• The exam is double-sided. Make sure to read both sides of each page.
• The time limit is 50 minutes.
• No calculators are permitted.
• You are permitted one page of notes, front and back.
• The textbook’s summary tables for the systems we have studied are provided on the last sheet. You may detach this sheet for easier reference.
• For any problem asking you to write a program, you may write in a language of your choice or in pseudocode, as long as your answer is sufficiently specific to tell the runtime of the program.
(1) Use Shanks's "babystep-giantstep" algorithm to compute log_5[13]_{23} (that is, find an integer \( x \) such that \( 5^x \equiv 13 \pmod{23} \)). Clearly label the two lists that you create and the common element between them. A multiplication table modulo 23 is provided at the back of the exam packet, for convenience.

Let \( B = 5 \), so that \( B^2 \cdot p - 1 = 22 \).

Two lists will be \( 5^i \) for \( i = 0, 1, 2, 3, 4 \) and \( 13 \cdot 5^{-i} \) for \( i = 0, 1, 2, 3, 4 \).

Powers of 5:

\[
\begin{align*}
5^0 & \equiv 1 \pmod{23} \\
5^1 & \equiv 5 \pmod{23} \\
5^2 & \equiv 5 \cdot 5 \equiv 2 \pmod{23} \\
5^3 & \equiv 2 \cdot 5 \equiv 10 \pmod{23} \\
5^4 & \equiv 10 \cdot 5 \equiv 4 \pmod{23} \\
5^5 & \equiv 4 \cdot 5 \equiv 20 \pmod{23}
\end{align*}
\]

So \( 5^{-5} \equiv 20^{-1} \equiv 15 \pmod{23} \)

Elements \( 13 \cdot 5^{-5i} \equiv 13 \cdot 15^i \pmod{23} \):

\[
\begin{align*}
13 \cdot 5^{-0} & \equiv 13 \pmod{23} \\
13 \cdot 5^{-5} & \equiv 13 \cdot 15 \equiv 11 \pmod{23} \\
13 \cdot 5^{-25} & \equiv 11 \cdot 15 \equiv 4 \pmod{23} \\
13 \cdot 5^{-35} & \equiv 4 \cdot 15 \equiv 14 \pmod{23} \\
13 \cdot 5^{-45} & \equiv 14 \cdot 15 \equiv 3 \pmod{23}
\end{align*}
\]

The common element is \( 4 \equiv 5^4 \equiv 13 \cdot 5^{-10} \pmod{23} \),

so \( 5^{4 + 10} \equiv 13 \pmod{23} \)

\[
\log_5 [13]_{23} = 14
\]

More space for work on reverse side. (6 points)
Additional space for problem 1.
(2) Let $p = 53$, $q = 13$, $g = 10$ be parameters for DSA (these satisfy the conditions in table 4.3). Suppose that Samantha has chosen the private signing key $a = 7$. Using $k = 2$ as the ephemeral key, compute a DSA signature for the document $D = 3$. (Note: you do not need to calculate the public key $A$ in order to solve this problem.)

$$S_1 = 10^2 \cdot 53 \mod 13$$
$$= 100 \cdot 53 \mod 13$$
$$= 47 \mod 13$$
$$= 8$$

$$S_2 \equiv 2^{-1} (3 + 7 \cdot 8) \mod 13$$
$$\equiv 7 \cdot (3 + 56) \mod 13$$
$$\equiv 7 \cdot 7 \mod 13$$
$$\equiv 49 \mod 13$$
$$\equiv 10 \mod 13$$

$$S_1, S_2 = (8, 10)$$

More space for work on reverse side. (6 points)
Additional space for problem 2.
(3) Integers $p$ and $q$ are both primes, exactly 42 bits in length. The numbers $p - 1$ and $q - 1$ factor into primes as follows.

$$ p - 1 = 2 \cdot 29 \cdot 353 \cdot 433 \cdot 601 \cdot 821 $$

$$ q - 1 = 2 \cdot 2199023249261 $$

You may assume, without proof, that 2 is a primitive root modulo $p$ and modulo $q$.

