One-way functions from the discrete logarithm and SIS

Noah Stephens-Davidowitz

September 10, 2019

1 One-way functions recap

Recall from the previous lecture the definition of a one-way function. In fact, we had two definitions, one strong and one weak. (Remember that a function \( \varepsilon(n) \) is negligible if it decays faster than any inverse polynomial. I.e., for every polynomial \( p(n) \), there exists an \( n_0 \geq 1 \) such that \( \varepsilon(n) \leq 1/p(n) \) for all \( n \geq n_0 \).)

Definition 1.1. A function \( f : \{0,1\}^* \rightarrow \{0,1\}^* \) is called (strongly) one-way if it satisfies the following.

1. Easy to compute. There is a probabilistic polynomial-time algorithm computing \( f \).

2. Hard to invert. For all probabilistic polynomial-time adversaries \( A \), there exists a negligible \( \varepsilon(n) \) such that

\[
\Pr_{x \leftarrow \{0,1\}^n} [A(1^n, f(x)) = x' : f(x') = f(x)] \leq \varepsilon(n)
\]

for all \( n \geq 1 \).

\( f \) is called weakly one-way if it satisfies the following instead of Item 2.

2’. Weakly hard to invert. There exists a polynomial \( p(n) \) such that

\[
\Pr_{x \leftarrow \{0,1\}^n} [A(1^n, f(x)) = x' : f(x') = f(x)] \leq 1 - 1/p(n)
\]

for all probabilistic polynomial-time adversaries \( A \) and \( n \geq 1 \).

We saw a candidate for a weak one-way function: multiplication. We also saw that weak one-way functions imply strong one-way functions. So, we already know how to build strong one-way functions (assuming that factoring is suitably hard). Here, we introduce two more constructions of one-way functions, as well as some of the mathematical background that will be useful for the rest of the course.

2 The discrete logarithm

2.1 Background: commutative groups

We will need to introduce the notion of a group. Actually, we will only need a special kind of group, which we will call commutative groups. They are often also called “Abelian groups” in honor of Abel (pronounced uh-beel’-lyun and ah’-bull respectively).
Definition 2.1. A commutative group \((G, \cdot)\) is a set \(G\) with a binary operation \(\cdot\) over \(G\), written \(g \cdot h\), with the following properties.

1. **Closure.** For all \(g, h \in G\), \(g \cdot h \in G\).

2. **Identity.** There exists an element \(e \in G\) (called the identity element) such that \(e \cdot g = g \cdot e = g\) for all \(g \in G\).

3. **Inverses.** For every \(g \in G\), there exists a \(g^{-1} \in G\) such that \(g \cdot g^{-1} = e\).

4. **Associativity.** For every \(g_1, g_2, g_3 \in G\), \((g_1 \cdot g_2) \cdot g_3 = g_1 \cdot (g_2 \cdot g_3)\).

5. **Commutativity.** For every \(g, h \in G\), \(g \cdot h = h \cdot g\).

We are only interested in groups with finitely many elements, and we refer to the number of elements \(|G|\) as the order of the group.

Our simplest example is \((\mathbb{Z}_q, +)\), the additive group modulo an integer \(q \geq 2\). I.e., the set is simply \(\mathbb{Z}_q := \{0, 1, \ldots, q - 1\}\), and the group operation is addition modulo \(q\), \(g + h \mod q\). Clearly, this is a commutative group. In particular, 0 is the identity element, and every \(g \in \mathbb{Z}_q\) has as its inverse \(q - g \in \mathbb{Z}_q\). Its order is \(q\).

The other example that interests us is the multiplicative group modulo \(q\), \((\mathbb{Z}_q^*, \cdot)\). The set is \(\mathbb{Z}_q^* := \{g \in \mathbb{Z}_q : \gcd(g, q) = 1\}\), and the group operation is multiplication modulo \(q\), i.e., \(g \cdot h \mod q\). It is a bit less clear that this is a commutative group. In particular, 0 is the identity element, and every \(g \in \mathbb{Z}_q\) has as its inverse \(q - g \in \mathbb{Z}_q\). Its order is \(q\).

For every integer \(k\) such that \(gh + kq = 1\) we recall that the extended Euclidean algorithm finds integers \(h, k\) satisfying this identity (sometimes called Bézout's identity). So, \(h = g^{-1} \mod q\) certainly exists (and we can even find it efficiently, given \(g\) and \(q\).)

