Cryptocomplexity II

Kryptokomplexität II

Sommersemester 2024

Chapter 2: Diffie–Hellman and the Discrete Logarithm Problem

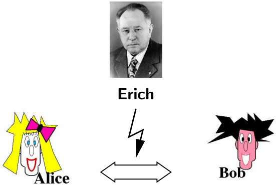
Dozent: Prof. Dr. J. Rothe

Merkle, Hellman, and Diffie 1977 at Stanford University

Diffie and Hellman Receive the 2015 Turing Award

Secret-Key Agreement Problem

- Key distribution for symmetric systems:



© The design of Alice and Bob is due to Crépeau.

  is an issue, and it is the more demanding, the more users are participating in the same system.

- **Secret-key agreement problem**:
  How can Alice and Bob agree on such a joint secret key, without meeting in private prior to exchanging encrypted messages and without using an expensive secure channel for key distribution?

Secret-Key Agreement Problem

- The secret-key agreement problem has been considered unsolvable since the beginnings of cryptography.
- Thus, it caused much surprise when Diffie and Hellman came up with an ingenious, simple idea to solve it.
- Using their secret-key agreement protocol, Alice and Bob can agree on a joint secret key by exchanging some messages.
- Eavesdropper Erich, however, does not have a clue about their key, even though he knows every single bit exchanged, provided that he cannot solve the discrete logarithm problem.

Diffie–Hellman Secret-Key Agreement Protocol

Step	Alice	Erich	Bob
1	Alice and Bob agree on a large prime p and a primitive element γ of p; p and γ are public		
2	chooses a large random number a, keeps it secret, and computes $\alpha = \gamma^a \bmod p$		chooses a large random number b, keeps it secret, and computes $\beta = \gamma^b \bmod p$
3		$\alpha \Rightarrow$ $\Leftarrow \beta$	
4	computes her key $k_A = \beta^a \bmod p$		computes his key $k_B = \alpha^b \bmod p$

Primitive Elements

Recall the multiplicative group

$$\mathbb{Z}_n^* = \{i \mid 1 \leq i \leq n-1 \text{ and } \gcd(i,n) = 1\}$$

of order $\varphi(n) = |\mathbb{Z}_n^*|$, where φ is the Euler function.

Definition

A *primitive element of a number* $n \in \mathbb{N}$ is an element $\gamma \in \mathbb{Z}_n^*$ satisfying

$$\gamma^d \not\equiv 1 \bmod n$$

for each d with $1 \leq d < \varphi(n)$.

Remark:

- A primitive element γ of n is a *generator* of the entire group $\mathbb{Z}_n^*$:

$$\mathbb{Z}_n^* = \langle\gamma\rangle = \{\gamma^i \mid 0 \leq i < \varphi(n)\}.$$

Primitive Elements

Remark:

- Not every integer has a primitive element; the number 8 is the smallest such example:

  $\mathbb{Z}_8^* = \{1,3,5,7\}$, so $\varphi(8) = 4$.

  $1^1 = 1, \;\; 3^2 = 9 \equiv 1 \bmod 8, \;\; 5^2 = 25 \equiv 1 \bmod 8, \;\; 7^2 = 49 \equiv 1 \bmod 8.$

- It is known from elementary number theory that a number n has a primitive element if and only if
  - n either is in $\{1,2,4\}$,
  - or is of the form $n = q^k$ or $n = 2q^k$ for some odd prime q.

Primitive Elements

Example (primitive element)

Consider $\mathbb{Z}_5^* = \{1,2,3,4\}$, so $\varphi(5) = 4$.

5 has two primitive elements: 2 and 3, both generating $\mathbb{Z}_5^*$:

$$\begin{array}{llllllllllll} 2^0 & = & 1, & 2^1 & = & 2, & 2^2 & = & 4, & 2^3 & \equiv & 3 \bmod 5; \\ 3^0 & = & 1, & 3^1 & = & 3, & 3^2 & \equiv & 4 \bmod 5, & 3^3 & \equiv & 2 \bmod 5. \end{array}$$

Primitive Elements

Fact

For each prime p, $\mathbb{Z}_p^*$ has exactly $\varphi(p-1)$ primitive elements.

Proof: Since a primitive element γ of p generates $\mathbb{Z}_p^*$, every $x \in \mathbb{Z}_p^*$ can be uniquely written as

$$x = \gamma^i \quad \text{for some } i,\ 0 \leq i < p-1.$$

The *order of x* is defined as the smallest $k > 0$ such that $x^k = 1$.

Note that $x = \gamma^i$ has order $\frac{p-1}{\texttt{gcd}(p-1,i)}$ (see next slide).

