13

Computational problems related to quadratic residues

13.1 Computing the Jacobi symbol

Suppose we are given an odd, positive integer n, along with an integer a, and we want to compute the Jacobi symbol $(a \mid n)$. Theorem 12.8 suggests the following algorithm:

$t \leftarrow 1$
repeat
    // *loop invariant: n is odd and positive*
    $a \leftarrow a \bmod n$
    if $a = 0$
        if $n = 1$ return t else return 0
    compute a', h such that $a = 2^h a'$ and a' is odd
    if $h \not\equiv 0 \pmod 2$ and $n \not\equiv \pm 1 \pmod 8$ then $t \leftarrow -t$
    if $a' \not\equiv 1 \pmod 4$ and $n \not\equiv 1 \pmod 4$ then $t \leftarrow -t$
    $(a, n) \leftarrow (n, a')$
forever

That this algorithm correctly computes the Jacobi symbol $(a \mid n)$ follows directly from Theorem 12.8. Using an analysis similar to that of Euclid's algorithm, one easily sees that the running time of this algorithm is $O(\mathrm{len}(a)\,\mathrm{len}(n))$.

Exercise 13.1. Develop a "binary" Jacobi symbol algorithm, that is, one that uses only addition, subtractions, and "shift" operations, analogous to the binary gcd algorithm in Exercise 4.1.

Exercise 13.2. This exercise develops a probabilistic primality test based

on the Jacobi symbol. For odd integer $n > 1$, define

$$G_n := \{\alpha \in \mathbb{Z}_n^* : \alpha^{(n-1)/2} = [J_n(\alpha)]_n\},$$

where $J_n : \mathbb{Z}_n^* \to \{\pm 1\}$ is the Jacobi map.

(a) Show that G_n is a subgroup of $\mathbb{Z}_n^*$.

(b) Show that if n is prime, then $G_n = \mathbb{Z}_n^*$.

(c) Show that if n is composite, then $G_n \subsetneq \mathbb{Z}_n^*$.

(d) Based on parts (a)–(c), design and analyze an efficient probabilistic primality test that works by choosing a random, non-zero element $\alpha \in \mathbb{Z}_n$, and testing if $\alpha \in G_n$.

13.2 Testing quadratic residuosity

In this section, we consider the problem of testing whether a is a quadratic residue modulo n, for given integers a and n, from a computational perspective.

13.2.1 Prime modulus

For an odd prime p, we can test if an integer a is a quadratic residue modulo p by either performing the exponentiation $a^{(p-1)/2} \bmod p$ or by computing the Legendre symbol $(a \mid p)$. Assume that $0 \leq a < p$. Using a standard repeated squaring algorithm, the former method takes time $O(\mathrm{len}(p)^3)$, while using the Euclidean-like algorithm of the previous section, the latter method takes time $O(\mathrm{len}(p)^2)$. So clearly, the latter method is to be preferred.

13.2.2 Prime-power modulus

For an odd prime p, we know that a is a quadratic residue modulo p^e if and only if a is a quadratic residue modulo p. So this case immediately reduces to the previous case.

13.2.3 Composite modulus

For odd, composite n, if we know the factorization of n, then we can also determine if a is a quadratic residue modulo n by determining if it is a quadratic residue modulo each prime divisor p of n. However, without knowledge of this factorization (which is in general believed to be hard to compute), there is no efficient algorithm known. We can compute the Jacobi symbol $(a \mid n)$;

if this is -1 or 0, we can conclude that a is not a quadratic residue; otherwise, we cannot conclude much of anything.

13.3 Computing modular square roots

In this section, we consider the problem of computing a square root of a modulo n, given integers a and n, where a is a quadratic residue modulo n.

13.3.1 Prime modulus

Let p be an odd prime, and let a be an integer such that $0 < a < p$ and $(a \mid p) = 1$. We would like to compute a square root of a modulo p. Let $\alpha := [a]_p \in \mathbb{Z}_p^*$, so that we can restate our problem of that of finding $\beta \in \mathbb{Z}_p^*$ such that $\beta^2 = \alpha$, given $\alpha \in (\mathbb{Z}_p^*)^2$.

