1) By using the Python program below, we find \( 6019208928 = 2^5 \times 3^3 \times 23 \times 101 \times 2999 \). It also prints out \( \phi(n) \), using the second formula for \( \phi(n) \), where we factor out \( n \).

```python
def primefact(n):
    i=2
    phi = n
    while i*i <= n:
        exp = 0
        while n%i == 0:
            n //= i
            exp+=1
        if exp > 0:
            print(i,"^",exp," * ",sep="",end="")
            phi = phi - phi/i
        i += 1
    if n > 1:
        print(n,"^1", sep="")
        phi = phi - phi//n
    print("\phi(\),n," = ", phi, sep="")
primefact(6019208928)
```

2) Alternatively, we can work out \( \phi(6019208928) \) by hand:

\[
\phi(6019208928) = \phi(2^5) \times \phi(3^3) \times \phi(23) \times \phi(101) \times \phi(2999) \\
= (2^5 - 2^4)(3^3 - 3^2)(23 - 1)(101 - 1)(2999 - 1) \\
= 16 \times 18 \times 22 \times 100 \times 2998 = 1899532800
\]

3) 15 and 131 are relatively prime so

\[15^{130} \equiv 1 (\text{mod } 131), \text{ via Fermat's Theorem}. \text{ It follows that:}\]

\[15^{2992} \equiv (15^{130})^{23} \times 15^2 \equiv 1 \times 225 \equiv 94 (\text{mod } 131)\]

4) 701 and 1224 are relatively prime so by Euler's Theorem we have

\[701^{\phi(1224)} \equiv 1 \pmod{1224}\]

First, let's prime factorize 1224 by trial division: 1224 = 2^3 \times 3^2 \times 17.
Thus, \( \phi(1224) = \phi(2^3) \times \phi(3^2) \times \phi(17) = (2^3 - 2^2)(3^2 - 3^1)(17 - 1) = 4 \times 6 \times 16 = 384.\)

\[701^{2689} \equiv 701^{2688+1} \equiv 701^{2688} \times 701^1 \equiv (701^{384})^7 \times 701 \equiv 1^7 \times 701 \equiv 701 (\text{mod } 1224)\]
Thus, we must determine $d = 143^{-1} \pmod{448}$. Use the Extended Euclidean Algorithm:

\[
\begin{align*}
448 &= 3 \times 143 + 19 \\
143 &= 7 \times 19 + 10 \\
19 &= 1 \times 10 + 9 \\
10 &= 1 \times 9 + 1 \\
10 - 9 &= 1 \\
10 - (19 - 1 \times 10) &= 1 \\
2 \times 10 - 1 \times 19 &= 1 \\
2(143 - 7 \times 19) - 1 \times 19 &= 1 \\
2 \times 143 - 14 \times 19 - 1 \times 19 &= 1 \\
2 \times 143 - 15 \times 19 &= 1 \\
2 \times 143 - 15(448 - 3 \times 143) &= 1 \\
2 \times 143 - 15 \times 448 + 45 \times 143 &= 1 \\
47 \times 143 - 15 \times 448 &= 1
\end{align*}
\]

Taking this equation mod 448, we find

\[47 \times 143 \equiv 1 \pmod{448}\]

It follows that $d = 47$.

6) We assume that at least one primitive root exists. Let’s call this $\alpha$. We know that of the $p-1$ values $1, 2, 3, \ldots, p-1$, exactly $\varphi(p-1)$ of them share no common factor with $p-1$, based on the definition of $\varphi$.

In order to prove the assertion, we must prove that $\alpha^k$ is a primitive root if and only if gcd($k, p-1$) = 1. If we can prove this, then from the list $\alpha, \alpha^2, \alpha^3, \alpha^4, \ldots, \alpha^{p-1}$, the terms that are primitive roots are precisely the terms with the exponents that don’t share a common factor with $p-1$, of which there are exactly $\varphi(p-1)$.