It follows that x itself is a primitive element of p if and only if $\texttt{gcd}(p-1,i) = 1$, and hence there are exactly $\varphi(p-1)$ primitive elements of p. ❑

Primitive Elements

The proof of this fact uses that the order of $x = \gamma^i$ is $\frac{p-1}{\gcd(p-1,i)}$.

Why?

Theorem

Let G be a multiplicative group with neutral element 1, let $g \in G$ be of finite order n, and let $k, \ell, m \in \mathbb{Z}$.

1. $g^m = 1 \iff n$ *divides* m.
2. $g^\ell = g^k \iff \ell \equiv k \bmod n$.

Primitive Elements

Proof:

1. $(\Rightarrow)$: Let $g^m = 1$ and $m = qn + r$ with $0 \leq r < n$.

   Since n is the smallest positive number with $g^n = 1$ and $0 \leq r < n$, we must have $r = 0$. Hence, $m = qn$, so n divides m.

   $(\Leftarrow)$: Assume $m = qn$. It follows that
   $$g^m = g^{qn} = (g^n)^q = 1^q = 1.$$

2. follows from the first statement with $m = \ell - k$ because
   $$g^\ell = g^k \iff \ell \equiv k \bmod n$$
   is equivalent to
   $$g^m = g^{\ell - k} = 1 \iff m = \ell - k \equiv 0 \bmod n. \quad ❑$$

Primitive Elements

Corollary

If $\gamma \in G$ is of finite order n and $i \in \mathbb{Z}$, then the order of γ^i is $\frac{n}{\gcd(n,i)}$.

Proof: We have $\left(\gamma^i\right)^{n/\gcd(n,i)} = (\gamma^n)^{i/\gcd(n,i)} = 1$.

By the theorem's first statement, the order of γ^i divides $\frac{n}{\gcd(n,i)}$.

Now let k be the order of γ^i, i.e., $1 = \left(\gamma^i\right)^k = \gamma^{i \cdot k}$.

Again, it follows from the theorem's first statement that n divides $i \cdot k$.

Hence, $\frac{n}{\gcd(n,i)}$ divides k.

Since k divides $\frac{n}{\gcd(n,i)}$ and $\frac{n}{\gcd(n,i)}$ divides k, we have $k = \frac{n}{\gcd(n,i)}$. ❑

Primitive Elements

Example (γ^i has order $\frac{p-1}{\gcd(p-1,i)}$)

Let $p = 17$ be a given prime number.

Note that $\gamma = 3$ is a primitive element of 17 (see next slide).

$3^4 \bmod 17 = 13$ has order $\frac{16}{\gcd(16,4)} = 4$, since

$$\begin{aligned}
13^1 &= 13 \neq 1, \\
13^2 &= 169 \equiv -1 \bmod 17 = 16 \neq 1, \\
13^3 &= -13 \bmod 17 = 4 \neq 1, \\
13^4 &= (-1)(-1) \bmod 17 = 1.
\end{aligned}$$

How to Determine All Primitive Elements

Example (computing all primitive elements)

- Let $p = 17$ be a given prime number.
- Note that $\mathbb{Z}_{16}^* = \{1, 3, 5, 7, 9, 11, 13, 15\}$, so $\varphi(16) = 8$ is the number of primitive elements modulo 17.
- It can be verified that 3 is a primitive element of 17, since 3 generates $\mathbb{Z}_{17}^* = \{1, 3, 9, 10, 13, 5, 15, 11, 16, 14, 8, 7, 4, 12, 2, 6\}$.
- The remaining primitive elements modulo 17 can be determined as follows. Recall from the previous proof that the order of $x = \gamma^i$ is

$$\frac{p-1}{\gcd(p-1, i)}. \tag{1}$$

How to Determine All Primitive Elements

Example (computing all primitive elements: continued)

- First, compute all successive powers of 3 modulo 17:

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$3^i \bmod 17$	1	3	9	10	13	5	15	11	16	14	8	7	4	12	2	6

Table: Computing the primitive elements modulo 17

- By (1), an element $3^i \bmod 17$ is primitive if and only if $\mathtt{gcd}(16, i) = 1$.

- The above table shows these primitive elements in gray boxes.

How to Test if a Given Element is Primitive

- If p is very large, it can be costly to compute $p-1$ powers of $\gamma \in \mathbb{Z}_p^*$.
- This can be speeded up if the prime factorization of $p-1$ is known.

Theorem

Let p be prime. An element $\gamma \in \mathbb{Z}_p^*$ is primitive for p if and only if

$$\gamma^{(p-1)/q} \not\equiv 1 \bmod p$$

for each prime q dividing $p-1$.

How to Test if a Given Element is Primitive

Proof: $(\Rightarrow)$: If γ is a primitive element of p, by definition we have

$$\gamma^i \not\equiv 1 \bmod p$$

for all i, $1 \leq i \leq p-2$, which implies the right-hand side.