We first consider the special case where $p \equiv 3 \pmod 4$, in which it turns out that this problem can be solved very easily. Indeed, we claim that in this case

$$\beta := \alpha^{(p+1)/4}$$

is a square root of α—note that since $p \equiv 3 \pmod 4$, the number $(p+1)/4$ is an integer. To show that $\beta^2 = \alpha$, suppose $\alpha = \tilde{\beta}^2$ for some $\tilde{\beta} \in \mathbb{Z}_p^*$. We know that there is such a $\tilde{\beta}$, since we are assuming that $\alpha \in (\mathbb{Z}_p^*)^2$. Then we have

$$\beta^2 = \alpha^{(p+1)/2} = \tilde{\beta}^{p+1} = \tilde{\beta}^2 = \alpha,$$

where we used Fermat's little theorem for the third equality. Using a repeated-squaring algorithm, we can compute β in time $O(\mathrm{len}(p)^3)$.

Now we consider the general case, where we may have $p \not\equiv 3 \pmod 4$. Here is one way to efficiently compute a square root of α, assuming we are given, in addition to α, an auxiliary input $\gamma \in \mathbb{Z}_p^* \setminus (\mathbb{Z}_p^*)^2$ (how one obtains such a γ is discussed below).

Let us write $p - 1 = 2^h m$, where m is odd. For any $\delta \in \mathbb{Z}_p^*$, δ^m has multiplicative order dividing 2^h. Since $\alpha^{2^{h-1}m} = 1$, α^m has multiplicative order dividing 2^{h-1}. Since $\gamma^{2^{h-1}m} = -1$, γ^m has multiplicative order precisely 2^h. Since there is only one subgroup of $\mathbb{Z}_p^*$ of order 2^h, it follows that γ^m generates this subgroup, and that $\alpha^m = \gamma^{mx}$ for $0 \leq x < 2^h$ and x is even. We can find x by computing the discrete logarithm of α^m to the base γ^m, using the algorithm in §11.2.3. Setting $\kappa = \gamma^{mx/2}$, we have

$$\kappa^2 = \alpha^m.$$

We are not quite done, since we now have a square root of α^m, and not of α. Since m is odd, we may write $m = 2t + 1$ for some non-negative integer t. It then follows that

$$(\kappa\alpha^{-t})^2 = \kappa^2\alpha^{-2t} = \alpha^m\alpha^{-2t} = \alpha^{m-2t} = \alpha.$$

Thus, $\kappa\alpha^{-t}$ is a square root of α.

Let us summarize the above algorithm for computing a square root of $\alpha \in (\mathbb{Z}_p^*)^2$, assuming we are given $\gamma \in \mathbb{Z}_p^* \setminus (\mathbb{Z}_p^*)^2$, in addition to α:

> Compute positive integers m, h such that $p - 1 = 2^h m$ with m odd
> $\gamma' \leftarrow \gamma^m$, $\alpha' \leftarrow \alpha^m$
> Compute $x \leftarrow \log_{\gamma'} \alpha'$ // *note that* $0 \le x < 2^h$ *and* x *is even*
> $\beta \leftarrow (\gamma')^{x/2}\alpha^{-\lfloor m/2 \rfloor}$
> output β

The total amount of work done outside the discrete logarithm calculation amounts to just a handful of exponentiations modulo p, and so takes time $O(\mathrm{len}(p)^3)$. The time to compute the discrete logarithm is $O(h\,\mathrm{len}(h)\,\mathrm{len}(p)^2)$. So the total running time of this procedure is

$$O(\mathrm{len}(p)^3 + h\,\mathrm{len}(h)\,\mathrm{len}(p)^2).$$

The above procedure assumed we had at hand a non-square γ. If $h = 1$, which means that $p \equiv 3 \pmod 4$, then $(-1 \mid p) = -1$, and so we are done. However, we have already seen how to efficiently compute a square root in this case.