(a) Explain briefly why discrete logarithms modulo $p$ can be computed much more rapidly than discrete logarithms modulo $q$ (be specific about which algorithms are involved; you do not need to describe the algorithms in detail).

The **Pohlig-Hellman algorithm** reduces mod$p$ DLP's to a sequence of six easier DLP's (one for each prime factor of $p-1$), with bases of orders 2, 29, ..., 821. All of these are less than 1000, so each of these DLP's are rapidly solved with BSGS. Recombining to obtain the overall solution requires only the **Chinese remainder theorem** & Euclidean algorithm.

P-H gives almost no traction on discrete logarithms mod$q$, since the prime factors of $q-1$ include one only slightly smaller than $q$ itself.

---

Part (b) on reverse side. (2 points)
(b) Let \( N = pq \). Suppose that Eve attempts to factor \( N \) by calling the following function (this is similar to the code provided on Problem Set 7, except that the initial value of \( a \) is chosen to be \( a = 2 \), rather than chosen at random, and it does not bother to check whether or not \( a \) is a unit initially).

\[
\text{def pollardWith2}(N):
\begin{align*}
& \quad a = 2 \\
& \quad j = 2 \\
& \quad \text{while } \text{fractions.gcd}(a-1,N) == 1: \\
& \quad \quad a = \text{pow}(a,j,N) \\
& \quad \quad j += 1 \\
& \quad \text{return fractions.gcd}(a-1,N)
\end{align*}
\]

What will be the return value of this function when called on \( N = pq \)? How many times will the while loop iterate before returning this value?

After \( n \) iterations, the value of \( a \) will be \( a \equiv 2^{(n+1)!} \mod N \).

Now, \( \text{gcd}(a-1,N) \neq 1 \) once either \( p \mid (a-1) \) or \( q \mid (a-1) \).

Since \( 2 \) is a primitive root \( \mod p \),
\[
p \mid \left(2^{(n+1)!} - 1\right) \iff 2^{(n+1)!} \equiv 1 \mod p \\
\iff (p-1) \mid (n+1)! \\
\iff n+1 > 821
\]
(since \( 821! = 821 \times 820 \times \ldots \times 2 \) includes all prime factors of \( p-1 \), while \( 820! \) does not include \( 821 \) as a factor).

Similarly, \( q \mid (2^{(n+1)!} - 1) \iff n+1 > \frac{q-1}{2} = 2199 \ldots 61 \).

So after \( 820 \) iterations, we have \( p \mid (a-1) \) & \( q \nmid (a-1) \), so the function returns \( p \), since \( \text{gcd}(a-1,N) = p \).

(4 points)
(4) (a) Prove that if \( p \) is a prime number, and \( a \) is an integer such that \( a^2 \equiv 1 \pmod{p} \), then either \( a \equiv 1 \pmod{p} \) or \( a \equiv -1 \pmod{p} \).

\[
a^2 \equiv 1 \pmod{p} \\
\Rightarrow \ a^2-1 \equiv 0 \pmod{p} \\
\Rightarrow \ (a+1)(a-1) \equiv 0 \pmod{p} \\
\Rightarrow \ p \mid (a+1)(a-1) \\
\Rightarrow \text{ either } p \mid (a+1) \text{ or } p \mid (a-1) \ (\text{since } p \text{ is prime}) \\
\Rightarrow \text{ either } a \equiv -1 \pmod{p} \text{ or } a \equiv 1 \pmod{p}.
\]

Part (b) on reverse side. (3 points)
(b) Suppose that $p$ is a prime number, $p - 1 = 2^k q$ for $q$ an odd integer, and $a$ is an integer with $1 \leq a \leq N - 1$. Deduce from part (a) that either $a^q \equiv 1 \pmod{p}$ or one of the numbers $a^q, a^{2q}, a^{4q}, \ldots, a^{2^{k-1}q}$ is congruent to $-1$ modulo $p$.