The other example that interests us is the multiplicative group modulo \(q\), \((\mathbb{Z}_q^*, \cdot)\). The set is \(\mathbb{Z}_q^* := \{g \in \mathbb{Z}_q : \gcd(g, q) = 1\}\), and the group operation is multiplication modulo \(q\), i.e., \(g \cdot h \mod q\). It is a bit less clear that this is a commutative group. In particular, 0 is the identity element, and every \(g \in \mathbb{Z}_q\) has as its inverse \(q - g \in \mathbb{Z}_q\). Its order is \(q\).

In fact, \(\phi(q) = q - 1\). More generally, if \(q = p_1^{a_1} \cdots p_\ell^{a_\ell}\) for distinct primes \(p_1, \ldots, p_\ell\), then \(\phi(q) = p_1^{a_1 - 1}(p_1 - 1) \cdot p_2^{a_2 - 1}(p_2 - 1) \cdots p_\ell^{a_\ell - 1}(p_\ell - 1)\).

2.2 Background: exponentiation in a group

We write \(g^k := g \cdot g \cdots g\) for the product of \(g \in G\) with itself \(k \geq 1\) times—or, if the operation is addition, we write \(kg := g + g + \cdots + g\). We also define \(g^{-k} := (g^{-1})^k\) and \(g^0 := e\). With these conventions, this operation satisfies the basic properties of exponentiation: \((g^k)^\ell = g^{k\ell}\), \(g^{-k} = (g^k)^{-1}\), \(g^k \cdot g^\ell = g^{k+\ell}\), and \((hg)^k = h^kg^k\).

Now, \(G\) is finite. Therefore, the sequence \(g^1, g^2, g^3, \ldots\), must eventually repeat itself. I.e., \(g^k = g^\ell\) for some \(k > \ell\). Multiplying by \(g^{-\ell}\) on both sides, we see that \(g^{k-\ell} = e\). So, for every element \(g\) in the group, there exists some \(k \geq 1\) such that \(g^k = e\). The minimal such \(k\) is called the order of \(g\). A key fact (proven in Appendix A) is that the order \(k\) of the element must divide the order \(|G|\) of the group.

A group is cyclic if all its elements can be written as a power of one fixed element \(g \in G\), i.e., \(G = \{e, g, g^2, \ldots, g^{|G|-1}\}\). Equivalently, a group is cyclic if it has an element with order \(|G|\). We call such a \(g\) a generator.
For example, the additive group \( \mathbb{Z}_q \) is cyclic, with 1 as a generator. More generally, any element coprime to \( q \) is a generator. So, there are actually \( \phi(q) \) generators of \( \mathbb{Z}_q \).

The multiplicative group \( \mathbb{Z}_q^* \) is not always cyclic, but it is cyclic when \( q \) is prime. (See Appendix B for a proof.) I.e., there is some element \( g \) whose order equals the order \( q - 1 \) of the entire group. In fact, since one such element exists, there must be many. To see this, we recall that every \( q \) coprime to \( q \) multiple of \( q - 1 \). It follows that the order of \( g^k \) is exactly \((q - 1)/\gcd(k, q - 1)\). In particular, \( g^k \) is a generator whenever \( k \) and \( q - 1 \) are coprime. So, the number of generators \( \phi(q - 1) \). (More generally, the number of generators of a cyclic group \( G \) is always \( \phi(|G|) \).)

### 2.3 Groups as computational objects

Since we are cryptographers interested in asymptotic complexity, we need a security parameter \( n \), and our group operation should be efficiently computable in the security parameter. This notion does not really make sense for a fixed group \( G \), so, we will actually need a sequence \( G_1, G_2, G_3, \ldots \), of groups, say with \( |G_n| \approx 2^n \). Let’s assume that group elements are represented by \( n \)-bit strings and that group operations are computable in poly\((n)\) time, given a description of the group \( G_n \) as advice. Later in the class, we will get tired of dragging the parameter \( n \) around. But, for now, we’ll keep it.

For example, \( G_n \) could be \( \mathbb{Z}_{q_n} \) or \( \mathbb{Z}_{q_n}^* \) for some \( n \)-bit number \( q_n \). Addition modulo \( q_n \) takes \( O(\log q_n) = O(n) \) time, and multiplication modulo \( q_n \) can be done in \( O(\log^2 q_n) = O(n^2) \) time (or even in \( O(n \log n) \) time [HvdH19]). So, both of these groups have efficiently computable group operations. For these two groups, the inverse \( g^{-1} \in G_n \) is efficiently computable as well.