If $h > 1$, we can find a non-square γ using a probabilistic search algorithm. Simply choose γ at random, test if it is a square, and if so, repeat. The probability that a random element of $\mathbb{Z}_p^*$ is a square is $1/2$; thus, the expected number of trials until we find a non-square is 2, and hence the expected running time of this probabilistic search algorithm is $O(\mathrm{len}(p)^2)$.

Exercise 13.3. Let p be an odd prime, and let $f \in \mathbb{Z}_p[\mathtt{X}]$ be a polynomial with $0 \le \deg(f) \le 2$. Design and analyze an efficient, probabilistic algorithm that determines if f has any roots in $\mathbb{Z}_p$, and if so, finds all of the roots. Hint: see Exercise 9.14.

Exercise 13.4. Show that the following two problems are deterministic, poly-time equivalent (see discussion just above Exercise 11.10 in §11.3):

(a) Given an odd prime p and $\alpha \in (\mathbb{Z}_p^*)^2$, find $\beta \in \mathbb{Z}_p^*$ such that $\beta^2 = \alpha$.

(b) Given an odd prime p, find an element of $\mathbb{Z}_p^* \setminus (\mathbb{Z}_p^*)^2$.

Exercise 13.5. Design and analyze an efficient, deterministic algorithm that takes as input primes p and q, such that $q \mid (p-1)$, along with an element $\alpha \in \mathbb{Z}_p^*$, and determines whether or not $\alpha \in (\mathbb{Z}_p^*)^q$.

Exercise 13.6. Design and analyze an efficient, deterministic algorithm that takes as input primes p and q, such that $q \mid (p-1)$ but $q^2 \nmid (p-1)$, along with an element $\alpha \in (\mathbb{Z}_p^*)^q$, and computes a qth root of α, that is, an element $\beta \in \mathbb{Z}_p^*$ such that $\beta^q = \alpha$.

Exercise 13.7. We are given a positive integer n, two elements $\alpha, \beta \in \mathbb{Z}_n$, and integers e and f such that $\alpha^e = \beta^f$ and $\gcd(e, f) = 1$. Show how to efficiently compute $\gamma \in \mathbb{Z}_n$ such that $\gamma^e = \beta$. Hint: use the extended Euclidean algorithm.

Exercise 13.8. Design and analyze an algorithm that takes as input primes p and q, such that $q \mid (p-1)$, along with an element $\alpha \in (\mathbb{Z}_p^*)^q$, and computes a qth root of α. (Unlike Exercise 13.6, we now allow $q^2 \mid (p-1)$.) Your algorithm may be probabilistic, and should have an expected running time that is bounded by $q^{1/2}$ times a polynomial in $\mathrm{len}(p)$. Hint: the previous exercise may be useful.

Exercise 13.9. Let p be an odd prime, γ be a generator for $\mathbb{Z}_p^*$, and α be any element of $\mathbb{Z}_p^*$. Define

$$B(p, \gamma, \alpha) := \begin{cases} 1 & \text{if } \log_\gamma \alpha \geq (p-1)/2; \\ 0 & \text{if } \log_\gamma \alpha < (p-1)/2. \end{cases}$$

Suppose that there is an algorithm that efficiently computes $B(p, \gamma, \alpha)$ for all p, γ, α as above. Show how to use this algorithm as a subroutine in an efficient, probabilistic algorithm that computes $\log_\gamma \alpha$ for all p, γ, α as above. Hint: in addition to the algorithm that computes B, use algorithms for testing quadratic residuosity and computing square roots modulo p, and "read off" the bits of $\log_\gamma \alpha$ one at a time.

13.3.2 Prime-power modulus

Let p be an odd prime, let a be an integer relatively prime to p, and let $e > 1$ be an integer. We know that a is a quadratic residue modulo p^e if and only if a is a quadratic residue modulo p. Suppose that a is a quadratic residue modulo p, and that we have found an integer z such that $z^2 \equiv a \pmod{p}$, using, say, one of the procedures described in §13.3.1. From this, we can easily compute a square root of a modulo p^e using the following technique, which is known as **Hensel lifting**.

