Hard problems for crypto

More notes for *Serious Cryptography* chapter 9

Jeffrey Goldberg
`jeffrey@goldmark.org`

August 26, 2024

We choose to go to the [factoring and the discrete logarithm problems] and do the other things, not because they are easy, but because they are hard.

(with apologies to) JFK

Problems

(Presumed) NP math problems that are turned into cryptography

- Factoring
- Discrete logarithm (over integer groups)
- Discrete logarithm (over elliptic curves)

Factoring

Factoring

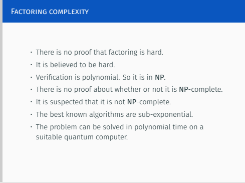
if $N = pq$ and p and q are large prime numbers (with a few additional restrictions) then computing p and q from N is believed to be hard. Verifying a candidate solution is easy.

A 67 digit example

- Factoring 1 025 046 047 436 407 593 990 706 183 629 376 939 000 352 221 561 805 965 005 888 683 119 takes about 9 seconds on my computer.
- Verifying that that the factors, 868 112 830 765 445 632 873 217 196 988 651 and 1 180 775 137 873 020 977 354 442 912 336 269, are correct takes about 5 microseconds.

Factoring complexity

- There is no proof that factoring is hard.
- It is believed to be hard.
- Verification is polynomial. So it is in NP.
- There is no proof about whether or not it is NP-complete.
- It is suspected that it is not NP-complete.
- The best known algorithms are sub-exponential.
- The problem can be solved in polynomial time on a suitable quantum computer.



Factoring complexity

- There is no proof that factoring is hard.
- It is believed to be hard.
- Verification is polynomial. So it is in **NP**.
- There is no proof about whether or not it is **NP**-complete.
- It is suspected that it is not **NP**-complete.
- The best known algorithms are sub-exponential.
- The problem can be solved in polynomial time on a suitable quantum computer.

That is a worrying amount of uncertainty.

The discrete logarithm problem

Why so much math?

Question

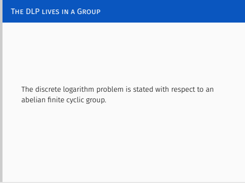
Why are we going to go into abstract mathematics learning about the discrete logarithm problem (DLP) when it is so much easier to understand the factoring problem?

Answer

Once we understand the DLP, it is easy to see how it can be used for public key encryption, while understanding how to turn the factoring problem into useful cryptography takes all the math of understanding the DLP and more.

The DLP lives in a Group

The discrete logarithm problem is stated with respect to an abelian finite cyclic group.


The DLP lives in a Group

The discrete logarithm problem is stated with respect to an abelian finite cyclic group.

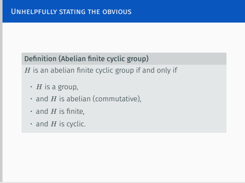
1. Sometimes, you will hear people talk about finite cyclic *fields*, but all fields are groups so don't worry about people saying "field" unless they are specifically drawing attention to it.

Unhelpfully stating the obvious

Definition (Abelian finite cyclic group)

H is an abelian finite cyclic group if and only if

- H is a group,
- and H is abelian (commutative),
- and H is finite,
- and H is cyclic.



UNHELPFULLY STATING THE OBVIOUS

Definition (Abelian finite cyclic group)
H is an abelian finite cyclic group if and only if

- H is a group,
- and H is abelian (commutative),
- and H is finite,
- and H is cyclic.

1. All of the groups we are talking about are abelian (commutative). If “$\circ$’ is the group opperation than the group is communative (abelian) iff $a \circ b = b \circ a$ for all a and b in the group.
2. So far no one has followed my suggestion that non-commutative groups be called “cainian”.

Integers +

Addition over the integers forms an abelian group because

- Addition is **closed**. Adding two integers gets you another integer. If a and b are both integers then $a + b$ is an integer.
- Addition is **commutative**. If a and b are elements of the group then $a + b = b + a$ is an integer.
- There is an **identity element** (0) such that if a is any integer, then $a + 0 = a$.
- Every integer has an **inverse**. If a is an integer then there is some integer which we will call ‘$-a$’ such that $a + -a = 0$ (where 0 is the identity element).
- The operation (addition) is **associative**. If a, b, c are integers then $a + (b + c) = (a + b) + c$.