Let gcd($k, p-1$) = 1. We will prove that $\alpha^k$ is a primitive root. We prove this using proof by contradiction. Assume the opposite, that $\alpha^k$ is NOT a primitive root. Then, we must have that two values in the list $\alpha^k, \alpha^{2k}, \alpha^{3k}, \ldots, \alpha^{k(p-1)}$ that are equivalent mod $p$. Let these two values be $\alpha^i$ and $\alpha^j$, where $0 < i < j < p$. Thus, we have:

\[
\begin{align*}
\alpha^i &\equiv \alpha^k \pmod{p} \\
\alpha^j - \alpha^i &\equiv 0 \pmod{p} \\
\alpha^i(\alpha^{j-i} - 1) &\equiv 0 \pmod{p}
\end{align*}
\]

We know that $p$ shares no common factors with $\alpha^k$. 

5) $\varphi(17 \times 29) = \varphi(17) \times \varphi(29) = 16 \times 28 = 448$
It follows that $p \mid \alpha^{ik} - 1$. Thus
\[
\alpha^{ik} - 1 \equiv 0 \mod p \\
\alpha^{(j-i)k} \equiv 1 \mod p
\]
Since $\alpha$ is a primitive root, we know that the exponent on the left must be a multiple of $p - 1$:
\[(p - 1) \mid (j - i)k.\]
We know that $\gcd(p - 1, k) = 1$. Thus it follows that $(p - 1) \mid (j - i)$. But this contradicts the fact that $0 < i < j < p$, which means that $i \geq 1$, $j \leq p - 1$, so $j - i > 0$ and $j - i \leq p - 2$.
This is our contradiction. It follows that our initial assumption that two values on the given list were equivalent mod $p$ is faulty. If no two of these values are equivalent mod $p$, we can conclude that $\alpha^k$ is a primitive root.

Now, the second part of the proof is that if $\gcd(p - 1, k) > 1$, then $\alpha^k$ is NOT a primitive root. Let $c = \gcd(p - 1, k) > 1$. Now, consider the term $(\alpha^k)^{(p-1)/c}$ mod $p$. The exponent $\frac{p-1}{c}$ is clearly less than $p - 1$. Secondly, this is equivalent to $\alpha^{k(p-1)/c}$ mod $p$. Notice that $c$ divides evenly into $k$ because $c = \gcd(k, p - 1)$, thus $c$ is a divisor of $k$. Let $m = \frac{k}{c}$, and $m \in \mathbb{Z}$. Thus $\alpha^{k(p-1)/c} \equiv (\alpha^{p-1})^k \equiv 1^m \equiv 1 (mod p)$. This means that $\alpha^k$ isn’t a primitive root since raising it to a power less than $p - 1$ yields 1.

Thus, we have shown that if AND only if $\gcd(p - 1, k) = 1$, is $\alpha^k$ a primitive root of $p$. Thus, to count the number of primitive roots, we simply look at the list $\alpha, \alpha^2, \alpha^3, ... , \alpha^{p-1}$ and count the number of terms that have exponents relatively prime to $p - 1$. By definition of $\alpha$, this number is exactly $\phi(p - 1)$. As a concrete example, if we know that 2 is a primitive root of $p = 19$, it follows that $2^1, 2^5, 2^7, 2^{11}, 2^{13},$ and $2^{17}$ are all primitive roots of 19, since 1, 5, 7, 11, 13 and 17 don’t share any common factors with 18, $p - 1$.

Note: This solution is written by Sushant Kulkarni, a past TA of the course.

7) Based on the previous proof, the 16 primitive roots must be $6^{a_1}, 6^{a_2}, ..., 6^{a_{16}}$, where each of the values $a_1, a_2, ..., a_{16}$ are relatively prime with 41 - 1 = 40. Since $a_1 < a_2 < ... < a_{16}$, it follows that this list is simply the 16 values relatively prime to 40, in between 1 and 40. These are:
\[1, 3, 7, 9, 11, 13, 17, 19, 21, 23, 27, 29, 31, 33, 37, 39\]
Thus, we find $a_{14} = 33$, $a_{15} = 37$ and $a_{16} = 39$.