More generally, suppose we have computed an integer z such that $z^2 \equiv a \pmod{p^f}$, for some $f \geq 1$, and we want to find an integer $\hat{z}$ such that $\hat{z}^2 \equiv a \pmod{p^{f+1}}$. Clearly, if $\hat{z}^2 \equiv a \pmod{p^{f+1}}$, then $\hat{z}^2 \equiv a \pmod{p^f}$, and so $\hat{z} \equiv \pm z \pmod{p^f}$. So let us set $\hat{z} = z + p^f u$, and solve for u. We have

$$\hat{z}^2 \equiv (z + p^f u)^2 \equiv z^2 + 2zp^f u + p^{2f} u^2 \equiv z^2 + 2zp^f u \pmod{p^{f+1}}.$$

So we want to find integer u such that

$$2zp^f u \equiv a - z^2 \pmod{p^{f+1}}.$$

Since $p^f \mid (z^2 - a)$, by Theorem 2.5, the above congruence holds if and only if

$$2zu \equiv \frac{a - z^2}{p^f} \pmod{p}.$$

From this, we can easily compute the desired value u, since $\gcd(2z, p) = 1$.

By iterating the above procedure, starting with a square root of a modulo p, we can quickly find a square root of a modulo p^e. We leave a detailed analysis of the running time of this procedure to the reader.

Exercise 13.10. Suppose you are given a polynomial $f \in \mathbb{Z}[\mathtt{X}]$, along with a prime p and a root z of f modulo p, that is, an integer z such that $f(z) \equiv 0 \pmod{p}$. Further, assume that z is simple root of f modulo p, meaning that $\mathbf{D}(f)(z) \not\equiv 0 \pmod{p}$, where $\mathbf{D}(f)$ is the formal derivative of f. Show that for any integer $e \geq 1$, f has a root modulo p^e, and give an efficient procedure to find it. Also, show that the root modulo p^e is uniquely determined, in the following sense: if two such roots are congruent modulo p, then they are congruent modulo p^e.

13.3.3 Composite modulus

To find square roots modulo n, where n is an odd composite modulus, if we know the prime factorization of n, then we can use the above procedures for finding square roots modulo primes and prime powers, and then use the algorithm of the Chinese remainder theorem to get a square root modulo n.

However, if the factorization of n is not known, then there is no efficient algorithm known for computing square roots modulo n. In fact, one can show that the problem of finding square roots modulo n is at least as hard as the problem of factoring n, in the sense that if there is an efficient algorithm for

computing square roots modulo n, then there is an efficient (probabilistic) algorithm for factoring n.

Here is an algorithm to factor n, using a modular square-root algorithm as a subroutine. For simplicity, we assume that n is of the form $n = pq$, where p and q are distinct, odd primes. Choose β to be a random, non-zero element of $\mathbb{Z}_n$. If $d := \gcd(\mathrm{rep}(\beta), n) > 1$, then output d (recall that $\mathrm{rep}(\beta)$ denotes the canonical representative of β). Otherwise, set $\alpha := \beta^2$, and feed n and α to the modular square-root algorithm, obtaining a square root $\beta' \in \mathbb{Z}_n^*$ of α. If the square-root algorithm returns $\beta' \in \mathbb{Z}_n^*$ such that $\beta' = \pm\beta$, then output "failure"; otherwise, output $\gcd(\mathrm{rep}(\beta - \beta'), n)$, which is a non-trivial divisor of n.