By Fermat's little theorem,

$$a^{p-1} = a^{2^k q} \equiv 1 \pmod{p}.$$

**Case 1:** $a^q \equiv 1 \pmod{p}$. Then there is nothing to prove in this case.

**Case 2**

$a^q \not\equiv 1 \pmod{p}$. Then some of the numbers $a^q, a^{2q}, \ldots, a^{2^{k-1}q} \pmod{p}$

are $1 \pmod{p}$, and others are not, including the first one.

Let $a^{2^{i_0}q}$ be the first one that isn't $1 \pmod{p}$.

Then $i_0 \neq k$, and the next element is $1$, i.e.

$$(a^{2^{i_0}q})^2 = a^{2^{i_0+k}q} \equiv 1 \pmod{p}.$$

By part (a), either $a^{2^{i_0}q} \equiv 1 \pmod{p}$ or $a^{2^{i_0}q} \equiv -1 \pmod{p}$.

By assumption, $a^{2^{i_0}q} \not\equiv 1 \pmod{p}$. So $a^{2^{i_0}q} \equiv -1 \pmod{p}$,

showing that one of these numbers is indeed $\equiv -1 \pmod{p}$.

(3 points)
(5) Suppose that \( p, g \) are public parameters for Elgamal signatures (you may assume that 
\( g \) is a primitive root modulo \( p \)), and that Samantha’s public verification key is \( A \). 
Samantha publishes a valid signature \((S_1, S_2)\) for a document \( D \), and Eve observes 
that \( S_1 \) is exactly equal to \( g \). This might occur if Samantha is not choosing her 
ephemeral key sufficiently randomly.

(a) Assuming that \( \gcd(g, p - 1) = 1 \), write a function `extract(p, g, A, S1, S2, D)` 
that extracts Samantha’s private signing key \( a \) from this information. You may 
assume that you have already implemented a function `ext_euclid(a, b)`, which 
returns a list \([u, v, g] \) such that \( g = \gcd(a, b) \) and \( au + bv = g \). Your code does 
not need to check that \( S_1 = g \), or that \( \gcd(g, p - 1) = 1 \); assume that it will only 
receive input meeting these conditions. Your code should be efficient enough to 
finish in a matter of seconds if all the arguments are 1024 bits long or shorter.

Eve knows that

\[
A^{S_1 \cdot S_2} \equiv g^D \mod p
\]

\[
\text{hence} \quad g^{S_2} \equiv A^{S_1}
\]

\[
hence \quad A^a \cdot g^{S_2} \equiv g^D \mod p
\]

\[
\Rightarrow A^a \equiv g^{D-S_2} \mod p
\]

\[
\Rightarrow g^a \cdot g \equiv g^{D-S_2} \mod p
\]

\[
\Rightarrow a \cdot g \equiv D - S_2 \mod (p-1) \quad \text{(since } g \text{ is order(p-1) })
\]

\[
\Rightarrow a \equiv g^{-1} \cdot (D - S_2) \mod (p-1).
\]

This isn't too hard to compute.

```python
def extract(p, g, A, S1, S2, D):
ginv = ext_euclid(g, p-1)[0]
return invv * (D - S2) \mod (p-1).
```

*Part (b) on reverse side.* (4 points)
(b) Describe briefly how you would modify your code to work in the more general situation where $\gcd(g, p-1)$ is relatively small, but may not be equal to 1. You do not need to write a second program; just clearly describe the steps that you would take.

We can still solve the congruence

$ga \equiv D - S_2 \mod (p-1)$

to obtain a number $a_0$ st.

$a \equiv a_0 \mod \frac{p-1}{\gcd(p-1, g)}$. 

