# Chapter 10

# Number-Theoretic Primitives

Number theory is a source of several computational problems that serve as primitives in the design of cryptographic schemes. Asymmetric cryptography in particular relies on these primitives. As with other beasts that we have been calling "primitives," these computational problems exhibit some intractability features, but by themselves do not solve any cryptographic problem directly relevant to a user security goal. But appropriately applied, they become useful to this end. In order to later effectively exploit them it is useful to first spend some time understanding them.

This understanding has two parts. The first is to provide precise definitions of the various problems and their measures of intractability. The second is to look at what is known or conjectured about the computational complexity of these problems.

There are two main classes of primitives. The first class relates to the discrete logarithm problem over appropriate groups, and the second to the factoring of composite integers. We look at them in turn.

This chapter assumes some knowledge of computational number theory as covered in the chapter on Computational Number Theory.

## 10.1 Discrete logarithm related problems

Let $G$ be a cyclic group and let $g$ be a generator of $G$. Recall this means that $G = \{g^0, g^1, \ldots, g^{m-1}\}$, where $m = |G|$ is the order of $G$. The discrete logarithm function $\mathrm{DLog}_{G,g}$: $G \to \mathbf{Z}_m$ takes input a group element $a$ and returns the unique $i \in \mathbf{Z}_m$ such that $a = g^i$. There are several computational problems related to this function that are used as primitives.

### 10.1.1 Informal descriptions of the problems

The computational problems we consider in this setting are summarized in Fig. 10.1. In all cases, we are considering an attacker that knows the group $G$ and the generator $g$. It is given the quantities listed in the column labeled "given," and is trying to compute the quantities, or answer the question, listed in the column labeled "figure out."

The most basic problem is the discrete logarithm (DL) problem. Informally stated, the attacker is given as input some group element $X$, and must compute $\mathrm{DLog}_{G,g}(X)$. This problem is conjectured to be computationally intractable in suitable groups $G$.

| **Problem** | **Given** | **Figure out** |
|---|---|---|
| Discrete logarithm (DL) | $g^x$ | $x$ |
| Computational Diffie-Hellman (CDH) | $g^x, g^y$ | $g^{xy}$ |
| Decisional Diffie-Hellman (DDH) | $g^x, g^y, g^z$ | Is $z \equiv xy \pmod{\lvert G\rvert}$? |

Figure 10.1: An informal description of three discrete logarithm related problems over a cyclic group $G$ with generator $g$. For each problem we indicate the input to the attacker, and what the attacker must figure out to "win." The formal definitions are in the text.

---

One might imagine "encrypting" a message $x \in \mathbf{Z}_m$ by letting $g^x$ be the ciphertext. An adversary wanting to recover $x$ is then faced with solving the discrete logarithm problem to do so. However, as a form of encryption, this has the disadvantage of being non-functional, because an intended recipient, namely the person to whom the sender is trying to communicate $x$, is faced with the same task as the adversary in attempting to recover $x$.

The Diffie-Hellman (DH) problems first appeared in the context of secret key exchange. Suppose two parties want to agree on a key which should remain unknown to an eavesdropping adversary. The first party picks $x \xleftarrow{\$} \mathbf{Z}_m$ and sends $X = g^x$ to the second party; the second party correspondingly picks $y \xleftarrow{\$} \mathbf{Z}_m$ and sends $Y = g^y$ to the first party. The quantity $g^{xy}$ is called the DH-key corresponding to $X, Y$. We note that

$$Y^x \;=\; g^{xy} \;=\; X^y \,. \tag{10.1}$$

Thus the first party, knowing $Y, x$, can compute the DH key, as can the second party, knowing $X, y$. The adversary sees $X, Y$, so to recover the DH-key the adversary must solve the Computational Diffie-Hellman (CDH) problem, namely compute $g^{xy}$ given $X = g^x$ and $Y = g^y$. Similarly, we will see later a simple asymmetric encryption scheme, based on Equation (10.1), where recovery of the encrypted message corresponds to solving the CDH problem.

The obvious route to solving the CDH problem is to try to compute the discrete logarithm of either $X$ or $Y$ and then use Equation (10.1) to obtain the DH key. However, there might be other routes that do not involve computing discrete logarithms, which is why CDH is singled out as a computational problem in its own right. This problem appears to be computationally intractable in a variety of groups.

We have seen before that security of a cryptographic scheme typically demands much more than merely the computational intractability of recovery of some underlying key. The computational intractability of the CDH problem turns out to be insufficient to guarantee the security of many schemes based on DH keys, including the secret key exchange protocol and encryption scheme mentioned above. The Decisional Diffie-Hellman (DDH) problem provides the adversary with a task that can be no harder, but possibly easier, than solving the CDH problem, namely to tell whether or not a given group element $Z$ is the DH key corresponding to given group elements $X, Y$. This problem too appears to be computationally intractable in appropriate groups.

