COSC 530: Background for Public-Key Cryptography

In symmetric-key cryptography our “source of hardness” were blockciphers or hash functions, which are “heuristically” designed. It turns out to be much harder to design sources of hardness in the public-key setting. For this we have to discuss a little background in group and number theory.

**Basic notation.** We define the set of integers \( \mathbb{Z} = \{ \ldots, -1, 0, 1, \ldots \} \), positive integers \( \mathbb{Z}_+ = \{1, 2, \ldots \} \), natural numbers \( \mathbb{N} = \{0, 1, \ldots \} \) (these are all countably infinite sets). For \( N \in \mathbb{Z}_+ \) we define the set of integers modulo \( N \), \( \mathbb{Z}_N = \{0, 1, \ldots, N - 1\} \) (pronounced as zee-enn) and \( \mathbb{Z}_N^* = \{a \in \mathbb{Z}_N \mid \gcd(a, N) = 1\} \) (pronounced as zee-enn-star), the set of all integers modulo \( N \) whose greatest common divisor with \( N \) is 1, aka. that are “relatively prime” to \( N \). It follows that \( |\mathbb{Z}_N^*| = \phi(N) \) where \( \phi(\cdot) \) is Euler’s phi-function. In particular if \( N \) is prime then \( |\mathbb{Z}_N^*| = N - 1 \).

**Groups.** The set \( \mathbb{Z}_N^* \) under the operation multiplication modulo \( N \), that is, the binary operation “\(^*\)” (or just juxtaposition) where for all \( a, b \in \mathbb{Z}_N^* \), \( ab = ab \mod N \), is an example of a group. This means a set \( \mathcal{G} \) and binary operation “\(^*\)” that satisfy some natural constraints: (1) Closure, (2) Associativity, (3) Existence of identity element, and (4) Existence of inverses. It is easy to see that the identity, often denoted \( 1_\mathcal{G} \), is unique. Groups like \( \mathbb{Z}_N^* \) that are furthermore commutative are often called “Abelian.”

For a group \( \mathcal{G} \), its order of the group is denoted \( |\mathcal{G}| \) and defined as its number of elements (here we are technically conflating the group with the underlying set, as is typical). For \( g \in \mathcal{G} \) define \( \langle g \rangle = \{g^i \mid i \in \mathbb{Z}\} \). This is an example of a subgroup of \( \mathcal{G} \) (a subset of the group which is itself a group). Lagrange’s Theorem is that the order of any subgroup divides the order of the group. (A partial converse is given by the Sylow theorems.)

Let \( \mathcal{G} \) be a group. We say that \( \mathcal{G} \) is cyclic if there exists \( g \in G \) such that \( \mathcal{G} = \langle g \rangle \). Such a \( g \) is called a generator of \( \mathcal{G} \). If follows from Lagrange’s Theorem that \( \mathbb{Z}_N^* \) is cyclic (of order \( N - 1 \)) when \( N \) is prime; in this case a generator is also referred to a primitive root. In general \( \mathbb{Z}_N^* \) is not cyclic; it is isomorphic to \( \mathbb{Z}_{p_1^{k_1}}^* \times \cdots \times \mathbb{Z}_{p_r^{k_r}}^* \) where \( p_1^{k_1} \cdots p_r^{k_r} \) is the prime factorization of \( N \).

**Efficient algorithms.** We care about groups that can be represented and operated on efficiently. In particular, elements in the group \( \mathbb{Z}_N^* \) can be represented in binary in the natural way using log \( N \) bits. The group operation can be computed efficiently in \( \log N \), which we also denote by \( |N| \). In fact, modular multiplication runs in quadratic time in this sense. In general, “efficient” here means efficient in the bit-length of the elements (not their magnitude).

Can exponentiation be efficiently computed? Let \( \mathcal{G} \) be a group and \( a \in \mathcal{G} \). For \( n \in \mathbb{N} \) we want to compute \( a^n \in \mathcal{G} \). One way is: set \( y ← 1 \) then for \( i = 1 \) to \( n \) do \( y ← y \cdot a \). This is correct but too slow, it runs in time exponential in \( |n| \), the bit-length of \( n \). However, fast exponentiation can be done using the so-called square-and-multiply method. The idea is to represent \( n = b_{k-1} \ldots b_0 \) in binary and then set \( y ← 1 \) and for \( i = k - 1 \) down to 0 do \( y ← y^2 \cdot a^{b_i} \). This takes \( |n| \) group operations so exponentiation in \( \mathbb{Z}_N^* \) takes cubic time.
Discrete logarithms. Let $G = \langle g \rangle$ be a cyclic group of order $m$. Then for every $a \in G$ there is a unique $i \in \mathbb{Z}_m$ such that $g^i = a$. We call $i$ the discrete logarithm of $a$ to the base $g$ and denote it by $\text{dlog}_{G,g}(a)$ or just $\text{dlog}(a)$ when clear from context. Associated to $G, g$ we have Game DLog$_{G,g}$ defined as:

```plaintext
proc Initialize
  $x \leftarrow Z_m$; $X \leftarrow g^x$
  Return $X$
proc Finalize($x'$)
  Return $(x = x')$
```

The best generic attack on discrete log runs in time $\sqrt{m}$. For example, consider the baby-step giant-step algorithm. The idea is to re-write $x$ as $x = iq + r$ where $q = \sqrt{m}$. Then we have

$$g^x(g^{-q})^i = g^j.$$ 

So for each value of $j$ try all the possible values of $i$. Surprisingly, it can actually be proven that there is no better generic attack. For some groups like appropriate elliptic curve groups, no attack better than generic is known. Thus we could use such groups of order $2^{160}$ for 80-bit security. For groups $\mathbb{Z}_p^*$, there are somewhat better attacks known (since this is in fact a finite field), about $2^{\log(p)^{1/3}}$ (versus $2^{\log(p)/2}$ for generic). As a result we need to take say $|p| = 1024$ in this case. The records for discrete log computation are roughly 113,596 respectively.