1.

One way to solve this problem is by dynamic programming.
Subproblems: Define $L(i, j)$ to be the probability of obtaining exactly j heads amongst the first i coin tosses.

Algorithm and Recursion: By the definition of L and the independence of the tosses, it is clear that:

$$L(i, j) = p_i L(i-1, j-1) + (1-p_i)L(i-1, j) \quad j = 0, 1, \ldots, i$$

We can then compute all $L(i, j)$ by initializing $L(0,0) = 1$, $L(i, j) = 0$ for $j < 0$, and proceeding incrementally (in the order $i = 1, 2, \ldots, n$, with inner loop $j = 0, 1, \ldots, i$). The final answer is given by $L(n, k)$.

Correctness and Running Time: The recursion is correct as we can get j heads in i coin tosses either by obtaining $j-1$ heads in the first $i-1$ coin tosses and throwing a head on the last coin, which takes place with probability $p_i L(i-1, j-1)$, or by having already j heads after $i-1$ tosses and throwing a tail last, which has probability $(1-p_i)L(i-1, j)$. Besides, these two events are disjoint, so the sum of their probabilities equals $L(i, j)$. Finally, computing each subproblem takes constant time, so the algorithm runs in $O(n^2)$ time.

Another approach is to observe that the probability is exactly the coefficient of x^k in the polynomial $g(x) = \prod_{i=1}^{n}(p_i x + (1-p_i))$. We can compute $g(x)$ using divide-and-conquer in $O(n \log^2 n)$ time by recursively computing $g_1(x) = \prod_{i=1}^{n/2}(p_i x + (1-p_i))$, $g_2(x) = \prod_{i=n/2+1}^{n}(p_i x + (1-p_i))$ and then using FFT to calculate $g(x) = g_1(x)g_2(x)$.

2.

Since FACTORING is in NP (we can check a factorization in polynomial time), $\mathbf{P} = \mathbf{NP}$ would mean the factors of a number can be *found* in polynomial time. Since in RSA, we know (N, e) as the public key, we can factor N to find p and q and in polynomial time. We can then compute $d = e^{-1} \bmod (p-1)(q-1)$ using Euclid's algorithm. If X is the encrypted message, then $X^e \bmod N$ gives the original message.

3.

We give a reduction from CLIQUE to KITE. Given an instance (G, k) of CLIQUE, we add a tail of k new vertices to every vertex of G to obtain a new graph G'. Since the added tails are just paths and cannot contribute to a clique, G' has a kite with $2k$ nodes if and only if G has a clique of size k.

4.

a) This is the same problem as finding a (s_1, s_2)min-cut, which can be done by a maximum flow computation in polynomial time.

b) Find a (s_1, s_2)mincut $E_1 \subseteq E$ using maximum flow. Suppose s_1 and s_3 fall on the same side of the cut (the other case is symmetric). Compute then a (s_1, s_3)mincut $E_2 \subseteq E$ and output $E_1 \cup E_2$. To see this is a 2-approximation, consider the optimal multiway cut E^*: because $E*$ is both a (s_1, s_2)cut and a (s_1, s_3) cut, we have $|E_1| \leq E^*$ and $|E_2| \leq E^*$. Hence, $|E_1 \cup E_2| \leq |E_1| + |E_2| \leq 2|E^*|$, as required.

c) We need to define a neighborhood structure on the subsets of E whose removal disconnects the input terminals. The most natural choice is to have two subsets be neighbors if the size of their symmetric difference is less than some fixed number t. Notice that the size of each neighborhood is at most $|E|^t$.