Article

Discrete Logarithms

Lukasz Matysiak *

Posted Date: 16 September 2024

doi: 10.20944/preprints202409.1141.v1

Keywords: algorithm; congruence; discrete logarithm; protocol

 doi:10.20944/preprints202409.1141.v

Article

Discrete Logarithms

Łukasz Matysiak

Military University of Technology, Warsaw, Poland; luk90mat@outlook.com

Abstract: In this paper we present two algorithms for computing discrete logarithms. We present some properties and show how we break the Diffie-Hellman protocol, which is considered to be very secure.

Keywords: algorithm; congruence; discrete logarithm; protocol

2020 Mathematics Subject Classification: Primary 11T71; Secondary 11A07

---

1. Introduction

In mathematics, for given real numbers a and b, the logarithm $\log_a b$ is a number x such that $a^x = b$. Analogously, in any group G, powers a^k can be defined for all integers k, and the discrete logarithm $\log_a b$ is an integer k such that $a^k = b$. In number theory, the more commonly used term is index: we can write $x = ind_r b \pmod m$ (read "the index of b to the base r modulo m") for $r^x \equiv b \pmod m$ if r is a primitive root of m and $\gcd(b, m) = 1$.

Discrete logarithms are quickly computable in a few special cases. However, no efficient method is known for computing them in general. In cryptography, the computational complexity of the discrete logarithm problem, along with its application, was first proposed in the Diffie–Hellman problem. Several important algorithms in public-key cryptography, such as ElGamal, base their security on the hardness assumption that the discrete logarithm problem (DLP) over carefully chosen groups has no efficient solution ([4]).

In the Section 2 we present two algorithms for computing discrete logarithms. The first is for logarithms $\log_a b \pmod n$, where $\gcd(a, n) = 1$. The second is for $\log_a b \pmod n$, where $\gcd(a, n) \neq 1$. Of course, in both algorithms we also provide the case when $\log_a b \pmod n$ has no solution (see Remark 2.2). In Theorem 2.6 we present a general criterion for when the logarithm $\log_a b \pmod n$ has a solution. And in Proposition 2.7 we present a criterion for when such a logarithm has no solution. In the Section 3 we present three key examples representing each of the cases considered.

In the Section 4 we present a very crucial application of discrete logarithms, i.e. how we are able to break the Diffie-Hellman protocol using the algorithms developed in this paper.

Diffie–Hellman (DH) key exchange is a mathematical method of securely exchanging cryptographic keys over a public channel and was one of the first public-key protocols as conceived by Ralph Merkle and named after Whitfield Diffie and Martin Hellman. From [5], [2] we know that DH is one of the earliest practical examples of public key exchange implemented within the field of cryptography. Published in 1976 by Diffie and Hellman, this is the earliest publicly known work that proposed the idea of a private key and a corresponding public key.

Traditionally, secure encrypted communication between two parties required that they first exchange keys by some secure physical means, such as paper key lists transported by a trusted courier. The Diffie–Hellman key exchange method allows two parties that have no prior knowledge of each other to jointly establish a shared secret key over an insecure channel. This key can then be used to encrypt subsequent communications using a symmetric-key cipher.

Diffie–Hellman is used to secure a variety of Internet services. However, research published in October 2015 ([1]) suggests that the parameters in use for many DH Internet applications at that time are not strong enough to prevent compromise by very well-funded attackers, such as the security services of some countries.

The scheme was published by Whitfield Diffie and Martin Hellman in 1976 in [2] but in 1997 it was revealed that James H. Ellis [3], Clifford Cocks, and Malcolm J. Williamson of GCHQ ([7]), the

 doi:10.20944/preprints202409.1141.v

British signals intelligence agency, had previously shown in 1969 [8] how public-key cryptography could be achieved ([7]).

Although Diffie–Hellman key exchange itself is a non-authenticated key-agreement protocol, it provides the basis for a variety of authenticated protocols, and is used to provide forward secrecy in Transport Layer Security's ephemeral modes (referred to as EDH or DHE depending on the cipher suite).

