Summary of Mathematical Groups in Cryptography

November 2020

Asymmetric cryptography can be achieved with a mathematical structure known as a "group". The key property a group must possess to be useful in cryptography is that the "discrete logarithm" must be hard to compute.

Group Definition: A *group* is a set of values G and an operation $\circ : G \times G \to G$, with the following requirements:

- The operation $\circ$ must be associative and commutative,
- There must be an identity element $e \in G$, and
- For every $x \in G$ there must be an inverse element $y \in G$ so that $x \circ y = e$.

Shorthand for specifying a group is $(G, \circ)$. Some examples of groups are $(\mathbb{Z}, +)$ where the identity is 0 and $(\mathbb{Q} - \{0\}, \times)$ where the identity is 1. Note that $(\mathbb{Q}, \times)$ and $(\mathbb{Z}, \times)$ are not groups because 0 has no inverse in either. These examples demonstrate that the operation can be addition or multiplication and that the set can be infinite. Cryptography always uses finite sets.

Example multiplicative group: The most commonly used multiplicative group in cryptography is $(\mathbb{Z}_n^*, \times \bmod n)$. The set $\mathbb{Z}_n^*$ is defined as $\{x \mid x \in \mathbb{Z}_n \text{ and } \gcd(x, n) = 1\}$. Because all the elements in the set share no factors with n, they all have multiplicative inverses mod n. The set $\mathbb{Z}_n^*$ is called the "multiplicative group mod n" and if $\mathbb{Z}_n^*$ is referred to as a group, it is implied that the operation is multiplication mod n.

For small n, you can compute the set $\mathbb{Z}_n^*$ by hand by eliminating from $\mathbb{Z}_n$ all of the elements that are a multiple of n's prime factors. For example, with $\mathbb{Z}_{12} = \{0, 1, 2, ..., 11\}$ if you remove the multiples of 2 (ie, 0, 2, 4, 6, 8, 10) and the multiples of 3 (ie, 0, 3, 6, 9), then you are left with $\mathbb{Z}_{12}^* = \{1, 5, 7, 11\}$ which are precisely the elements whose gcd with 12 is 1. You can verify that $\mathbb{Z}_{12}^*$ is a group by reviewing the definition of a group and checking that $\mathbb{Z}_{12}^*$ satisfies them all. A quicker way to arrive at a multiplicative group is to let $n = p$ where p is prime. Since a prime number p has no prime factors, $\gcd(x, p) = 1$ for every $x \in \mathbb{Z}_p$ except $\gcd(0, p) = p$. This means the multiplicative group mod prime p includes all of $\mathbb{Z}_p$ except 0, and so $\mathbb{Z}_p^* = \{1, 2, \ldots, p-1\}$. In fact, $\mathbb{Z}_p^*$, with prime p, is a very commonly used group in cryptography. □

Some definitions and facts

From here on, this note will use × for the group operation and 1 for the group identity. But, you should understand that this is just for convenience and to allow you some intuition. All the facts and definitions given below apply equally well if you use an additive group instead by simply switching to group operator + and group identity 0.

Notation: If G is a multiplicative group and $x \in G$, we write x^k as shorthand for $1 \times x \times x \times \cdots \times x$ where $\times$ is applied k times[1]. If G is an additive group and $x \in G$, we write xk or kx as shorthand for $0 + x + x + \cdots + x$ where $+$ is applied k times. Abstractly, both mean to begin with the identity and then apply the group operation k times with x.

Fact: If G is a finite group and $x \in G$ then $x^{|G|} = 1$. For example in $\mathbb{Z}_{12}^*$, $1^4 = 5^4 = 7^4 = 11^4 = 1$.

Definition: If G is a finite group and $x \in G$ then the *order of element* x is the smallest $k > 0$ such that $x^k = 1$. One way to determine the order of x is to write out x^1, x^2, x^3, etc, until you encounter your first 1. Here's an example

[1] Note that because we start with the identity 1, this explains why $x^0 = 1$.

finding the order of each element in $\mathbb{Z}_5^*$.

$$\begin{array}{rcl}
x & : & x^1, x^2, x^3, \ldots, 1 \\
1 & : & 1 \\
2 & : & 2, 4, 3, 1 \\
3 & : & 3, 4, 2, 1 \\
4 & : & 4, 1
\end{array}$$

You can tell from this exercise that, in $\mathbb{Z}_5^*$, ord(1) = 1, ord(2) = 4, ord(3) = 4, and ord(4) = 2. You may notice from these orders, that the size of the group is a multiple of each. This is always true.

Fact: If G is a finite group and $x \in G$ then $|G|$ is a multiple of the order of x. Here's another example, $\mathbb{Z}_{15}^* = \{1, 2, 4, 7, 8, 11, 13, 14\}$. The order of each of its eight elements is a factor of eight.

$$\begin{array}{rcl}
x & : & x^1, x^2, x^3, \ldots, 1 \\
1 & : & 1 \\
2 & : & 2, 4, 8, 1 \\
4 & : & 4, 1 \\
7 & : & 7, 4, 13, 1 \\
8 & : & 8, 4, 2, 1 \\
11 & : & 11, 1 \\
13 & : & 13, 4, 7, 1 \\
14 & : & 14, 1
\end{array}$$

Definition: $(H, \circ)$ is a *subgroup* of $(G, \circ)$ if $H \subseteq G$ and the group operation is identical for both groups.