$(\Leftarrow)$: Suppose that γ is not a primitive element of p. Let k be the order of γ, i.e., the smallest positive number with $\gamma^k \equiv 1 \bmod p$.

Then $k < p-1$ because γ is not primitive.

By Lagrange's theorem, k divides $p-1$, the order of the group $\mathbb{Z}_p^*$.

Hence, $\frac{p-1}{k}$ is an integer larger than 1.

How to Test if a Given Element is Primitive

Let q be a prime divisor of $\frac{p-1}{k}$.

Then k divides $\frac{p-1}{q}$ because:

$$\frac{p-1}{k} = a \cdot q \quad \text{implies} \quad \frac{p-1}{q} = a \cdot k.$$

Since k divides $\frac{p-1}{q}$, it follows that

$$\left(\gamma^{k}\right)^{a} = \gamma^{a \cdot k} = \gamma^{\frac{p-1}{q}} \equiv 1 \bmod p$$

by the first statement of the previous theorem. ❑

How to Test if a Given Element is Primitive

Example (for the previous theorem)

1. Consider $p = 17$, so $p - 1 = 16 = 2^4$, i.e., $q = 2$ is the only prime divisor of 16. $\gamma = 3$ is a primitive element of 17, since
$$\gamma^{\frac{p-1}{q}} = 3^{\frac{16}{2}} = 3^8 \bmod 17 = 16 \not\equiv 1 \bmod 17.$$
However, $\gamma = 4$ is **not** a primitive element of 17, since $4^8 \bmod 17 = 1$.
2. Now let $p = 19$, so $p - 1 = 18 = 2 \cdot 3^2$. Check $2, 3, \ldots, 17$:
$$\begin{array}{cccc}
2^9 \bmod 19 = 18 & \text{and} & 2^6 \bmod 19 = 7 & \checkmark \\
3^9 \bmod 19 = 18 & \text{and} & 3^6 \bmod 19 = 7 & \checkmark \\
4^9 \bmod 19 = 1 & ✗ & & \\
5^9 \bmod 19 = 1 & ✗ & & \\
 & \vdots & &
\end{array}$$

Diffie–Hellman Secret-Key Agreement Protocol

Step	Alice	Erich	Bob
1	Alice and Bob agree on a large prime p and a primitive element γ of p; p and γ are public		
2	chooses a large random number a, keeps it secret, and computes $\alpha = \gamma^a \bmod p$		chooses a large random number b, keeps it secret, and computes $\beta = \gamma^b \bmod p$
3		$\alpha \Rightarrow$ $\Leftarrow \beta$	
4	computes her key $k_A = \beta^a \bmod p$		computes his key $k_B = \alpha^b \bmod p$

Diffie–Hellman Secret-Key Agreement Protocol

Remark:

- The Diffie–Hellman protocol works, since in the arithmetics modulo p:

$$k_A = \beta^a = \gamma^{ba} = \gamma^{ab} = \alpha^b = k_B$$

- Thus, Alice and Bob indeed compute the same key.

- Using the "square-and-multiply" algorithm to perform exponentiation fast, both Alice and Bob can efficiently determine this key.

Diffie–Hellman Secret-Key Agreement Protocol

Example (Diffie–Hellman)

Suppose that Alice und Bob have chosen the prime number $p = 17$ and the primitive element $\gamma = 12$ of 17. (Check: $12^8 \not\equiv 1 \bmod 17$.)

Further, Alice chooses the secret number $a = 10$ at random.

She wants to send the number $\alpha = 12^{10} \bmod 17$ to Bob.

Applying the "square-and-multiply" algorithm, she first computes the binary expansion of the exponent, $10 = 2^1 + 2^3$, and then the values $12^{2^i} \bmod 17$ for $0 \leq i \leq 3$:

$12^{2^0} \bmod 17$	$12^{2^1} \bmod 17$	$12^{2^2} \bmod 17$	$12^{2^3} \bmod 17$	$\alpha = 12^{10} \bmod 17$
12	8	13	16	9

Diffie–Hellman Secret-Key Agreement Protocol

Example (Diffie–Hellman: continued)

Multiplying the values in the gray boxes, she obtains

$$\alpha = 12^{10} \equiv 9 \bmod 17$$

and sends $\alpha = 9$ to Bob.

Meanwhile, Bob has chosen his secret exponent $b = 15$ and has computed his value $\beta = 12^{15} \equiv 10 \bmod 17$ by the same procedure.

Bob sends $\beta = 10$ to Alice. Now, Alice and Bob compute

$$k_A = 10^{10} \equiv 2 \bmod 17 \quad \text{and} \quad k_B = 9^{15} \equiv 2 \bmod 17$$

to determine their joint secret key, $k_A = 2 = k_B$.