We now proceed to define the problems more formally. Having done that we will provide more specific discussions about their hardness in various different groups and their relations to each other.

### 10.1.2 The discrete logarithm problem

The description of the discrete logarithm problem given above was that the adversary is given as input some group element $X$, and is considered successful if it can output $\mathrm{DLog}_{G,g}(X)$. We would like to associate to a specific adversary $A$ some advantage function measuring how well it does in solving this problem. The measure adopted is to look at the fraction of group elements for which the adversary is able to compute the discrete logarithm. In other words, we imagine the group element $X$ given to the adversary as being drawn at random.

**Definition 10.1.1** Let $G$ be a cyclic group of order $m$, let $g$ be a generator of $G$, and let $A$ be an algorithm that returns an integer in $\mathbf{Z}_m$. We consider the following experiment:

Experiment $\mathbf{Exp}^{\mathrm{dl}}_{G,g}(A)$
$x \xleftarrow{\$} \mathbf{Z}_m$ ; $X \leftarrow g^x$
$\overline{x} \leftarrow A(X)$
If $g^{\overline{x}} = X$ then return 1 else return 0

The *dl-advantage* of $A$ is defined as

$$\mathbf{Adv}^{\mathrm{dl}}_{G,g}(A) \;=\; \Pr\left[\mathbf{Exp}^{\mathrm{dl}}_{G,g}(A) = 1\right] \;.\;\blacksquare$$

Recall that the discrete exponentiation function takes input $i \in \mathbf{Z}_m$ and returns the group element $g^i$. The discrete logarithm function is the inverse of the discrete exponentiation function. The definition above simply measures the one-wayness of the discrete exponentiation function according to the standard definition of one-way function. It is to emphasize this that certain parts of the experiment are written the way they are.

The discrete logarithm problem is said to hard in $G$ if the dl-advantage of any adversary of reasonable resources is small. Resources here means the time-complexity of the adversary, which includes its code size as usual.

### 10.1.3 The Computational Diffie-Hellman problem

As above, the transition from the informal description to the formal definition involves considering the group elements $X, Y$ to be drawn at random.

**Definition 10.1.2** Let $G$ be a cyclic group of order $m$, let $g$ be a generator of $G$, and let $A$ be an algorithm that returns an element of $G$. We consider the following experiment:

Experiment $\mathbf{Exp}^{\mathrm{cdh}}_{G,g}(A)$
$x \xleftarrow{\$} \mathbf{Z}_m$ ; $y \xleftarrow{\$} \mathbf{Z}_m$
$X \leftarrow g^x$ ; $Y \leftarrow g^y$
$Z \leftarrow A(X, Y)$
If $Z = g^{xy}$ then return 1 else return 0

The *cdh-advantage* of $A$ is defined as

$$\mathbf{Adv}^{\mathrm{cdh}}_{G,g}(A) \;=\; \Pr\left[\mathbf{Exp}^{\mathrm{cdh}}_{G,g}(A) = 1\right] \;.\;\blacksquare$$

Again, the CDH problem is said to be hard in $G$ if the cdh-advantage of any adversary of reasonable resources is small, where the resource in question is the adversary's time complexity.

### 10.1.4 The Decisional Diffie-Hellman problem

The formalization considers a "two worlds" setting. The adversary gets input $X, Y, Z$. In either world, $X, Y$ are random group elements, but the manner in which $Z$ is chosen depends on the world. In World 1, $Z = g^{xy}$ where $x = \mathrm{DLog}_{G,g}(X)$ and $y = \mathrm{DLog}_{G,g}(Y)$. In World 0, $Z$ is chosen at random from the group, independently of $X, Y$. The adversary must decide in which world it is. (Notice that this is a little different from the informal description of Fig. 10.1 which said that the adversary is trying to determine whether or not $Z = g^{xy}$, because if by chance $Z = g^{xy}$ in World 0, we will declare the adversary unsuccessful if it answers 1.)

