Chapter 8
Public-Key Cryptosystems Based on the Discrete Logarithm Problem

http://www.crypto-textbook.com/

Understanding Cryptography:
A Textbook for Students and Practitioners

by Christof Paar and Jan Pelzl
1st ed. 2010 Edition

In the previous chapter we learned about the RSA public-key scheme. As we have seen, RSA is based on the hardness of factoring large integers. The integer factorization problem is said to be the *one-way function* of RSA. As we saw earlier, roughly speaking a function is *one-way* if it is computationally easy to compute the function $f(x) = y$, but computationally infeasible to invert the function: $f^{-1}(y) = x$. The question is whether we can find other one-way functions for building asymmetric crypto schemes. It turns out that most non-RSA public-key algorithms with practical relevance are based on another one-way function, the discrete logarithm problem.

In this chapter you will learn:

- The Diffie–Hellman key exchange
- Cyclic groups which are important for a deeper understanding of Diffie–Hellman key exchange
- The discrete logarithm problem, which is of fundamental importance for many practical public-key algorithms
- Encryption using the Elgamal scheme

The security of many cryptographic schemes relies on the computational intractability of finding solutions to the *Discrete Logarithm Problem (DLP)*. Well-known examples of such schemes are the Diffie–Hellman key exchange and the Elgamal encryption scheme, both of which will be introduced in this chapter. Also, the Elgamal digital signature scheme (cf. Section 8.5.1) and the digital signature algorithm (cf. Section 10.2) are based on the DLP, as are cryptosystems based on elliptic curves (Section 9.3).

We start with the basic Diffie–Hellman protocol, which is surprisingly simple and powerful. The discrete logarithm problem is defined in what are called *cyclic groups*. The concept of this algebraic structure is introduced in Section 8.2. A formal definition of the DLP as well as some illustrating examples are provided, followed by a brief description of attack algorithms for the DLP. With this knowledge we will revisit the Diffie–Hellman protocol and more formally talk about its security. We will then develop a method for encrypting data using the DLP that is known as the Elgamal cryptosystem.

C. Paar, J. Pelzl, *Understanding Cryptography*,
DOI 10.1007/978-3-642-04101-3_8, © Springer-Verlag Berlin Heidelberg 2010

8.1 Diffie–Hellman Key Exchange

The *Diffie–Hellman key exchange (DHKE)*, proposed by Whitfield Diffie and Martin Hellman in 1976 [58], was the first asymmetric scheme published in the open literature. The two inventors were also influenced by the work of Ralph Merkle. It provides a practical solution to the key distribution problem, i.e., it enables two parties to derive a common secret key by communicating over an insecure channel[1]. The DHKE is a very impressive application of the discrete logarithm problem that we'll study in the subsequent sections. This fundamental key agreement technique is implemented in many open and commercial cryptographic protocols like Secure Shell (SSH), Transport Layer Security (TLS), and Internet Protocol Security (IPSec).The basic idea behind the DHKE is that exponentiation in $\mathbb{Z}_p^*$, p prime, is a one-way function and that exponentiation is commutative, i.e.,

$$k = (\alpha^x)^y \equiv (\alpha^y)^x \bmod p$$

The value $k \equiv (\alpha^x)^y \equiv (\alpha^y)^x \bmod p$ is the joint secret which can be used as the session key between the two parties.

Let us now consider how the Diffie–Hellman key exchange protocol over $\mathbb{Z}_p^*$ works. In this protocol we have two parties, Alice and Bob, who would like to establish a shared secret key. There is possibly a trusted third party that properly chooses the public parameters which are needed for the key exchange. However, it is also possible that Alice or Bob generate the public parameters. Strictly speaking, the DHKE consists of two protocols, the set-up protocol and the main protocol, which performs the actual key exchange. The set-up protocol consists of the following steps:

Diffie–Hellman Set-up

1. Choose a large prime p.
2. Choose an integer $\alpha \in \{2, 3, \ldots, p-2\}$.
3. Publish p and α.

These two values are sometimes referred to as *domain parameters*. If Alice and Bob both know the public parameters p and α computed in the set-up phase, they can generate a joint secret key k with the following key-exchange protocol:

[1] The channel needs to be authenticated, but that will be discussed later in this book.

Diffie–Hellman Key Exchange

Alice		**Bob**
choose $a = k_{pr,A} \in \{2,\ldots,p-2\}$		choose $b = k_{pr,B} \in \{2,\ldots,p-2\}$
compute $A = k_{pub,A} \equiv \alpha^a \bmod p$		compute $B = k_{pub,B} \equiv \alpha^b \bmod p$
	$\xrightarrow{k_{pub,A}=A}$	
	$\xleftarrow{k_{pub,B}=B}$	
$k_{AB} = k_{pub,B}^{k_{pr,A}} \equiv B^a \bmod p$		$k_{AB} = k_{pub,A}^{k_{pr,B}} \equiv A^b \bmod p$

Here is the proof that this surprisingly simple protocol is correct, i.e., that Alice and Bob in fact compute the same session key k_{AB}.

Proof. Alice computes

$$B^a \equiv (\alpha^b)^a \equiv \alpha^{ab} \bmod p$$

while Bob computes

$$A^b \equiv (\alpha^a)^b \equiv \alpha^{ab} \bmod p$$

and thus Alice and Bob both share the session key $k_{AB} \equiv \alpha^{ab} \bmod p$. The key can now be used to establish a secure communication between Alice and Bob, e.g., by using k_{AB} as key for a symmetric algorithm like AES or 3DES. □

We'll look now at a simple example with small numbers.