Security of Diffie–Hellman

How secure is the Diffie–Hellman protocol?

Answer: Security of the Diffie–Hellman protocol rests on the hardness of computing discrete logarithms.

- Direct (passive) attack on Diffie–Hellman:
  Erich solves the "Diffie–Hellman problem."
- Active "Man-in-the-Middle" attack:
  Erich changes the protocol to his advantage.

Man-in-the-Middle Attack on Diffie–Hellman

Step	Alice	Erich	Bob
1	Alice and Bob agree on a large prime p and a primitive element γ of p; p and γ are public		
2	chooses a large random number a, keeps it secret, and computes $\alpha = \gamma^a \bmod p$	chooses a secret number e and computes $\alpha_E = \beta_E = \gamma^e \bmod p$	chooses a large random number b, keeps it secret, and computes $\beta = \gamma^b \bmod p$
3		$\alpha \Rightarrow \mid \alpha_E \Rightarrow$ $\Leftarrow \beta_E \mid \Leftarrow \beta$	
4	computes her key $k_{A,E} = (\beta_E)^a \bmod p$	computes his keys $k_{E,A} = \alpha^e \bmod p$, $k_{E,B} = \beta^e \bmod p$	computes his key $k_{B,E} = (\alpha_E)^b \bmod p$

Modular Exponentiation and Discrete Logarithm

Definition

Let p be a prime, and let γ be a primitive element of p.

1. The *modular exponential function with base γ and modulus p* is the function $\exp_{\gamma,p}$ mapping from $\mathbb{Z}_{p-1}$ to $\mathbb{Z}_p^*$ and defined by

$$\exp_{\gamma,p}(a) = \gamma^a \bmod p.$$

2. Its inverse function is called the *discrete logarithm* and maps, for fixed p and γ, the value $\alpha = \exp_{\gamma,p}(a)$ to a. If $\alpha = \exp_{\gamma,p}(a)$, we write

$$a = \log_\gamma \alpha \bmod (p-1) \qquad \text{or, for short,} \qquad a = \log_\gamma \alpha.$$

The Discrete Logarithm Problem

Example

Let $p = 13$. A primitive element of 13 is 2. We have $2^4 = 16 \equiv 3 \bmod 13$.

α	1	2	3	4	5	6	7	8	9	10	11	12
$\log_2 \alpha$	0	1	4	2	9	5	11	3	8	10	7	6

Remark:

- In the following, we will consider only cyclic, multiplicative groups G of finite order n, such as $\mathbb{Z}_p^*$ with $n = p - 1$ for prime p.
- If G is not cyclic, the discrete logarithm may not always exist.
- Every cyclic group of finite order n is isomorphic to the *additive* group $\mathbb{Z}_n$. However, this group is not suitable for implementing Diffie–Hellman, as discrete logarithms can be computed efficiently.

The Discrete Logarithm Problem

Definition

The (functional) *discrete logarithm problem*, denoted by DLOG, is defined as follows: Given

- a cyclic, multiplicative group $(G,\cdot)$, represented by a primitive element $\gamma \in G$ of order n, and
- an element $\alpha \in \langle \gamma \rangle$,

compute the unique element a with $0 \leq a \leq n-1$ such that

$$a = \log_{\gamma} \alpha.$$

Equivalently, given γ and α, compute the unique element a with

$$\gamma^{a} = \alpha.$$

Direct Attack on Diffie–Hellman: Diffie–Hellman Problem

Definition

The (functional) *Diffie–Hellman problem*, denoted by DIFFIE-HELLMAN, is defined as follows: Given

- an element $\gamma \in \mathbb{Z}_p^*$ of order $n = p - 1$ for some prime number p, and
- two elements α and β in $\langle \gamma \rangle = \mathbb{Z}_p^*$,

compute an element $\delta \in \langle \gamma \rangle$ such that

$$\log_\gamma \delta \equiv (\log_\gamma \alpha)(\log_\gamma \beta) \bmod n.$$

Equivalently, given γ, $\alpha = \gamma^a \bmod p$, and $\beta = \gamma^b \bmod p$, compute

$$\gamma^{ab} \bmod p.$$

Discrete Logarithm Problem vs. Diffie–Hellman Problem

- If Erich were able to compute discrete logarithms efficiently, he would be able to solve the Diffie–Hellman problem, since he could determine
  - Alice's private exponent $a = \log_\gamma \alpha \bmod (p-1)$ from p, γ, and α, and
  - Bob's private exponent $b = \log_\gamma \beta \bmod (p-1)$ from p, γ, and β.
- Thus, computing discrete logarithms is no easier than solving the Diffie–Hellman problem.
- This argument can easily be generalized from $\mathbb{Z}_p^*$ to arbitrary cyclic, multiplicative groups and thus proves the following fact.