Let us analyze this algorithm. If $d > 1$, we split n, so assume that $d = 1$, which means that $\beta \in \mathbb{Z}_n^*$. In this case, β is uniformly distributed over $\mathbb{Z}_n^*$, and α is uniformly distributed over $(\mathbb{Z}_n^*)^2$. Let us condition on an a *fixed* value of α, and on fixed random choices made by the modular square-root algorithm (in general, this algorithm may be probabilistic). In this conditional probability distribution, the value β' returned by the algorithm is completely determined. If $\theta : \mathbb{Z}_p \times \mathbb{Z}_q \to \mathbb{Z}_n$ is the ring isomorphism of the Chinese remainder theorem, and $\beta' = \theta(\beta'_1, \beta'_2)$, then in this conditional probability distribution, β is uniformly distributed over the four square roots of α, which we may write as $\theta(\pm\beta'_1, \pm\beta'_2)$.

With probability $1/4$, we have $\beta = \theta(\beta'_1, \beta'_2) = \beta'$, and with probability $1/4$, we have $\beta = \theta(-\beta'_1, -\beta'_2) = -\beta'$, and so with probability $1/2$, we have $\beta = \pm\beta'$, in which case we fail to factor n. However, with probability $1/4$, we have $\beta = \theta(-\beta'_1, \beta'_2)$, in which case $\beta - \beta' = \theta(-2\beta'_1, 0)$, and since $2\beta'_1 \neq 0$, we have $p \nmid \mathrm{rep}(\beta - \beta')$ and $q \mid \mathrm{rep}(\beta - \beta')$, and so $\gcd(\mathrm{rep}(\beta - \beta'), n) = q$. Similarly, with probability $1/4$, we have $\beta = \theta(\beta'_1, -\beta'_2)$, in which case $\beta - \beta' = \theta(0, -2\beta'_2)$, and since $2\beta'_2 \neq 0$, we have $p \mid \mathrm{rep}(\beta - \beta')$ and $q \nmid \mathrm{rep}(\beta - \beta')$, and so $\gcd(\mathrm{rep}(\beta - \beta'), n) = p$. Thus, with probability $1/2$, we have $\beta \neq \pm\beta'$, and $\gcd(\mathrm{rep}(\beta - \beta'), n)$ splits n.

Since we split n with probability $1/2$ conditioned on any fixed choice $\alpha \in (\mathbb{Z}_n^*)^2$ and any fixed random choices of the modular square-root algorithm, it follows that we split n with probability $1/2$ conditioned simply on the event that $\beta \in \mathbb{Z}_n^*$. Also, conditioned on the event that $\beta \notin \mathbb{Z}_n^*$, we split n with certainty, and so we may conclude that the above algorithm splits n with probability at least $1/2$.

Exercise 13.11. Generalize the algorithm above to efficiently factor arbi-

trary integers, given a subroutine that computes arbitrary modular square roots.

13.4 The quadratic residuosity assumption

Loosely speaking, the **quadratic residuosity (QR)** assumption is the assumption that it is hard to distinguish squares from non-squares in $\mathbb{Z}_n^*$, where n is of the form $n = pq$, and p and q are distinct primes. This assumption plays an important role in cryptography. Of course, since the Jacobi symbol is easy to compute, for this assumption to make sense, we have to restrict our attention to elements of $\ker(J_n)$, where $J_n : \mathbb{Z}_n^* \to \{\pm 1\}$ is the Jacobi map. We know that $(\mathbb{Z}_n^*)^2 \subseteq \ker(J_n)$ (see Exercise 12.2). Somewhat more precisely, the QR assumption is the assumption that it is hard to distinguish a random element in $\ker(J_n) \setminus (\mathbb{Z}_n^*)^2$ from a random element in $(\mathbb{Z}_n^*)^2$, given n (but not its factorization!).