The method was followed shortly afterwards by RSA, an implementation of public-key cryptography using asymmetric algorithms.

Expired US patent 4,200,770 ([9]) from 1977 describes the now public-domain algorithm. It credits Hellman, Diffie, and Merkle as inventors.

The simplest and the original implementation in [2], later formalized as Finite Field Diffie-Hellman in RFC 7919,([6]) of the protocol uses the multiplicative group of integers modulo p, where p is prime, and g is a primitive root modulo p. These two values are chosen in this way to ensure that the resulting shared secret can take on any value from 1 to p˘1. Here is an example of the protocol, with non-secret values, and secret values.

2. Discrete Logarithm Calculation Algorithms

In this Section we describe two algorithms for computing discrete logarithms, namely computing $\log_a b \pmod n$ in the case where $\gcd(a,n) = 1$ and in the case where $\gcd(a,n) \neq 1$.

By definition, the problem comes down to calculating x in the congruence $a^x \equiv b \pmod n$.

Algorithm for computing discrete logarithms $\log_a b \pmod n$**, where** $\mathrm{GCD}(a,n) = 1$**:**

(1) We compute the congruence $ay_1 \equiv b \pmod n$ and find y_1.
(2) If $y_1 \equiv a^{r_1} \pmod n$ for some $r_1 \in \mathbb{N}_0$, then $x = r_1 + 1$ is a particular solution and we go to point (6). If $y_1 \not\equiv a^r \pmod n$, then we go to the next step.
(3) Substituting $ay_1 \equiv b$ into the initial congruence and after appropriate reduction, we obtain $a^{x-1} \equiv y_1 \pmod n$.
(4) We calculate the congruence $ay_2 \equiv y_1 \pmod n$ and find y_2.
(5) If $y_2 \equiv a^{r_2} \pmod n$ for some $r_2 \in \mathbb{N}_0$, then $x = r_2 + 2$ is a particular solution and we go to step (6). If $y_2 \not\equiv a^{r_2} \pmod n$, we return to step (1).
(6) If x is a particular solution, then the general solution is $X = x + \varphi(n)k$, where $\varphi(n)$ is an Euler function and the number $k \in \mathbb{Z}$ satisfies $-\frac{x}{\varphi(n)} < k < \frac{n-x}{\varphi(n)}$.

Corollary 2.1. *If the algorithm terminates after t steps, then the particular solution is $x = r_t + t$.*

Remark 2.2. It is possible that the algorithm will not terminate. Specifically, for some s the sequences $(y_1, y_2, \ldots, y_s) = (y_{s+1}, y_{s+2}, \ldots, y_{2s})$ will be equal. Then this will mean that $\log_a b \pmod n$ has no solution. Unfortunately, we cannot define the conditions when $\log_a b \pmod n$ has no solution.

Corollary 2.3. *The algorithm implies that if x is a particular solution, then $y_1 = a^{x-1}$, $y_2 = a^{x-2}$, $\ldots$, $y_t = a$, if the algorithm terminates in the t-step.*

Proof of the algorithm correctness: Assume that a and n are relatively prime. We are looking for x such that $a^x \equiv b \pmod n$.

Assume that x is a solution of the equation $a^x \equiv b \pmod n$. In step 1, we compute y_1 from the congruence $ay_1 \equiv b \pmod n$. If $y_1 \equiv a^{r_1} \pmod n$, then $x = r_1 + 1$ is a particular solution. If $y_1 \not\equiv a^{r_1} \pmod n$, then we substitute $ay_1 \equiv b$ into the initial congruence, which gives $a^{x-1} \equiv y_1 \pmod n$. In step 3, we compute y_2 from the congruence $ay_2 \equiv y_1 \pmod n$. If $y_2 \equiv a^{r_2} \pmod n$, then $x = r_2 + 2$ is a particular solution. If $y_2 \not\equiv a^{r_2} \pmod n$, we return to step 1 with a new value $y_1 = y_2$.