Now we use trial-and-error: for each element $a^\circ$ of

\[
\{a_0 + k \cdot \frac{p-1}{\gcd(p-1, g)} : k = 0, 1, 2, \ldots, \gcd(p-1, g)\}
\]

check whether $g^{a^\circ} \equiv A \mod p$ or not, until success. It's unlikely for $\gcd(p-1, g)$ to be terribly large, so this will likely produce the key $a$ in very short order.

(2 points)
Additional space for work.
## Multiplication table modulo 23

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
<th>21</th>
<th>22</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td>21</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>10</td>
<td>12</td>
<td>14</td>
<td>16</td>
<td>18</td>
<td>20</td>
<td>22</td>
<td>1</td>
<td>3</td>
<td>5</td>
<td>7</td>
<td>9</td>
<td>11</td>
<td>13</td>
<td>15</td>
<td>17</td>
<td>19</td>
<td>21</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>6</td>
<td>9</td>
<td>12</td>
<td>15</td>
<td>18</td>
<td>21</td>
<td>2</td>
<td>5</td>
<td>8</td>
<td>11</td>
<td>14</td>
<td>17</td>
<td>20</td>
<td>23</td>
<td>1</td>
<td>4</td>
<td>7</td>
<td>10</td>
<td>13</td>
<td>16</td>
<td>19</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>8</td>
<td>12</td>
<td>16</td>
<td>20</td>
<td>24</td>
<td>28</td>
<td>4</td>
<td>8</td>
<td>12</td>
<td>16</td>
<td>20</td>
<td>24</td>
<td>28</td>
<td>3</td>
<td>7</td>
<td>11</td>
<td>15</td>
<td>19</td>
<td>23</td>
<td>17</td>
<td>21</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>10</td>
<td>15</td>
<td>20</td>
<td>5</td>
<td>10</td>
<td>15</td>
<td>20</td>
<td>5</td>
<td>10</td>
<td>15</td>
<td>20</td>
<td>5</td>
<td>10</td>
<td>15</td>
<td>20</td>
<td>5</td>
<td>10</td>
<td>15</td>
<td>20</td>
<td>5</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>12</td>
<td>18</td>
<td>24</td>
<td>30</td>
<td>36</td>
<td>42</td>
<td>6</td>
<td>12</td>
<td>18</td>
<td>24</td>
<td>30</td>
<td>36</td>
<td>42</td>
<td>6</td>
<td>12</td>
<td>18</td>
<td>24</td>
<td>30</td>
<td>36</td>
<td>42</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>14</td>
<td>21</td>
<td>28</td>
<td>35</td>
<td>42</td>
<td>49</td>
<td>7</td>
<td>14</td>
<td>21</td>
<td>28</td>
<td>35</td>
<td>42</td>
<td>49</td>
<td>7</td>
<td>14</td>
<td>21</td>
<td>28</td>
<td>35</td>
<td>42</td>
<td>49</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>16</td>
<td>24</td>
<td>32</td>
<td>40</td>
<td>48</td>
<td>56</td>
<td>8</td>
<td>16</td>
<td>24</td>
<td>32</td>
<td>40</td>
<td>48</td>
<td>56</td>
<td>8</td>
<td>16</td>
<td>24</td>
<td>32</td>
<td>40</td>
<td>48</td>
<td>56</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>18</td>
<td>27</td>
<td>36</td>
<td>45</td>
<td>54</td>
<td>63</td>
<td>9</td>
<td>18</td>
<td>27</td>
<td>36</td>
<td>45</td>
<td>54</td>
<td>63</td>
<td>9</td>
<td>18</td>
<td>27</td>
<td>36</td>
<td>45</td>
<td>54</td>
<td>63</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>20</td>
<td>30</td>
<td>40</td>
<td>50</td>
<td>60</td>
<td>70</td>
<td>10</td>
<td>20</td>
<td>30</td>
<td>40</td>
<td>50</td>
<td>60</td>
<td>70</td>
<td>10</td>
<td>20</td>
<td>30</td>
<td>40</td>
<td>50</td>
<td>60</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>22</td>
<td>33</td>
<td>44</td>
<td>55</td>
<td>66</td>
<td>77</td>
<td>11</td>
<td>22</td>
<td>33</td>
<td>44</td>
<td>55</td>
<td>66</td>
<td>77</td>
<td>11</td>
<td>22</td>
<td>33</td>