Discrete Logarithm Problem vs. Diffie–Hellman Problem

Fact

The Diffie–Hellman problem reduces to the discrete logarithm problem under polynomial-time Turing reductions: DIFFIE-HELLMAN *is in* $\mathrm{FP}^{\mathrm{DLOG}}$.

- The converse question of whether the discrete logarithm problem is at least as hard as the Diffie–Hellman problem remains an unproven conjecture.
- The Diffie–Hellman protocol currently has no proof of security, not even in the sense that it is as hard as the discrete logarithm, which itself is a problem whose precise complexity is an open issue.

Exhaustive Search Algorithm

The discrete logarithm problem can be solved by exhaustive search:

- Given γ and α, successively compute

$$\gamma,\ \gamma^2,\ \gamma^3,\ \ldots,$$

  until the unique exponent a with

$$\gamma^a = \alpha$$

  is found.

- This can be done by computing $\gamma^i = \gamma \cdot \gamma^{i-1}$ for $1 < i < n$.
- Hence, assuming that executing one group operation costs constant time, this naive brute-force algorithm requires time $\mathcal{O}(n)$, which is exponential in the length of n and thus exponential in the input size.

Shanks' Baby-Step Giant-Step Algorithm

SHANKS(n, γ, α) {

(* $G = \langle \gamma \rangle$ *is a cyclic, multiplicative group, generated by a primitive element* γ *of order* n, *and* $\alpha \in G$ *)

$s := \lceil \sqrt{n} \rceil$;

`for` $(i = 0, 1, \ldots, s-1)$ { add (γ^{is}, i) to a list $\mathscr{L}_1$; }

Sort the elements of $\mathscr{L}_1$ with respect to their first coordinates;

`for` $(j = 0, 1, \ldots, s-1)$ { add $(\alpha\gamma^{-j}, j)$ to a list $\mathscr{L}_2$; }

Sort the elements of $\mathscr{L}_2$ with respect to their first coordinates;

Find a pair $(\delta, i) \in \mathscr{L}_1$ and a pair $(\delta, j) \in \mathscr{L}_2$, i.e., find two pairs with identical first coordinates;

`return` "$\log_\gamma \alpha = is + j$" and halt;

}

Figure: Shanks' baby-step giant-step algorithm

Shanks' Baby-Step Giant-Step Algorithm: Explanation

In order to compute $\log_\gamma \alpha$ for given values α and γ, where γ is a primitive element of order n, Shanks' algorithm first determines $s = \lceil \sqrt{n} \rceil$.

If we now set

$$a = is + j, \quad 0 \leq j < s,$$

we have

$$\alpha = \gamma^a = \gamma^{is+j}. \tag{2}$$

We want to determine $a = \log_\gamma \alpha$.

Equation (2) implies $\alpha\gamma^{-j} = (\gamma^s)^i$.

The pairs $(\alpha\gamma^{-j}, j)$ with $0 \leq j < s$ are the elements of the list $\mathscr{L}_2$, sorted with respect to the first coordinates, which represent the **"baby steps."**

Shanks' Baby-Step Giant-Step Algorithm: Explanation

If the pair $(1, j)$ is in $\mathscr{L}_2$ for some j, we are done, since $\alpha\gamma^{-j} = 1$ implies $\alpha = \gamma^j$, so setting $a = j$ solves the discrete logarithm problem in this case.

Otherwise, we determine

$$\delta = \gamma^s$$

and search for a group element δ^i, $1 \leq i < s$, occurring as the first coordinate of some element in $\mathscr{L}_2$.

The elements $(\gamma^s)^i = \gamma^{is}$ are collected in the list $\mathscr{L}_1$, again sorted with respect to the first coordinates, and represent the **"giant steps."**

Once a pair (γ^{is}, i) is found in $\mathscr{L}_1$ such that (γ^{is}, j) occurs in the list $\mathscr{L}_2$ of baby steps, we have solved the discrete logarithm problem, since

$$\alpha\gamma^{-j} = \delta^i = \gamma^{is}$$

implies $\alpha = \gamma^{is+j}$, so $a = \log_\gamma \alpha = is + j$.

Shanks' Baby-Step Giant-Step Algorithm: Example

Example

- Suppose we want to find $a = \log_2 47 \bmod 100$ in the group $\mathbb{Z}_{101}^*$, using Shanks' algorithm. That is, $p = 101$, $\gamma = 2$, and $\alpha = 47$ are given.

- Note that 101 is a prime number and 2 is a primitive element of 101.

- Since $n = p - 1 = 100$ is the order of 2, we have $s = \left\lceil \sqrt{100} \right\rceil = 10$.