Example 8.1. The Diffie–Hellman domain parameters are $p = 29$ and $\alpha = 2$. The protocol proceeds as follows:

Alice		**Bob**
choose $a = k_{pr,A} = 5$		choose $b = k_{pr,B} = 12$
$A = k_{pub,A} = 2^5 \equiv 3 \bmod 29$		$B = k_{pub,B} = 2^{12} \equiv 7 \bmod 29$
	$\xrightarrow{A=3}$	
	$\xleftarrow{B=7}$	
$k_{AB} = B^a \equiv 7^5 = 16 \bmod 29$		$k_{AB} = A^b = 3^{12} \equiv 16 \bmod 29$

As one can see, both parties compute the value $k_{AB} = 16$, which can be used as a joint secret, e.g., as a session key for symmetric encryption.
◇

The computational aspects of the DHKE are quite similar to those of RSA. During the set-up phase, we generate p using the probabilistic prime-finding algorithms discussed in Section 7.6. As shown in Table 6.1, p should have a similar length as the RSA modulus n, i.e., 1024 or beyond, in order to provide strong security. The integer α needs to have a special property: It should be a primitive element, a topic which we discuss in the following sections. The session key k_{AB} that is being computed in the protocol has the same bit length as p. If we want to use it as a symmetric key for algorithms such as AES, we can simply take the 128 most significant bits. Alternatively, a hash function is sometimes applied to k_{AB} and the output is then used as a symmetric key.

During the actual protocol, we first have to choose the private keys a and b. They should stem from a true random generator in order to prevent an attacker from guessing them. For computing the public keys A and B as well as for computing the session key, both parties can make use of the square-and-multiply algorithm. The public keys are typically precomputed. The main computation that needs to be done for a key exchange is thus the exponentiation for the session key. In general, since the bit lengths and the computations of RSA and the DHKE are very similar, they require a similar effort. However, the trick of using short public exponents that was shown in Section 7.5 is not applicable to the DHKE.

What we showed so far is the classic Diffie–Hellman key exchange protocol in the group $\mathbb{Z}_p^*$, where p is a prime. The protocol can be generalized, in particular to groups of elliptic curves. This gives rise to elliptic curve cryptography, which has become a very popular asymmetric scheme in practice. In order to better understand elliptic curves and schemes such as Elgamal encryption, which are also closely related to the DHKE, we introduce the discrete logarithm problem in the following sections. This problem is the mathematical basis for the DHKE. After we have introduced the discrete logarithm problem, we will revisit the DHKE and discuss its security.

8.2 Some Algebra

This section introduces some fundamentals of abstract algebra, in particular the notion of groups, subgroups, finite groups and cyclic groups, which are essential for understanding discrete logarithm public-key algorithms.

8.2.1 Groups

For convenience, we restate here the definition of groups which was introduced in the Chapter 4:

Definition 8.2.1 Group
A group *is a set of elements G together with an operation $\circ$ which combines two elements of G. A group has the following properties.*

1. *The group operation $\circ$ is* closed. *That is, for all $a, b, \in G$, it holds that $a \circ b = c \in G$.*
2. *The group operation is* associative. *That is, $a \circ (b \circ c) = (a \circ b) \circ c$ for all $a, b, c \in G$.*
3. *There is an element $1 \in G$, called the* neutral element *(or* identity element*), such that $a \circ 1 = 1 \circ a = a$ for all $a \in G$.*
4. *For each $a \in G$ there exists an element $a^{-1} \in G$, called the* inverse *of a, such that $a \circ a^{-1} = a^{-1} \circ a = 1$.*
5. *A group G is* abelian (or commutative) *if, furthermore, $a \circ b = b \circ a$ for all $a, b \in G$.*

Note that in cryptography we use both multiplicative groups, i.e., the operation "$\circ$" denotes multiplication, and additive groups where "$\circ$" denotes addition. The latter notation is used for elliptic curves as we'll see later.

Example 8.2. To illustrate the definition of groups we consider the following examples.

- $(\mathbb{Z}, +)$ is a group, i.e., the set of integers $\mathbb{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$ together with the usual addition forms an abelian group, where $e = 0$ is the identity element and $-a$ is the inverse of an element $a \in \mathbb{Z}$.
- $(\mathbb{Z}$ without $0, \cdot)$ is **not** a group, i.e., the set of integers $\mathbb{Z}$ (without the element 0) and the usual multiplication does not form a group since there exists no inverse a^{-1} for an element $a \in \mathbb{Z}$ with the exception of the elements -1 and 1.
- $(\mathbb{C}, \cdot)$ is a group, i.e., the set of complex numbers $u + iv$ with $u, v \in \mathbb{R}$ and $i^2 = -1$ together with the complex multiplication defined by

$$(u_1 + iv_1) \cdot (u_2 + iv_2) = (u_1 u_2 - v_1 v_2) + i(u_1 v_2 + v_1 u_2)$$

  forms an abelian group. The identity element of this group is $e = 1$, and the inverse a^{-1} of an element $a = u + iv \in \mathbb{C}$ is given by $a^{-1} = (u - i)/(u^2 + v^2)$.

◇

However, all of these groups do not play a significant role in cryptography because we need groups with a finite number of elements. Let us now consider the group $\mathbb{Z}_n^*$ which is very important for many cryptographic schemes such as DHKE, Elgamal encryption, digital signature algorithm and many others.