Neither finite nor cyclic

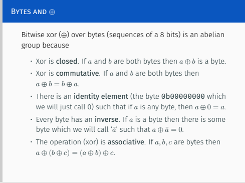
Addition over the integers is an abelian group but it is not a finite group, and it is not a cyclic group.

- The number of integers is not finite.
- This group is not cyclic (we will get to what that means)

Bytes and $\oplus$

Bitwise xor ($\oplus$) over bytes (sequences of a 8 bits) is an abelian group because

- Xor is **closed**. If a and b are both bytes then $a \oplus b$ is a byte.
- Xor is **commutative**. If a and b are both bytes then $a \oplus b = b \oplus a$.
- There is an **identity element** (the byte `0b00000000` which we will just call 0) such that if a is any byte, then $a \oplus 0 = a$.
- Every byte has an **inverse**. If a is a byte then there is some byte which we will call '$\bar{a}$' such that $a \oplus \bar{a} = 0$.
- The operation (xor) is **associative**. If a, b, c are bytes then $a \oplus (b \oplus c) = (a \oplus b) \oplus c$.



Bytes and ⊕

Bitwise xor (⊕) over bytes (sequences of a 8 bits) is an abelian group because

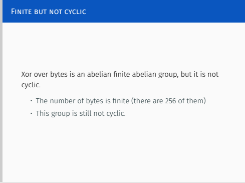
- Xor is **closed**. If a and b are both bytes then $a \oplus b$ is a byte.
- Xor is **commutative**. If a and b are both bytes then $a \oplus b = b \oplus a$.
- There is an **identity element** (the byte `0b00000000` which we will just call 0) such that if a is any byte, then $a \oplus 0 = a$.
- Every byte has an **inverse**. If a is a byte then there is some byte which we will call '$\bar{a}$' such that $a \oplus \bar{a} = 0$.
- The operation (xor) is **associative**. If a, b, c are bytes then $a \oplus (b \oplus c) = (a \oplus b) \oplus c$.

1. In the case of xor, each element is its own inverse. This makes xor magical and dangerous.

Finite but not cyclic

Xor over bytes is an abelian finite abelian group, but it is not cyclic.

- The number of bytes is finite (there are 256 of them)
- This group is still not cyclic.


Finite but not cyclic

Xor over bytes is an abelian finite abelian group, but it is not cyclic.

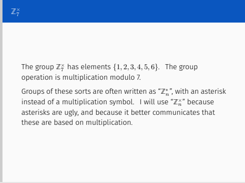
- The number of bytes is finite (there are 256 of them)
- This group is still not cyclic.

1. xor is cyclic on a per bit basis, but it is not for multi-bit sequences.

$\mathbb{Z}_7^\times$

The group $\mathbb{Z}_7^\times$ has elements $\{1, 2, 3, 4, 5, 6\}$. The group operation is multiplication modulo 7.

Groups of these sorts are often written as “$\mathbb{Z}_n^*$”, with an asterisk instead of a multiplication symbol. I will use “$\mathbb{Z}_n^\times$” because asterisks are ugly, and because it better communicates that these are based on multiplication.


$\mathbb{Z}_7^\times$

The group $\mathbb{Z}_7^\times$ has elements $\{1, 2, 3, 4, 5, 6\}$. The group operation is multiplication modulo 7.

Groups of these sorts are often written as "$\mathbb{Z}_n^*$", with an asterisk instead of a multiplication symbol. I will use "$\mathbb{Z}_n^\times$" because asterisks are ugly, and because it better communicates that these are based on multiplication.

1. The elements of the group are actually equivalence classes instead of actual integers. But I hope to avoid having to go into that.
2. I suspect that this is because the notation developed after typewriters but before $\TeX$.
3. Mathematicians tend to write "$\mathbb{Z}/n\mathbb{Z}$". Many fields (all puns intended) in math make use of groups of that sort and have varients of their own notation, depending on the distinctions they need. In cryptography, we can get away with $\mathbb{Z}_p^\times$.

Modular arithmatic

As a refresher, consider *addition* modulo 7.

