Chapter 4

The Discrete Logarithm with Cryptographic Applications

Computing discrete logarithms is believed—like factoring large integers—to be a hard problem. This serves as basis of many cryptographic procedures.

A useful aspect of most of these procedures is that they rely only on the group property of the multiplicative groups of the residue class rings of integers. Therefore they often have an immediate translation to other groups such as elliptic curves. Should discrete logarithms for residue class rings happen to be efficiently computable there remains a chance that the procedures remain secure for other groups.

4.1 The Discrete Logarithm

Let G be a group (multiplicatively written) and $a \in G$ be an element of order s (maybe ∞). Then the **exponential function** to base a in G

$$\exp_a \colon \mathbb{Z} \longrightarrow G, \quad x \mapsto a^x,$$

is a group homomorphism (since $a^{x+y} = a^x a^y$) and has period s (since $a^{x+s} = a^x a^s = a^x$ if $s < \infty$). By the homomorphy theorem the induced homomorphism h

$$\begin{tikzcd}
\mathbb{Z} \arrow[r, "\exp_a"] \arrow[d] & \langle a \rangle \subseteq G \arrow[dl, shift left, "\log_a"] \\
\mathbb{Z}/s\mathbb{Z} \arrow[ur, shift left, "h"] &
\end{tikzcd}$$

is an isomorphism, hence has an inverse map

$$\log_a \colon \langle a \rangle \longrightarrow \mathbb{Z}/s\mathbb{Z}$$

defined on the cyclic subgroup $\langle a \rangle \subseteq G$, the **discrete logarithm** to base a that is an isomorphism of groups. [The case $s = \infty$ fits into this scenario for $s\mathbb{Z} = 0$ and $\mathbb{Z}/s\mathbb{Z} = \mathbb{Z}$.]

We apply this to the multiplicative group $\mathbb{M}_n$: For an integer $a \in \mathbb{Z}$ with $\gcd(a, n) = 1$ the exponential function $\bmod n$ to base a,

$$\exp_a \colon \mathbb{Z} \longrightarrow \mathbb{M}_n, \quad x \mapsto a^x \bmod n,$$

has period $s = \operatorname{ord} a | \lambda(n) | \varphi(n)$. The inverse function

$$\log_a \colon \langle a \rangle \longrightarrow \mathbb{Z}/s\mathbb{Z}$$

is the discrete logarithm $\bmod n$ to base a.

We know of no efficient algorithm that computes the discrete logarithm $\log_a$ for large $s = \operatorname{ord} a$, or to invert the exponential function—not even a probabilistic one.

> **Informal definition:** A function $f : M \longrightarrow N$ is called **one-way function** if for "almost all" images $y \in N$ there is no efficient way to compute a pre-image $x \in M$ with $f(x) = y$.
>
> This definition can be given a mathematically precise (although not completely satisfying) formulation in terms of complexity theory, see Appendix B.

Discrete logarithm assumption: The exponential function $\exp_a \bmod n$ is a one-way function for "almost all" bases a.

Note that this is an unproven conjecture.

The most important special case is a prime module $p \geq 3$, and a primitive element $a \in [2, \ldots, p-2]$, i. e., $\operatorname{ord} a = p-1$.

$$\begin{tikzcd}
\mathbb{Z} \arrow[r, "\exp_a"] \arrow[d] & \mathbb{F}_p^\times \arrow[dl, shift left, "\log_a"] \\
\mathbb{Z}/(p-1)\mathbb{Z} \arrow[ur, shift left, "\text{bij}"] &
\end{tikzcd}$$

To make the computation of discrete logarithms hard in practice we have to choose a prime module p of about the same size as an RSA module. Thus according to the state of the art 1024-bit primes are completely obsolete, 2048-bit primes are safe for short-time applications only.

The book by SHPARLINSKI (see the references for these lecture notes) contains some lower bounds for the complexity of discrete logarithm computations in various computational models.

4.2 Diffie-Hellman Key Exchange

We treat some exemplary applications that provide astonishingly elegant solutions for seemingly unsolvable problems under the discrete logarithm assumption.