Theorem 8.2.1
The set $\mathbb{Z}_n^*$ which consists of all integers $i = 0, 1, \ldots, n-1$ for which $\gcd(i,n) = 1$ *forms an abelian group under multiplication modulo n. The identity element is $e = 1$.*

Let us verify the validity of the theorem by considering the following example:

Example 8.3. If we choose $n = 9$, $\mathbb{Z}_n^*$ consists of the elements $\{1,2,4,5,7,8\}$.

Table 8.1 Multiplication table for $\mathbb{Z}_9^*$

$\times$ mod 9	**1**	**2**	**4**	**5**	**7**	**8**
1	1	2	4	5	7	8
2	2	4	8	1	5	7
4	4	8	7	2	1	5
5	5	1	2	7	8	4
7	7	5	1	8	4	2
8	8	7	5	4	2	1

By computing the *multiplication table* for $\mathbb{Z}_9^*$, depicted in Table 8.1, we can easily check most conditions from Definition 8.2.1. Condition 1 (closure) is satisfied since the table only consists of integers which are elements of $\mathbb{Z}_9^*$. For this group Conditions 3 (identity) and 4 (inverse) also hold since each row and each column of the table is a permutation of the elements of $\mathbb{Z}_9^*$. From the symmetry along the main diagonal, i.e., the element at row i and column j equals the element at row j and column i, we can see that Condition 5 (commutativity) is satisfied. Condition 2 (associativity) cannot be directly derived from the shape of the table but follows immediately from the associativity of the usual multiplication in $\mathbb{Z}_n$.
$\diamond$

Finally, the reader should remember from Section 6.3.1 that the inverse a^{-1} of each element $a \in \mathbb{Z}_n^*$ can be computed by using the extended Euclidean algorithm.

8.2.2 Cyclic Groups

In cryptography we are almost always concerned with *finite* structures. For instance, for AES we needed a finite field. We provide now the straightforward definition of a finite group:

Definition 8.2.2 Finite Group
A group (G, ∘) is finite *if it has a finite number of elements. We denote the* cardinality *or* order *of the group G by* $|G|$.

Example 8.4. Examples of finite groups are:

- $(\mathbb{Z}_n, +)$: the cardinality of $\mathbb{Z}_n$ is $|\mathbb{Z}_n| = n$ since $\mathbb{Z}_n = \{0, 1, 2, \ldots, n-1\}$.
- $(\mathbb{Z}_n^*, \cdot)$: remember that $\mathbb{Z}_n^*$ is defined as the set of positive integers smaller than n which are relatively prime to n. Thus, the cardinality of $\mathbb{Z}_n^*$ equals Euler's phi function evaluated for n, i.e., $|\mathbb{Z}_n^*| = \Phi(n)$. For instance, the group $\mathbb{Z}_9^*$ has a cardinality of $\Phi(9) = 3^2 - 3^1 = 6$. This can be verified by the earlier example where we saw that the group consist of the six elements $\{1, 2, 4, 5, 7, 8\}$.

◇

The remainder of this section deals with a special type of groups, namely cyclic groups, which are the basis for discrete logarithm-based cryptosystems. We start with the following definition:

Definition 8.2.3 Order of an element
The order $ord(a)$ *of an element a of a group* $(G, \circ)$ *is the smallest positive integer k such that*

$$a^k = \underbrace{a \circ a \circ \ldots \circ a}_{k \text{ times}} = 1,$$

where 1 *is the identity element of G.*

We'll examine this definition by looking at an example.

Example 8.5. We try to determine the order of $a = 3$ in the group $\mathbb{Z}_{11}^*$. For this, we keep computing powers of a until we obtain the identity element 1.

$$\begin{aligned}
a^1 &= 3 \\
a^2 &= a \cdot a \;\; = 3 \cdot 3 = 9 \\
a^3 &= a^2 \cdot a = 9 \cdot 3 = 27 \equiv 5 \bmod 11 \\
a^4 &= a^3 \cdot a = 5 \cdot 3 = 15 \equiv 4 \bmod 11 \\
a^5 &= a^4 \cdot a = 4 \cdot 3 = 12 \equiv 1 \bmod 11
\end{aligned}$$

From the last line it follows that $\text{ord}(3) = 5$.

◇

It is very interesting to look at what happens if we keep multiplying the result by a:

$$
\begin{aligned}
a^6 &= a^5 \cdot a \equiv 1 \cdot a \equiv 3 \bmod 11 \\
a^7 &= a^5 \cdot a^2 \equiv 1 \cdot a^2 \equiv 9 \bmod 11 \\
a^8 &= a^5 \cdot a^3 \equiv 1 \cdot a^3 \equiv 5 \bmod 11 \\
a^9 &= a^5 \cdot a^4 \equiv 1 \cdot a^4 \equiv 4 \bmod 11 \\
a^{10} &= a^5 \cdot a^5 \equiv 1 \cdot 1 \equiv 1 \bmod 11 \\
a^{11} &= a^{10} \cdot a \equiv 1 \cdot a \equiv 3 \bmod 11 \\
&\vdots
\end{aligned}
$$

We see that from this point on, the powers of a run through the sequence $\{3,9,5,4,1\}$ indefinitely. This cyclic behavior gives rise to following definition:

Definition 8.2.4 Cyclic Group
A group G which contains an element α with maximum order $ord(\alpha) = |G|$ is said to be cyclic. *Elements with maximum order are called* primitive elements *or* generators.

An element α of a group G with maximum order is called a generator since every element a of G can be written as a power $\alpha^i = a$ of this element for some i, i.e., α *generates* the entire group. Let us verify these properties by considering the following example.