**Definition 10.1.3** Let $G$ be a cyclic group of order $m$, let $g$ be a generator of $G$, let $A$ be an algorithm that returns a bit, and let $b$ be a bit. We consider the following experiments:

| Experiment $\mathbf{Exp}^{\text{ddh-1}}_{G,g}(A)$ | Experiment $\mathbf{Exp}^{\text{ddh-0}}_{G,g}(A)$ |
|---|---|
| $x \xleftarrow{\$} \mathbf{Z}_m$ | $x \xleftarrow{\$} \mathbf{Z}_m$ |
| $y \xleftarrow{\$} \mathbf{Z}_m$ | $y \xleftarrow{\$} \mathbf{Z}_m$ |
| $z \leftarrow xy \bmod m$ | $z \xleftarrow{\$} \mathbf{Z}_m$ |
| $X \leftarrow g^x$ ; $Y \leftarrow g^y$ ; $Z \leftarrow g^z$ | $X \leftarrow g^x$ ; $Y \leftarrow g^y$ ; $Z \leftarrow g^z$ |
| $d \leftarrow A(X, Y, Z)$ | $d \leftarrow A(X, Y, Z)$ |
| Return $d$ | Return $d$ |

The *ddh-advantage* of $A$ is defined as

$$\mathbf{Adv}^{\text{ddh}}_{G,g}(A) \;=\; \Pr\left[\mathbf{Exp}^{\text{ddh-1}}_{G,g}(A) = 1\right] - \Pr\left[\mathbf{Exp}^{\text{ddh-0}}_{G,g}(A) = 1\right] \;. \quad \blacksquare$$

Again, the DDH problem is said to be hard in $G$ if the ddh-advantage of any adversary of reasonable resources is small, where the resource in question is the adversary's time complexity.

### 10.1.5 Relations between the problems

Relative to a fixed group $G$ and generator $g$ for $G$, if you can solve the DL problem then you can solve the CDH problem, and if you can solve the CDH problem then you can solve the DDH problem. So if DL is easy then CDH is easy, and if CDH is easy then DDH is easy. Equivalently, if DDH is hard then CDH is hard, and if CDH is hard then DL is hard.

We note that the converses of these statements are not known to be true. There are groups where DDH is easy, while CDH and DL appear to be hard. (We will see examples of such groups later.) Correspondingly, there could be groups where CDH is easy but DL is hard.

The following Proposition provides the formal statement and proof corresponding to the above claim that if you can solve the DL problem then you can solve the CDH problem, and if you can solve the CDH problem then you can solve the DDH problem.

**Proposition 10.1.4** Let $G$ be a cyclic group and let $g$ be a generator of $G$. Let $A_{\text{dl}}$ be an adversary (against the DL problem). Then there exists an adversary $A_{\text{cdh}}$ (against the CDH problem) such that

$$\mathbf{Adv}^{\text{dl}}_{G,g}(A_{\text{dl}}) \;\le\; \mathbf{Adv}^{\text{cdh}}_{G,g}(A_{\text{cdh}}) \;. \tag{10.2}$$

Furthermore the running time of $A_{\text{cdh}}$ is the that of $A_{\text{dl}}$ plus the time to do one exponentiation in $G$. Similarly let $A_{\text{cdh}}$ be an adversary (against the CDH problem). Then there exists an adversary $A_{\text{ddh}}$ (against the DDH problem) such that

$$\mathbf{Adv}^{\text{cdh}}_{G,g}(A_{\text{cdh}}) \;\le\; \mathbf{Adv}^{\text{ddh}}_{G,g}(A_{\text{ddh}}) + \frac{1}{|G|} \;. \tag{10.3}$$

Furthermore the running time of $A_{\rm ddh}$ is the same as that of $A_{\rm cdh}$. ▮

**Proof of Proposition 10.1.4:** Adversary $A_{\rm cdh}$ works as follows:

Adversary $A_{\rm cdh}(X, Y)$
  $\overline{x} \leftarrow A(X)$
  $Z \leftarrow Y^{\overline{x}}$
  Return $Z$

Let $x = \mathrm{DLog}_{G,g}(X)$ and $y = \mathrm{DLog}_{G,g}(y)$. If $A_{\rm dl}$ is successful then its output $\overline{x}$ equals $x$. In that case

$$Y^{\overline{x}} \;=\; Y^x = (g^y)^x = g^{yx} = g^{xy}$$

is the correct output for $A_{\rm cdh}$. This justifies Equation (10.2).

We now turn to the second inequality in the proposition. Adversary $A_{\rm ddh}$ works as follows:

Adversary $A_{\rm ddh}(X, Y, Z)$
  $\overline{Z} \leftarrow B(X, Y)$
  If $\overline{Z} = Z$ then return 1 else return 0

We claim that

$$\begin{aligned} \Pr\left[\mathbf{Exp}^{\text{ddh-1}}_{G,g}(A_{\rm ddh}) = 1\right] &= \mathbf{Adv}^{\rm cdh}_{G,g}(A_{\rm cdh}) \\ \Pr\left[\mathbf{Exp}^{\text{ddh-0}}_{G,g}(A_{\rm ddh}) = 1\right] &= \frac{1}{|G|}\,, \end{aligned}$$

which implies Equation (10.3). To justify the above, let $x = \mathrm{DLog}_{G,g}(X)$ and $y = \mathrm{DLog}_{G,g}(y)$. If $A_{\rm cdh}$ is successful then its output $\overline{Z}$ equals $g^{xy}$, so in world 1, $A_{\rm ddh}$ returns 1. On the other hand in world 0, $Z$ is uniformly distributed over $G$ and hence has probability $1/|G|$ of equalling $\overline{Z}$. ▮

## 10.2 The choice of group

The computational complexity of the above problems depends of course on the choice of group $G$. (But not perceptibly on the choice of generator $g$.) The issues are the type of group, and also its size. Let us look at some possibilities.