If the algorithm terminates after t steps, then the particular solution is $x = r_t + t$. Indeed, we have the substitutions $b \equiv ay_1$, $y_1 \equiv ay_2$, $y_2 = ay_3$, $\ldots$, $y_{t-2} = ay_{t-1}$, $y_{t-1} = ay_t = a^{r_t} \pmod n$. Then:

 doi:10.20944/preprints202409.1141.v

$b \equiv a^x \equiv ay_1 \equiv a^2y_2 \equiv a^3y_3 \equiv \cdots \equiv a^{t-1}y_{t-1} \equiv a^ty_t \equiv a^ta^{r_t} \equiv a^{t+r_t}$. So the particular solution is $x = r_t + t$.

Since $\mathrm{GCD}(a, n) = 1$, we know from Euler's Theorem that $a^{\varphi(n)} \equiv 1 \pmod{n}$. From the previous paragraph, we also know that $b \equiv a^{t+r_t} \pmod{n}$. Therefore, $a^X \equiv b \equiv b \cdot 1 \equiv a^{t+r_t}\left(a^{\varphi(n)}\right)^k \pmod{n}$. So, $X = t + r_t + k\varphi(n)$ is a general solution. Such a general solution should satisfy the inequalities $0 < X < n$. Therefore, from the inequalities $0 < t + r_t + k\varphi(n) < n$ we get the inequalities $-\dfrac{t + r_t}{\varphi(n)} < k < \dfrac{n - (t + r_t)}{\varphi(n)}$, that is, $-\dfrac{x}{\varphi(n)} < k < \dfrac{n - x}{\varphi(n)}$.

The algorithm may not terminate if for some s the sequences $(y_1, y_2, \ldots, y_s)$, $(y_{s+1}, y_{s+2}, \ldots, y_{2s})$ are equal. Indeed, let's assume that $y_1 = y_{s+1}$ and at that point we terminate the algorithm. Then $b \equiv a^2y_2 \equiv \cdots \equiv a^sy_s \equiv a^{s+1}y_{s+1} \equiv a^{s+1}y_1 \equiv a^{s+1}ay_2 \equiv a^{s+2}y_2 \pmod{n}$. Then the algorithm never terminates and every s steps the coefficients y_i are repeated. Therefore $\log_a b \pmod{n}$ has no solution. □

We now present an algorithm for computing discrete logarithms when $\gcd(a, n) \neq 1$.

Algorithm for computing discrete logarithms $\log_a b \pmod{n}$, where $\mathrm{GCD}(a, n) \neq 1$:

(1) We factor n: $n = p_1^{\alpha_1} \ldots p_r^{\alpha_r}$.
(2) For each $p_i^{\alpha_i}$ such that $p_i^{\alpha_i} \nmid a$, $p_i^{\alpha_i} \nmid b$ we proceed to calculate $a^x \equiv b \pmod{p_i^{\alpha_i}}$.
(3) Since a and $p_i^{\alpha_i}$ are coprime, we use the previous algorithm to compute x.
(4) If we compute different x for different i, we use the Chinese Remainder Theorem to obtain the unique result x.
(5) If x is a particular solution, then the general solution is $X = x + \varphi(n)k$, where $\varphi(n)$ is an Euler function and the number $k \in \mathbb{Z}$ satisfies $-\dfrac{x}{\varphi(n)} < k < \dfrac{n - x}{\varphi(n)}$.

Remark 2.4. In the above algorithm, in step (3) we can obtain that the given congruence has no solution. Then $\log_a b \pmod{n}$ has no solution.

Before we proceed to prove the correctness of the algorithm, we need the following lemma.