Example 8.6. We want to check whether $a = 2$ happens to be a primitive element of $\mathbb{Z}_{11}^* = \{1,2,3,4,5,6,7,8,9,10\}$. Note that the cardinality of the group is $|\mathbb{Z}_{11}^*| = 10$. Let's look at all the elements that are generated by powers of the element $a = 2$:

$$
\begin{aligned}
a &= 2 & a^6 &\equiv 9 \bmod 11 \\
a^2 &= 4 & a^7 &\equiv 7 \bmod 11 \\
a^3 &= 8 & a^8 &\equiv 3 \bmod 11 \\
a^4 &\equiv 5 \bmod 11 & a^9 &\equiv 6 \bmod 11 \\
a^5 &\equiv 10 \bmod 11 & a^{10} &\equiv 1 \bmod 11
\end{aligned}
$$

From the last result it follows that

$$\text{ord}(a) = 10 = |\mathbb{Z}_{11}^*|.$$

This implies that (i) $a = 2$ is a primitive element and (ii) $|\mathbb{Z}_{11}^*|$ is cyclic.

We now want to verify whether the powers of $a = 2$ actually generate all elements of the group $\mathbb{Z}_{11}^*$. Let's look again at all the elements that are generated by powers of two.

i	1	2	3	4	5	6	7	8	9	10
a^i	2	4	8	5	10	9	7	3	6	1

By looking at the bottom row, we see that that the powers 2^i in fact generate all elements of the group $\mathbb{Z}_{11}^*$. We note that the order in which they are generated looks quite arbitrary. This seemingly random relationship between the exponent i and the group elements is the basis for cryptosystems such as the Diffie–Hellman key exchange.

◇

From this example we see that the group $\mathbb{Z}_{11}^*$ has the element 2 as a generator. It is important to stress that the number 2 is not necessarily a generator in other cyclic groups $\mathbb{Z}_n^*$. For instance, in $\mathbb{Z}_7^*$, $\text{ord}(2) = 3$, and the element 2 is thus not a generator in that group.

Cyclic groups have interesting properties. The most important ones for cryptographic applications are given in the following theorems.

Theorem 8.2.2 *For every prime p, $(\mathbb{Z}_p^*, \cdot)$ is an abelian finite cyclic group.*

This theorem states that the multiplicative group of every prime field is cyclic. This has far reaching consequences in cryptography, where these groups are the most popular ones for building discrete logarithm cryptosystems. In order to underline the practical relevance of these somewhat esoteric looking theorem, consider that almost every Web browser has a cryptosystem over $\mathbb{Z}_p^*$ built in.

Theorem 8.2.3
Let G be a finite group. Then for every $a \in G$ it holds that:

1. $a^{|G|} = 1$
2. $ord(a)$ *divides* $|G|$

The first property is a generalization of Fermat's Little Theorem for all cyclic groups. The second property is very useful in practice. It says that in a cyclic group only element orders which divide the group cardinality exist.

Example 8.7. We consider again the group $\mathbb{Z}_{11}^*$ which has a cardinality of $|\mathbb{Z}_{11}^*| = 10$. The only element orders in this group are 1, 2, 5, and 10, since these are the only integers that divide 10. We verify this property by looking at the order of all elements in the group:

$$\begin{array}{ll} \text{ord}(1) = 1 & \text{ord}(6) \ = 10 \\ \text{ord}(2) = 10 & \text{ord}(7) \ = 10 \\ \text{ord}(3) = 5 & \text{ord}(8) \ = 10 \\ \text{ord}(4) = 5 & \text{ord}(9) \ = 5 \\ \text{ord}(5) = 5 & \text{ord}(10) = 2 \end{array}$$

Indeed, only orders that divide 10 occur.

◇

Theorem 8.2.4 *Let G be a finite cyclic group. Then it holds that*
1. *The number of primitive elements of G is $\Phi(|G|)$.*
2. *If $|G|$ is prime, then all elements $a \neq 1 \in G$ are primitive.*

The first property can be verified by the example above. Since $\Phi(10) = (5-1)(2-1) = 4$, the number of primitive elements is four, which are the elements 2, 6, 7 and 8. The second property follows from the previous theorem. If the group cardinality is prime, the only possible element orders are 1 and the cardinality itself. Since only the element 1 can have an order of one, all other elements have order p.

8.2.3 Subgroups

In this section we consider subsets of (cyclic) groups which are groups themselves. Such sets are referred to as *subgroups*. In order to check whether a subset H of a group G is a subgroup, one can verify if all the properties of our group definition in Section 8.2.1 also hold for H. In the case of cyclic groups, there is an easy way to generate subgroups which follows from this theorem:

Theorem 8.2.5 ***Cyclic Subgroup Theorem***
Let $(G, \circ)$ be a cyclic group. Then every element $a \in G$ with $ord(a) = s$ is the primitive element of a cyclic subgroup with s elements.

This theorem tells us that any element of a cyclic group is the generator of a subgroup which in turn is also cyclic.