Imagine A (Alice) and B (Bob) want to exchange a key for a symmetric cipher. In 1976 Diffie and Hellman proposed the following protocol whose security relies on the dicrete logarithm assumption:

1. A and B (publicly) agree on a prime p and a primitive element a mod p.
2. A generates a random integer x, computes $u = a^x \bmod p$, and sends u to B.
3. B generates a random integer y, computes $v = a^y \bmod p$, and sends v to A.
4. A computes $k = v^x \bmod p$, and B computes $k = u^y \bmod p$.

Now A and B share a secret k that may be used as key. The fact that A and B compute the same key k lies in the equation

$$v^x \equiv a^{xy} \equiv u^y \pmod{p}.$$

An eavesdropper can intercept the values p, a, u, and v. But this doesn't enable her to efficiently compute k, or x, or y.

This protocol realizes a kind of hybrid encryption. A difference with a "proper" asymmetric cipher concerns the need for synchronization between A and B, preventing spontaneous messages (for example by e-mail that follows an asynchroneous protocol).

An attacker who is able to efficiently compute discrete logarithms is also able to efficiently break the Diffie-Hellman protocol. It is unknown whther the converse also holds.

The British Secret Service CESC knew the procedure already in 1974 but of course kept it secret.

Here is a mathematical model for a somewhat more abstract protocol:

1. A and B (publicly) agree on a set X, an element $a \in X$, and a commutative subsemigroup $H \subseteq \mathrm{Map}(X, X)$.
2. A chooses a random map $\varphi_A \in H$, computes $u = \varphi_A(a)$, and sends u to B.
3. B chooses a random map $\varphi_B \in H$, computes $v = \varphi_B(a)$, and sends v to A.
4. A computes $\varphi_A(v)$, and B computes $\varphi_B(u)$.

Then A and B share the secret value

$$k = \varphi_A(v) = \varphi_A(\varphi_B(a)) = \varphi_B(\varphi_A(a)) = \varphi_B(u)$$

and may use it as key for their secret communication—at least if an attacker has no method to derive φ_A, φ_B, or k from the entities X, a, u, and v she knows or intercepts.

For the adaption of this protocol to elliptic curves an even more abstract scenario is useful that is visualized by a commutative diagram as follows:

$$\begin{tikzcd} X \arrow[r, "\varphi_A"] \arrow[d, "\varphi_B"'] & Y \arrow[d, "\tilde{\varphi}_B"] \\ Y \arrow[r, "\tilde{\varphi}_A"'] & Z \end{tikzcd}$$

4.3 The Man in the Middle

In this section we consider a communication protocol with asymmetric encryption, and note that the same attack works against the DIFFIE-HELLMAN key exchange. The basic problem is that an attacker can plant his own key into the procedure. In some more detail:

Suppose A = Alice and B = Bob want to exchange messages. First A sends her public key E_A to B, and B sends his public key E_B to A.

The attacker E = Eve who only listens cannot use these public data for eavesdropping. However the attacker M = Mallory, the "man in the middle" who actively forges messages, intercepts the key exchange, and each time replaces the intercepted public key by his own key E_M. From now on M is able to monitoring and even counterfeiting the complete communication of A and B. Figure 4.1 illustrates the attack.

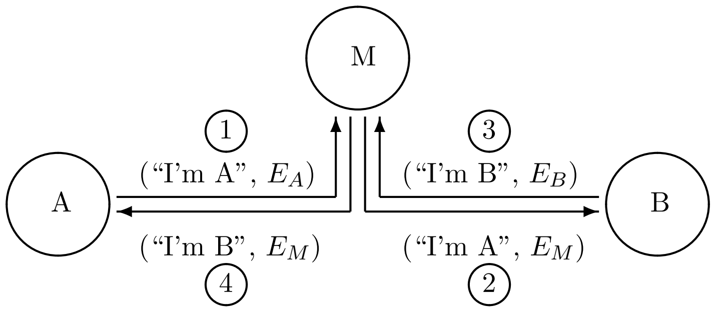

Figure 4.1: The man in the middle

There are different ways to prevent this attack. But all of them make asymmetric encryption more complex. The usual way is the use of certificates: The public keys of all participants of a communication network get a digital signature by a "trusted third party".

