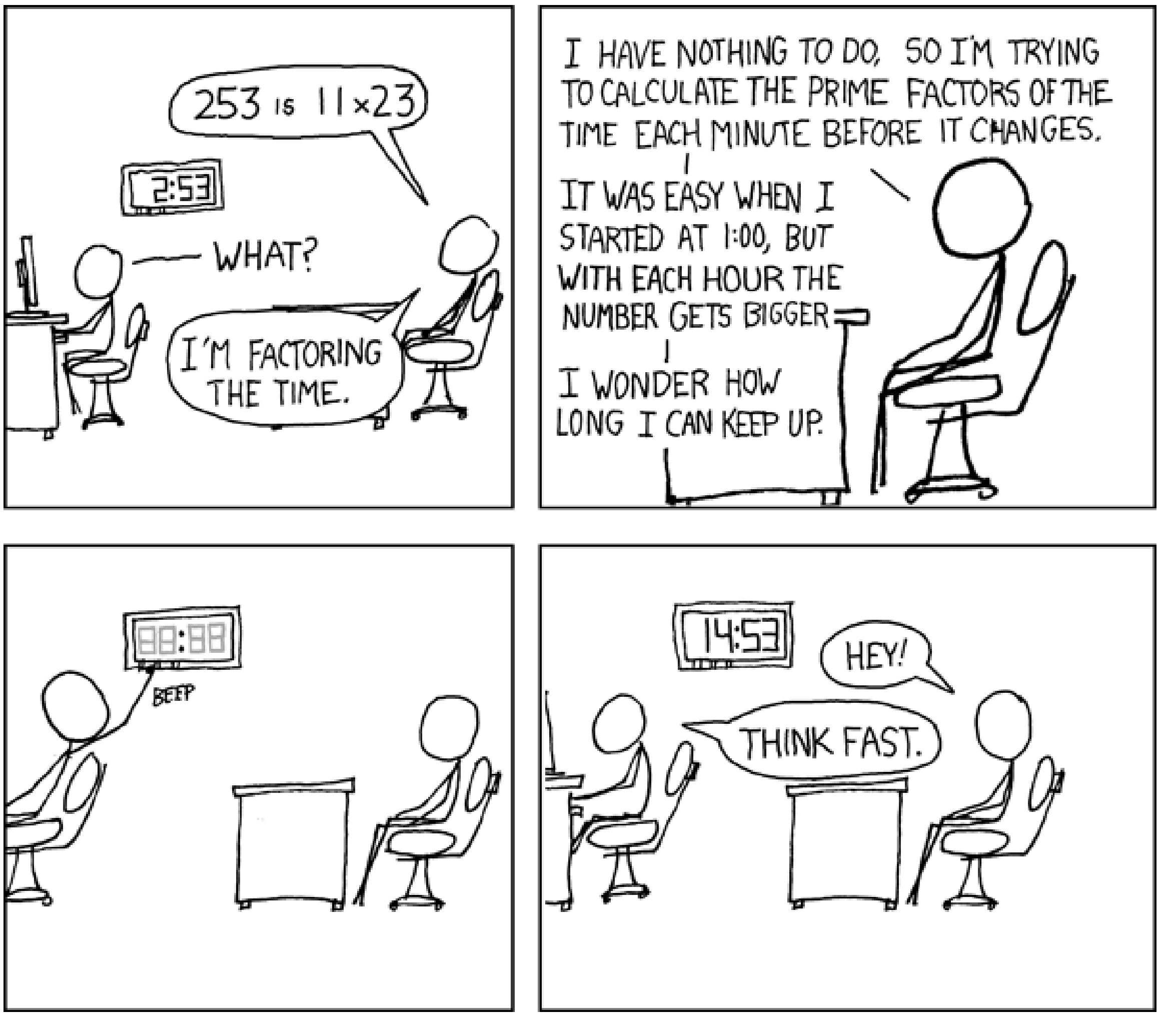
TUTORIAL 8



1 Pollard's ρ for the factorization problem

In this exercise we develop a variant of the Pollard's ρ method for factoring N. We assume that $p|N$ is the smallest (but still large for brute-force search) divisor of N. Pollard's ρ algorithm is an heuristic method: we assume that a certain deterministic sequence behaves like a truly random one.