Example 8.8. Let us verify the above theorem by considering a subgroup of $G = \mathbb{Z}_{11}^*$. In an earlier example we saw that $\mathrm{ord}(3) = 5$, and the powers of 3 generate the subset $H = \{1,3,4,5,9\}$ according to Theorem 8.2.5. We verify now whether this set is actually a group by having a look at its multiplication table:

Table 8.2 Multiplication table for the subgroup $H = \{1,3,4,5,9\}$

× mod 11	**1**	**3**	**4**	**5**	**9**
1	1	3	4	5	9
3	3	9	1	4	5
4	4	1	5	9	3
5	5	4	9	3	1
9	9	5	3	1	4

H is closed under multiplication modulo 11 (Condition 1) since the table only consists of integers which are elements of H. The group operation is obviously associative and commutative since it follows regular multiplication rules (Conditions 2 and 5, respectively). The neutral element is 1 (Condition 3), and for every element $a \in H$ there exists an inverse $a^{-1} \in H$ which is also an element of H (Condition 4). This can be seen from the fact that every row and every column of the table contains the identity element. Thus, H is a subgroup of $\mathbb{Z}_{11}^*$ (depicted in Figure 8.1).

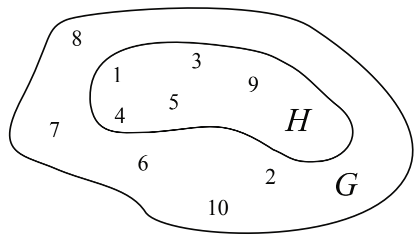

Fig. 8.1 Subgroup H of the cyclic group $G = \mathbb{Z}_{11}^*$

More precisely, it is a subgroup of prime order 5. It should also be noted that 3 is not the only generator of H but also 4, 5 and 9, which follows from Theorem 8.2.4. $\diamond$

An important special case are subgroups of prime order. If this group cardinality is denoted by q, all non-one elements have order q according to Theorem 8.2.4.

From the Cyclic Subgroup Theorem we know that each element $a \in G$ of a group G generates some subgroup H. By using Theorem 8.2.3, the following theorem follows.

Theorem 8.2.6 Lagrange's theorem
Let H be a subgroup of G. Then $|H|$ divides $|G|$.

Let us now consider an application of Lagrange's theorem:

Example 8.9. The cyclic group $\mathbb{Z}_{11}^*$ has cardinality $|\mathbb{Z}_{11}^*| = 10 = 1 \cdot 2 \cdot 5$. Thus, it follows that the subgroups of $\mathbb{Z}_{11}^*$ have cardinalities 1, 2, 5 and 10 since these are all possible divisors of 10. All subgroups H of $\mathbb{Z}_{11}^*$ and their generators α are given below:

subgroup	elements	primitive elements
H_1	$\{1\}$	$\alpha = 1$
H_2	$\{1, 10\}$	$\alpha = 10$
H_3	$\{1, 3, 4, 5, 9\}$	$\alpha = 3, 4, 5, 9$

$\diamond$

The following final theorem of this section fully characterizes the subgroups of a finite cyclic group:

Theorem 8.2.7
Let G be a finite cyclic group of order n and let α be a generator of G. Then for every integer k that divides n there exists exactly one cyclic subgroup H of G of order k. This subgroup is generated by $\alpha^{n/k}$. H consists exactly of the elements $a \in G$ which satisfy the condition $a^k = 1$. There are no other subgroups.

This theorem gives us immediately a construction method for a subgroup from a given cyclic group. The only thing we need is a primitive element and the group cardinality n. One can now simple compute $\alpha^{n/k}$ and obtains a generator of the subgroup with k elements.

Example 8.10. We again consider the cyclic group $\mathbb{Z}_{11}^*$. We saw earlier that $\alpha = 8$ is a primitive element in the group. If we want to have a generator β for the subgroup of order 2, we compute:

$$\beta = \alpha^{n/k} = 8^{10/2} = 8^5 = 32768 \equiv 10 \bmod 11.$$

We can now verify that the element 10 in fact generates the subgroup with two elements: $\beta^1 = 10$, $\beta^2 = 100 \equiv 1 \bmod 11$, $\beta^3 \equiv 10 \bmod 11$, etc.

Remark: Of course, there are smarter ways of computing $8^5 \bmod 11$, e.g., through $8^5 = 8^2\,8^2\,8 \equiv (-2)(-2)8 \equiv 32 \equiv 10 \bmod 11$.
$\diamond$

8.3 The Discrete Logarithm Problem

After the somewhat lengthy introduction to cyclic groups one might wonder how they are related to the rather straightforward DHKE protocol. It turns out that the underlying one-way function of the DHKE, the discrete logarithm problem (DLP), can directly be explained using cyclic groups.

8.3.1 The Discrete Logarithm Problem in Prime Fields

We start with the DLP over $\mathbb{Z}_p^*$, where p is a prime.

Definition 8.3.1 Discrete Logarithm Problem (DLP) in $\mathbb{Z}_p^*$
Given is the finite cyclic group $\mathbb{Z}_p^*$ of order $p-1$ and a primitive element $\alpha \in \mathbb{Z}_p^*$ and another element $\beta \in \mathbb{Z}_p^*$. The DLP is the problem of determining the integer $1 \leq x \leq p-1$ such that:

$$\alpha^x \equiv \beta \mod p$$

Remember from Section 8.2.2 that such an integer x must exist since α is a primitive element and each group element can be expressed as a power of any primitive element. This integer x is called the *discrete logarithm of* β to the base α, and we can formally write:

$$x = \log_\alpha \beta \bmod p.$$

Computing discrete logarithms modulo a prime is a very hard problem if the parameters are sufficiently large. Since exponentiation $\alpha^x \equiv \beta \mod p$ is computationally easy, this forms a one-way function.