Lemma 2.5. *If $a \equiv b \pmod{n}$, where $\mathrm{GCD}(a, n) \neq 1$, then $a \equiv b \pmod{d}$, where $d \mid n$ and $\mathrm{GCD}(a, d) = 1$.*

Proof. Suppose that $a \equiv b \pmod{n}$, where $\gcd(a, n) \neq 1$. We want to show that $a \equiv b \pmod{d}$, where d is a divisor of n and $\gcd(a, d) = 1$.

By assumption, $a \equiv b \pmod{n}$ means that $n \mid (a - b)$. In other words, there is an integer k such that $a - b = kn$. Let d be a factor of n. This means that $n = md$ for some integer m. Substituting $n = md$ into the equation $a - b = kn$, we get:

$$a - b = k(md) = (km)d$$

which means that $d \mid (a - b)$. So $a \equiv b \pmod{d}$.

Now we need to show that $\gcd(a, d) = 1$. Since $\gcd(a, n) \neq 1$, there is a common divisor $g > 1$ such that $g \mid a$ and $g \mid n$. Since d is a divisor of n, we can write $d = g \cdot d'$ for some integer d'. If $\gcd(a, d) \neq 1$, then $g \mid d$, which contradicts the assumption that $\gcd(a, d) = 1$. Therefore, $\gcd(a, d) = 1$. □

Now we give a general criterion when the congruence $a^x \equiv b \pmod{n}$ has a solution if $\gcd(a, n) \neq 1$.

Theorem 2.6. *The following conditions are equivalent:*

(1) *If $\gcd(a, b, n) \neq 1$, then*
(2) *$a^x \equiv b \pmod{n}$ has a solution where $\gcd(a, n) \neq 1$.*

 doi:10.20944/preprints202409.1141.v

Proof. ($\Rightarrow$) Assume that $\gcd(a,b,n) = d \neq 1$. This means that d divides both a, b, and n. Since d divides a, b, and n, we can write: $a = d \cdot a_1$, $b = d \cdot b_1$, $n = d \cdot n_1$. Then the congruence $a^x \equiv b \pmod{n}$ can be transformed by substituting the above expressions:

$$(d \cdot a_1)^x \equiv d \cdot b_1 \pmod{d \cdot n_1}.$$

We can divide both sides by d, since d is a common divisor:

$$d^x \cdot a_1^x \equiv d \cdot b_1 \pmod{d \cdot n_1}$$

$$d^{x-1} \cdot a_1^x \equiv b_1 \pmod{n_1}$$

Now we can see that d^{x-1} is a multiple of d, and since d divides n, we can simplify the congruence to:

$$a_1^x \equiv b_1 \pmod{n_1}.$$

In this way, due to the divisibility of d by n, we can simplify the congruence to the form where a_1 and n_1 are relatively prime (their gcd is 1). This allows us to use theorems from number theory that say that such an equation has a solution.
($\Leftarrow$) Now suppose that the congruence $a^x \equiv b \pmod{n}$ has a solution where $\gcd(a,n) \neq 1$. We will show that $\gcd(a,b,n) \neq 1$.

By assumption, we have that d divides both a and n. Since $a^x \equiv b \pmod{n}$, there is an integer k such that:

$$a^x = b + kn.$$

Since d divides a, we can write $a = d \cdot a_1$, where a_1 is an integer. Substituting this into the equation, we have:

$$(d \cdot a_1)^x = b + kn$$

$$d^x \cdot a_1^x = b + kn$$

Since d divides $d^x \cdot a_1^x$, then d must divide $b + kn$. That is, d divides b, since d divides kn (since d divides n). Since d divides a, b, and n, then $\gcd(a,b,n) \geq d \neq 1$. □

Proposition 2.7. *The following conditions are equivalent:*

(1) $\gcd(a,b) = 1$ *or* $\gcd(b,n) = 1$,
(2) $a^x \equiv b \pmod{n}$, *where* $\gcd(a,n) > 1$, *has no solution.*