<td>44</td>
<td>55</td>
<td>66</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>12</td>
<td>24</td>
<td>36</td>
<td>48</td>
<td>60</td>
<td>72</td>
<td>12</td>
<td>24</td>
<td>36</td>
<td>48</td>
<td>60</td>
<td>72</td>
<td>84</td>
<td>12</td>
<td>24</td>
<td>36</td>
<td>48</td>
<td>60</td>
<td>72</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>13</td>
<td>26</td>
<td>39</td>
<td>52</td>
<td>65</td>
<td>78</td>
<td>91</td>
<td>13</td>
<td>26</td>
<td>39</td>
<td>52</td>
<td>65</td>
<td>78</td>
<td>91</td>
<td>13</td>
<td>26</td>
<td>39</td>
<td>52</td>
<td>65</td>
<td>78</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>14</td>
<td>28</td>
<td>42</td>
<td>56</td>
<td>70</td>
<td>84</td>
<td>98</td>
<td>14</td>
<td>28</td>
<td>42</td>
<td>56</td>
<td>70</td>
<td>84</td>
<td>98</td>
<td>14</td>
<td>28</td>
<td>42</td>
<td>56</td>
<td>70</td>
<td>84</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>15</td>
<td>30</td>
<td>45</td>
<td>60</td>
<td>75</td>
<td>90</td>
<td>105</td>
<td>15</td>
<td>30</td>
<td>45</td>
<td>60</td>
<td>75</td>
<td>90</td>
<td>105</td>
<td>15</td>
<td>30</td>
<td>45</td>
<td>60</td>
<td>75</td>
<td>90</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>16</td>
<td>32</td>
<td>48</td>
<td>64</td>
<td>80</td>
<td>96</td>
<td>112</td>
<td>16</td>
<td>32</td>
<td>48</td>
<td>64</td>
<td>80</td>
<td>96</td>
<td>112</td>
<td>16</td>
<td>32</td>
<td>48</td>
<td>64</td>
<td>80</td>
<td>96</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>17</td>
<td>34</td>
<td>51</td>
<td>68</td>
<td>85</td>
<td>102</td>
<td>119</td>
<td>17</td>
<td>34</td>
<td>51</td>
<td>68</td>
<td>85</td>
<td>102</td>
<td>119</td>
<td>17</td>
<td>34</td>
<td>51</td>
<td>68</td>
<td>85</td>
<td>102</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>18</td>
<td>36</td>
<td>54</td>
<td>72</td>
<td>90</td>
<td>108</td>
<td>126</td>
<td>18</td>
<td>36</td>
<td>54</td>
<td>72</td>
<td>90</td>
<td>108</td>
<td>126</td>
<td>18</td>
<td>36</td>
<td>54</td>
<td>72</td>
<td>90</td>
<td>108</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>19</td>
<td>38</td>
<td>57</td>
<td>76</td>
<td>95</td>
<td>114</td>
<td>133</td>
<td>19</td>
<td>38</td>
<td>57</td>
<td>76</td>
<td>95</td>
<td>114</td>
<td>133</td>
<td>19</td>
<td>38</td>
<td>57</td>
<td>76</td>
<td>95</td>
<td>114</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>20</td>
<td>40</td>
<td>60</td>
<td>80</td>
<td>100</td>
<td>120</td>
<td>140</td>
<td>20</td>
<td>40</td>
<td>60</td>
<td>80</td>
<td>100</td>
<td>120</td>
<td>140</td>
<td>20</td>
<td>40</td>
<td>60</td>
<td>80</td>
<td>100</td>
<td>120</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>21</td>
<td>42</td>
<td>63</td>
<td>84</td>
<td>105</td>
<td>126</td>
<td>147</td>
<td>21</td>
<td>42</td>
<td>63</td>
<td>84</td>
<td>105</td>
<td>126</td>
<td>147</td>
<td>21</td>
<td>42</td>
<td>63</td>
<td>84</td>
<td>105</td>
<td>126</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>22</td>
<td>44</td>
<td>66</td>
<td>88</td>
<td>110</td>
<td>132</td>
<td>154</td>
<td>22</td>
<td>44</td>
<td>66</td>
<td>88</td>
<td>110</td>
<td>132</td>
<td>154</td>
<td>22</td>
<td>44</td>
<td>66</td>
<td>88</td>
<td>110</td>
<td>132</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Table 2.2: Diffie-Hellman key exchange