### 10.2.1 General groups

For any "reasonable" group $G$, there is an algorithm that can solve the discrete logarithm problem in time $|G|^{1/2} \cdot O(|p|^3)$. (The exceptions are groups lacking succinct representations of group elements, and we will not encounter such groups here.) In thinking about this running time we neglect the $|p|^3$ factor since it is very small compared to $|G|^{1/2}$, so that we view this as a $O(|G|^{1/2})$ algorithm.

There are several different algorithms with this running time. Shank's baby-step giant-step algorithm is the simplest, and is deterministic. Pollard's algorithm is randomized, and, although taking time on the same order as that taken by Shank's algorithm, is more space efficient, and preferred in practice.

Let us present Shank's baby-step giant-step algorithm. Let $m = |G|$ and let $n = \lceil \sqrt{m} \rceil$. Given $X = g^x$ we seek $x$. We note that there exist integers $x_0, x_1$ such that $0 \le x_0, x_1 \le n$ and

$x = nx_1 + x_0$. This means that $g^{nx_1+x_0} = X$, or $Xg^{-x_0} = (g^n)^{x_1}$. The idea of the algorithm is to compute two lists:

$$Xg^{-b} \quad \text{for} \quad b = 0, 1, \ldots, n$$
$$(g^n)^a \quad \text{for} \quad a = 0, 1, \ldots, n$$

and then find a group element that is contained in both lists. The corresponding values of $a, b$ satisfy $Xg^{-b} = (g^n)^a$, and thus $\mathrm{DLog}_{G,g}(X) = an + b$. The details follow.

Algorithm $A_{\mathrm{bsgs}}(X)$
  $n \leftarrow \lceil \sqrt{m} \rceil$ ; $N \leftarrow g^n$
  For $b = 0, \ldots, n$ do $B[Xg^{-b}] \leftarrow b$
  For $a = 0, \ldots, n$ do
    $Y \leftarrow N^a$
    If $B[Y]$ is defined then $x_0 \leftarrow B[Y]$ ; $x_1 \leftarrow a$
  Return $ax_1 + x_0$

This algorithm is interesting because it shows that there is a better way to compute the discrete logarithm of $X$ than to do an exhaustive search for it. However, it does not yield a practical discrete logarithm computation method, because one can work in groups large enough that an $O(|G|^{1/2})$ algorithm is not really feasible. There are however better algorithms in some specific groups.

### 10.2.2 Integers modulo a prime

Naturally, the first specific group to consider is the integers modulo a prime, which we know is cyclic. So let $G = \mathbf{Z}_p^*$ for some prime $p$ and let $g$ be a generator of $g$. We consider the different problems in turn.

We begin by noting that the Decisional Diffie-Hellman problem is easy in this group. Some indication of this already appeared in the chapter on Computational Number Theory. In particular we saw there that the DH key $g^{xy}$ is a square with probability 3/4 and a non-square with probability 1/4 if $x, y$ are chosen at random from $\mathbf{Z}_{p-1}$. However, we know that a random group element is a square with probability 1/2. Thus, a strategy to tell which world we are in when given a triple $X, Y, Z$ is to test whether or not $Z$ is a square mod $p$. If so, bet on World 1, else on World 0. (We also know that the Jacobi symbol can be computed via an exponentiation mod $p$, so testing for squares can be done efficiently, specifically in cubic time.) A computation shows that this adversary has advantage 1/4, enough to show that the DDH problem is easy. The Proposition below presents a slightly better attack that achieves advantage 1/2, and provides the details of the analysis.