Proof. ($\Rightarrow$) Let $d \mid a$ and $d \mid n$. Then $a = dk$, $n = dm$, where $k, m \in \mathbb{Z}$. So we have

$$d^x \cdot k^x \equiv b \pmod{d \cdot m}.$$

Since $d \nmid b$ (since $\gcd(a,b) = 1$, the left side of the congruence $d^x \cdot k^x$ will always be divisible by d, while the right side of b will not be divisible by d. This means that the left side of the equation will have a factor of d, and the right side will not, which leads to a contradiction. Therefore, $a^x \equiv b \pmod{n}$ cannot have a solution when $GCD(a,n) \neq 1$.

The case when $\gcd(b,n) = 1$ is proved analogously.
($\Leftarrow$) Now suppose that $\gcd(a,b) \neq 1$ and $\gcd(b,n) \neq 1$. This means that there are numbers $d_1 > 1$ and $d_2 > 1$, such that d_1 divides a and b, and d_2 divides b and n. If d_1 divides a and b, then we can write $a = d_1 \cdot k_1$ and $b = d_1 \cdot k_2$ for some integers k_1 and k_2. Similarly, if d_2 divides b and n, then we can write $b = d_2 \cdot m_1$ and $n = d_2 \cdot m_2$ for some integers m_1 and m_2. Since d_1 divides a and b, and d_2 divides b and n, the left side of the congruence $a^x \equiv b \pmod{n}$ will have a factor of d_1, and the right side a factor of d_2. If d_1 and d_2 are distinct, then b must be divisible by both d_1 and d_2. However, if $\gcd(a,n) > 1$, then a and n have a common factor that does not divide b, leading to a contradiction. □

 doi:10.20944/preprints202409.1141.v

Proof of the algorithm correctness: By Theorem 2.6 we know that the congruence $a^x \equiv b \pmod{n}$, where $\gcd(a,n) \neq 1$ has a solution if and only if $\gcd(a,b,n) \neq 1$.

If $n = p_1^{\alpha_1} \dots p_r^{\alpha_r}$, then the congruence $a^x \equiv b \pmod{n}$ is equivalent to the system of congruences:

$$\begin{cases} a^x \equiv b \pmod{p_1^{\alpha_1}}, \\ a^x \equiv b \pmod{p_2^{\alpha_2}}, \\ \quad \dots\dots\dots \\ a^x \equiv b \pmod{p_r^{\alpha_r}}. \end{cases}$$

For each $p_i^{\alpha_i}$ such that $p_i^{\alpha_i} \nmid a$, $p_i^{\alpha_i} \nmid b$ we proceed to calculate $a^x \equiv b \pmod{p_i^{\alpha_i}}$.
Using Lemma 2.5 we have $\gcd(a, p_i^{\alpha_i}) = 1$ and we can then use the first algorithm that was proved.
By the Chinese Remainder Theorem we can combine the results from such congruences. □

3. Some Examples

In this Section we will show three examples illustrating our three cases. In the first example we have the case of using the first algorithm and obtaining the result. In the second case similarly but for the logarithm that has no solution. In the third example we have the use of the second algorithm.

Example 3.1. We will calculate $\log_3 4 \pmod 7$. So we need to calculate the congruence $3^x \equiv 4 \pmod 7$. We will use our algorithm.

