Queen Mary
University of London

MTH6115 Cryptography

Solutions 6

---

1 (a) 1.

(b) This rather depends on the algorithm you employ. If you use trial division, you have to divide p by all numbers m such that $2 \leqslant m \leqslant \lfloor\sqrt{p}\rfloor$. Here $\lfloor\sqrt{6038068681}\rfloor = 77705$ and $\lfloor\sqrt{9673941253}\rfloor = 98356$. Thus slightly fewer than 180000 divisions (176059 in fact) would be used. Improvements like dividing only by 2 and all odd numbers or $2, 3$ and all numbers $\equiv \pm 1 \pmod 6$ reduce the 180000 to 90000 and 60000 respectively. However, we are still doing about $C\sqrt{p}$ arithmetic operations for some constant C, which does not make for a polynomial time algorithm. Dividing only by primes in the relevant ranges would involve lots of primality tests (which would make the algorithm take longer), or the availability of a massive list of all 'small' primes (not entirely helpful, though lists of primes relevant to these ranges do exist).

There are quicker primality tests, and the quickest of these is probably the ECPP (Elliptic Curve Primality Prover), which you do not need to know about in this course. Below I give the primality certificates for these primes. Primes < 1000 will be presumed proved prime by trial division (by all the primes < 32 say).

Prime	Primitive Element	Factors of $p-1$
6038068681	17	$2^3 \cdot 3^2 \cdot 5 \cdot 7 \cdot 569 \cdot 4211$
4211	6	$2 \cdot 5 \cdot 421$
9673941253	2	$2^2 \cdot 3 \cdot 3209 \cdot 251219$
251219	2	$2 \cdot 11 \cdot 19 \cdot 601$
3209	3	$2^3 \cdot 401$

Verifying these certificates is a lot faster than doing the trial division. Finding them however requires factorising $p-1$ for various primes p, which is not so attractive. Other primality certificates exist.

(c) Like the previous part, this rather depends on the algorithm you employ. With trial division, one would have to divide by every number up to and including 6038068681, about 6.04×10^9 operations, or say by every odd number, which is about 3.02×10^9 operations. Ouch!

Trial division takes about time $C\sqrt{N}$ [arithmetic operations] to factorise N. A faster method is *Pollard Rho*, which seems to take about $C'N^{1/4}$ operations to factor N. A certain algebra package running on my machine claimed the following times for factorising 58411921701573197293.

Algorithm	CPU Time (seconds)
Trial Division	177 (estimate)
Pollard Rho	0.05
Default	0.01

Actually, the package set a limit of seemingly 2^{30} (about 1.07×10^9) for what it would allow for trial division, so I must estimate the above time. For this example, the default algorithm did a small amount of trial division (up to 10000), a small amount of Pollard Rho, before finally succeeding with ECM (Elliptic Curve Method). Clearly the more sophisticated algorithms use far far fewer operations to achieve their goals than trial division, even on this relatively *small* example.

[You do not need to know about the algorithms Pollard Rho and ECM in this course, but it is good to be aware that these more complicated and faster factorising algorithms do exist. It is a matter of active research to find (even) quicker factorisation algorithms, or prove they do not exist, partly because of the existence of cryptosystems like RSA.]

2 (a) Let $k = \log_2(a)$ and $l = \log_2(b)$ be the sizes of a and b, respectively. Then, there is a polynomial P such that computing the product ab takes (at most) $P(n)$ steps, where $n = k+l$ is the size of the problem. (Where did $k+l$ come from?) Similarly, there is a polynomial Q such that computing the sum $a+b$ takes $Q(n)$ steps.

(b) For an M-by-M matrix A denote the size of A by k; this is the sum of the sizes of the entries of A. Similarly, denote the size of B by l. So, the total size of the problem is $n = k + l$.

Computing the product AB involves M^3 multiplications of integers a_{ij} and $b_{i'j'}$, where the size of a_{ij} is less than k and the size of $b_{i'j'}$ is less than l; by the previous part, this requires $M^3P(n)$ steps. Computing AB also involves $M^2(M-1)$ additions of numbers of the form $a_{ij}b_{i'j'}$, each of which has size less than $k+l$; this requires $M^2(M-1)Q(n+n)$ steps. So, in total, computing AB requires $M^3P(n) + M^2(M-1)Q(2n)$ steps. This is a polynomial in n.

(c) We can prove this by induction. Suppose we have shown that computing a^q requires $F(n)$ steps, where $F(n)$ is a polynomial in the size $n = \log_2(a)$ of a. Then, by part (a), computing $a^{q+1} = (a^q)(a)$ requires $P(n + qn)$ steps; this is a polynomial in n. Here, I have used the fact that the size of a^q is $\log_2(a^q) = qn$.