<table>
<thead>
<tr>
<th><strong>Public parameter creation</strong></th>
<th><strong>Key creation</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>A trusted party chooses and publishes a large prime ( p ) and an integer ( g ) having large prime order ( p ).</td>
<td>Choose secret primes ( p ) and ( q ).</td>
</tr>
<tr>
<td></td>
<td>Choose secret exponent ( e ) with ( \gcd(e, \phi(p-1)(q-1)) = 1 ).</td>
</tr>
<tr>
<td>Alice</td>
<td>Bob</td>
</tr>
<tr>
<td>Choose a secret integer ( a ). Compute ( A = g^a \ (\mod p) ).</td>
<td>Choose a secret integer ( b ). Compute ( B = g^b \ (\mod p) ).</td>
</tr>
<tr>
<td>Compute ( \text{gcd}(e(p-1)(q-1), \phi(p-1)(q-1)) = 1 ). Publish ( N = \phi(p-1)(q-1) ).</td>
<td>Publish ( N = pq ) and ( e ).</td>
</tr>
</tbody>
</table>

#### Public exchange of values

1. Alice sends \( A \) to Bob.
2. Bob sends \( B \) to Alice.

#### Further private computations

- Compute \( B^a \ (\mod p) \). Compute \( A^b \ (\mod p) \).
- The shared secret value is \( B^a = g^{ab} = g^{b \cdot a} = (g^b)^a = A^a \ (\mod p) \).

### Table 2.3: ElGamal key creation, encryption, and decryption

<table>
<thead>
<tr>
<th><strong>Public parameter creation</strong></th>
<th><strong>Key creation</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>A trusted party chooses and publishes a large prime ( p ) and an element ( g ) modulo ( p ) of large (prime) order.</td>
<td>Choose secret signing key ( 1 \leq a \leq p-1 ).</td>
</tr>
<tr>
<td></td>
<td>Compute ( A = g^a \ (\mod p) ). Publish the public key ( A ).</td>
</tr>
<tr>
<td>Alice</td>
<td>Bob</td>
</tr>
<tr>
<td>Choose private key ( 1 \leq a \leq p-1 ). Compute ( A = g^a \ (\mod p) ). Publish the public key ( A ).</td>
<td></td>
</tr>
<tr>
<td>Choose plaintext ( m ). Choose random element ( k ).</td>
<td></td>
</tr>
<tr>
<td>Use Alice's public key ( A ) to compute ( c_1 = g^k \ (\mod p) ) and ( c_2 = ma^k \ (\mod p) ). Send ciphertext ( (c_1, c_2) ) to Alice.</td>
<td></td>
</tr>
</tbody>
</table>

#### Encryption

<table>
<thead>
<tr>
<th><strong>Decryption</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Compute ( (c_1)^{-1} \cdot c_2 \ (\mod p) ).</td>
</tr>
<tr>
<td>This quantity is equal to ( m ).</td>
</tr>
</tbody>
</table>

