# The Pohlig-Hellman Algorithm

Joshua Ottoson

April 14, 2023

## 1 Introduction

This paper considers an attack on the Discrete Logarithm Problem that was not discussed in class called the Pohlig-Hellman Algorithm. We begin with a restatement of the problem.

**Definition 1.1** (The Discrete Logarithm Problem (DLP))**.** Let $g$ be a primitive root for $\mathbb{F}_p$ and let $h$ be a nonzero element of $\mathbb{F}_p$. The DLP is the problem of finding an exponent $x$ such that

$$g^x \equiv h \pmod{p}.$$

The number $x$ is called the *discrete logarithm of $h$ to the base $g$* and is denoted by $\log_g(h)$. [HPS14]

I would like to write a few reminders for the reader before we begin. $\mathbb{F}_p$ is the notation used for a finite field with a prime number of elements, which can also be denoted by $\mathbb{Z}/p\mathbb{Z}$. A primitive root of $\mathbb{F}_p$ can generate every nonzero element of $\mathbb{F}_p$. In other words, each element of $\mathbb{F}_p$ can be written as some power of a primitive root, say $g$. So

$$1, g, g^2, \ldots, g^{p-2}$$

are all the elements of $\mathbb{F}_p$. Recall $g^{p-1} \equiv 1$ modulo $p$, as primitive roots have order $p-1$. It is important to note that the list of numbers above are most likely *not* in numerical order.

A naive method to solve this problem is to simply plug in values for $x$, starting with $x = 1, 2, 3, \ldots$ and so on, where the successive squaring algoritm discussed in class can be used to compute the value $g^x$ modulo some prime $p$. This brute-force approach works well enough for small values of $p$, but is not practical when $p$ is very large. We would like to find some "shortcut" method to find $x$ that would require far fewer steps for large values of $p$.

Such a shortcut was first discovered by Roland Silver, but first published in a paper by Martin Hellman and Stephen Pohlig in 1978 [HP78]. They describe a method to significantly speed up the computation of the discrete logarithm, now known as the Pohlig-Hellman algorithm. In this paper, I will provide several

examples of the method described in their paper and summarize their results. I will also prove the correctness of their algorithm more generally and implement the algorithm using the Python Programming Language. Lastly, I will discuss some of the limitations of their method.

## 2 Historical Notes

Before I continue further, I would like to briefly provide a few historical details.

Martin Hellman was one of the architects of the Diffie-Hellman Key Exchange, the first Public Key Cryptosystem and a system based on the DLP. He grew up in the Bronx in New York City as a second-generation immigrant. He credits his father, a high school Physics teacher and HAM radio operator, as one of the factors that stoked his interest in Electrical Engineering.

Hellman did his graduate studies at Stanford University and received his Masters and PhD in Electrical Engineering. He decided not to go into teaching because, as he know from his father, "[Teachers] don't make a lot of money." He worked at IBM's Research Department in 1968-1969. Here, he would regularly have lunch with Horst Feistel - the main creator of the Data Encryption Standard (DES). It was during these informal lunches that Hellman first became interested in Cryptography and the rest, as they say, is history.

After Hellman realized that "[...] you didn't have to be poor to be a University Professor," he joined the faculty of Stanford, where he met Stephen Pohlig. Pohlig was a student of Hellman, and it was there at Stanford where the pair did a lot of exciting mathematics, including the development of the Pohlig-Hellman algorithm for calculating discrete logarithms. For more information, see [Hel04].

## 3 The Pohlig-Hellman Algorithm

We begin with a definition from the textbook.

**Definition 3.1.** The *order of a modulo* $p$ is the smallest exponent $k \geq 1$ such that $a^k \equiv 1 \pmod{p}$. [HPS14]

In solving the discrete logarithm problem $g^x \equiv h \pmod{p}$, we know that since $g$ is a primitive root, it has order $p-1$ modulo $p$. But recall that as a result of Fermat's Little Theorem, the exponents are related to each other modulo $p-1$. In other words, the solution to the discrete logarithm problem exists modulo $p-1$.