**Proposition 10.2.1** Let $p \geq 3$ be a prime, let $G = \mathbf{Z}_p^*$, and let $g$ be a generator of $G$. Then there is an adversary $A$, with running time $O(|p|^3)$ such that

$$\mathbf{Adv}^{\mathrm{ddh}}_{G,g}(A) \;=\; \frac{1}{2}\,. \quad \blacksquare$$

**Proof of Proposition 10.2.1:** The input to our adversary $A$ is a triple $X, Y, Z$ of group elements, and the adversary is trying to determine whether $Z$ was chosen as $g^{xy}$ or as a random group element, where $x, y$ are the discrete logarithms of $X$ and $Y$, respectively. We know that if we know $J_p(g^x)$ and $J_p(g^y)$, we can predict $J_p(g^{xy})$. Our adversary's strategy is to compute $J_p(g^x)$ and $J_p(g^y)$ and then see whether or not the challenge value $Z$ has the Jacobi symbol value that $g^{xy}$ ought to have. In more detail, it works as follows:

Adversary $A(X, Y, Z)$
  If $J_p(X) = 1$ or $J_p(Y) = 1$
    Then $s \leftarrow 1$ Else $s \leftarrow -1$
  If $J_p(Z) = s$ then return 1 else return 0

We know that the Jacobi symbol can be computed via an exponentiation modulo $p$, which we know takes $O(|p|^3)$ time. Thus, the time-complexity of the above adversary is $O(|p|^3)$. We now claim that

$$\begin{aligned} \Pr\left[\mathbf{Exp}^{\text{ddh-1}}_{G,g}(A) = 1\right] &= 1 \\ \Pr\left[\mathbf{Exp}^{\text{ddh-0}}_{G,g}(A) = 1\right] &= \frac{1}{2}\,. \end{aligned}$$

Subtracting, we get

$$\mathbf{Adv}^{\text{ddh}}_{G,g}(A) \;=\; \Pr\left[\mathbf{Exp}^{\text{ddh-1}}_{G,g}(A) = 1\right] - \Pr\left[\mathbf{Exp}^{\text{ddh-0}}_{G,g}(A) = 1\right] \;=\; 1 - \frac{1}{2} = \frac{1}{2}$$

as desired. Let us now see why the two equations above are true.

Let $x = \text{DLog}_{G,g}(X)$ and $y = \text{DLog}_{G,g}(Y)$. We know that the value $s$ computed by our adversary $A$ equals $J_p(g^{xy} \bmod p)$. But in World 1, $Z = g^{xy} \bmod p$, so our adversary will always return 1. In World 0, $Z$ is distributed uniformly over $G$, so

$$\Pr\left[J_p(Z) = 1\right] \;=\; \Pr\left[J_p(Z) = -1\right] \;=\; \frac{(p-1)/2}{p-1} \;=\; \frac{1}{2}\,.$$

Since $s$ is distributed independently of $Z$, the probability that $J_p(Z) = s$ is 1/2. ▌

Now we consider the CDH and DL problems. It appears that the best approach to solving the CDH in problem in $\mathbf{Z}_p^*$ is via the computation of discrete logarithms. (This has not been proved in general, but there are proofs for some special classes of primes.) Thus, the main question is how hard is the computation of discrete logarithms. This depends both on the size and structure of $p$.

The currently best algorithm is the GNFS (General Number Field Sieve) which has a running time of the form

$$O(e^{(C+o(1))\cdot \ln(p)^{1/3}\cdot(\ln\ln(p))^{2/3}}) \tag{10.4}$$

where $C \approx 1.92$. For certain classes of primes, the value of $C$ is even smaller. These algorithms are heuristic, in the sense that the run time bounds are not proven, but appear to hold in practice.

If the prime factorization of the order of the group is known, the discrete logarithm problem over the group can be decomposed into a set of discrete logarithm problems over subgroups. As a result, if $p-1 = p_1^{\alpha_1} \cdots p_n^{\alpha_n}$ is the prime factorization of $p-1$, then the discrete logarithm problem in $\mathbf{Z}_p^*$ can be solved in time on the order of

$$\sum_{i=1}^{n} \alpha_i \cdot (\sqrt{p_i} + |p|)\;.$$

If we want the discrete logarithm problem in $\mathbf{Z}_p^*$ to be hard, this means that it must be the case that at least one of the prime factors $p_i$ of $p-1$ is large enough that $\sqrt{p_i}$ is large.

The prime factorization of $p-1$ might be hard to compute given only $p$, but in fact we usually choose $p$ in such a way that we know the prime factorization of $p-1$, because it is this that gives us a way to find a generator of the group $\mathbf{Z}_p^*$, as discussed in the chapter on Computational Number Theory So the above algorithm is quite relevant.

From the above, if we want to make the DL problem in $\mathbf{Z}_p^*$ hard, it is necessary to choose $p$ so that it is large and has at least one large prime factor. A common choice is $p = sq + 1$ where $s \geq 2$ is some small integer (like $s = 2$) and $q$ is a prime. In this case, $p - 1$ has the factor $q$, which is large.