Today is Friday. What day of the week will it be after another

- 7 days?
- 14 days?
- 7000 days?
- $7k$ days? (where k is an integer)
- 8 days?
- $7k + 1$ days (where k is an integer)

$\mathbb{Z}_7^\times$ IS A GROUP

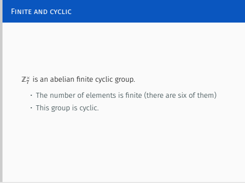
$\mathbb{Z}_7^\times$ is a group because

- It is **closed**. If a and b are elements then $a \times b \pmod 7$ is a is an element.
- Multiplication modulo 7 is **commutative**. If a and b are elements then $a \times b = b \times a \pmod 7$.
- There is an **identity element** (1) such that if a is an element then $a \times 1 = a \pmod 7$.
- Every element has an **inverse**. If a is an element then there is some element which we will write “a^{-1}” such that $a \times a^{-1} = 1 \pmod 7$.
- The operation is **associative**. If a, b, c are elements then $a \times (b \times c) = (a \times b) \times c \pmod 7$.

Finite and cyclic

$\mathbb{Z}_7^\times$ is an abelian finite cyclic group.

- The number of elements is finite (there are six of them)
- This group is cyclic.



Finite and cyclic

$\mathbb{Z}_7^\times$ is an abelian finite cyclic group.

- The number of elements is finite (there are six of them)
- This group is cyclic.

All of the examples for now will have n be prime. $\mathbb{Z}_n^\times$ for non-prime n is something we will get to when we start going into the math needed for RSA.

The discrete logarithm problem

Cycles and generators

$\mathbb{Z}_7^\times$ MULTIPLICATION

$\times$	1	2	3	4	5	6
1	1	2	3	4	5	6
2	2	4	6	1	3	5
3	3	6	2	5	1	4
4	4	1	5	2	6	3
5	5	3	1	6	4	2
6	6	5	4	3	2	1

Figure 1: Multiplication table for $\mathbb{Z}_7^\times$

3 DOES THE ROUNDS IN $\mathbb{Z}_7^\times$

$$\begin{tikzcd}
& 5 \arrow[rr, bend left, "\times 3"] & & 1 \arrow[dr, bend left, "\times 3"] & \\
4 \arrow[ur, bend left, "\times 3"] & & & & 3 \arrow[dl, bend left, "\times 3"] \\
& 6 \arrow[ul, bend left, "\times 3"] & & 2 \arrow[ll, bend left, "\times 3"] &
\end{tikzcd}$$

3 does the rounds in $\mathbb{Z}_7^\times$

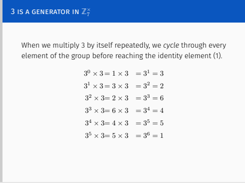
$$\begin{tikzcd}
& 5 \arrow[r, bend left, "\times 3"] & 1 \arrow[dr, bend left, "\times 3"] & \\
4 \arrow[ur, bend left, "\times 3"] & & & 3 \arrow[dl, bend left, "\times 3"] \\
& 6 \arrow[ul, bend left, "\times 3"] & 2 \arrow[l, bend left, "\times 3"] &
\end{tikzcd}$$

I use Ti*k*Z so rarely that I need to relearn it each time.

3 IS A GENERATOR IN $\mathbb{Z}_7^\times$

When we multiply 3 by itself repeatedly, we *cycle* through every element of the group before reaching the identity element (1).

$$\begin{aligned}
3^0 \times 3 &= 1 \times 3 &= 3^1 = 3 \\
3^1 \times 3 &= 3 \times 3 &= 3^2 = 2 \\
3^2 \times 3 &= 2 \times 3 &= 3^3 = 6 \\
3^3 \times 3 &= 6 \times 3 &= 3^4 = 4 \\
3^4 \times 3 &= 4 \times 3 &= 3^5 = 5 \\
3^5 \times 3 &= 5 \times 3 &= 3^6 = 1
\end{aligned}$$



3 IS A GENERATOR IN $\mathbb{Z}_7^\times$

When we multiply 3 by itself repeatedly, we *cycle* through every element of the group before reaching the identity element (1).

$$\begin{aligned}
3^0 \times 3 &= 1 \times 3 &= 3^1 = 3\\
3^1 \times 3 &= 3 \times 3 &= 3^2 = 2\\
3^2 \times 3 &= 2 \times 3 &= 3^3 = 6\\
3^3 \times 3 &= 6 \times 3 &= 3^4 = 4\\
3^4 \times 3 &= 4 \times 3 &= 3^5 = 5\\
3^5 \times 3 &= 5 \times 3 &= 3^6 = 1
\end{aligned}$$

For our purposes the distinction between “generator” and ”primative element” is not useful. So I am avoiding that second term.