Now $p$ obviously cannot be factored (as it is prime), but $p-1$ is not prime, and so can be written as a product of prime numbers $p-1 = p_1^{e_1} \cdot p_2^{e^2} \cdots p_n^{e_n}$. Then $x$ is the solution modulo $p-1 = p_1^{e_1} \cdot p_2^{e^2} \cdots p_n^{e_n}$. This can split into individual congruences

$$x \equiv a_1 \mod p_1^{e^1}, \quad x \equiv a_2 \mod p_2^{e^2}, \quad \ldots, \quad x \equiv a_n \mod p_n^{e^n},$$

where each $a_i$ will be explained later. Since the modulii are all relatively prime, the Chinese Remainder Theorem guarantees a unique solution for the system of congruences modulo $p_1^{e_1} \cdot p_2^{e^2} \cdots p_n^{e_n} = p - 1$. So we can piece back together these congruences to find the the solution to the discrete logarithm problem $g^x \equiv h \pmod{p}$! Before we state the general result, we look at an example to see the specifics of how the algorithm works.

**Example 3.1.** Solve $3^x \equiv 22 \pmod{31}$.

Since 3 is a primitive root modulo 31, we know that such an $x$ will exist. We compute the prime factorization of $p - 1 = 30 = 2 \cdot 3 \cdot 5$.

We then choose one of the prime factors, say 5, and compute

$$\begin{aligned}
(22)^{\frac{p-1}{5}} &\equiv (22)^{\frac{2\cdot 3\cdot 5}{5}} \mod p \\
&\equiv (3^x)^{\frac{2\cdot 3\cdot 5}{5}} \mod p \quad (3^x \equiv 22 \mod p) \\
&\equiv (3^{5q_1+r_1})^{\frac{2\cdot 3\cdot 5}{5}} \mod p \quad \text{(Division Algorithm)} \\
&\equiv (3^{5q_1})^{\frac{2\cdot 3\cdot 5}{5}} \cdot (3^{r_1})^{\frac{2\cdot 3\cdot 5}{5}} \mod p \\
&\equiv (3^{2\cdot 3\cdot 5})^{q_1} \cdot (3^{r_1})^{2\cdot 3} \mod p \\
&\equiv (3^{p-1})^{q_1} \cdot (3^{r_1\cdot 2\cdot 3}) \mod p \\
&\equiv 3^{r_1\cdot 2\cdot 3} \mod p \quad \text{(Fermat's Little Theorem)}
\end{aligned}$$

So $(22)^{\frac{2\cdot 3\cdot 5}{5}} \equiv (22)^{2\cdot 3} \equiv 3^{r_1\cdot 2\cdot 3} \mod p$. By the division algorithm, $0 \leq r_1 < 5$, so we must plug in $r_1 = 0, 1, \ldots, 4$ until we find the correct value. It turns out that $r_1 = 2$ satifies the congruence $(22)^{2\cdot 3} \equiv 3^{r_1\cdot 2\cdot 3} \mod p$. Then since $x = 5q_1 + r_1$, we have that $x \equiv r_1 \mod 5$. Thus $r_1 = 2 \implies x \equiv 2 \mod 5$.

We repeat with another prime factor, say 3. Some details will be omitted this time.

$$\begin{aligned}
(22)^{\frac{p-1}{3}} &\equiv (3^x)^{\frac{2\cdot 3\cdot 5}{3}} \mod p \\
&\equiv (3^{3q_2+r_2})^{\frac{2\cdot 3\cdot 5}{3}} \mod p \quad \text{(Division Algorithm)} \\
&\equiv (3^{3q_2})^{\frac{2\cdot 3\cdot 5}{3}} \cdot (3^{r_1})^{\frac{2\cdot 3\cdot 5}{3}} \mod p \\
&\equiv (3^{p-1})^{q_2} \cdot (3^{r_2\cdot 2\cdot 5}) \mod p \\
&\equiv 3^{r_2\cdot 2\cdot 5} \mod p \quad \text{(Fermat's Little Theorem)}
\end{aligned}$$

So $(22)^{\frac{2\cdot 3\cdot 5}{3}} \equiv (22)^{2\cdot 5} \equiv 3^{r_2\cdot 2\cdot 5} \mod p$. By trial and error, $r_2 = 2$ so we have in this step $x \equiv 2 \mod 3$.

