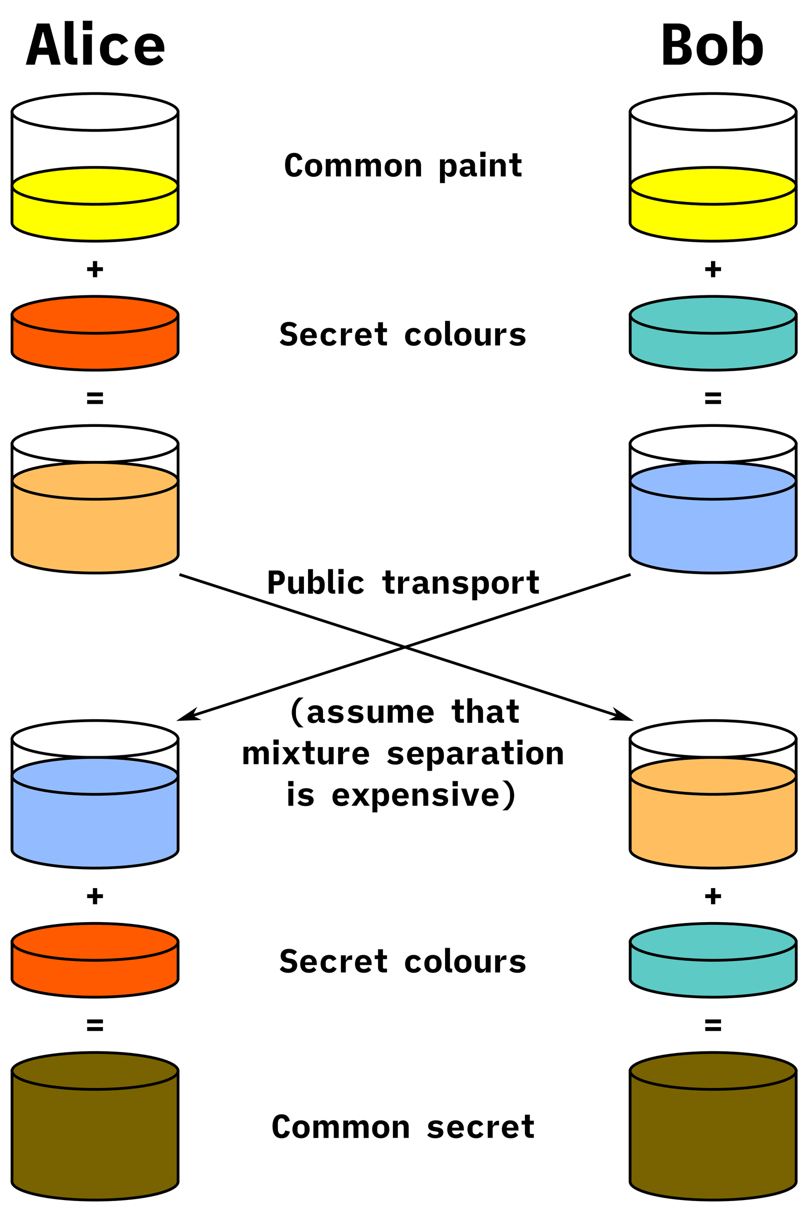
We begin by introducing some basic concepts.

- **Cryptography** is the study of methods of sending messages in disguised form so that only the intended recipients can remove the disguise and read the message. (Koblitz, 1994)
- **Cryptographic key**, or simply **key**, is a piece of information used to encode or decode the messages.
- **Key exchange** is a method where keys are exchanged between parties, so that further cryptographic algorithms could be used.
- **Diffie–Hellman key exchange** is a key exchange algorithm named after Whitfield Diffie and Martin Hellman.

This poster outlines the principle and implementation of the original Diffie–Hellman key exchange for 2 parties (hereinafter **2-DH**) and generalises the algorithm to n parties (hereinafter ***n*-DH**).

Here is a figure that illustrates how 2-DH works.

FIGURE 1



Alice and Bob wish to create a common secret (key) b from some information a that they have publicly agreed on. To do this, Alice and Bob each uses their own encoding algorithms f and g to encrypt their own messages, creating $f(a)$ and $g(a)$ respectively. After that, they send them to each other and encode once again, getting $f(g(a))$ for Alice and $g(f(a))$ for Bob. If f and g commute, we have $b := f(g(a)) = g(f(a))$. An eavesdropper could overhear the whole conversation, obtaining $f(a)$ and $g(a)$, yet they cannot work out b if it is hard to find f^{-1} and g^{-1}.

ORIGINAL IMPLEMENTATION (Diffie and Hellman, 1976)

The original implementation of 2-DH makes use of the primitive root. A number g is called a **primitive root modulo n** if for every $a \in \mathbb{N}$ coprime to n, there exists some $k \in \mathbb{N}$ such that $g^k \equiv a \pmod{n}$.