Precise estimates of the size of a prime necessary to make a discrete logarithm algorithm infeasible are hard to make based on asymptotic running times of the form given above. Ultimately, what actual implementations can accomplish is the most useful data. In April 2001, it was announced that discrete logarithms had been computed modulo a 120 digit (ie. about 400 bit) prime (Joux and Lercier, 2001). The computation took 10 weeks and was done on a 525MHz quadri-processor Digital Alpha Server 8400 computer. The prime $p$ did not have any special structure that was exploited, and the algorithm used was the GNFS. A little earlier, discrete logarithms had been computed modulo a slightly larger prime, namely a 129 digit one, but this had a special structure that was exploited [1].

Faster discrete logarithm computation can come from many sources. One is exploiting parallelism and the paradigm of distributing work across available machines on the Internet. Another is algorithmic improvements. A reduction in the constant $C$ of Equation (10.4) has important impact on the running time. A reduction in the exponents from $1/3, 2/3$ to $1/4, 3/4$ would have an even greater impact. There are also threats from hardware approaches such as the design of special purpose discrete logarithm computation devices. Finally, the discrete logarithm probably can be solved in polynomial time with a quantum computer. Whether a quantum computer can be built is not known.

Predictions are hard to make. In choosing a prime $p$ for cryptography over $\mathbf{Z}_p^*$, the security risks must be weighed against the increase in the cost of computations over $\mathbf{Z}_p^*$ as a function of the size of $p$.

### 10.2.3 Other groups

In elliptic curve groups, the best known algorithm is the $O(\sqrt{|G|})$ one mentioned above. Thus, it is possible to use elliptic curve groups of smaller size than groups of integers modulo a prime for the same level of security, leading to improved efficiency for implementing discrete log based cryptosystem.

## 10.3 The RSA system

The RSA system is the basis of the most popular public-key cryptography solutions. Here we provide the basic mathematical and computational background that will be used later.

### 10.3.1 The basic mathematics

We begin with a piece of notation:

**Definition 10.3.1** Let $N, f \geq 1$ be integers. The *RSA function associated to* $N, f$ is the function $\mathsf{RSA}_{N,f}$: $\mathbf{Z}_N^* \to \mathbf{Z}_N^*$ defined by $\mathsf{RSA}_{N,f}(w) = w^f \bmod N$ for all $w \in \mathbf{Z}_N^*$. ∎

The RSA function associated to $N, f$ is thus simply exponentiation with exponent $f$ in the group $\mathbf{Z}_N^*$, but it is useful in the current context to give it a new name. The following summarizes a basic property of this function. Recall that $\varphi(N)$ is the order of the group $\mathbf{Z}_N^*$.

**Proposition 10.3.2** Let $N \geq 2$ and $e, d \in \mathbf{Z}^*_{\varphi(N)}$ be integers such that $ed \equiv 1 \pmod{\varphi(N)}$. Then the RSA functions $\mathsf{RSA}_{N,e}$ and $\mathsf{RSA}_{N,d}$ are both permutations on $\mathbf{Z}^*_N$ and, moreover, are inverses of each other, ie. $\mathsf{RSA}^{-1}_{N,e} = \mathsf{RSA}_{N,d}$ and $\mathsf{RSA}^{-1}_{N,d} = \mathsf{RSA}_{N,e}$. ▌

A permutation, above, simply means a bijection from $\mathbf{Z}^*_N$ to $\mathbf{Z}^*_N$, or, in other words, a one-to-one, onto map. The condition $ed \equiv 1 \pmod{\varphi(N)}$ says that $d$ is the inverse of $e$ in the group $\mathbf{Z}^*_{\varphi(N)}$.

**Proof of Proposition 10.3.2:** For any $x \in \mathbf{Z}^*_N$, the following hold modulo $N$:

$$\mathsf{RSA}_{N,d}(\mathsf{RSA}_{N,e}(x)) \equiv (x^e)^d \equiv x^{ed} \equiv x^{ed \bmod \varphi(N)} \equiv x^1 \equiv x \ .$$

The third equivalence used the fact that $\varphi(N)$ is the order of the group $\mathbf{Z}^*_N$. The fourth used the assumed condition on $e, d$. Similarly, we can show that for any $y \in \mathbf{Z}^*_N$,

$$\mathsf{RSA}_{N,e}(\mathsf{RSA}_{N,d}(y)) \equiv y$$

modulo $N$. These two facts justify all the claims of the Proposition. ▌

With $N, e, d$ as in Proposition 10.3.2 we remark that

- For any $x \in \mathbf{Z}^*_N$: $\mathsf{RSA}_{N,e}(x) = \text{MOD-EXP}(x, e, N)$ and so one can efficiently compute $\mathsf{RSA}_{N,e}(x)$ given $N, e, x$.
- For any $y \in \mathbf{Z}^*_N$: $\mathsf{RSA}_{N,d}(y) = \text{MOD-EXP}(y, d, N)$ and so one can efficiently compute $\mathsf{RSA}_{N,d}(y)$ given $N, d, y$.

