Cryptography and Security Protocols **IST-UL, 2024/2025**

The discrete logarithm assumption and one-way functions

Lecturer: João Ribeiro

1 Introduction

We saw that constructing a PRG with expansion $\ell(n) > n$ suffices to construct a computationally secure encryption scheme encrypting $\ell(n)$-bit messages using an n-bit key. This is an improvement over the one-time pad because our keys are now potentially much shorter than the messages. We have also seen that to construct a PRG with *any* polynomial expansion $\ell(n)$ it suffices to come up with a permutation $f\colon \{0,1\}^* \to \{0,1\}^*$ (i.e., f is length-preserving and bijective) and a hardcore predicate $P\colon \{0,1\}^* \to \{0,1\}$ for f. In these notes we will discuss a concrete construction of such a PRG from the *discrete logarithm assumption*, and will also see that these objects are useful for cryptography beyond encryption.

2 The discrete logarithm problem

As we have mentioned in the lecture, we do not actually know how to construct PRGs unconditionally. Rather, what we *can* do is construct candidate PRGs whose security is rigorously based on the conjectured hardness of solving certain mathematical problems, in the sense that any efficient security break on the PRG can be translated into an efficient algorithm that solves the mathematical problem. Such a candidate PRG is only as good as the underlying hardness assumption.

We will construct a candidate PRG based on the hardness of the *discrete logarithm problem* in cyclic groups. Fix a finite cyclic group G with generator g. If these words are not familiar to you, please refer to the group theory crash course. For such a group G and $h \in G$, we call an integer $k \in \{0, 1, \ldots, |G| - 1\}$ the discrete logarithm of h in G if $h = g^k$. Since G is cyclic and g is a generator, every group element has a discrete logarithm, and it is unique.

The *discrete logarithm problem* for G is the following: Given $h = g^k$ for uniformly random $k \leftarrow \{0, 1, \ldots, |G| - 1\}$, find k.

2.1 The DLog assumption

The DLog assumption is just the assumption that the discrete logarithm problem above cannot be solved by efficient adversaries, except with negligible success probability.

More formally, consider the following discrete logarithm experiment $\mathsf{DLog}_{\mathcal{A},\mathcal{G}}(n)$ parameterized by the security parameter n, a group generation algorithm $\mathcal{G}$, and an adversary $\mathcal{A}$:

- $\mathcal{G}(1^n)$ is a PPT algorithm that outputs (G, q, g), where G is a cyclic group of order q and g is a generator of G. We require that there is an algorithm running in time polynomial in n that given $a, b \in G$ outputs $a * b$, where $*$ is the operation of G.
- Sample $h \leftarrow G$.
- The adversary computes $k \leftarrow \mathcal{A}(1^n, G, q, g, h)$ and wins if $g^k = h$. In this case we set $\mathsf{DLog}_{\mathcal{A},\mathcal{G}}(n) = 1$, and we set $\mathsf{DLog}_{\mathcal{A},\mathcal{G}}(n) = 0$ otherwse.

We say that the DLog problem is hard relative to $\mathcal{G}$ if for any PPT adversary $\mathcal{A}$ there exists a negligible function $\varepsilon(n)$ such that

$$\Pr[\mathsf{DLog}_{\mathcal{A},\mathcal{G}}(n) = 1] \leq \varepsilon(n).$$

When faced with a new computational assumption, it is important to explore its basic properties:

- First, we can ask whether the DLog problem can be hard relative to all families of cyclic groups. This is not true! There are families of cyclic groups (even with order $q \approx 2^n$) in which the discrete logarithm problem is very easy. For example, if $G = (\mathbb{Z}_N, +)$, then $h = k \cdot g \pmod N$, and so it is easy to recover k if we know g and h. This means that we need to be careful about how we set up $\mathcal{G}$.

  It is widely believed that the DLog problem is hard (for classical algorithms) relative to multiplicative groups $\mathbb{Z}_p^*$ with p prime. In this case $\mathcal{G}(1^n)$ should output a prime $p \approx 2^n$, which also implicitly describes $\mathbb{Z}_p^*$.

  There is a deep body of research on algorithms for solving the DLog problem over $\mathbb{Z}_p^*$. In particular, we know how to solve this problem faster than brute-force, but the best known classical algorithms still require much more than polynomial time. If you are curious, see Section 8.2 of Katz-Lindell and Chapters 11 and 15 of Shoup's book.

  The DLog problem is also believed to be hard for groups consisting of points on elliptic curves. These are widely used in cryptography because, unlike for $\mathbb{Z}_p^*$, we do not know any subexponential time algorithm for solving the DLog problem over these groups.