To give a rough idea as to how this assumption may be used in cryptography, assume that $p \equiv q \equiv 3 \pmod 4$, so that $[-1]_n \in \ker(J_n) \setminus (\mathbb{Z}_n^*)^2$, and moreover, $\ker(J_n) \setminus (\mathbb{Z}_n^*)^2 = [-1]_n(\mathbb{Z}_n^*)^2$ (see Exercise 12.3). The value n can be used as a public key in a public-key cryptosystem (see §7.8). Alice, knowing the public key, can encrypt a single bit $b \in \{0, 1\}$ as $\beta := (-1)^b\alpha^2$, where Alice chooses $\alpha \in \mathbb{Z}_n^*$ at random. The point is, if $b = 0$, then β is uniformly distributed over $(\mathbb{Z}_n^*)^2$, and if $b = 1$, then β is uniformly distributed over $\ker(J_n) \setminus (\mathbb{Z}_n^*)^2$. Now Bob, knowing the secret key, which is the factorization of n, can easily determine if $\beta \in (\mathbb{Z}_n^*)^2$ or not, and hence deduce the value of the encrypted bit b. However, under the QR assumption, an eavesdropper, seeing just n and β, cannot effectively figure out what b is.

Of course, the above scheme is much less efficient than the RSA cryptosystem presented in §7.8, but nevertheless, has attractive properties; in particular, its security is very closely tied to the QR assumption, whereas the security of RSA is a bit less well understood.

Exercise 13.12. Suppose that A is a probabilistic algorithm that takes as input n of the form $n = pq$, where p and q are distinct primes such that $p \equiv q \equiv 3 \pmod 4$. The algorithm also takes as input $\alpha \in \ker(J_n)$, and outputs either 0 or 1. Furthermore, assume that A runs in strict polynomial time. Define two random variables, X_n and Y_n, as follows: X_n is defined to be the output of A on input n and a value α chosen at random from $\ker(J_n) \setminus (\mathbb{Z}_n^*)^2$, and Y_n is defined to be the output of A on input n and a value α chosen at random from $(\mathbb{Z}_n^*)^2$. In both cases, the value of the random variable is determined by the random choice of α, as well as the random

choices made by the algorithm. Define $\epsilon(n) := |\mathsf{P}[X_n = 1] - \mathsf{P}[Y_n = 1]|$. Show how to use A to design a probabilistic, expected polynomial time algorithm A' that takes as input n as above and $\alpha \in \ker(J_n)$, and outputs either "square" or "non-square," with the following property:

> if $\epsilon(n) \geq 0.001$, then *for all* $\alpha \in \ker(J_n)$, the probability that A' correctly identifies whether $\alpha \in (\mathbb{Z}_n^*)^2$ is at least 0.999.

Hint: use the Chernoff bound.

Exercise 13.13. Assume the same notation as in the previous exercise. Define the random variable X'_n to be the output of A on input n and a value α chosen at random from $\ker(J_n)$. Show that $|\mathsf{P}[X'_n = 1] - \mathsf{P}[Y_n = 1]| = \epsilon(n)/2$. Thus, the problem of distinguishing $\ker(J_n)$ from $(\mathbb{Z}_n^*)^2$ is essentially equivalent to the problem of distinguishing $\ker(J_n) \setminus (\mathbb{Z}_n^*)^2$ from $(\mathbb{Z}_n^*)^2$.

13.5 Notes

Exercise 13.2 is based on Solovay and Strassen [94].

The probabilistic algorithm in §13.3.1 for computing square roots modulo p can be made deterministic under a generalization of the Riemann hypothesis. Indeed, as discussed in §10.7, under such a hypothesis, Bach's result [10] implies that the least positive integer that is not a quadratic residue modulo p is at most $2 \log p$ (this follows by applying Bach's result with the subgroup $(\mathbb{Z}_p^*)^2$ of $\mathbb{Z}_p^*$). Thus, we may find the required element $\gamma \in \mathbb{Z}_p^* \setminus (\mathbb{Z}_n^*)^2$ in deterministic polynomial time, just by brute-force search. The best *unconditional* bound on the smallest positive integer that is not a quadratic residue modulo p is due to Burgess [22], who gives a bound of $p^{\alpha + o(1)}$, where $\alpha := 1/(4\sqrt{e}) \approx 0.15163$.

Goldwasser and Micali [39] introduced the quadratic residuosity assumption to cryptography (as discussed in §13.4). This assumption has subsequently been used as the basis for numerous cryptographic schemes.