But, what about computing \( g^k \in G_n \)? The naive algorithm just computes \( g, g^2, g^3, \ldots, g^k \), which requires us to compute \( k \) group operations. This is not a polynomial in the bit length \( \log k \) of \( k \). But there is a better way that allows us to use only \( O(\log k) \) operations! We can compute \( g^1, g^2, g^4, \ldots, g^{2^{\ell-1}}, g^{2^\ell} \in G_n \), where \( \ell := \lceil \log_2 k \rceil \), using a total of \( \ell \) group operations by noticing that \( g^{2^k} = g^{2^i} \cdot g^{2^j} \). Since we can write \( k \) as a sum of \( O(\log k) \) numbers of the form \( 1, 2, 4, \ldots, 2^\ell \) (by writing \( k \) in binary), we can write \( g^k \) as \( O(\log k) \) products of the \( g^{2^i} \in G_n \). This allows us to compute \( g^k \) in just \( O(\log k) \) total group operations, as claimed.

### 2.4 The discrete logarithm

We are now ready to present our one-way function! Assume we have a sequence \( G_1, G_2, G_3, \ldots \), of groups together with generators \( g_1, g_2, g_3, \ldots \). Our one-way function \( f \) is simply \( f(k) := g_n^k \in G_n \), where \( n \) is the bit length of \( k \). The problem of inverting \( f \) is called the discrete logarithm problem. I.e., given a generator \( g_n \in G_n \) and another element \( h \in G_n \), the discrete logarithm problem is to find \( k \) such that \( g_n^k = h \). We write \( \log_{g_n}(h) \) for the unique \( 0 \leq k \leq |G_n| - 1 \) such that \( g_n^k = h \).

(Notice the similarity with the “continuous logarithm.” E.g., \( \log_2 128 \) is the number \( x \) such that \( 2^x = 128 \).)

So, when is this hard? For the additive group \( \mathbb{Z}_q \), the discrete logarithm problem is easy, i.e., solvable in time \( \text{poly}(\log q) \). Indeed, the discrete logarithm over \( \mathbb{Z}_q \) is the following. We are given \( g, h \in \mathbb{Z}_q \) with \( g \) coprime to \( q \), and we are asked to find \( k \) such that \( kg = h \mod q \). To do so, it suffices to compute the inverse of \( g \mod q \), i.e., the element \( r \in G \) such that \( gr = 1 \mod q \). We observed earlier that the Euclidean algorithm lets us compute this efficiently. Then, we can find \( k \) by computing \( k = \log_q h \mod q \).
For the multiplicative group \( \mathbb{Z}_q^* \), things are more interesting. As far as we know, the discrete logarithm is in fact hard over \( \mathbb{Z}_q^* \). Specifically, the best known algorithm for the discrete logarithm over \( \mathbb{Z}_{q_n}^* \) runs in time \( 2^{O(\log^{1/3} q (\log \log q)^{2/3})} = 2^{O(n^{1/3} \log^{2/3} n)} \). So, we can fix some sequence \( q_n \) of \( n \)-bit primes and generators \( g_n \) of \( \mathbb{Z}_{q_n}^* \), and we believe that the function \( f(k) := g_n^k \mod q_n^2 \) is in fact one way. (There are groups of size roughly \( 2^n \) over which the fastest known algorithm for the discrete logarithm runs in time \( 2^{n/2} \). Because of this, these groups, which are based on elliptic curves, are used in practice—and sometimes in theory as well.)

### 2.5 Where do \( G_n \) and \( g_n \) come from?

Since we are okay with non-uniform algorithms, we do not formally need to worry so much about where \( G_n \) and \( g_n \) come from. Technically, we can just provide them as advice to the algorithm that computes our one-way function \( f \). In other words, we can hard code \( G_n \) and \( g_n \) into our algorithm.

But, it’s a bit unsatisfying to say that \( G_n \) and \( g_n \) just fall from the sky. For this one-way function to be useful, there obviously has to be some way to find them. So, we now show how to efficiently find an \( n \)-bit prime \( q \) together with a generator \( g \) of \( \mathbb{Z}_q^* \).

So, first of all, how do we find \( n \)-bit primes \( q \)? Even this is not entirely trivial. There is no deterministic poly(\( n \))-time algorithm known (though very simple algorithms work under certain very strong number-theoretic conjectures). But, with randomness, it is relatively straightforward. We pick a random \( n \)-bit number, use our favorite efficient primality testing algorithm to test if it is prime, and repeat this until we find one. The Prime Number Theorem tells us that our guess will be prime with probability roughly \( 1/n \), so that we are likely to find a prime after \( n \) or so tries.