Example 8.11. We consider a discrete logarithm in the group $\mathbb{Z}_{47}^*$, in which $\alpha = 5$ is a primitive element. For $\beta = 41$ the discrete logarithm problem is: Find the positive integer x such that

$$5^x \equiv 41 \bmod 47$$

Even for such small numbers, determining x is not entirely straightforward. By using a brute-force attack, i.e., systematically trying all possible values for x, we obtain the solution $x = 15$.
◇

In practice, it is often desirable to have a DLP in groups with prime cardinality in order to prevent the Pohlig–Hellman attack (cf. Section 8.3.3). Since groups $\mathbb{Z}_p^*$ have cardinality $p-1$, which is obviously not prime, one often uses DLPs in subgroups of $\mathbb{Z}_p^*$ with prime order, rather than using the group $\mathbb{Z}_p^*$ itself. We illustrate this with an example.

Example 8.12. We consider the group $\mathbb{Z}_{47}^*$ which has order 46. The subgroups in $\mathbb{Z}_{47}^*$ have thus a cardinality of 23, 2 and 1. $\alpha = 2$ is an element in the subgroup with 23 elements, and since 23 is a prime, α is a primitive element in the subgroup. A possible discrete logarithm problem is given for $\beta = 36$ (which is also in the subgroup): Find the positive integer x, $1 \leq x \leq 23$, such that

$$2^x \equiv 36 \bmod 47$$

By using a brute-force attack, we obtain a solution for $x = 17$.
◇

8.3.2 The Generalized Discrete Logarithm Problem

The feature that makes the DLP particularly useful in cryptography is that it is not restricted to the multiplicative group $\mathbb{Z}_p^*$, p prime, but can be defined over any cyclic groups. This is called the *generalized discrete logarithm problem (GDLP)* and can be stated as follows.

Definition 8.3.2 Generalized Discrete Logarithm Problem
Given is a finite cyclic group G with the group operation $\circ$ and cardinality n. We consider a primitive element $\alpha \in G$ and another element $\beta \in G$. The discrete logarithm problem is finding the integer x, where $1 \leq x \leq n$, such that:

$$\beta = \underbrace{\alpha \circ \alpha \circ \ldots \circ \alpha}_{x \;\; times} = \alpha^x$$

As in the case of the DLP in $\mathbb{Z}_p^*$, such an integer x must exist since α is a primitive element, and thus each element of the group G can be generated by repeated application of the group operation on α.

It is important to realize that there are cyclic groups in which the DLP is *not* difficult. Such groups cannot be used for a public-key cryptosystem since the DLP is not a one-way function. Consider the following example.

Example 8.13. This time we consider the additive group of integers modulo a prime. For instance, if we choose the prime $p = 11$, $G = (\mathbb{Z}_{11}, +)$ is a finite cyclic group with the primitive element $\alpha = 2$. Here is how α generates the group:

i	1	2	3	4	5	6	7	8	9	10	11
$i\alpha$	2	4	6	8	10	1	3	5	7	9	0

We try now to solve the DLP for the element $\beta = 3$, i.e., we have to compute the integer $1 \leq x \leq 11$ such that

$$x \cdot 2 = \underbrace{2 + 2 + \ldots + 2}_{x \text{ times}} \equiv 3 \bmod 11$$

Here is how an "attack" against this DLP works. Even though the group operation is addition, we can express the relationship between α, β and the discrete logarithm x in terms of *multiplication*:

$$x \cdot 2 \equiv 3 \bmod 11.$$

In order to solve for x, we simply have to invert the primitive element α:

$$x \equiv 2^{-1} 3 \bmod 11$$

Using, e.g., the extended Euclidean algorithm, we can compute $2^{-1} \equiv 6 \bmod 11$ from which the discrete logarithm follows as:

$$x \equiv 2^{-1}3 \equiv 7 \bmod 11$$

The discrete logarithm can be verified by looking at the small table provided above.

We can generalize the above trick to any group $(\mathbb{Z}_n, +)$ for arbitrary n and elements $\alpha, \beta \in \mathbb{Z}_n$. Hence, we conclude that the generalized DLP is computationally easy over $\mathbb{Z}_n$. The reason why the DLP can be solved here easily is that we have mathematical operations which are not in the additive group, namely multiplication and inversion.

◇

After this counterexample we now list discrete logarithm problems that have been proposed for use in cryptography:

1. The multiplicative group of the prime field $\mathbb{Z}_p$ or a subgroup of it. For instance, the classical DHKE uses this group, but also Elgamal encryption or the Digital Signature Algorithm (DSA). These are the oldest and most widely used types of discrete logarithm systems.
2. The cyclic group formed by an elliptic curve. Elliptic curve cryptosystems are introduced in Chapter 9. They have become popular in practice over the last decade.
3. The multiplicative group of a Galois field $GF(2^m)$ or a subgroup of it. These groups can be used completely analogous to multiplicative groups of prime fields, and schemes such as the DHKE can be realized with them. They are not as popular in practice because the attacks against them are somewhat more powerful than those against the DLP in $\mathbb{Z}_p$. Hence DLPs over $GF(2^m)$ require somewhat higher bit lengths for providing the same level of security than those over $\mathbb{Z}_p$.
4. Hyperelliptic curves or algebraic varieties, which can be viewed as generalization as elliptic curves. They are currently rarely used in practice, but in particular hyperelliptic curves have some advantages such as short operand lengths.

There have been proposals for other DLP-based cryptosystems over the years, but none of them have really been of interest in practice. Often it was found that the underlying DL problem was not difficult enough.