We now consider an adversary that is given $N, e, y$ and asked to compute $\mathsf{RSA}^{-1}_{N,e}(y)$. If it had $d$, this could be done efficiently by the above, but we do not give it $d$. It turns out that when the paremeters $N, e$ are properly chosen, this adversarial task appears to be computationally infeasible, and this property will form the basis of both asymmetric encryption schemes and digital signature schemes based on RSA. Our goal in this section is to lay the groundwork for these later applications by showing how RSA parameters can be chosen so as to make the above claim of computational difficulty true, and formalizing the sense in which it is true.

### 10.3.2 Generation of RSA parameters

We begin with a computational fact.

**Proposition 10.3.3** There is an $O(k^2)$ time algorithm that on inputs $\varphi(N), e$ where $e \in \mathbf{Z}^*_{\varphi(N)}$ and $N < 2^k$, returns $d \in \mathbf{Z}^*_{\varphi(N)}$ satisfying $ed \equiv 1 \pmod{\varphi(N)}$. ▌

**Proof of Proposition 10.3.3:** Since $d$ is the inverse of $e$ in the group $\mathbf{Z}^*_{\varphi(N)}$, the algorithm consists simply of running MOD-INV$(e, \varphi(N))$ and returning the outcome. Recall that the modular inversion algorithm invokes the extended-gcd algorithm as a subroutine and has running time quadratic in the bit-length of its inputs. ▌

To choose RSA parameters, one runs a generator. We consider a few types of geneators:

**Definition 10.3.4** A *modulus generator* with *associated security parameter* $k$ (where $k \geq 2$ is an integer) is a randomized algorithm that takes no inputs and returns integers $N, p, q$ satisfying:

1. $p, q$ are distinct, odd primes
2. $N = pq$
3. $2^{k-1} \leq N < 2^k$ (ie. $N$ has bit-length $k$).

An *RSA generator* with *associated security parameter* $k$ is a randomized algorithm that takes no inputs and returns a pair $((N,e),(N,p,q,d))$ such that the three conditions above are true, and, in addition,

**4.** $e, d \in \mathbf{Z}^*_{(p-1)(q-1)}$

**5.** $ed \equiv 1 \pmod{(p-1)(q-1)}$

We call $N$ an *RSA modulus*, or just modulus. We call $e$ the *encryption exponent* and $d$ the *decryption exponent.* ▮

Note that $(p-1)(q-1) = \varphi(N)$ is the size of the group $\mathbf{Z}^*_N$. So above, $e, d$ are relatively prime to the order of the group $\mathbf{Z}^*_N$. As the above indicates, we are going to restrict attention to numbers $N$ that are the product of two distinct odd primes. Condition (4) for the RSA generator translates to $1 \le e, d < (p-1)(q-1)$ and $\gcd(e,(p-1)(q-1)) = \gcd(d,(p-1)(q-1)) = 1$.

For parameter generation to be feasible, the generation algorithm must be efficient. There are many different possible efficient generators. We illustrate a few.

In modulus generation, we usually pick the primes $p, q$ at random, with each being about $k/2$ bits long. The corresponding modulus generator $\mathcal{K}^{\$}_{\mathrm{mod}}$ with associated security parameter $k$ works as follows:

Algorithm $\mathcal{K}^{\$}_{\mathrm{mod}}$
$\ell_1 \leftarrow \lfloor k/2 \rfloor$ ; $\ell_2 \leftarrow \lceil k/2 \rceil$
Repeat
$p \xleftarrow{\$} \{2^{\ell_1-1}, \ldots, 2^{\ell_1}-1\}$ ; $q \xleftarrow{\$} \{2^{\ell_2-1}, \ldots, 2^{\ell_2}-1\}$
Until the following conditions are all true:
– TEST-PRIME$(p) = 1$ and TEST-PRIME$(q) = 1$
– $p \ne q$
– $2^{k-1} \le N$
$N \leftarrow pq$
Return $(N,e),(N,p,q,d)$

Above, TEST-PRIME denotes an algorithm that takes input an integer and returns 1 or 0. It is designed so that, with high probability, the former happens when the input is prime and the latter when the input is composite.

Sometimes, we may want modulii product of primes having a special form, for example primes $p, q$ such that $(p-1)/2$ and $(q-1)/2$ are both prime. This corresponds to a different modulus generator, which works as above but simply adds, to the list of conditions tested to exit the loop, the conditions TEST-PRIME$((p-1)/2)) = 1$ and TEST-PRIME$((q-1)/2)) = 1$. There are numerous other possible modulus generators too.