- It follows that

$$\gamma^s \bmod p = 2^{10} \bmod 101 = 14.$$

Shanks' Baby-Step Giant-Step Algorithm

Example (continued)

- Now, the sorted lists $\mathscr{L}_1$ and $\mathscr{L}_2$ can be determined as follows:

$\mathscr{L}_1$	$(1,0)$	$(14,1)$	$(95,2)$	$(17,3)$	$(36,4)$	$(100,5)$	$(87,6)$	$(6,7)$	$(84,8)$	$(65,9)$
$\mathscr{L}_1$ sorted	$(1,0)$	$(6,7)$	$(14,1)$	$(17,3)$	$(36,4)$	$(65,9)$	$(84,8)$	$(87,6)$	$(95,2)$	$(100,5)$
$\mathscr{L}_2$	$(47,0)$	$(74,1)$	$(37,2)$	$(69,3)$	$(85,4)$	$(93,5)$	$(97,6)$	$(99,7)$	$(100,8)$	$(50,9)$
$\mathscr{L}_2$ sorted	$(37,2)$	$(47,0)$	$(50,9)$	$(69,3)$	$(74,1)$	$(85,4)$	$(93,5)$	$(97,6)$	$(99,7)$	$(100,8)$

- Since $(100,5)$ is in $\mathscr{L}_1$ and $(100,8)$ is in $\mathscr{L}_2$, we obtain

$$a = 5 \cdot 10 + 8 = 58.$$

- It can be verified that $2^{58} \bmod 101 = 47$, as desired.

Analysis of Shanks' Baby-Step Giant-Step Algorithm

- The first `for` loop can be implemented so as to first compute γ^s and then raising its powers by multiplying by γ^s.
- Similarly, the second `for` loop is performed by first computing the inverse element γ^{-1} of γ in the group and then computing its powers.
- Both `for` loops require time $\mathcal{O}(s)$.
- Using an efficient sorting algorithm such as quicksort, the lists $\mathscr{L}_1$ and $\mathscr{L}_2$ can be sorted in time $\mathcal{O}(s \log s)$.
- Finally, the two pairs whose first coordinate occurs in both lists can be found in time $\mathcal{O}(s)$ by simultaneously passing through both lists.

Analysis of Shanks' Baby-Step Giant-Step Algorithm

- Summing up, Shanks' algorithm can be implemented to run
  - in time $\mathcal{O}^*(s) = \mathcal{O}^*(\sqrt{n})$ and
  - to require the same amount of space,

  where $\mathcal{O}^*$ indicates that logarithmic factors are neglected as is usually done in the analysis of discrete logarithm algorithms.

- Although Shanks' algorithm is more efficient than the exhaustive search algorithm, it is not an efficient algorithm.

- There are many other (also inefficient) algorithms for the discrete logarithm problem, some of which are better than Shanks' algorithm:
  - Pollard's ρ algorithm,
  - the Pohlig–Hellman algorithm,
  - the index calculus method, and variants thereof.

The Pohlig–Hellman Algorithm

- Let a primitive element γ of a prime p and $\alpha \in \langle\gamma\rangle$ be given.
- Suppose we know the factorization of the group order

$$n = \prod_{i=1}^{k} p_i^{c_i} = p - 1$$

  for distinct prime numbers p_i.

- The value of

$$a = \log_\gamma \alpha \bmod (p-1)$$

  is uniquely determined.

- If we can compute $a = \log_\gamma \alpha \bmod p_i^{c_i}$ for each i, $1 \leq i \leq k$, then we obtain $a \bmod n$ by the Chinese Remainder Theorem.

The Pohlig–Hellman Algorithm

- Let q be a prime number and $c \geq 1$ be a constant such that

$$n \equiv 0 \bmod q^c \quad \text{and} \quad n \not\equiv 0 \bmod q^{c+1}.$$

- We show how to compute

$$x = a \bmod q^c, \quad 0 \leq x < q^c.$$

- In q-ary representation, we have

$$x = \sum_{i=0}^{c-1} a_i \cdot q^i, \quad 0 \leq a_i \leq q-1 \text{ for } 0 \leq i \leq c-1.$$

- Since $a = x + s \cdot q^c$ for some $s \in \mathbb{Z}$, we have

$$a = \left(\sum_{i=0}^{c-1} a_i \cdot q^i \right) + s \cdot q^c.$$

The Pohlig–Hellman Algorithm

- So we have to determine the coefficients a_i, $0 \leq i \leq c-1$.
- Starting with a_0, we first show:

$$\alpha^{n/q} = \gamma^{a_0 (n/q)}. \tag{3}$$

Proof of (3):

$$\begin{aligned}
\alpha^{n/q} &= (\gamma^a)^{n/q} \\
&= \left(\gamma^{a_0 + a_1 \cdot q + \cdots + a_{c-1} \cdot q^{c-1} + s \cdot q^c}\right)^{n/q} \\
&= \left(\gamma^{a_0 + k \cdot q}\right)^{n/q}, \quad \text{where } k \in \mathbb{N} \\
&= \gamma^{a_0 \cdot n/q} \cdot \gamma^{k \cdot n} \quad \text{and since } \gamma^n = 1 \\
&= \gamma^{a_0 \cdot n/q}. \quad \square
\end{aligned}$$

