Binary Modular Exponentiation

- In cryptography, it is important to be able to find b^n **mod** m efficiently, where b, n, and m are large integers.
- Use the binary expansion of n, $n = (a_{k-1},\dots,a_1,a_0)_2$, to compute b^n .
  Note that:

$$b^n = b^{a_{k-1}\cdot 2^{k-1}+\cdots+a_1\cdot 2+a_0} = b^{a_{k-1}\cdot 2^{k-1}} \cdots b^{a_1\cdot 2}\cdot b^{a_0}.$$

- Therefore, to compute b^n, we need only compute the values of b, b^2, $(b^2)^2 = b^4$, $(b^4)^2 = b^8$, …, b^{2^k} and the multiply the terms b^{2^j} in this list, where $a_j = 1$.

Example: Compute 3^{11} using this method.
Solution: Note that $11 = (1011)_2$ so that $3^{11} = 3^8\, 3^2\, 3^1 = ((3^2)^2)^2\, 3^2\, 3^1 = (9^2)^2 \cdot 9 \cdot 3 = (81)^2 \cdot 9 \cdot 3 = 6561 \cdot 9 \cdot 3 = 117{,}147.$

continued →

Binary Modular Exponentiation Algorithm

- The algorithm successively finds b **mod** m, b^2 **mod** m, b^4 **mod** m, …, $b^{2^{k-1}}$ **mod** m, and multiplies together the terms b^{2^j} where $a_j = 1$.

procedure *modular exponentiation*(b: integer, $n = (a_{k-1}a_{k-2}\ldots a_1a_0)_2$, m: positive integers)
$x := 1$
power := b **mod** m
for $i := 0$ to $k - 1$
    if $a_i = 1$ **then** $x := (x \cdot power)$ **mod** m
    power := (*power* · *power*) **mod** m
return x {x equals b^n **mod** m }

- $O((\log m)^2 \log n)$ bit operations are used to find b^n **mod** m.