8.3.3 Attacks Against the Discrete Logarithm Problem

This section introduce methods for solving discrete logarithm problems. Readers only interested in the constructive use of DL schemes can skip this section.

As we have seen, the security of many asymmetric primitives is based on the difficulty of computing the DLP in cyclic groups, i.e., to compute x for a given α and β in G such that

$$\beta = \underbrace{\alpha \circ \alpha \circ \ldots \circ \alpha}_{x \text{ times}} = \alpha^x$$

holds. We still do not know the exact difficulty of computing the discrete logarithm x in any given actual group. What we mean by this is that even though some attacks are known, one does not know whether there are any better, more powerful algorithms for solving the DLP. This situation is similar to the hardness of integer factorization, which is the one-way function underlying RSA. Nobody really knows what the *best possible* factorization method is. For the DLP some interesting general results exist regarding its computational hardness. This section gives a brief overview of algorithms for computing discrete logarithms which can be classified into *generic algorithms* and *nongeneric algorithms* and which will be discussed in a little more detail.

Generic Algorithms

Generic DL algorithms are methods which only use the group operation and no other algebraic structure of the group under consideration. Since they do not exploit special properties of the group, they work in any cyclic group. Generic algorithms for the discrete logarithm problem can be subdivided into two classes. The first class encompasses algorithms whose running time depends on the size of the cyclic group, like the *brute-force search*, the *baby-step giant-step* algorithm and *Pollard's rho* method. The second class are algorithms whose running time depends on the size of the prime factors of the group order, like the *Pohlig–Hellman* algorithm.

Brute-Force Search

A brute-force search is the most naïve and computationally costly way for computing the discrete logarithm $\log_\alpha \beta$. We simply compute powers of the generator α successively until the result equals β:

$$\begin{aligned} \alpha^1 &\stackrel{?}{=} \beta \\ \alpha^2 &\stackrel{?}{=} \beta \\ &\vdots \\ \alpha^x &\stackrel{?}{=} \beta \end{aligned}$$

For a random logarithm x, we do expect to find the correct solution after checking half of all possible x. This gives us a complexity of $\mathcal{O}(|G|)$ steps[2], where $|G|$ is the cardinality of the group.

To avoid brute-force attacks on DL-based cryptosystems in practice, the cardinality $|G|$ of the underlying group must thus be sufficiently large. For instance, in the case of the group $\mathbb{Z}_p^*$, p prime, which is the basis for the DHKE, $(p-1)/2$ tests are required on average to compute a discrete logarithm. Thus, $|G| = p-1$ should be at least in the order of 2^{80} to make a brute-force search infeasible using today's computer technology. Of course, this consideration only holds if a brute-force attack is the only feasible attack which is never the case. There exist much more powerful algorithms to solve discrete logarithms as we will see below.

Shanks' Baby-Step Giant-Step Method

Shanks' algorithm is a time-memory tradeoff method, which reduces the time of a brute-force search at the cost of extra storage. The idea is based on rewriting the discrete logarithm $x = \log_\alpha \beta$ in a two-digit representation:

$$x = x_g\, m + x_b \qquad \text{for } 0 \le x_g, x_b < m. \tag{8.1}$$

The value m is chosen to be of the size of the square root of the group order, i.e., $m = \lceil \sqrt{|G|} \rceil$. We can now write the discrete logarithm as $\beta = \alpha^x = \alpha^{x_g m + x_b}$ which leads to

$$\beta \cdot (\alpha^{-m})^{x_g} = \alpha^{x_b}. \tag{8.2}$$

The idea of the algorithm is to find a solution (x_g, x_b) for Eq. (8.2), from which the discrete logarithm then follows directly according to Eq. (8.1). The core idea for the algorithm is that Eq. (8.2) can be solved by searching for x_g and x_b separatedly, i.e., using a divide-and-conquer approach. In the first phase of the algorithm we compute and store all values α^{x_b}, where $0 \le x_b < m$. This is the *baby-step phase* that requires $m \approx \sqrt{|G|}$ steps (group operations) and needs to store $m \approx \sqrt{|G|}$ group elements.

In the *giant-step phase*, the algorithm checks for all x_g in the range $0 \le x_g < m$ whether the following condition is fulfilled:

$$\beta \cdot (\alpha^{-m})^{x_g} \stackrel{?}{=} \alpha^{x_b}$$

for some stored entry α^{x_b} that was computed during the baby-step phase. In case of a match, i.e., $\beta \cdot (\alpha^{-m})^{x_{g,0}} = \alpha^{x_{b,0}}$ for some pair $(x_{g,0}, x_{b,0})$, the discrete logarithm is given by

$$x = x_{g,0}\, m + x_{b,0}.$$

The baby-step giant-step method requires $\mathcal{O}(\sqrt{|G|})$ computational steps and an equal amount of memory. In a group of order 2^{80}, an attacker would only need

[2] We use the popular "big-Oh" notation here. A complexity function $f(x)$ has big-Oh notation $\mathcal{O}(g(x))$ if $f(x) \le c \cdot g(x)$ for some constant c and for input values x greater than some value x_0.

approximately $2^{40} = \sqrt{2^{80}}$ computations and memory locations, which is easily achievable with today's PCs and hard disks. Thus, in order to obtain an attack complexity of 2^{80}, a group must have a cardinality of at least $|G| \geq 2^{160}$. In the case of groups $G = \mathbb{Z}_p^*$, the prime p should thus have at least a length of 160 bit. However, as we see below, there are more powerful attacks against DLPs in $\mathbb{Z}_p^*$ which forces even larger bit lengths of p.