The Pohlig–Hellman Algorithm

- By (3), we can determine a_0 as follows:
  Compute $\delta = \gamma^{n/q}$, δ^2, ... until, for some $i \leq q-1$,
  $$\delta^i = \alpha^{n/q}.$$
  Then $a_0 = i$.
- If $c = 1$, we are done.
- If $c > 1$, we have to determine now $a_1, a_2, \ldots, a_{c-1}$, similarly to a_0.

The Pohlig–Hellman Algorithm

- Let $\alpha_0 = \alpha$. Define for $1 \leq j \leq c-1$:
$$\alpha_j = \alpha \cdot \gamma^{-\left(a_0 + a_1 q + \cdots + a_{j-1} q^{j-1}\right)}.$$
- Generalizing (3) (i.e., (4) is (3) for $j = 0$), we now show:
$$\alpha_j^{n/q^{j+1}} = \gamma^{a_j (n/q)}. \tag{4}$$

Proof of (4):

$$\begin{aligned}
\alpha_j^{n/q^{j+1}} &= \left(\gamma^{a-(a_0 + a_1 \cdot q + \cdots + a_{j-1} \cdot q^{j-1})}\right)^{n/q^{j+1}} \\
&= \left(\gamma^{a_j \cdot q^j + \cdots + a_{c-1} \cdot q^{c-1} + s \cdot q^c}\right)^{n/q^{j+1}} \\
&= \left(\gamma^{a_j \cdot q^j + k_j \cdot q^{j+1}}\right)^{n/q^{j+1}}, \quad \text{where } k_j \in \mathbb{N} \\
&= \gamma^{a_j \cdot n/q} \cdot \gamma^{k_j \cdot n} \quad \text{and since } \gamma^n = 1 \\
&= \gamma^{a_j \cdot n/q}. \quad \square
\end{aligned}$$

The Pohlig–Hellman Algorithm

- By (4), we can determine a_j from α_j as follows:
  Compute $\delta = \gamma^{n/q},\ \delta^2,\ \ldots$ until, for some $i \leq q-1$,
  $$\delta^i = \alpha_j^{n/q^{j+1}}.$$
  Then $a_j = i$.

- **How do we get α_j?**

- If a_j is known, we can determine α_{j+1} from α_j using the recurrence
  $$\alpha_{j+1} = \alpha_j \cdot \gamma^{-a_j q^j}, \tag{5}$$
  which follows immediately from the definition of α_j.

The Pohlig–Hellman Algorithm

- Thus, applying (4) and (5) alternately, we can compute:

$$a_0,\ \alpha_1,\ a_1,\ \alpha_2,\ a_2,\ \ldots,\ \alpha_{c-1},\ a_{c-1}.$$

- Summing up, if γ is a primitive element of order n and q is a prime such that

$$n \equiv 0 \bmod q^c \quad \text{and} \quad n \not\equiv 0 \bmod q^{c+1},$$

  then the Pohlig–Hellman algorithm computes coefficients $(a_0, a_1, \ldots, a_{c-1})$ with

$$\log_\gamma \alpha \bmod q^c = \sum_{i=0}^{c-1} a_i \cdot q^i.$$

The Pohlig–Hellman Algorithm

Pohlig-Hellman$(n, \gamma, \alpha, q, c)$ {

(γ is a primitive element of order n, $\alpha \in \langle \gamma \rangle$, q is a prime, and c is a constant satisfying $n \equiv 0 \bmod q^c$ and $n \not\equiv 0 \bmod q^{c+1}$ *)*

$j := 0$;

$\alpha_j := \alpha$;

`while` $(j \leq c-1)$ {

Set $\delta := \alpha_j^{n/q^{j+1}}$ and find an i with $\delta = \gamma^{i(n/q)}$;

$a_j := i$; *(* according to (4) *)*

$\alpha_{j+1} := \alpha_j \cdot \gamma^{-a_j q^j}$; *(* according to (5) *)*

$j := j+1$;

}

`return` "$(a_0, a_1, \ldots, a_{c-1})$" and halt;

}

Analysis of the Pohlig–Hellman Algorithm

- Direct implementation of Pohlig–Hellman:
  - There are c `while` loops.
  - The most expensive step per loop is (4): "*Find an i with $\delta = \gamma^{i(n/q)}$.*"
  - This step requires at most q multiplications, since $\gamma^{q(n/q)} = \gamma^n = 1$.
  - Thus, we have a running time of $\mathcal{O}^*(c \cdot q)$.