- In the definition of the DLog experiment above, we only require the adversary $\mathcal{A}$ to solve the discrete logarithm problem on *random* group elements. This means that a PPT adversary $\mathcal{A}$ that solves the discrete logarithm problem *on average* breaks the DLog assumption. Potentially, this could be much easier than solving the discrete logarithm problem on *arbitrary* instances.

  It turns out that these two settings are equivalent for the discrete logarithm problem! More precisely, if we have a PPT algorithm $\mathcal{A}$ such that

  $$\Pr[\mathsf{DLog}_{\mathcal{A},\mathcal{G}}(n) = 1] \geq \frac{1}{p(n)}$$

  for some polynomial $p(n)$, then we can construct another PPT algorithm $\mathcal{A}'$ that, for an *arbitrary* $h \in G$, computes the discrete logarithm of h with high probability. The key insight behind $\mathcal{A}'$ is that we can "randomize" any discrete logarithm problem instance (the technical

term is "random self-reducibility"). If $h \in G$ is arbitrary and we pick $r \leftarrow \{0, 1, \ldots, q-1\}$, then $h' = g^r * h$ is a uniformly random element of G. Moreover, if we successfully compute the discrete logarithm k' of h', then we recover the discrete logarithm of h as $k = k' - r \pmod q$. This suggests the following strategy for $\mathcal{A}'$ on input an arbitrary $h \in G$:

1. With some hindsight, define the polynomial $s(n) = n \cdot p(n)$.
2. For $i = 1, \ldots, s(n)$:
   (a) Sample $r_i \leftarrow \{0, 1, \ldots, q-1\}$ and compute $h_i = g^{r_i} * h$.
   (b) Obtain a guess for the discrete logarithm of h_i as $k_i \leftarrow \mathcal{A}(1^n, G, q, g, h_i)$.
   (c) Check whether $g^{k_i} = h_i$. If this holds, stop and output $k = k_i - r_i \pmod q$.
3. If the check failed in all iterations, output a random $k \leftarrow \{0, 1, \ldots, q-1\}$.

This procedure is clearly PPT since $\mathcal{A}$ is PPT. We now upper bound the failure probability of $\mathcal{A}'$. Observe that $\mathcal{A}'$ fails to find the discrete logarithm of h only if $\mathcal{A}$ fails to find the discrete logarithm of h_i in every iteration i. Since, the h_i's are uniformly distributed over G, the probability that $\mathcal{A}$ fails in the i-th iteration is at most $1 - \frac{1}{p(n)}$. Since the h_i's are independent, this means that the failure probability of $\mathcal{A}'$ is at most

$$\left(1 - \frac{1}{p(n)}\right)^{s(n)} \leq e^{-\frac{s(n)}{p(n)}} = e^{-n},$$

which uses the inequality $1 + x \leq e^x$ valid for all $x \in \mathbb{R}$. In other words, $\mathcal{A}'$ succeeds in outputting the discrete logarithm of h with probability at least $1 - e^{-n}$.

2.2 What about quantum computers?

The relevance of our discussion hinges on the assumption that the DLog problem is hard relative to some appropriate family of groups. However, an astonishing breakthrough by Shor [Sho97] put a dent in this assumption by giving a polynomial-time algorithm that solves the DLog problem *using a quantum computer*.

So, why are we still talking about the DLog assumption? The first reason is that scalable quantum computing is still a work in progress. We still believe that the DLog assumption holds against *classical* algorithms, and cryptography based on DLog underlies many real-world systems.[1] The second reason is pedagogical – I think that the DLog assumption is a nice way of introducing students to basing cryptography on the hardness of mathematical problems. Later on we will construct cryptographic schemes based on the hardness of problems conjectured to be hard even for quantum computers.

[1] There are ongoing efforts by NIST and other standardization bodies to migrate from cryptographic schemes based on assumptions broken by quantum computers to new schemes based on assumptions conjectured to be hard for quantum computers – this is what we call *post-quantum cryptography*.

3 A PRG from the DLog assumption

Now that we have set up things appropriately, we will finally construct (conditionally!) our first PRG (this constructions is due to Blum and Micali [BM84]). Fix the security parameter n and let $(G = \mathbb{Z}_p^*, q = p-1, g) \leftarrow \mathcal{G}(1^n)$. We take our permutation to be $f(k) = g^k$. Note that $f(k)$ is efficiently computable by repeated squaring.