Finding generators is harder. We mentioned earlier that \( \mathbb{Z}_q^* \) has a lot of generators, \( \phi(q - 1) \) of them. (\( \phi(m) \) is always at least \( \Omega(m / \log \log m) \). I.e., for \( n \)-bit primes, at least a \( 1/\log n \) fraction of the elements are generators.) So, if we had some way to test whether an element \( g \in \mathbb{Z}_q^* \) is a generator, then we could find one using the same guess-and-check trick that allows us to find primes. And, since the order of \( g \) must divide the order of the group, the possible orders for the element \( g \) correspond to the factors of \( |\mathbb{Z}_q^*| = q - 1 \). If we knew the non-trivial factors of \( q - 1 \), say, \( d_1, \ldots , d_t \), then we could test whether \( g \) is a generator by checking whether \( g^{d_i} = 1 \mod q \) for all \( i \). \( g \) is a generator if and only if \( g^{d_i} \neq 1 \mod q \) for all \( i \). There are at most \( \log_2 q \) factors, so we could do this efficiently, given the factors.

Unfortunately, factoring \( q - 1 \) seems to be hard. (Maybe that’s actually fortunate.) But, we can use a trick to get around this: instead of sampling a prime \( q \) and then trying to factor \( q - 1 \), we can sample a factorization of \( q - 1 \) first and then test whether \( q \) is prime. There are beautiful algorithms to do this that achieve uniformly random \( q \) [Kal02], but in practice, we do the following. We sample a uniformly random \( (n - 1) \)-bit prime \( p \) using the guess-and-check technique described above. If \( q = 2p + 1 \) is prime, then take this to be \( q \). Otherwise, resample \( p \) until this is true.

Primes \( p, q \) satisfying \( q = 2p + 1 \) are called Sophie Germain primes (after Marie-Sophie Germain). More specifically, \( p \) is called a Sophie Germain prime, and \( q \) is called a safe prime. We believe that a random \( n \)-bit number will be a Sophie Germain prime with probability roughly \( 1/n^2 \) (i.e., the probability that two random \( n \)-bit numbers \( p \) and \( q \) are prime). But, like many things in cryptography, we do not know how to prove it. (Sophie Germain primes are used quite a lot in practice, and in practice, this works.)

Notice that we conveniently know the factorization of \( q - 1 \) when \( q \) is a safe prime. Specifically, \( q - 1 = 2p \). So, to find a generator of \( \mathbb{Z}_q^* \), we sample a random element \( g \) and check if \( g^2 = 1 \mod q \).
or \( g^p = 1 \mod q \). If not, then \( g \) is a generator. (Safe primes are quite useful in number-theoretic cryptography in general because of this nice property.)

To make this approach fit with our notion of a one-way function, we need to modify the definition slightly. Instead of one function \( f \), we have a family of functions \( f_k \) and an efficient function \( G \) that takes as input \( 1^n \) and outputs \( k \). E.g., \( G \) can be the algorithm described above to find an \( n \)-bit safe prime \( q \) and a generator \( g \) of \( \mathbb{Z}_q^* \). It outputs \( q \) and \( g \). The formal definition is below.

**Definition 2.2.** A one-way function family is an algorithm \( G \) that takes as input \( 1^n \) and outputs \( k \in \mathcal{K} \) and a function \( f : \mathcal{K} \times \{0,1\}^* \rightarrow \{0,1\}^* \) satisfying the following properties.

1. **Easy to compute.** There is a probabilistic polynomial-time algorithm computing \( f \).

2. **Hard to invert.** For all probabilistic polynomial-time adversaries \( A \), there exists a negligible \( \varepsilon(n) \) such that

   \[
   \Pr_{x \leftarrow \{0,1\}^n, k \leftarrow G(1^n) \left[ A(1^n, k, f(k, x)) = x' : f(k, x') = f(k, x) \right] \leq \varepsilon(n)
   \]

   for all \( n \geq 1 \).

In fact, the existence of a one-way function family implies the existence of a one-way function. We will not show this here, but the proof is relatively simple. Given \( f \) and \( G \), we construct \( f' \) whose input consists of the randomness \( r \) used by \( G(1^n) \) together with the input \( x \) to \( f \). \( f' \) outputs the key \( k = G(1^n; r) \) together with \( f(k, x) \).

## 3 Short Integer Solutions

Our next candidate family of function is simpler. We just apply a linear transformation to our input modulo \( q \). I.e., we have a matrix \( A_n \in \mathbb{Z}_{q_n}^{m_n \times n} \), and we take \( f(x) := A_n x \mod q_n \), where we interpret our input as a vector \( x \in \mathbb{Z}_{q_n}^n \) and the resulting output lies in \( \mathbb{Z}_{q_n}^{m_n} \).

Of course, as described, this function is certainly not secure. In particular, inverting \( f \) is equivalent to solving a system of linear equations over \( \mathbb{Z}_q \), which is easy. It can be done, e.g., using Gaussian elimination. To make this function secure, we modify it in two clever ways, due to Ajtai [Ajt96].

