Cryptography Coursework: Aleksandar Yordanov – jvgw34

1: Function Explanation:

For this assignment I chose to implement the Pohlig-Hellman algorithm. The algorithm uses a hybrid prime factorisation algorithm and Pollard-Rho algorithm which are explained below:

1a: Prime Factorisation:

For prime factorisation, I wrote a recursive hybrid trial division and Pollard-Brent algorithm that takes an integer n as input.[i] For this algorithm we leverage two sets of primes generated using the Sieve of Eratosthenes: One with primes below 100000 for primality verification, and another with primes below 1000 for trial division.

The algorithm begins with the latter set, performing trial division with every prime in it. If, after this step, $n > 1$; we use the former set along with the Miller-Rabin primality test to verify whether n is prime, if so, we return n and, if applicable, any previously found factors to the caller of the function.

If n is not prime, we use Pollard-Brent to find a factor q of n. The process then recursively repeats to see if q can be factorised further. Once a result has been returned, we combine $q's$ factors with the current factors and divide n by q, repeating the process until $n = 1$. For brevity, I will exclude the explanations for the Sieve of Eratosthenes, Pollard-Brent algorithm and Miller-Rabin primality test and assume prior knowledge.

1b: Pollard-Rho to solve discrete logarithms:

For Pollard Rho, I created a modified version of the algorithm that finds solutions to $g^k \, mod \, p \equiv A$ in subgroups of Z_p^*, denoted n. The core idea of the algorithm is to find integers $a_1, b_1, x_1, a_2, b_2, x_2$ such that $x_1 = g^{a_1}A^{b_1} = g^{a_2}A^{b_2} = x_2$, leading to a collision $x_1 = x_2$ To accomplish this, we first partition n into three disjoint sets:
$S_0 \colon x \, mod \, 3 = 2$, $S_1 \colon x \, mod \, 3 = 0$, $S_2 \colon x \, mod \, 3 = 1$ and define the maps $f(x),\ g(x,a),\ h(x,b)$.[ii] We then use Floyd's cycle finding algorithm applying these maps to x, a, b respectively until $x_1 = x_2$.

Once a collision has occurred, we have $(b_1 - b_2)k = (a_2 - a_1)$. For further use, I will denote $(b_1 - b_2)$ as r. if r and n are coprime, r will have a multiplicative inverse and there will be only one solution: $k = r^{-1}(a_2 - a_1) \, mod \, n$. If this is not the case, we must check all possible solutions. First, we must use the Extended Euclidian algorithm to obtain $gcd(r, n)$ and the Bézout coefficient s. With this we can obtain the offset between each solution $\mathrm{o} = n/gcd(r, n)$ and the first potential solution $k_1 = (s \cdot (a_2 - a_1)/gcd(r, n)) \bmod \mathrm{n}$.
We then verify each candidate $k_m = k_1 + (m \cdot o)$ by checking $g^{k_m} \equiv A \, mod \, p$
for $0 \leq m \leq gcd(r, n) - 1$ returning k_m that satisfies the discrete logarithm.[iii]

1c: Pohlig-Hellman to solve logarithms:

For my implementation of the Pohlig-Hellman algorithm I created a version of the algorithm based on 3.6.4[iv]. The algorithm finds solutions to $g^k \, mod \, p \equiv A$. The core idea of the algorithm is, first, to obtain a prime factorisation of the group order n.
$n = p_1^{e_1} p_2^{e_2} \dots p_r^{e_r}$. From here, the approach is to determine $A_i = A \, mod \, p_i^{e_i}$ for $1 \leq i \leq r$ and use CRT to recover $A \, mod \, n$. An enhancement this implementation uses is compute $k_i = l_0 + l_1 p_1 + \cdots + l_{e_i - 1} p_i^{e_i - 1}$. Where l_j are the p_i-ary digits of k_i and $0 \leq l_j \leq p_i - 1$.

To calculate l_j we need to $\overline{g}$, the adjusted generator and $\overline{A}$, the adjusted A to contribute to the digits of k_i according to 3.6.4[v]. From there we calculate $log_{\overline{g}}\overline{A} = l_j$ using Exhaustive search for $p < 100$ and Pollard-Rho for $p > 100$. Once we have calculated all k_i, we use Gauss's algorithm to solve the system of congruences $k =_{p_i^{e_i}} k_i$ and return k.

2: Pohlig-Hellman Justification:

I decided to go with the Pohlig-Hellman algorithm for its ability to exploit smooth group orders and efficiently calculate the discrete logarithm for primes by breaking down the problem into sub-problems by n's prime factorisation. Considering that, in the worst case, Pohlig-Hellman's time complexity decays to $O(q \cdot log(n))$ (Pollard Rho); Pohlig-Hellman in most cases provides a clear benefit over solely using Pollard-Rho. Furthermore, for $p < 100$[vi], I use exhaustive search as Pollard-Rho has too high of a chance of failing for small subgroups. This ensures the correctness of the algorithm.

2a: Factorisation Justification:

I decided to go with this hybrid approach as it leverages the strengths of both trial division and Pollard-Brent for small and large factors respectively. Trial division has a time complexity of $O(\sqrt{n})$ while Pollard-Brent has a time complexity of $O\left(\sqrt[4]{n} \cdot log(n)\right)$.
E.g., $n = 10^3$, $\sqrt{n} = 31.6, \sqrt[4]{n} \cdot log(n) = 38.8$, | $n = 10^{10}$, $\sqrt{n} = 10^5, \sqrt[4]{n} \cdot log(n) = 7281.4$.
To note: time complexity of modular arithmetic is $O(log(n))$.

2b: Pollard-Rho Justification:

I decided to use Pollard-Rho instead of Shank's algorithm for Pohlig-Hellman as they have the same time complexity while the memory requirements are essentially eliminated when using Pollard-Rho.[vii]

3: Comparison with other methods:

Foreword:

The main comparison I will be doing is against Index Calculus. All algorithms discussed have been compared and are in exponential time, but index calculus is one of the few sub-exponential time complexity algorithms for the DLP. The main justification I have for implementing Pohlig-Hellman as opposed to Index Calculus is the high implementation complexity of index calculus. I also believe that, given the constraints of this assignment (30-digit non-smooth group order, 50-digit smooth group order), Index Calculus is not necessary to implement to solve these DLP's in sufficient time.

A lot of cases where the group order has large to moderately large smoothness, Pohlig-Hellman can outperform Index Calculus. For example, the 107-digit prime in section 3.66 of the Handbook of Applied Cryptography. The largest factor of $p - 1$ in this case is 350377. My implementation of Pohlig-Hellman can solve DLP's with this prime in under a second while Index Calculus (using an online implementation) takes an indeterminately long amount of time. Now granted, this is a cherry-picked example, but it goes to show the clear benefit of Pohlig-Hellman in such cases, for which there happen to be many.

---

[i] Factorisation algorithm
[ii] Handbook of Applied cryptography 3.6.3
[iii] Solving modular equations
[iv] Handbook of Applied Cryptography 3.6.4
[v] Handbook of Applied Cryptography 3.6.4
[vi] EFFP – Pohlig-Hellman Algorithm (Section at the bottom that details his implementation)
[vii] Handbook of Applied Cryptography 3.6.7 (iii)