- This running time analysis can be improved, since $\delta = \gamma^{i(n/q)}$ is itself an instance of the discrete logarithm problem:

$$\delta = \gamma^{i(n/q)} \iff i = \log_{\gamma^{n/q}} \delta.$$

  - The element $\gamma^{n/q}$ has order q.
  - Thus, each i can be found in time $\mathcal{O}^*(\sqrt{q})$ (e.g., by Shanks' algorithm).
  - This gives a total running time of $\mathcal{O}^*(c\sqrt{q})$.

Analysis of the Pohlig–Hellman Algorithm

Remark: The running time is dominated by $\sqrt{q}$.

If q (the largest prime divisor of the group order n) is too small, discrete logarithms can be easily computed.

For example,

$$p = 2 \cdot 3 \cdot 5^{278} + 1$$

is a prime number of binary length 649.

$\mathbb{Z}_p^*$ has order

$$p - 1 = 2 \cdot 3 \cdot 5^{278}.$$

But since 5 is the largest prime divisor, p cannot be used for cryptographic purposes.

Pohlig–Hellman Algorithm: Example

Example (Pohlig–Hellman Algorithm)

Let $p = 29$ and $\gamma = 2$ a primitive element of 29. We have

$$n = p - 1 = 28 = 2^2 \cdot 7^1.$$

Suppose $\alpha = 18$, so we want to determine

$$a = \log_2 18 \bmod 28,$$

by computing

1. first $a \bmod 4$,
2. then $a \bmod 7$.

Pohlig–Hellman Algorithm: Example

Example (Pohlig–Hellman Algorithm: continued)

1. **Computing a** mod **4:** $q = 2$ and $c = 2$.

$j = 0$: $\alpha_0 = \alpha = 18$ and $\delta = \alpha_0^{n/q^{j+1}} = 18^{28/2} = 18^{14} \equiv 28 \bmod 29$.

$\Rightarrow$ For $i = 1$, we have $\delta = \gamma^{i \cdot n/q} = 2^{i \cdot 14} \equiv 28 \bmod 29$.

$\Rightarrow a_0 = 1$

$\Rightarrow \alpha_1 = \alpha_0 \gamma^{-a_0 q^0} = 18 \cdot 2^{-1} \equiv 9 \bmod 29$

$j = 1$: $\delta = \alpha_1^{n/q^{j+1}} = 9^{28/4} = 9^7 \equiv 28 \bmod 29$.

$\Rightarrow$ For $i = 1$, we have $\delta = \gamma^{i \cdot n/q} = 2^{i \cdot 14} \equiv 28 \bmod 29$.

$\Rightarrow a_1 = 1$

Hence, $a = a_0 q^0 + a_1 q^1 = 1 \cdot 2^0 + 1 \cdot 2^1 = 3$, so $\mathbf{a} \equiv \mathbf{3} \bmod \mathbf{4}$.

Pohlig–Hellman Algorithm: Example

Example (Pohlig–Hellman Algorithm: continued)

2. **Computing a** mod **7:** $q = 7$ and $c = 1$.

$j = 0$: $\alpha_0 = \alpha = 18$ and $\delta = \alpha_0^{n/q^{j+1}} = 18^{28/7} = 18^4 \equiv 25 \bmod 29$.

$\gamma^{n/q} = 2^{28/7} = 16$

$\Rightarrow$ For $i = 4$, we have $\delta = \gamma^{i \cdot n/q} = 2^{4 \cdot 4} = 2^{16} \equiv 25 \bmod 29$.

$\Rightarrow$ $a_0 = 4$, so $\mathbf{a} \equiv \mathbf{4} \bmod \mathbf{7}$.

Pohlig–Hellman Algorithm: Example

Example (Pohlig–Hellman Algorithm: continued)

Applying the Chinese Remainder Theorem to

$$a \equiv 3 \bmod 4$$
$$a \equiv 4 \bmod 7$$

with $q_1 = {}^{28}/_{4} = 7$ and $q_1^{-1} = 7$ (check: $7 \cdot 7 \equiv 1 \mod 4$) and
with $q_2 = {}^{28}/_{7} = 4$ and $q_2^{-1} = 2$ (check: $4 \cdot 2 \equiv 1 \mod 7$), we get

$$a = 3 \cdot 7 \cdot 7 + 4 \cdot 4 \cdot 2 = 179 \equiv 11 \bmod 28.$$

Check: $2^{11} \equiv 18 \bmod 29$.