Definition. A certificate is a public key signed by a trusted third party.

Mnemonic. *A key exchange can be secure from the man in the middle only if the partners are mutually authenticated.*

Exercise. What information in the DIFFIE-HELLMAN protocol is suited to be used in a certificate?

4.4 Secret Communication without Key Exchange

Even without exchanging keys in advance a confidential conversation is possible. (Note that this protocol also is not secure from the man in the middle.)

An analogy from veryday life illustrates the idea:

- Alice puts her message in a box and locks it with a padlock whose key is hers and not available to anyone else.
- Of course Bob is unable to open the box. Instead he locks it with another padlock of his own. He returns the doubly locked box to Alice.
- Alice removes her padlock and returns the box that is locked with Bob's padlock only.
- Bob removes his padlock, opens the box, and reads the message.

This cryptographic protocol is called the MASSEY-OMURA scheme or SHAMIR's no-key algorithm. It may be implemented with the discrete exponential function. Its security relies on the discrete logarithm assumption:

The procedure uses a public large prime number p. Alice and Bob each choose a pair of exponents d and e with $ed \equiv 1 \pmod{p-1}$, hence $a^{de} \equiv a \pmod{p}$ for all integers $a \in \mathbb{Z}$. Each one keeps *both* of their exponents secret.

Then Alice sends a message a to Bob according to the following protocol:

$$\begin{tikzcd}
\textcircled{\textbf{A}} & & \textcircled{\textbf{B}} \\
\boxed{a^{e_A} \bmod p} & \arrow[r] & {} \\
 & {} & \boxed{a^{e_A e_B} \bmod p} \arrow[l] \\
\boxed{a^{e_A e_B d_A} = a^{e_B} \bmod p} & \arrow[r] & {} \\
 & & \boxed{a^{e_B d_B} = a \bmod p}
\end{tikzcd}$$

An attacker who is able to compute discrete logarithms is also able to compute the exponent e_B from the intercepted ciphertexts $a^{e_A} \bmod p$ and $a^{e_A e_B} \bmod p$. From this she computes d_B by congruence division and solves for a.

This is the only known attack. Hence the protocol is secure from Eve as long as the discrete logarithm assumption holds. To be secure from Mallory the protocol must be supplemented by an authentication phase.

4.5 ElGamal Cipher—Idea

The ElGamal cipher is an asymmetric cipher—or more precisely a hybrid cipher—that also relies on the complexity of the discrete logarithm.

The basic public parameters are a prime p and an element $g \in [2 \dots p-2]$. The order of g in $\mathbb{F}_p^\times$ should be high, preferably g should be a primitive element $\bmod p$.

> p and g may be shared by all participants but also may be individually chosen.

Each participant chooses a random integer

$$d \in [2 \dots p-2]$$

as private key, und computes

$$e = g^d \bmod p$$

as corresponding public key. Computing d from e is computing a discrete logarithm, hence presumably hard.

The definitioon of the cipher needs one more idea: How to transform a message a in such a way that it can be reconstructed only with knowledge of d?

> The naive idea of sending $e^a = g^{da} \bmod p$ is useless—knowing d doesn't help with decrypting a. Also sending $r = g^a \bmod p$ is useless—the receiver can compute $r^d = e^a \bmod p$ but not a.

The idea is to first generate a message key to be used with a hybrid procedure:

- Alice chooses a random $k \in [2 \dots p-2]$. As key she will use $K = e^k \bmod p$ where e is the Bob's public key, thus Alice can compute K.
- To share the key K with Bob Alice sends the *key information* $r = g^k \bmod p$ together with the encrypted message.
- Bob computes $r^d = g^{kd} = e^k = K \bmod p$ using his private key d.

As symmetric component of the hybrid encryption the shift cipher in $\mathbb{F}_p^\times$ is used with K as one-time key. So Alice has to generate a new key K for each plaintext block and to send the corresponding key information, doubling the length of the message.