First, we take \( m_n \) to be much smaller than \( n \). As a result, every output vector \( y \in \mathbb{Z}_{q_n}^{m_n} \) has many different preimages \( x \in \mathbb{Z}_{q_n}^n \) such that \( A_n x \mod q_n \) (assuming that \( A_n \) has full rank). Second, instead of allowing arbitrary input vectors \( x \in \mathbb{Z}_{q_n}^n \), we only allow \( \{0,1\} \)-vectors as input, \( x \in \{0,1\}^n \). (We do this for security, but it is also rather convenient to work with bit vectors.)

Notice that, if we did one of these things but not the other, then \( f \) would not be secure. For example, if \( m_n \geq n \) then \( A_n x \) will typically have a single preimage, which can be found efficiently. Restricting our input to bit vectors will not change that. On the other hand, if we take \( m_n \) to be small but allow arbitrary input, then we can still use Gaussian elimination to find a preimage \( x' \in \mathbb{Z}_q^n \) with \( A_n x = A_n x' \). In fact, we can find a whole affine subspace of preimages \( x' \).

But, if there are many preimages, it seems hard to pick out one whose coordinates happen to be bits. In fact, Ajtai showed that this problem is hard if certain well-studied and seemingly hard geometric problems called lattice problems are hard. Ajtai even showed that this (conditional) hardness holds if we choose \( A \in \mathbb{Z}_{q_n}^{m_n \times n} \) uniformly at random [Ajt96, MR07], giving a simple one-way function family.
The problem of finding $x \in \{0,1\}^n$ such that $Ax = y \mod q$ is called Short Integer Solutions or just SIS.

A Proof that the order of an element divides the order of the group

For each $h \in G$, let $C_h := \{h, h \cdot g, h \cdot g^2, h \cdot g^3, \ldots, h \cdot g^{k-1}\}$, where $k$ is the order of $g$. Notice that $|C_h| = k$.

We claim that for any two $h, h' \in G$, either $C_h = C_{h'}$ or $C_h \cap C_{h'} = \emptyset$. Indeed, suppose that $r \in C_h \cap C_{h'}$. I.e., $r = h \cdot g^\ell = h' \cdot g'^{\ell'}$ for some $\ell, \ell'$. Then, $h = h' \cdot g^{\ell'-\ell} \in C_{h'}$, and it follows that $h \cdot g^i = h' \cdot g^{i+\ell'-\ell} \in C_{h'}$ for any $i$. (Which parts of the definition of a group did we use here?)

So, $G$ can be partitioned into disjoint sets $C_h$ with $|C_h| = k$. It follows that $k$ divides $|G|$. (The quotient $|G|/k$ is the number of distinct sets of the form $C_h$.)

B Proof that $\mathbb{Z}_q^*$ is cyclic for prime $q$

The elements of $\mathbb{Z}_q^*$ must have order dividing $|\mathbb{Z}_q^*| = \phi(q) = q-1$. For a divisor $d$ of $q-1$, let $N_d$ be the number of elements $g \in \mathbb{Z}_q^*$ with order $d$. We claim that $N_d = \phi(d)$. In particular, $N_{q-1} = \phi(q-1)$, so that there are actually many generators of $\mathbb{Z}_q^*$, and it is therefore cyclic.

To see that $N_d = \phi(d)$, first suppose that there exists one element $g \in \mathbb{Z}_q^*$ with order $d$. Then, $g^k$ has order $d$ for any $k$ coprime to $d$. There are of course $\phi(d)$ such elements $g^k$. Furthermore, for all $1 \leq k \leq d$, $g^k$ is a distinct root of the polynomial $x^d - 1 \mod q$. Since $\mathbb{Z}_q$ is a field, this polynomial has at most $d$ roots. So, there cannot be any other elements with order $d$.

The above shows that either $N_d = 0$ or $N_d = \phi(d)$. To finish the proof, we use Euler’s identity:

$$\sum_{d|m} \phi(d) = m.$$ 

Applying this to $m = q - 1$ and using the fact that $N_d \leq \phi(d)$, we see that $\sum N_d \leq q - 1$ with equality if and only if $N_d = \phi(d)$ for all $d$. Since $\sum N_d = |\mathbb{Z}_q^*| = q - 1$, we must have $N_d = \phi(d)$ for all $d$.

References


[HvdH19] David Harvey and Joris van der Hoeven. Integer multiplication in time $O(n \log n)$. https://hal.archives-ouvertes.fr/hal-02070778, 2019.