An RSA generator, in addition to $N, p, q$, needs to generate the exponents $e, d$. There are several options for this. One is to first choose $N, p, q$, then pick $e$ at random subject to $\gcd(N, \varphi(N)) = 1$, and compute $d$ via the algorithm of Proposition 10.3.3. This *random-exponent RSA generator*, denoted $\mathcal{K}^{\$}_{\mathrm{rsa}}$, is detailed below:

Algorithm $\mathcal{K}^{\$}_{\mathrm{rsa}}$
$(N,p,q) \xleftarrow{\$} \mathcal{K}^{\$}_{\mathrm{mod}}$
$M \leftarrow (p-1)(q-1)$
$e \xleftarrow{\$} \mathbf{Z}^*_M$

Compute $d$ by running the algorithm of Proposition 10.3.3 on inputs $M, e$
Return $((N, e), (N, p, q, d))$

In order to speed-up computation of $\mathsf{RSA}_{N,e}$, however, we often like $e$ to be small. To enable this, we begin by setting $e$ to some small prime number like 3, and then picking the other parameters appropriately. In particular we associate to any odd prime number $e$ the following *exponent-e RSA generator*:

Algorithm $\mathcal{K}^e_{\text{rsa}}$
Repeat
$(N, p, q) \xleftarrow{\$} \mathcal{K}^{\$}_{\text{mod}}(k)$
Until
– $e < (p-1)$ and $e < (q-1)$
– $\gcd(e, (p-1)) = \gcd(e, (q-1)) = 1$
$M \leftarrow (p-1)(q-1)$
Compute $d$ by running the algorithm of Proposition 10.3.3 on inputs $M, e$
Return $((N, e), (N, p, q, d))$

### 10.3.3 One-wayness problems

The basic assumed security property of the RSA functions is one-wayness, meaning given $N, e, y$ it is hard to compute $\mathsf{RSA}^{-1}_{N,e}(y)$. One must be careful to formalize this properly though. The formalization chooses $y$ at random.

**Definition 10.3.5** Let $\mathcal{K}_{\text{rsa}}$ be an RSA generator with associated security parameter $k$, and let $A$ be an algorithm. We consider the following experiment:

Experiment $\mathbf{Exp}^{\text{ow-kea}}_{\mathcal{K}_{\text{rsa}}}(A)$
$((N, e), (N, p, q, d)) \xleftarrow{\$} \mathcal{K}_{\text{rsa}}$
$x \xleftarrow{\$} \mathbf{Z}^*_N$ ; $y \leftarrow x^e \bmod N$
$x' \xleftarrow{\$} A(N, e, y)$
If $x' = x$ then return 1 else return 0

The *ow-kea-advantage* of $A$ is defined as

$$\mathbf{Adv}^{\text{ow-kea}}_{\mathcal{K}_{\text{rsa}}}(A) \quad = \quad \Pr\left[\mathbf{Exp}^{\text{ow-kea}}_{\mathcal{K}_{\text{rsa}}}(A) = 1\right] . \quad \blacksquare$$

Above, "kea" stands for "known-exponent attack." We might also allow a chosen-exponent attack, abbreviated "cea," in which, rather than having the encryption exponent specified by the instance of the problem, one allows the adversary to choose it. The only condition imposed is that the adversary not choose $e = 1$.

**Definition 10.3.6** Let $\mathcal{K}_{\text{mod}}$ be a modulus generator with associated security parameter $k$, and let $A$ be an algorithm. We consider the following experiment:

Experiment $\mathbf{Exp}^{\text{ow-cea}}_{\mathcal{K}_{\text{rsa}}}(A)$
$(N, p, q) \xleftarrow{\$} \mathcal{K}_{\text{mod}}$
$y \xleftarrow{\$} \mathbf{Z}^*_N$
$(x, e) \xleftarrow{\$} A(N, y)$
If $x^e \equiv y \pmod N$ and $e > 1$
then return 1 else return 0.

The *ow-cea-advantage* of $A$ is defined as

$$\mathbf{Adv}^{\text{ow-cea}}_{\mathcal{K}_{\text{mod}}}(A) \;=\; \Pr\left[\mathbf{Exp}^{\text{ow-cea}}_{\mathcal{K}_{\text{mod}}}(A) = 1\right] . \;\blacksquare$$

## 10.4 Historical notes

## 10.5 Exercises and Problems

# Bibliography

[1] T. Denny and D. Weber The solution of Mccurley's discrete logchallenge. *Advances in Cryptology – CRYPTO '98*, Lecture Notes in Computer Science Vol. 1462, H. Krawczyk ed., Springer-Verlag, 1998.