(1) We calculate y_1 from the congruence $3y_1 \equiv 4 \pmod 7$. It is easy to calculate that $y_1 \equiv 6 \pmod 7$.
(2) Since $6 \not\equiv 3^t \pmod 7$, we substitute $4 \equiv 3 \cdot 6 \pmod 7$ into $3^x \equiv 4 \pmod 7$. After reduction we get $3^{x-1} \equiv 6 \pmod 7$.
(3) We calculate y_2 from the congruence $3y_2 \equiv 6 \pmod 7$. It is easy to calculate that $y_2 \equiv 2 \pmod 7$.
(4) Since $2 \not\equiv 3^t \pmod 7$, we substitute $6 \equiv 3 \cdot 2 \pmod 7$ into $3^{x-1} \equiv 6 \pmod 7$. After reduction we get $3^{x-2} \equiv 2 \pmod 7$.
(5) We calculate y_3 from the congruence $3y_3 \equiv 2 \pmod 7$. It is easy to calculate that $y_3 \equiv 3 \pmod 7$.
(6) Since $3 \equiv 3^1 \pmod 7$, we substitute $2 \equiv 3 \cdot 3 \pmod 7$ into $3^{x-2} \equiv 2 \pmod 7$. We get $3^{x-2} \equiv 3^2 \pmod 7$.
(7) From the congruence property we get $x - 2 = 2$, so $x = 4$, which is our particular solution.
(8) Since $\varphi(7) = 6$, the general solution is $X = 4 + 6k$ for all $k \in \mathbb{Z}$.

Example 3.2. We will calculate $\log_4 7 \pmod{11}$. So we need to calculate the congruence $4^x \equiv 7 \pmod{11}$. We will use our algorithm.

(1) We calculate y_1 from the congruence $4y_1 \equiv 7 \pmod{11}$. It is easy to calculate that $y_1 \equiv 10 \pmod{11}$.
(2) Since $10 \not\equiv 4^t \pmod{11}$, we substitute $7 \equiv 4 \cdot 10 \pmod{11}$ into $4^x \equiv 7 \pmod{11}$. After reduction we obtain $4^{x-1} \equiv 10 \pmod{11}$.
(3) We calculate y_2 from the congruence $4y_2 \equiv 10 \pmod{11}$. It is easy to calculate that $y_2 \equiv 8 \pmod{11}$.
(4) Since $8 \not\equiv 4^t \pmod{11}$, we substitute $10 \equiv 4 \cdot 8 \pmod{11}$ into $4^{x-1} \equiv 10 \pmod{11}$. After reduction we get $4^{x-2} \equiv 8 \pmod{11}$.
(5) We calculate y_3 from the congruence $4y_3 \equiv 8 \pmod{11}$. It is easy to calculate that $y_3 \equiv 2 \pmod{11}$.
(6) Since $2 \not\equiv 4^t \pmod{11}$, we substitute $8 \equiv 4 \cdot 2 \pmod{11}$ into $4^{x-2} \equiv 8 \pmod{11}$. After reduction we get $4^{x-3} \equiv 6 \pmod{11}$.
(7) We calculate y_4 from the congruence $4y_4 \equiv 6 \pmod{11}$. It is easy to calculate that $y_4 \equiv 6 \pmod{11}$.

 doi:10.20944/preprints202409.1141.v

(8) Since $6 \not\equiv 4^t \pmod{11}$, we substitute $6 \equiv 4 \cdot 6 \pmod{11}$ into $4^{x-3} \equiv 6 \pmod{11}$. After reduction we get $4^{x-4} \equiv 6 \pmod{11}$.
(9) We calculate y_5 from the congruence $4y_5 \equiv 6 \pmod{11}$. It is easy to calculate that $y_5 \equiv 7 \pmod{11}$.
(10) Since $7 \not\equiv 4^t \pmod{11}$, we substitute $6 \equiv 4 \cdot 7 \pmod{11}$ into $4^{x-4} \equiv 6 \pmod{11}$. After reduction we get $4^{x-5} \equiv 7 \pmod{11}$.
(11) We calculate y_6 from the congruence $4y_6 \equiv 7 \pmod{11}$. It is easy to calculate that $y_6 \equiv 10 \pmod{11}$.
(12) Since $10 \not\equiv 4^t \pmod{11}$, we substitute $7 \equiv 4 \cdot 10 \pmod{11}$ into $4^{x-5} \equiv 7 \pmod{11}$. After reduction we get $4^{x-6} \equiv 10 \pmod{11}$.
(13) Notice that we repeat the value $y_6 = y_1$. Counting further we see that $y_7 = y_2$, $y_8 = y_3, \ldots$. Therefore $\log_4 7 \pmod{11}$ has no solution.