Cyclic

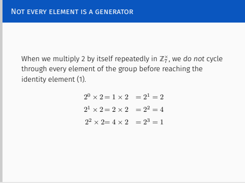
Definition (Cyclic)

A group is cyclic if and only if it has at least one generator that cycles through all members of the group.

Not every element is a generator

When we multiply 2 by itself repeatedly in $\mathbb{Z}_7^\times$, we *do not* cycle through every element of the group before reaching the identity element (1).

$$\begin{aligned} 2^0 \times 2 &= 1 \times 2 &= 2^1 = 2 \\ 2^1 \times 2 &= 2 \times 2 &= 2^2 = 4 \\ 2^2 \times 2 &= 4 \times 2 &= 2^3 = 1 \end{aligned}$$



Not every element is a generator

When we multiply 2 by itself repeatedly in $\mathbb{Z}_7^\times$, we *do not* cycle through every element of the group before reaching the identity element (1).

$$\begin{aligned} 2^0 \times 2 &= 1 \times 2 &= 2^1 = 2 \\ 2^1 \times 2 &= 2 \times 2 &= 2^2 = 4 \\ 2^2 \times 2 &= 4 \times 2 &= 2^3 = 1 \end{aligned}$$

This generator defines a finite cyclic *subgroup* of $\mathbb{Z}_7^\times$ with elements in $\{1, 2, 4\}$ while still using mod 7 multiplication.

Logs and exponents

Consider this equation where g is a generator of the finite cyclic group $\mathbb{Z}_p^\times$

$$A = g^a \pmod{p} \tag{1}$$

If we know g, p and a then it is easy to compute A.

Now suppose we know g, p, and A and we wish to find a. This is written as

$$a = \log_g A \pmod{p} \tag{2}$$

and is called the discrete logarithm.

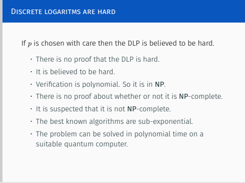
The discrete logarithm problem

Back to the DLP

Discrete logaritms are hard

If p is chosen with care then the DLP is believed to be hard.

- There is no proof that the DLP is hard.
- It is believed to be hard.
- Verification is polynomial. So it is in **NP**.
- There is no proof about whether or not it is **NP**-complete.
- It is suspected that it is not **NP**-complete.
- The best known algorithms are sub-exponential.
- The problem can be solved in polynomial time on a suitable quantum computer.



Discrete logaritms are hard

If p is chosen with care then the DLP is believed to be hard.

- There is no proof that the DLP is hard.
- It is believed to be hard.
- Verification is polynomial. So it is in **NP**.
- There is no proof about whether or not it is **NP**-complete.
- It is suspected that it is not **NP**-complete.
- The best known algorithms are sub-exponential.
- The problem can be solved in polynomial time on a suitable quantum computer.

This slide should look familiar.

Quick exponentiation

Our example group is $\mathbb{Z}_p^\times$ where p is 112 bit prime number

$$p = 3\,198\,594\,135\,065\,974\,402\,303\,823\,714\,395\,979$$

and our generator g is 2.

If we set a be some other 112 bit number,

$$a = 1\,729\,842\,084\,772\,514\,752\,626\,713\,784\,179\,499$$

it takes less than 20 microseconds to compute $A = g^a \pmod p$.

$$A = g^a \pmod p = 1\,142\,203\,535\,203\,822\,438\,059\,381\,484\,091\,159$$

Slow logarithm

Given g, p, and A from the previous slide, computing the discrete logarithm $a = \log_g A \pmod p$ takes more than 11 seconds.

DLP in elliptic curve groups

Generalizing the DLP

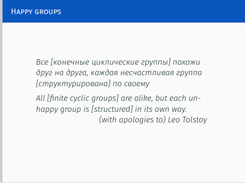
The cryptographically useful properties of the discrete logarithm problem requires a finite cyclic group (with a few other conditions on the group).

