Ex2

April 2, 2024

1 Trabalho Prático 2

André Freitas PG54707

Bruna Macieira PG54467

1.1 Exercício 2

Uma das aplicações mais importantes do teorema chinês dos restos (CRT) em criptografia é a transformada NTT "Number Theoretic Transform".
Esta transformada é uma componente importantes de "standards" PQC como o Kyber e o Dilithium mas também de outros algoritmos submetidos ao concurso NIST PQC.
A transformação NTT tem várias opções e aquela que está apresentada no +Capítulo 4: Problemas Difíceis usa o CRT. Neste problema pretende-se uma implementação Sagemath do NTT-CRT tal como é descrito nesse documento.

[ ]:
```
%pip install sagemath-standard
from sage.all import next_prime
from sage.rings.finite_rings.finite_field_constructor import FiniteField
from sage.arith.misc import CRT
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: sagemath-standard in /usr/lib/python3/dist-
packages (9.5)
Requirement already satisfied: cysignals>=1.10.2 in
/home/fura/.sage/local/lib/python3.10/site-packages (from sagemath-standard)
(1.11.4)
Note: you may need to restart the kernel to use updated packages.
```

A definição de parâmetros é feita. Para tal, utilizamos N para determinar o tamanho dos polinómios (deve ser uma potência de 2), e q, que é o próximo primo após 2^(N+1) e deve satisfazer q 1 (mod 2N).

[ ]:
```
# Escolher um valor N e um primo q que satisfaçam as condições dadas
N = 8  #Dá qualquer potência de 2
q = next_prime(2**(N+1))  # Escolher um primo que satisfaça q  1 (mod 2N)
```

A seguir, é definido um corpo finito k com base no q calculado anteriormente, e um anel de polinómios Zw sobre o corpo finito k com uma variável w.

[ ]:
```
# Definir o corpo finito
K = FiniteField(q)

# Definir o anel de polinómios RqN
Zw.<w> = PolynomialRing(K, 'w')
```

São calculadas as raízes N-ésimas da unidade usando um elemento primitivo do corpo finito K e é armazenado em raizes.

[ ]:
```
# Calcular as raízes s_i
omega = K.primitive_element()^((q - 1) // N)
raizes = [omega^(2*i + 1) for i in range(N)]
print(raizes)
```

```
[315, 43, 206, 478, 315, 43, 206, 478]
```

São escolhidos dois polinómios aleatórios f e g de grau N-1 no anel de polinómios.

[ ]:
```
# Escolher duas funções f e g
f = Zw.random_element(degree=N-1)
g = Zw.random_element(degree=N-1)
print(f)
print(g)
```

```
171*w^7 + 198*w^6 + 135*w^5 + 465*w^4 + 249*w^3 + 273*w^2 + 312*w + 130
328*w^7 + 189*w^6 + 116*w^5 + 156*w^4 + 188*w^3 + 403*w^2 + 315*w + 129
```

É aplicada o algoritmo da Transformada Rápida de Fourier sobre Corpos Finitos (NTT) para calcular os resíduos de f e g em cada raiz N-ésima da unidade

[ ]:
```
# Aplicar o algoritmo NTT para calcular os resíduos de f e g
residuos_f = [lift(f(w)) % q for w in raizes]
residuos_g = [lift(g(w)) % q for w in raizes]
```

São definidos os módulos CRT q_i (polinómios (w - raiz) para cada raiz N-ésima da unidade)

[ ]:
```
# Recolher os módulos CRT q_i
modulos_crt_f = [Zw(w - raiz) for raiz in raizes]
modulos_crt_g = [Zw(w - raiz) for raiz in raizes]
```

[ ]:
```
# Multiplicar componente a componente os resíduos de f e g para obter os␣
↪resíduos resultantes
residuos_resultantes = [residuos_f[i] * residuos_g[i] for i in range(N)]
```

[ ]:
```
# Usar o algoritmo CRT para reconstruir a função resultante
funcao_resultante = CRT(residuos_resultantes, modulos_crt_f)
```

Este código realiza a multiplicação rápida de polinómios usando o algoritmo do Teorema do Resto Chinês (CRT) sobre corpos finitos, aproveitando a estrutura das raízes N-ésimas da unidade para obter eficiência computacional. Os resultados obtidos estão presentes abaixo.

[ ]:
```
print(f'f = {f}')
print(f'g = {g}')
print(f'residuos = {residuos_resultantes}')
print(f'resultante = {funcao_resultante}')
```

```
f = 171*w^7 + 198*w^6 + 135*w^5 + 465*w^4 + 249*w^3 + 273*w^2 + 312*w + 130
g = 328*w^7 + 189*w^6 + 116*w^5 + 156*w^4 + 188*w^3 + 403*w^2 + 315*w + 129
residuos = [40959, 16170, 141722, 132660, 40959, 16170, 141722, 132660]
resultante = 170*w^3 + 41*w^2 + 514*w + 169
```