Example 3.3. We will calculate $\log_{12} 18 \pmod{30}$. So we need to calculate the congruence $12^x \equiv 18 \pmod{30}$. We will use our algorithm.

(1) We have $30 = 2 \cdot 3 \cdot 5$. So we can consider the system of congruences: $\begin{cases} 12^x \equiv 18 \pmod{2} \\ 12^x \equiv 18 \pmod{3} \\ 12^x \equiv 18 \pmod{5} \end{cases}$
(2) The first two congruences in the system are identical. So let's calculate only the last one, which is $12^x \equiv 18 \pmod{5}$. After reduction we have $2^x \equiv 3 \pmod{5}$.
(3) We calculate y_1 from the congruence $2y_1 \equiv 3 \pmod{5}$. It is easy to calculate that $y_1 \equiv 4$.
(4) Since $4 = 2^2 \pmod{5}$, then substituting $3 \equiv 2 \cdot 2^2$ into $2^x \equiv 3 \pmod{5}$ we have $2^x \equiv 2^3 \pmod{5}$.
(5) The particular solution is $x = 3$. The general solution is $X = 3 + 4k$, $k \in \mathbb{Z}$.

4. Breaking the Diffie-Hellman Protocol

In this Section, we will show one concrete example showing that our algorithm breaks the Diffie–Hellman protocol. Unfortunately, we are not able to show this in a formal proof due to the complexity of the algorithm.

Alice and Bob publicly agree to use a modulus $p = 23$ and base $g = 5$ (which is a primitive root modulo 23). Alice chooses a secret integer $a = 4$, then sends Bob $A \equiv g^a \pmod{p}$, $A \equiv 54 \pmod{23} \equiv 4$ (in this example both A and a have the same value 4, but this is usually not the case). Bob chooses a secret integer $b = 3$, then sends Alice $B \equiv g^b \pmod{p}$, $B \equiv 53 \pmod{23} \equiv 10$. Alice computes $s \equiv B^a \pmod{p}$, $s \equiv 104 \pmod{23} \equiv 18$. Bob computes $s \equiv A^b \pmod{p}$, $s \equiv 43 \pmod{23} \equiv 18$. Alice and Bob now share a secret (the number 18). Both Alice and Bob have arrived at the same values because under modulo p,

$$A^b \equiv g^{ab} \equiv g^{ba} \equiv B^a \pmod{p}.$$

More specifically,

$$(g^a)^b \equiv \left(g^b\right)^a \pmod{p}.$$

Only a and b are kept secret. All the other values – p, g, $g^a \pmod{p}$, and $g^b \pmod{p}$ – are sent in the clear. The strength of the scheme comes from the fact that $g^{ab} \pmod{p} \equiv g^{ba} \pmod{p}$ take extremely long times to compute by any known algorithm just from the knowledge of p, g, $g^a \pmod{p}$, and $g^b \pmod{p}$. Such a function that is easy to compute but hard to invert is called a one-way function. Once Alice and Bob compute the shared secret they can use it as an encryption key, known only to them, for sending messages across the same open communications channel.

Of course, much larger values of a, b, and p would be needed to make this example secure, since there are only 23 possible results of $n \pmod{23}$. However, if p is a prime of at least 600 digits, then even the fastest modern computers using the fastest known algorithm cannot find a given only g, p and $g^a \pmod{p}$. Such a problem is called the discrete logarithm problem ([1]).

 doi:10.20944/preprints202409.1141.v

To summarize, in our example, only Alice's key $a = 4$ and Bob's key $b = 3$ are private and we don't know about these numbers. We know about $g = 5$, $p = 23$, $A = 4$, $B = 10$.

Now we will try to break this protocol with this example.