The group does not need to have integer elements and modular multiplication. It can be constructed from other things if it has the right structure. The term “generalized DLP” (GDLP) is sometimes used to talk about this generalization of the the DLP.

Happy groups

Все [конечные циклические группы] похожи друг на друга, каждая несчастливая группа [структурирована] по своему

All [finite cyclic groups] are alike, but each unhappy group is [structured] in its own way.
(with apologies to) Leo Tolstoy


Happy groups

Все [конечные циклические группы] похожи друг на друга, каждая несчастливая группа [структурирована] по своему

All [finite cyclic groups] are alike, but each unhappy group is [structured] in its own way.
(with apologies to) Leo Tolstoy

1. I spent way too much time on this slide. Particularly because I thought that ‘*г*’ and ‘*и*’ were some sort of error from a bad font mapping or encoding. It turns out that these are the correct Cyrillic letter forms for lowercase *italic* ‘г’ and ‘и’.
2. I really should have this about *abelian* finite cyclic groups, but I don’t want to bother Tim with more translation requests.
3. Misquoted from *Ana Karenina*, which is less well known than his very Russian novel *Special Military Operations and Peace*.

Elliptic curves

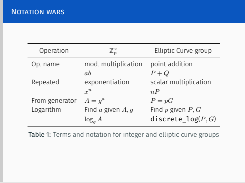
It is possible to define a group with the right structure using elliptic curves. See the computation-examples document for more information.

The operation is called "point addition" and so the notation is different.

Notation wars

Operation	$\mathbb{Z}_p^\times$	Elliptic Curve group
Op. name	mod. multiplication	point addition
	ab	$P + Q$
Repeated	exponentiation	scalar multiplication
	x^n	nP
From generator	$A = g^a$	$P = pG$
Logarithm	Find a given A, g	Find p given P, G
	$\log_g A$	`discrete_log`(P, G)

Table 1: Terms and notation for integer and elliptic curve groups



Notation wars

Operation	$\mathbb{Z}_p^\times$	Elliptic Curve group
Op. name	mod. multiplication	point addition
	ab	$P+Q$
Repeated	exponentiation	scalar multiplication
	x^n	nP
From generator	$A = g^a$	$P = pG$
Logarithm	Find a given A, g	Find p given P, G
	$\log_g A$	`discrete_log`(P, G)

Table 1: Terms and notation for integer and elliptic curve groups

1. ECs have a history in geometry, and so points are typically referred to using capital letters like P and Q, while in integer groups p is often used for the prime modulous.
2. When talking abstractly about groups, G is often used to refer to a group, but with elliptic curves, it is often used to describe the generator or base-point.
3. “Scalar” means ordinary number. n is a point.

DLP on elliptic curves

Given a suitably defined elliptic curve group $\mathcal{C}$ and a generator point G within the group and some integer d, a point P is easy to compute as in equation 3.

$$P = dG \qquad \text{(within group } \mathcal{C}\text{)} \tag{3}$$

Computing d from P is hard and is called finding the discrete logarithm of P.

GDLP Assymmetry

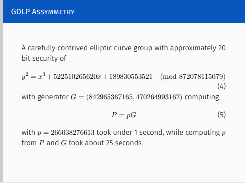
A carefully contrived elliptic curve group with approximately 20 bit security of

$$y^2 = x^3 + 522510265620x + 189830553521 \pmod{872078115079} \tag{4}$$

with generator $G = (842965367165, 470264993162)$ computing

$$P = pG \tag{5}$$

with $p = 266038276613$ took under 1 second, while computing p from P and G took about 25 seconds.



GDLP Assymmetry

A carefully contrived elliptic curve group with approximately 20 bit security of

$$y^2 = x^3 + 522510265620x + 189830553521 \pmod{872078115079} \tag{4}$$

with generator $G = (842965367165, 470264993162)$ computing

$$P = pG \tag{5}$$

with $p = 266038276613$ took under 1 second, while computing p from P and G took about 25 seconds.

The software that I happen to be using is not nearly as well optimized for elliptic curve operations as it is for integer operations.