The idea is to find two distinct $x, x' \in \mathbb{Z}_N$, s.t. $x - x' = 0 \mod p$. The tuple (x, x') defines a *collision*. To find a collision efficiently, we define a random walk on $\mathbb{Z}_N$ as

$$f(x) = x^2 + a \mod N, \quad a \in \mathbb{Z}_N$$

and consider a sequence $x_0, x_1, x_2, \ldots$ such that $x_i = f(x_{i-1})$ (we fix some initial x_0 to be a random element from $\mathbb{Z}_N$).

1. Since f takes values in a finite set, the sequence $(x_i)_i$ should eventually repeat itself. Show that you can expect to find a collision after $\mathcal{O}(\sqrt{p})$ steps. (*Hint*: recall Birthday Paradox.) You should also be able to determine the constant in front of $\sqrt{p}$.

2. Describe a Pollard's ρ algorithm for factoring having the running time of order $\widetilde{\mathcal{O}}(\sqrt{p})$.

3. Explain why the following choices for $f(x)$ are bad:
   - $f(x) = ax + b \mod N, a, b \in \mathbb{Z}_N$,
   - $f(x) = x^2 \mod N$,
   - $f(x) = x^2 - 2 \mod N$.

2 Modular roots and factoring

The first goal of this exercise is to design an efficient algorithm to compute square roots in the group $\mathbb{Z}_N$. This problem is closely related to the one of factoring N. As a first step we study the Tonelli-Shanks algorithm to compute square roots modulo a prime p. In the next tutorial, we will extend it to the non-prime moduli.

The Euler criterion states that, for any odd prime p and any $a \in \mathbb{Z}_p^\times$, we have

$$a^{(p-1)/2} \equiv \begin{cases} 1 \bmod p, \text{if } a \text{ is a square modulo } p \\ -1 \bmod p, \text{if } a \text{ is not a square modulo } p. \end{cases}$$

1. Assume $p \equiv 3 \bmod p$ and let a be a square modulo p. Give an algorithm of binary complexity $O(\log^3 p)$ to compute a square root of a.

2. We now assume that $p \equiv 1 \bmod 4$, and write $p = 2^v m + 1$ with v maximal and m odd. Give a probabilistic algorithm that, given p, return $c \in \mathbb{Z}_p^\times$ which is not a square. What is its bit complexity? Show that c^m generates the (unique) subgroup of order 2^v in $\mathbb{Z}_p^\times$.

3. Let a be a square modulo p, and c be the output of the previous algorithm. Show that a^m belongs to the subgroup generated by c^m. Next, show how that computing a square root of a modulo p amounts to computing a discrete logarithm.

4. **Pohlig-Hellman's trick:** Let G be a cyclic group of order p^k, where $k \geq 1$. For a given generator g and $h = g^x$, show that we can compute x amounts to computations of discrete logarithms in a group of order p. (Hint: write x in base p.)

5. Deduce an algorithm to compute square roots modulo p.

6. (**Bonus**) Prove Euler's criterion.

3 Why not to choose primes close to $\sqrt{N}$ for RSA

Assume one of the RSA primes is close to $\sqrt{N}$, more precisely,

$$|q - \sqrt{N}| < \sqrt[4]{N}.$$

Show how to factor N in time $poly(\log N)$. *Hint.* You might want to use the following fact: for $N = pq$, $N = \left(\frac{p+q}{2}\right)^2 - \left(\frac{p-q}{2}\right)^2$. Note that the first summand is $\approx \sqrt{N}$, while the second one is small.