Note: $6^3 \equiv 216 \equiv 11 (mod 41)$
$6^{33} \equiv (6^2)^{16}6 \equiv (36)^{16}6 \equiv (-5)^{16}6 \equiv (-125)^5(-5)(6) \equiv (-2)^5(-30) \equiv (-32)(-30) \equiv 960 \equiv 17 (mod 41)$
$6^{37} \equiv 6^{33} \times 6^4 \times 6 \equiv 17 \times 11 \times 6 \equiv 15 (mod 41)$
$6^{39} \equiv 6^{37} \times 6^2 \equiv 15 \times 36 \equiv 7 (mod 41)$
8) We need to find

Alice Sends to Bob $= 13^8 \equiv 9 \pmod{37}$, (using Python)

Bob Sends to Alice $= 13^{19} \equiv 24 \pmod{37}$, (using Python)

Shared Key $= 9^{19} \equiv 9 \pmod{37}$

Alternatively, the Shared Key $= 24^8 \equiv 9 \pmod{37}$

9) The code that produced the results below is attached as a separate file, FactoringHW5.java.

<table>
<thead>
<tr>
<th>Number</th>
<th>Pollard-Rho</th>
<th>Trial Division</th>
<th>Fermat</th>
</tr>
</thead>
<tbody>
<tr>
<td>441075437627829133</td>
<td>78 ms</td>
<td>2209 ms</td>
<td></td>
</tr>
<tr>
<td>733561193479131791</td>
<td>16 ms</td>
<td>3974 ms</td>
<td></td>
</tr>
<tr>
<td>611217877192686991</td>
<td>47 ms</td>
<td>3135 ms</td>
<td></td>
</tr>
<tr>
<td>1442059257386438303</td>
<td>47 ms</td>
<td>9069 ms</td>
<td>9437 ms</td>
</tr>
<tr>
<td>3008502085141882717</td>
<td>62 ms</td>
<td>13435 ms</td>
<td>906 ms</td>
</tr>
</tbody>
</table>

Since Fermat took much longer on the first three cases, these times are guesstimated below as follows:

A million tests for a were clocked at 8.75 seconds.

Based on the factorizations obtained by both of the other methods, we can determine precisely how many values of a have to be tried before the solution is found and use the scale factor above to guesstimate the time. Here is that chart:

<table>
<thead>
<tr>
<th>Number</th>
<th>Fermat Guesstimate</th>
</tr>
</thead>
<tbody>
<tr>
<td>441075437627829133</td>
<td>2563 seconds</td>
</tr>
<tr>
<td>733561193479131791</td>
<td>1303 seconds</td>
</tr>
<tr>
<td>611217877192686991</td>
<td>1610 seconds</td>
</tr>
</tbody>
</table>

Incidentally, the factorizations are provided below:

$441075437627829133 = 267583457 \times 1648365869$

$733561193479131791 = 479056439 \times 1531262569$

$611217877192686991 = 398592973 \times 1533438667$

$1442059257386438303 = 1255635859 \times 1148469317$

$3008502085141882717 = 1713733097 \times 1755525461$
In addition, Safa wrote python code to do all three tests (factor.py) and here are his run-times:

<table>
<thead>
<tr>
<th>Number</th>
<th>Pollard-Rho1</th>
<th>Pollard-Rho2</th>
<th>Trial Division</th>
<th>Fermat</th>
</tr>
</thead>
<tbody>
<tr>
<td>441075437627829133</td>
<td>601 ms</td>
<td>267 ms</td>
<td>16062 ms</td>
<td>3042906 ms</td>
</tr>
<tr>
<td>733561193479131791</td>
<td>479 ms</td>
<td>398 ms</td>
<td>31859 ms</td>
<td>1490593 ms</td>
</tr>
<tr>
<td>611217877192686991</td>
<td>384 ms</td>
<td>290 ms</td>
<td>23828 ms</td>
<td>1754468 ms</td>
</tr>
<tr>
<td>1442059257386438303</td>
<td>606 ms</td>
<td>380 ms</td>
<td>84390 ms</td>
<td>9953 ms</td>
</tr>
<tr>
<td>3008502085141882717</td>
<td>989 ms</td>
<td>475 ms</td>
<td>138203 ms</td>
<td>1000 ms</td>
</tr>
</tbody>
</table>

The Java versions were faster than Python except for Fermat (which probably means my Fermat method is implemented extremely inefficiently!)

In general, for numbers in this range (products of two primes around $10^9$), we find that Pollard-Rho is most effective, followed by trial division. Fermat is fairly terrible because the number of steps it takes can come close to trial division, but the individual actions (squaring a, determining if the leftover is a perfect square) take much more time than a single mod (trial division).