Pollard's Rho Method

Pollard's rho method has the same expected run time $\mathcal{O}(\sqrt{|G|})$ as the baby-step giant-step algorithm but only negligible space requirements. The method is a probabilistic algorithm which is based on the birthday paradox (cf. Section 11.2.3). We will only sketch the algorithm here. The basic idea is to pseudorandomly generate group elements of the form $\alpha^i \cdot \beta^j$. For every element we keep track of the values i and j. We continue until we obtain a collision of two elements, i.e., until we have:

$$\alpha^{i_1} \cdot \beta^{j_1} = \alpha^{i_2} \cdot \beta^{j_2}. \tag{8.3}$$

If we substitute $\beta = \alpha^x$ and compare the exponents on both sides of the equation, the collision leads to the relation $i_1 + x j_1 \equiv i_2 + x j_2 \bmod |G|$. (Note that we are in a cyclic group with $|G|$ elements and have to take the exponent modulo $|G|$.) From here the discrete logarithm can easily computed as:

$$x \equiv \frac{i_2 - i_1}{j_1 - j_2} \bmod |G|$$

An important detail, which we omit here, is the exact way to find the collision (8.3). In any case, the pseudorandom generation of the elements is a random walk through the group. This can be illustrated by the shape of the Greek letter rho, hence the name of this attack.

Pollard's rho method is of great practical importance because it is currently the best known algorithm for computing discrete logarithms in elliptic curve groups. Since the method has an attack complexity of $\mathcal{O}(\sqrt{|G|})$ computations, elliptic curve groups should have a size of at least 2^{160}. In fact, elliptic curve cryptosystems with 160-bit operands are very popular in practice.

There are still much more powerful attacks known for the DLP in $\mathbb{Z}_p^*$, as we will see below.

Pohlig–Hellman Algorithm

The Pohlig–Hellman method is an algorithm which is based on the Chinese Remainder Theorem (not introduced in this book); it exploits a possible factorization of the order of a group. It is typically not used by itself but in conjunction with any of the other DLP attack algorithms in this section. Let

$$|G| = p_1^{e_1} \cdot p_2^{e_2} \cdot \ldots \cdot p_l^{e_l}$$

be the prime factorization of the group order $|G|$. Again, we attempt to compute a discrete logarithm $x = \log_\alpha \beta$ in G. This is also a divide-and-conquer algorithm. The basic idea is that rather than dealing with the large group G, one computes smaller discrete logarithms $x_i \equiv x \bmod p_i^{e_i}$ in the subgroups of order $p_i^{e_i}$. The desired discrete logarithm x can then be computed from all x_i, $i = 1, \ldots, l$, by using the Chinese Remainder Theorem. Each individual small DLP x_i can be computed using Pollard's rho method or the baby-step giant-step algorithm.

The run time of the algorithm clearly depends on the prime factors of the group order. To prevent the attack, the group order must have its largest prime factor in the range of 2^{160}. An important practical consequence of the Pohlig–Hellman algorithm is that one needs to know the prime factorization of the group order. Especially in the case of elliptic curve cryptosystems, computing the order of the cyclic group is not always easy.

Nongeneric Algorithms: The Index-Calculus Method

All algorithms introduced so far are completely independent of the group being attacked, i.e., they work for discrete logarithms defined over any cyclic group. Nongeneric algorithms efficiently exploit special properties, i.e., the inherent structure, of certain groups. This can lead to much more powerful DL algorithms. The most important nongeneric algorithm is the index-calculus method.

Both the baby-step giant-step algorithm and Pollard's rho method have a run time which is exponential in the bit length of the group order, namely of about $2^{n/2}$ steps, where n is the bit length of $|G|$. This greatly favors the crypto designer over the cryptanalyst. For instance, increasing the group order by a mere 20 bit increases the attack effort by a factor of $1024 = 2^{10}$. This is a major reason why elliptic curves have better long-term security behavior than RSA or cryptosystems based on the DLP in $\mathbb{Z}_p^*$. The question is whether there are more powerful algorithms for DLPs in certain specific groups. The answer is yes.

The index-calculus method is a very efficient algorithm for computing discrete logarithms in the cyclic groups $\mathbb{Z}_p^*$ and $GF(2^m)^*$. It has a subexponential running time. We will not introduce the method here but just provide a very brief description. The index-calculus method depends on the property that a significant fraction of elements of G can be efficiently expressed as products of elements of a small subset of G. For the group $\mathbb{Z}_p^*$ this means that many elements should be expressable as a product of small primes. This property is satisfied by the groups $\mathbb{Z}_p^*$ and $GF(2^m)^*$. However, one has not found a way to do the same for elliptic curve groups. The index calculus is so powerful that in order to provide a security of 80 bit, i.e., an attacker has to perform 2^{80} steps, the prime p of a DLP in $\mathbb{Z}_p^*$ should be at least 1024 bit long. Table 8.3 gives an overview on the DLP records achieved since the early 1990s. The index-calculus method is somewhat more powerful for solving the DLP in $GF(2^m)^*$. Hence the bit lengths have to be chosen somewhat longer to achieve the same level of security. For that reason, DLP schems in $GF(2^m)^*$ are not as widely used in practice.

Table 8.3 Summary of records for computing discrete logarithms in $\mathbb{Z}_p^*$

Decimal digits	Bit length	Date
58	193	1991
65	216	1996
85	282	1998
100	332	1999
120	399	2001
135	448	2006
160	532	2007