We repeat with the last prime factor 2.

$$\begin{aligned}
(22)^{\frac{p-1}{2}} &\equiv (3^x)^{\frac{2\cdot 3\cdot 5}{2}} \mod p \\
&\equiv (3^{2q_3+r_3})^{\frac{2\cdot 3\cdot 5}{2}} \mod p \quad \text{(Division Algorithm)} \\
&\equiv (3^{2q_3})^{\frac{2\cdot 3\cdot 5}{2}} \cdot (3^{r_3})^{\frac{2\cdot 3\cdot 5}{2}} \mod p \\
&\equiv (3^{p-1})^{q_2} \cdot (3^{r_3\cdot 3\cdot 5}) \mod p \\
&\equiv 3^{r_3\cdot 3\cdot 5} \mod p \quad \text{(Fermat's Little Theorem)}
\end{aligned}$$

So $(22)^{\frac{2\cdot 3\cdot 5}{2}} \equiv (22)^{3\cdot 5} \equiv 3^{r_3\cdot 3\cdot 5} \mod p$. By trial and error, $r_3 = 1$ so we have in this step $x \equiv 1 \mod 2$.

We use the Chinese Remainder Theorem to solve the three congruences

$$x \equiv 2 \mod 5, \quad x \equiv 2 \mod 3, \quad x \equiv 1 \mod 2,$$

which produce the unique solution $x \equiv 17 \mod 30$. Indeed, this is the solution to $3^x \equiv 22 \mod 31$.

In the previous example, each prime factor had multiplicity 1. But what happens when a prime factor is repeated? We will demonstrate this with one more example.

**Example 3.2.** Solve $6^x \equiv 30 \pmod{41}$.

6 is a primitive root modulo 41, so such an $x$ will exist. Note that $p - 1 = 40 = 2^3 \cdot 5$. If we select prime factor 5, then it can be shown that $x \equiv 3 \mod 5$. The details are left as an exercise.

Next, we select prime factor 2. Since it is repeated three times, then we compute

$$\begin{aligned}
(30)^{\frac{p-1}{2^3}} &\equiv (6^x)^{\frac{2^3\cdot 5}{8}} \mod p \\
&\equiv (6^{8q_1+r_1})^{\frac{2^3\cdot 5}{8}} \mod p \quad \text{(Division Algorithm)} \\
&\equiv (6^{8q_1})^{\frac{2^3\cdot 5}{8}} \cdot (6^{r_1})^{\frac{2^3\cdot 5}{8}} \mod p \\
&\equiv (6^{2^3\cdot 5})^{q_2} \cdot (6^{r_1\cdot 5}) \mod p \\
&\equiv (6^{r_1\cdot 5}) \mod p \quad \text{(Fermat's Little Theorem)}
\end{aligned}$$

So $(30^{\frac{2^3\cdot 5}{2^3}}) \equiv (30)^5 \equiv 6^{r_1\cdot 5} \mod p$. By the Division Algorithm, $0 \leq r_1 < 8$. By trial and error, $r_2 = 7$, so $x_2 \equiv 7 \mod 8$.

Then we are left with $x \equiv 7 \mod 8$ and $x \equiv 3 \mod 5$, so by the Chinese Remainder Theorem we have that $x \equiv 23 \mod 40$ is the unique solution to the pair of congruences. Note that $3^{23} \equiv 22 \mod 41$, so this solves the problem.

*Remark* 3.2. In the examples above, trial and error was used to compute the solution for each congruence. We had to plug in values for each remainder $r_i = 0, \ldots, p_i - 1$. This was easy enough to do in the examples, but this suggests that if the prime factors of $p-1$ are very large, then this algorithm will not be very efficient.

**Theorem 3.3** (Pohlig-Hellman Algorithm)**.** *Let $G$ be a group of prime order $p$, and suppose that we have an algorithm to solve the discrete logarithm problem in $G$ for an element whose order is a power of a prime. To be concrete, if $g \in G$ has order $q^e$, suppose that we can solve $g^x = h$ in $\mathcal{O}(S_{q^e})$ steps.*

*Now let $g \in G$ be a primitive root, and suppose that $p-1$ factors into a product of prime powers as*

$$p - 1 = p_1^{e_1} \cdot p_2^{e_2} \cdots p_t^{e_t}.$$

*Then the discrete logarithm problem $g^x = h$ can be solved in*

$$\mathcal{O}(\sum_{i=1}^{t} S_{p_i^{e_i}} + \log(p-1)) \quad \textit{steps}$$

*using the following procedure:*

*(1) For each $1 \leq i \leq t$, let*

$$g_i = g^{\frac{p-1}{p_i^{e_i}}} \qquad \textit{and} \qquad h_i = h^{\frac{p-1}{p_i^{e_i}}}.$$