Given that, Alice and Bob need to agree on a pair of values (p, g) where p is a prime number and g is a primitive root modulo p. This is to ensure that the resulting common secret could take any value between 1 and $p - 1$. Then they each generate a secret natural number a and b and compute $A := g^a \bmod p$ and $B := g^b \bmod p$ respectively. After that, Bob receives A from Alice and vice versa. They perform the modulo operation once again, giving $B^a \bmod p$ and $A^b \bmod p$. As $(g^b \bmod p)^a \bmod p = (g^a \bmod p)^b \bmod p$, Alice and Bob now have a common value c. Even if someone intercepts the whole transmission, they would still not be able to work out c from g, A and B easily.

EXTENSION TO FINITE CYCLIC GROUPS

Recall that a group G is called **cyclic** if there exists an element $g \in G$, called its **generator**, such that $G = \{g^k \mid k \in \mathbb{Z}\}$. In particular, for every $a \in G$, there exists an integer k such that $a = g^k$. This number k is called the **discrete logarithm** of a with respect to g. It is assumed (and believed) that, in general, there is no efficient algorithm finding the discrete logarithm, thus ensuring the vulnerability of 2-DH is minimised.

Now Alice and Bob need to find a pair of numbers (n, g) where $n \in \mathbb{N}$ and g is the generator of some finite cyclic group G of order n. They subsequently pick some number a and b where $1 < a, b < n$, compute g^a and g^b, exchange the results, then raise the power once again, giving $c := (g^b)^a = (g^a)^b = g^{ab}$.

Notice that taking $G = (\mathbb{Z}/p\mathbb{Z})^\times$ gives back the original version.

FROM 2-DH TO 3-DH

Now that we have considered 2 parties, let us try to generalise the algorithm to the case of 3 parties by adding another party, Carol.

- Alice, Bob and Carol agree on their (n, g) as defined above and generate their private keys a, b, c.
- Alice computes g^a and sends it to Bob.
- Bob computes $(g^a)^b = g^{ab}$ and sends it to Carol.
- Carol computes $(g^{ab})^c = g^{abc}$.
- Repeat the process, now from Bob to Carol to Alice and from Carol to Alice to Bob.
- They each have a common value g^{abc} which cannot be calculated easily from any combination of $g, g^a, g^b, g^c, g^{ab}, g^{ac}, g^{bc}$.

FROM 3-DH TO n-DH

It is easy to generalise the algorithm above to any number of parties. Denote these parties as A_1 to A_n, each possessing their own private key a_1 to a_n. Then they transfer the information in the following directions: $A_1 \to A_2 \to \cdots \to A_n$, $A_2 \to A_3 \to \cdots \to A_1$, $\cdots$, $A_n \to A_1 \to \cdots \to A_{n-1}$, and they get a common value $g^{\prod_{i=1}^n a_i}$.

However, this is not efficient when n gets large. Notice that each party needs to perform n times of exponentiation, giving a time complexity of $O(n^2)$. By using a **divide-and-conquer** algorithm, one might reduce it to $O(n \log n)$.

We first consider a simpler case where there are 2^m ($m \in \mathbb{N}$) parties, from A_1 to A_{2^m}. Perform the following process recursively:

- Divide the parties into two groups, denoting one from A_1 to A_n and another from A_{n+1} to A_{2n}.
- The first group, having their own private key as a_1 to a_n, compute $g^{\prod_{i=1}^n a_i}$ and send it to the other group. The other group does the same thing.
- Each group replaces their original value of g with the value received.
- Each group divides again and does the same thing until there is only one party left. In that case, they perform a final exponentiation with their private key.

After that, each party should get a common key $g^{\prod_{i=1}^{2^m} a_i}$. Notice that each party only performs $\log_2 2^m + 1 = m + 1$ rounds of exponentiation. Indeed, if the number of parties, n, is not a power of 2, we can still do the same thing by rounding n to $2^{\lceil \log_2 n \rceil}$. Thus, the revised algorithm is $O(n \log n)$ as desired.

To make it clearer, consider the following table illustrating the 4-DH case. Each cell represents the current value of g a party possesses.

TABLE 1

Party (Key)	A_1 (a_1)	A_2 (a_2)	A_3 (a_3)	A_4 (a_4)
Before	g	g	g	g
1st Round	$g^{a_3 a_4}$	$g^{a_3 a_4}$	$g^{a_1 a_2}$	$g^{a_1 a_2}$
2nd Round	$g^{a_2 a_3 a_4}$	$g^{a_1 a_3 a_4}$	$g^{a_1 a_2 a_4}$	$g^{a_1 a_2 a_3}$
3nd Round	$g^{\prod_{i=1}^4 a_i}$	$g^{\prod_{i=1}^4 a_i}$	$g^{\prod_{i=1}^4 a_i}$	$g^{\prod_{i=1}^4 a_i}$

One can see that, while the usual algorithm requires 4 rounds, our algorithm requires only $\log_2 4 + 1 = 3$ rounds. As n gets larger, $n \log n \ll n^2$, so this is a huge improvement.

REFERENCES

Diffie, Whitefield and Martin E. Hellman (1976). “New Directions in Cryptography”. In: 22, pp. 644–654. DOI: 10.1109/TIT.1976.1055638.

Koblitz, Neal (1994). *A Course in Number Theory and Cryptography*. Springer.

Some of the ideas are taken from the intro lecture by Paolo Cascini. The figure is https://commons.wikimedia.org/wiki/File:Diffie-Hellman_Key_Exchange.svg which is in public domain.