Thus, after generating the key K and the key information r:

- the formula for encryption is $c = Ka \bmod p$,
- and the message to be sent is (c, r).

Bob computes the key K from r, and then decrypts

- $a = K^{-1}c \bmod p$ by congruence division.

4.6 Computing Discrete Logarithms

The classical algorithm for computing discrete logarithms is the **index calculus** by Adleman—"index" was Gauss' denotation of the discrete logarithm.

Let $p \geq 3$ be a prime and a be a primitive element for p.

The naive algorithm for computing $\log_a y$ for $y \in \mathbb{F}_p^\times$ is punished by exponentially growing costs, as usual. It computes $a, a^2, a^3, \ldots$ in order until x with $a^x = y$ is found. In the mean it needs $\frac{p}{2} - 1$ trials, in the worst case, $p - 2$ (omitting the trivial value $y = 1$).

Preliminary Steps

For given p and a we need to execute this precomputation only once.

Let $p_1 = 2$, $p_2 = 3$, ..., p_k be the first k primes.

If we randomly choose an exponent r, then it could happen that $a^r \bmod p$—considered as integer $\in \mathbb{Z}$—has only prime divisors in $\{p_1, \ldots, p_k\}$. After h strokes of luck we have a system of h equations:

$$\begin{aligned} a^{r_1} \bmod p &= p_1^{\alpha_{11}} \cdots p_k^{\alpha_{1k}}, \\ &\vdots \\ a^{r_h} \bmod p &= p_1^{\alpha_{h1}} \cdots p_k^{\alpha_{hk}}. \end{aligned}$$

in $\mathbb{Z}$ and a forteriori in $\mathbb{F}_p$. Taking logarithms results in a system of linear equations over the ring $\mathbb{Z}/(p-1)\mathbb{Z}$ for the k unknowns $\log_a p_i$:

$$\begin{aligned} r_1 &= \alpha_{11} \cdot \log_a p_1 + \cdots + \alpha_{1k} \cdot \log_a p_k, \\ &\vdots \\ r_h &= \alpha_{h1} \cdot \log_a p_1 + \cdots + \alpha_{hk} \cdot \log_a p_k. \end{aligned}$$

From Chapter I we know efficient algorithms for solving it. If h is sufficiently large—at least $h \geq k$—, then we can compute $\log_a p_1$, ..., $\log_a p_k$.

The random search for "strokes of luck" makes the precomputation probabilistic.

Computation

Let $y \in \mathbb{F}_p^\times$ be given. We want to compute $\log_a y$.

For a randomly chosen exponent s it could happen that

$$y \cdot a^s \bmod p = p_1^{\beta_1} \cdots p_k^{\beta_k}$$

in $\mathbb{Z}$. Then we easily compute

$$\log_a y = \beta_1 \cdot \log_a p_1 + \cdots + \beta_k \cdot \log_a p_k - s\,.$$

This observation reduces the computation of the discrete logarithm of any element to the computation for the elements of the **factor basis** $(p_1, \ldots, p_k)$. This reduction is also probabilistic.

Variants

The presented approach has several variants that result in different running times. They vary in the choice of the factor basis—that might be adapted to y and need not consist of the first primes without gap—and in the strategy of choosing the exponents r and s.

The fastest known variant uses a number field sieve such as applied for factoring large integers and has expenses of

$$\approx e^{c \cdot \sqrt[3]{\log p \cdot (\log \log p)^2}},$$

the same order of magnitude as is needed for factoring an integer of the same size. By the state of the art 1024-bit primes are insecure, and 2048-bit primes secure only for short-term cryptographic applications.

As an oddity we mention that the "Secure NFS" protocol deployed by SUN used a 192-bit prime (58 decimal places) even in the 1990s.

Special Primes

There are reasons to choose p as a special prime of the form $p = 2p' + 1$ with p' prime:

1. Some algorithms are very fast if $p - 1$ has only small prime divisors. This argument is no longer considered as solid since the advantage of special algorithms over the current versions of the number field sieve is only small. Moreover the probability of choosing such a "bad" prime by accident is extremely small.

2. Finding a primitive element is easy, see Section A.9 in the appendix.