*Notice that $g_i$ has prime power order $p_i^{e^i}$, so use the given algorithm to solve the discrete logarithm problem*

$$g_i^r = h_i.$$

*Let $r = r_i$ be a solution to the above equation.*

*(2) Use the Chinese Remainder Theorem to solve*

$$x \equiv r_1 \mod p_1^{e_1}, \quad x \equiv r_2 \mod p_2^{e_2}, \quad \ldots, \quad x \equiv r_t \mod p_t^{e_t}.$$

*Proof* First, we show that the algorithm described in steps (1) and (2) give a solution to $g^x \equiv h \mod p$. Suppose that $x$ is the solution to the system of congruences in step (2) above. By the Chinese Remainder Theorem, this solution is unique modulo $p_1^{e_1} p_2^{2_3} \cdots p_t^{e_t}$. We would like to show that this $x$ is the same solution to the discrete logarithm problem. We note that by definition, $p_i^{e_i} | (x - r_i) \implies x - r_i = p_i^{e_i} q_i \implies x = p_i^{e_i} q_i + r_i$ for some $q_i \in \mathbb{Z}$ with $0 \leq r_i < p_i^{e_i}$.

We can then write

$$
\begin{aligned}
h_i &\equiv (h)^{\frac{p-1}{p_i^{e_i}}} \\
&\equiv (h)^{\frac{p_1^{e_1}p_2^{e_3}\cdots p_t^{e_t}}{p_i^{e_i}}} \\
&\equiv (g^x)^{\frac{p_1^{e_1}p_2^{e_3}\cdots p_t^{e_t}}{p_i^{e_i}}} \\
&\equiv (g^{p_i^{e_i}q_i+r_i})^{\frac{p_1^{e_1}p_2^{e_3}\cdots p_t^{e_t}}{p_i^{e_i}}} \\
&\equiv (g^{p_i^{e_i}q_i})^{\frac{p_1^{e_1}p_2^{e_3}\cdots p_t^{e_t}}{p_i^{e_i}}} \cdot (g^{r_i})^{\frac{p_1^{e_1}p_2^{e_3}\cdots p_t^{e_t}}{p_i^{e_i}}} \\
&\equiv (g^{\frac{p-1}{p_i^{e_i}}}) \cdot (g^{r_i \cdot \frac{p_1^{e_1}p_2^{e_3}\cdots p_t^{e_t}}{p_i^{e_i}}}) \\
&\equiv (g^{p-1})^{p_i^{e_i}} \cdot (g^{r_i \cdot \frac{p-1}{p_i^{e_i}}}) \\
&\equiv (g^{r_i \cdot \frac{p-1}{p_i^{e_i}}})
\end{aligned}
$$

Then we have that $h_i = (h)^{\frac{p-1}{p_i^{e_i}}} = (g^{r_i \cdot \frac{p-1}{p_i^{e_i}}})$. By the one-to-one property of logarithms, this implies

$$\log_g(h^{\frac{p-1}{p_i^{e_i}}}) = \log_g(g^{r_i \cdot \frac{p-1}{p_i^{e_i}}}),$$

which we may rewrite with properties of logarithms as $\frac{p-1}{p_i^{e_i}} \cdot \log_g(h) = r_i \cdot \frac{p-1}{p_i^{e_i}}$. Here we note that since $x = p_i^{e_i}q_i + r_i$, then $x \equiv r_i \mod p_i^{e_i}$. Thus $\frac{p-1}{p_i^{e_i}} \cdot \log_g(h) \equiv x_i \cdot \frac{p-1}{p_i^{e_i}} \mod p_i^{e_i}$. Since $\gcd(\frac{p-1}{p_i^{e_i}}, p_i^{e_i}) = 1$, we may use cancellation to rewrite this as

$$x_i \equiv \log_g(h) \mod p_i^{e_i}$$

# References


[Hel04] Martin Hellman, *Oral history interview with Martin Hellman*, Charles Babbage Institute (2004), available at https://hdl.handle.net/11299/107353.

[HP78] Martin Hellman and Stephen Pohlig, *An improved algorithm for computing logarithms over GF(p) and its cryptographic significance*, IEEE Transactions on Information Theory **24** (1978), no. 1, 106-110.

[HPS14] Jeffrey Hoffstein, Jill Pipher, and Joseph Silverman, *An Introduction to Mathematical Cryptograpy*, 2nd ed., Springer, New York, 2014.