(1) We want to calculate Alice's key a from the congruence $A \equiv g^a \pmod{p}$, which is $5^a \equiv 4 \pmod{23}$.
(2) According to the algorithm, we calculate the congruence $5y_1 \equiv 4 \pmod{23}$ and get $y_1 \equiv 10$, which is not a power of 5.
(3) We replace the number 4 with $5y_1 = 5 \cdot 10$ and insert it into $5^a \equiv 4 \pmod{23}$. After reduction, we get $5^{a-1} \equiv 10 \pmod{23}$.
(4) We calculate the congruence of $5y_2 \equiv 10 \pmod{23}$ and get $y_2 \equiv 2$, which is not a power of 5.
(5) We replace the number 10 with $5y_2 = 5 \cdot 2$ and insert it into $5^{a-1} \equiv 2 \pmod{23}$. After reduction we get $5^{a-2} \equiv 2 \pmod{23}$.
(6) We calculate the congruence of $5y_3 \equiv 2 \pmod{23}$ and get $y_3 = 5$, which is a power of 5.
(7) We replace the number 2 with the number $5y_3 = 5 \cdot 5 = 5^2$ and insert it into $5^{a-2} \equiv 2 \pmod{23}$. We get $5^{a-2} \equiv 5^2 \pmod{23}$.
(8) So $a - 2 = 2$, so $a = 4$ is the number we are looking for.

(1) We want to calculate Bob's key b from the congruence $B \equiv g^b \pmod{p}$, which is $5^a \equiv 10 \pmod{23}$.
(2) According to the algorithm, we calculate the congruence $5y_1 \equiv 10 \pmod{23}$ and get $y_1 \equiv 2$, which is not a power of 5.
(3) We replace the number 10 with $5y_1 = 5 \cdot 2$ and insert it into $5^{b-1} \equiv 2 \pmod{23}$. After reduction, we get $5^{b-1} \equiv 2 \pmod{23}$.
(4) We calculate the congruence $5y_2 \equiv 2 \pmod{23}$ and get $y_2 = 5$ which is a power of 5.
(5) We replace the number 2 with the number $5y_2 = 5 \cdot 5 = 5^2$ and insert it into $5^{b-1} \equiv 2 \pmod{23}$. We get $5^{b-1} \equiv 5^2 \pmod{23}$.
(6) So $b - 1 = 2$, so $b = 3$ is the number we are looking for.

Now, without Alice and Bob knowing, we can calculate their common secret number s:

$$s \equiv A^b \equiv B^a \equiv g^{ab} \equiv 5^{12} \equiv 18 \pmod{23}.$$

We broke the Diffie-Hellman protocol.

References

1. Adrian, David; *Imperfect Forward Secrecy: How Diffie–Hellman Fails in Practice*, Archived (PDF) from the original on 2015-09-06, (October 2015).
2. Diffie, Whitfield; Hellman, Martin E., *New Directions in Cryptography*, IEEE Transactions on Information Theory. 22 (6): 644–654, (November 1976).
3. Ellis, J. H., *The possibility of Non-Secret digital encryption*, CESG Research Report. Retrieved 2015-08-28. (January 1970).
4. Menezes, A. J.; van Oorschot, P. C.; Vanstone, S. A., *Chapter 8.4 ElGamal public-key encryption*, Handbook of Applied Cryptography. CRC Press.
5. Merkle, Ralph C., *Secure Communications Over Insecure Channels*. Communications of the ACM. 21 (4): 294–299, (April 1978), CiteSeerX 10.1.1.364.5157
6. Wong, David, *Key exchange standards*, Real World Cryptography. Manning. ISBN 9781617296710 – via Google Books, 2021.
7. *GCHQ trio recognised for key to secure shopping online*, BBC News. 5 October 2010. Retrieved 5 August 2014.

 doi:10.20944/preprints202409.1141.v

8. *The Possibility of Secure Secret Digital Encryption*, Retrieved 2017-07-08.
9. US patent 4200770