TAG-CRIPTOGRAFIA
Nome:Mariane Ferreira

Challenge 1: Basics - Convert hex to base64
O código está no arquivo challenge1.txt.

Explicação do código:
O código tem 21 linhas no total, sendo a linha 5 a string em hexadecimal a ser convertida para a base64. Inicialmente, pensei em implementar da seguinte forma: Um algoritmo que dividisse a string em partes com 2 hexadecimais.
49 27 6d 20 6b 69 6c 6c 69 6e 67 20 79 6f 75 72 20 62 72 61 69 6e 20 6c 69 6b 65 20 61 20 70 6f 69 73 6f 6e 6f 75 73 20 6d 75 73 68 72 6f 6f 6d

E converter cada parte para binário
01001001 00100111 01101101 00100000 01101011 01101001

Nesse caso convertemos até o 6º grupo "69" para exemplificar, juntamos tudo e dividimos em grupos de 6 bits.
010010010010011101101101001000000110101101101001
010010 010010 011101 101101 001000 000110 101101 101001

E convertemos os grupos de 6 bits para decimal.
18 18 29 45 8 6 45 41

Olhando na tabela da base64 e encontrando os caracteres correspondentes aos decimais, temos:
SSdtIGtp

Que é correspondente aos 6 primeiros dígitos da chave em base64 dada. Mas encontrei dificuldades para implementar desta forma e fiz da seguinte forma:

Semelhante ao passo anterior, dividimos os hexadecimais em grupos de 2, mas ao inves de convertê-los para binário, convertemos direto para decimal.
49 27 6d 20 6b 69 6c 6c 69 6e 67 20 79 6f 75 72 20 62 72 61 69 6e 20 6c 69 6b 65 20 61 20 70 6f 69 73 6f 6e 6f 75 73 20 6d 75 73 68 72 6f 6f 6d

Convertemos os 6 primeiros grupos, temos:
73 39 109 32 107 105

Olhando na tabela ASCII, temos:
I'm ki
A frase completa é I'm killing your brain like a poisonous mushroom.

Para a base 64, convertemos os números correspondentes ao decimal ASCII para binário.
01001001 00100111 01101101 00100000 01101011 01101001

Juntamos tudo e dividimos em grupos de 6 bits:
010010010010011101101101001000000110101101101001
010010 010010 011101 101101 001000 000110 101101 101001

Ao converter para decimal, dá os seguintes números:
18 18 29 45 8 6 45 41

Olhando na tabela da base64 e encontrando os caracteres correspondentes aos decimais, temos:
SSdtIGtp
Que é correspondente aos 6 primeiros dígitos da chave em base64 dada.

Resumo do código:
Criei um objeto do tipo StringBuilder (trabalha com strings maiores do que o String), fiz um for que soma +2 a cada laço e o substring corta a string de 2 em 2. Converti os hexadecimais para inteiro e fiz um typecasting de int para char, que retorna um caractere correspondente ao número de acordo com a tabela ASCII, então conforme ele vai rodando o laço, ele vai concatenando todos os caracteres gerados na saida. No final, todos são convertidos para uma string e é jogado na base. Usa-se um método do java chamado Base64 para converter a string gerada no código solicitado.

Challenge 2: Basics - Fixed XOR
O código está no arquivo challenge2.txt.

Explicação do código:
O objetivo era fazer uma soma de XOR entre 2 strings hexadecimais, porem eu nao conseguia fazer o Java entender que aquelas strings eram em hexadecimais, então a conversão para binário estava dando errado porque ele estava usando a tabela ASCII como referência. Fiz a conversão manualmente usando 2 for (um para cada string) e convertendo caractere por caractere através de um array de char, e jogando o resultado em uma StringBuilder. Fiz o XOR entre as duas strings geradas e joguei o resultado em uma terceira StringBuilder. No final, converter o resultado para hexadecimal.

O que é XOR?
Basicamente, soma de binários na qual
1+1=0,
1+0=1,
0+1=1,
0+0=0.
Tem que ter cuidado ao fazer a soma, as strings devem ter o mesmo tamanho e caso não tenha, completar com zeros na frente a string de tamanho menor.

Challenge 39: Set 5 - Implement RSA
O código está no arquivo challenge3.txt.

Explicação do código:
Aviso: o código tem erros e talvez não funcione como esperado, mas explicarei o que era pra ser.
Escolher 2 primos p e q aleatoriamente, preferi de no maximo 1021 (quatro algarismos) para facilitar as coisas, embora o RSA exija primos com 10^100 algarismos. Teremos n que é p*q e et=(p-1)*(q-1). O desafio pediu para colocar e=3. Deveríamos ter d= a inversa entre (e,et), porém, esta parte está mal implementada pois podemos gerar um número que não tem inversa ou que a inversa dê negativo, não soube como validar isso. Para os números dados (17,3120), o método funciona perfeitamente.
As chaves públicas e privadas são printadas. Para encriptar e decriptar, talvez não funcione, pois precisaria de um invmod devidamente implementado e funcional, mas basicamente era escolher a mensagem (o enunciado pede para encriptar o número 42) e decriptá-lo logo em seguida.

RSA fora dos códigos:

1. escolha primos p e q.
2. gere n como produto de p e q (n=p*q).
3. o et seria (p-1)*(q-1).
4. o e seria escolha pessoal, porém tem que ser primo entre n e distante.
5. o d seria gerado pelo inverso multiplicativo de d*e é congruente a 1 (mod n), usando o Algoritmo Euclidiano Estendido
6. Chave pública (usada para encriptar) seria o par (n, e)
7. Chave privada (usada para decriptar) seria o trio (p,q,d) sendo d o essencial para os cálculos
8. Encriptar: escolhe-se uma mensagem m e faz m^e é congruente a c (mod n), sendo c a mensagem encriptada
9. Decriptar: c^e é congruente a m (mod n), sendo m a mensagem original

Resumo do código:
Temos funções para gerar primos e os dados solicitados, a única função com problema é a que deveria calcular o inverso modular entre e e et, fora isto, funciona.

Considerações finais:
Algumas partes dos códigos são pedaços de códigos pegos na internet e alterados, outras exigiram um bom estudo do Java, e outras são geradas de programação básica. Tentei escrever tudo do meu jeito para entender como funciona, até porque se fosse pra só para entregar, teria feito em python e colado todos os códigos prontos na internet (com uma pesquisa básica você acha todos os desafios do cryptopals prontos em python na internet).)