For example $\mathbb{Z}_5^*$ is not a subgroup of $\mathbb{Z}_7^*$. Even though $\mathbb{Z}_5^* \subseteq \mathbb{Z}_7^*$, the two groups use different operations (mod 5 in the first case and mod 7 in the second). On the other hand, $\{1, 2, 4, 8\}$ is a subgroup of $\mathbb{Z}_{15}^*$ when they both use multiplication mod 15 as their operation.

Fact: If G is a finite group and $x \in G$ then $\{x^i \mid 0 < i \leq \text{ord}(x)\}$ is a subgroup of G (using G's group operation).

This is basically saying that if you write out $x^1, x^2, x^3, \ldots$ until you reach 1, the values you generate form a subgroup. Looking at the table just above, this means $\{1\}$, $\{1, 2, 4, 8\}$, $\{1, 4\}$, $\{1, 4, 7, 13\}$, $\{1, 11\}$, and $\{1, 14\}$ are all subgroups of $\mathbb{Z}_{15}^*$ when multiplication mod 15 is the operation used.

Definition: If G is a finite group and $x \in G$ then x is called a *primitive element* (or *generator*) of G if and only if the order of x is $|G|$. If a group has at least one primitive element, the group is called *cyclic.* It is known that for every prime p, $\mathbb{Z}_p^*$ is a cyclic group.

Hard problem: In the *discrete logarithm* problem, the definition of a cyclic group and a generator g of the group are given along with a value g^x for some unknown x. The goal for the solver is to find x. For example, let's say you know the group in questions is is $\mathbb{Z}_{101}^*$ and $g = 3$ is the generator used. If you were told that $g^x = 5$, could you find x? There's no obvious way to do it, so you'd probably resort to "brute force" and try all possible x's until you find the one that works[2].

There is a powerful number-theoretic algorithm, called the Number Field Sieve, for attacking the discrete logarithm problem over a prime group $\mathbb{Z}_p^*$, but it only accelerates the search rather than finding the solution quickly. So, for security reasons cryptography uses large prime moduli of thousands of bits when defining these groups. This makes the logarithm hard. For efficiency of exponentiation, however, a much smaller subgroup of the larger group is often used to accelerate it.

Application (finding subgroup of $\mathbb{Z}_p^*$ of desired size): To find a large group $\mathbb{Z}_p^*$ of size $p-1$ and a size q subgroup of it, choose p and q to be primes with the relationship $p = qN + 1$ for some integer N. The large group is $\mathbb{Z}_p^*$ and has $p-1$ elements. Every element x of $\mathbb{Z}_p^*$ has the property that $x^{p-1} = 1$. But since $p = qN + 1$, this means $x^{qN} = 1$. Using laws of exponents, $(x^N)^q = 1$. Let's call $g = x^N$. The value g will be 1 if ord(x) is a factor of N, but if it's not 1, then g generates a subgroup of size q. To see this, consider the sequence $g^1, g^2, \ldots, g^q$. Because q is prime, it is impossible for g^i and g^q to both be 1 for any $0 < i < q$. They will all be different numbers, forming the desired subgroup.

Given that g generates a size q subgroup of $\mathbb{Z}_p^*$, calculating $g^x \bmod p$ can be accelerated. If x is chosen so that $0 \le x < p$, we can rewrite $x = qn + r$ using the division algorithm for some integers n and r, then g^x can be rewritten $g^{qn+r} = g^{qn} \cdot g^r = (g^q)^n \cdot g^r = 1^n \cdot g^r = g^r$. But, r is just $x \bmod q$, so in practice $g^x \bmod p$ can be calculated much more quickly as $g^{x \bmod q} \bmod p$. Alternatively, if it is known that g generates a size q subgroup of $\mathbb{Z}_p^*$, x can be chosen with $0 \le x < q$ rather than $0 \le x < p$, which is faster and gives the same distribution. □

Application (Diffie-Hellman key exchange): If g generates a group of size n, then Alice can choose a random $0 \le x < n$ and send g^x to Bob. Bob can choose a random $0 \le y < n$ and send g^y to Alice. Both now have enough information to compute $k = g^{xy}$. If the discrete logarithm problem is hard for the group being used, then an adversary knows only g^x and g^y, which is not enough information to compute g^{xy}.

Diffie-Hellman (DH) is the basis of several protocols in cryptography, but is susceptible to an adversary pretending to be Bob to Alice and Alice to Bob, known as a man-in-the-middle-attack. Further authentication is required for the key exchange to be trustworthy. □

Application (Elgamal asymmetric encryption): Elgamal encryption uses a DH key to directly encrypt a message. During key setup a fixed DH contribution is made as part of the public key, and then each time someone wants to encrypt using the public key a new second random DH contribution is chosen and made part of the ciphertext. It can be thought of as a DH key exchange over a longer stretch of time. For simplicity the following shows Elgamal using group $\mathbb{Z}_p^*$ where p is prime.

Setup: (i) Choose prime p and generator g. (ii) Choose random d from $\mathbb{Z}_p^*$. Publish (p, g, g^d); keep secret d.

Encrypt x: (i) Choose random e from from $\mathbb{Z}_p^*$. (ii) Compute $k = (g^d)^e$ and $y = xk \bmod p$. Send as the ciphertext the pair (g^e, y). Note that each encryption will have a different random e, so k and y will differ even if x is repeated.

Decrypt y: (i) Compute $k = (g^e)^d$ and $k^{-1} \bmod p$. (ii) Then $x = yk^{-1} = (xk)k^{-1} = x(kk^{-1}) = x$ (all mod p). □

Additive group, elliptic curves: See notes on course webpage.

[2] The following one-line python program finds x for $3^x \bmod 101 = 5$: `print([x for x in range(101) if (3**x)%101==5])`.