Then, the predicate[2] $P(k) = \mathbf{1}_{\{k \leq \frac{q}{2}\}}$ is known to be a hardcore predicate for f, provided that the DLog problem is hard relative to $\mathcal{G}$. We give some brief intuition about why this is true.

Suppose that $\mathcal{A}$ actually computes $P(k)$ perfectly, i.e., $\mathcal{A}(1^n, g^k) = P(k)$ holds with probability 1 over the choice of k. In this case, we can use $\mathcal{A}$ to perform binary search and find k in $O(\log q) = \text{poly}(n)$ steps. Indeed, if $h = g^k$ then we can get $P(k) = \mathcal{A}(1^n, h)$, which tells us whether $k < q/2$ or not. If $k < q/2$, then we efficiently compute[3] $h_1 = h \cdot g^{q/4}$ and $P(k + q/4) = \mathcal{A}(1^n, h_1)$, which will equal 1 if and only if $k < q/4$, and so on. In general, if we know that $k \in [a, b]$, then we compute $h' = h \cdot g^{\frac{b-a}{2}} = g^{k'}$ with $k' = k + \frac{b-a}{2}$. From $\mathcal{A}(1^n, h') = P(k')$ we can then determine whether $k < \frac{b-a}{2}$ or not. Of course, this is not a full proof because $\mathcal{A}$ may only guess $P(k)$ with probability slightly larger than $1/2$, and not probability 1.

4 Hardness of inverting efficiently computable functions and minimal assumptions in cryptography

It is useful to reflect on what we have seen above. The DLog assumption relative to $\mathcal{G}$ tells us that the permutation $f(k) = g^k$, which is efficiently computable, cannot be inverted efficiently on average. Then, we discussed how the hardness of inverting this permutation implies the existence of a hardcore predicate for f (and so it also implies the existence of PRGs with any polynomial expansion based on the conjectured hardness of inverting this permutation). The full proof for this in [BM84] uses specific properties of the discrete logarithm problem.

Let's define "easy to compute but hard to invert" more formally.

Definition 1 (One-way function) *A function $f : \{0,1\}^* \to \{0,1\}^*$ is said to be a* one-way function *(OWF) if it is:*

- ***Easy to compute:*** *There exists a deterministic polynomial time algorithm that given x outputs $f(x)$.*
- ***Hard to invert:*** *For any PPT algorithm $\mathcal{A}$ there exists a negligible function $\varepsilon(n)$ such that*

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

If f is a permutation, then we say that f is a one-way permutation *(OWP).*

[2] We use the notation $\mathbf{1}_{\{E\}}$ to denote a value that is 1 if E holds and 0 otherwise.

[3] For simplicity, we ignore ceilings and floors in the exponents.

We can rephrase the DLog assumption relative to $\mathcal{G}$ in terms of this definition. It states that the function $f(k) = g^k$ is a one-way permutation.

It is natural to wonder whether the connection between one-way functions and functions with hardcore predicates holds more generally. In fact, OWFs have appeared in our discussions before, although implicitly. Any PRG G with expansion $\ell(n) \geq 2n$ is an OWF, and any permutation f with a hardcore predicate P is also an OWP (I leave proving this as homework). Therefore, the existence of OWFs is a necessary assumption for the construction of the objects we have been discussing (PRGs and permutations with hardcore predicates).

Given that we only have conditional constructions of cryptographic objects, it makes sense to search for the *minimal assumption* on which we can base our constructions. This type of questions falls into a topic called *complexity-theoretic cryptography*. Since the existence of OWFs is necessary for the existence of the objects we have seen so far, we can ask whether it is also *sufficient*. More precisely, can any OWF be used to obtain a function with a hardcore predicate? The answer to this question turns out to be "yes"! Later on we will establish this result for OWPs – this is a weaker version of the Goldreich-Levin theorem [GL89]. This theorem has important applications well beyond its original scope (in coding theory and learning theory, for example).

The existence of one-way functions (equivalently, the existence of PRGs) is a minimal assumption for several interesting cryptographic objects. Remarkably, many other fundamental cryptographic objects we will discuss later on actually seem to require even stronger minimal assuptions!

References

[BM84] Manuel Blum and Silvio Micali. How to generate cryptographically strong sequences of pseudorandom bits. *SIAM Journal on Computing*, 13(4):850–864, 1984.

[GL89] Oded Goldreich and Leonid A. Levin. A hard-core predicate for all one-way functions. In *Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing*, STOC '89, page 25–32, New York, NY, USA, 1989. Association for Computing Machinery.

[Sho97] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, 1997.