### Table 3.1: RSA key creation, encryption, and decryption

<table>
<thead>
<tr>
<th><strong>Public parameter creation</strong></th>
<th><strong>Key creation</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>A trusted party chooses and publishes a large prime ( p ) and ( q ) satisfying ( p \equiv 1 \ (\mod q) ) and ( q ) of order ( q ) modulo ( p ).</td>
<td>Choose secret signing key ( 1 \leq a \leq q-1 ).</td>
</tr>
<tr>
<td></td>
<td>Compute ( A = g^a \ (\mod p) ). Publish the verification key ( A ).</td>
</tr>
<tr>
<td>Samantha</td>
<td>Victor</td>
</tr>
<tr>
<td>Choose secret primes ( p ) and ( q ). Choose encryption exponent ( e ) with ( \gcd(e, p-1)(q-1) = 1 ). Publish ( N = pq ) and ( e ).</td>
<td></td>
</tr>
<tr>
<td>Choose document ( D ) mod ( p ). Choose random element ( 1 \leq k &lt; q ) satisfying ( \gcd(k, \phi(p-1)(q-1)) = 1 ). Compute signature ( S_1 = g^k \ (\mod p) ) and ( S_2 = (D - aS_1)k^{-1} \ (\mod p-1) ).</td>
<td></td>
</tr>
</tbody>
</table>

#### Signing

<table>
<thead>
<tr>
<th><strong>Verification</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Compute ( A^{S_1}S_2^{-1} \ (\mod p) ). Verify that it is equal to ( g^D \ (\mod p) ).</td>
</tr>
</tbody>
</table>

### Table 4.1: RSA digital signatures

#### Public parameter creation

<table>
<thead>
<tr>
<th><strong>Public parameter creation</strong></th>
<th><strong>Key creation</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>A trusted party chooses and publishes a large prime ( p ) and ( q ) primitive root ( g ) modulo ( p ).</td>
<td>Choose secret primes ( p ) and ( q ).</td>
</tr>
<tr>
<td></td>
<td>Choose secret signing key ( 1 \leq a \leq p-1 ).</td>
</tr>
<tr>
<td></td>
<td>Compute ( A = g^a \ (\mod p) ).</td>
</tr>
<tr>
<td>Samantha</td>
<td>Victor</td>
</tr>
<tr>
<td>Choose secret primes ( p ) and ( q ). Choose encryption exponent ( e ) with ( \gcd(e, \phi(p-1)(q-1)) = 1 ). Publish ( N = pq ) and ( e ).</td>
<td></td>
</tr>
<tr>
<td>Choose signing key ( 1 \leq a \leq q-1 ). Compute ( A = g^a \ (\mod p) ). Publish the verification key ( A ).</td>
<td></td>
</tr>
</tbody>
</table>

#### Signing

<table>
<thead>
<tr>
<th><strong>Verification</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Compute ( A^{S_1}S_2^{-1} \ (\mod p) ) and ( V_2 = S_1S_2^{-1} \ (\mod q) ). Verify that ( (g^{V_2}A^{V_2}) \ (\mod p) ) mod ( q = S_1 ).</td>
</tr>
</tbody>
</table>

### Table 4.2: The ElGamal digital signature algorithm

#### Public parameter creation

<table>
<thead>
<tr>
<th><strong>Public parameter creation</strong></th>
<th><strong>Key creation</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>A trusted party chooses and publishes large prime ( p ) and ( q ) satisfying ( p \equiv 1 \ (\mod q) ) and ( q ) of order ( q ) modulo ( p ).</td>
<td>Choose secret signing key ( 1 \leq a \leq p-1 ).</td>
</tr>
<tr>
<td></td>
<td>Choose signing key ( 1 \leq a \leq q-1 ).</td>
</tr>
<tr>
<td></td>
<td>Compute ( A = g^a \ (\mod p) ). Publish the verification key ( A ).</td>
</tr>
<tr>
<td>Samantha</td>
<td>Victor</td>
</tr>
<tr>
<td>Choose secret primes ( p ) and ( q ). Choose encryption exponent ( e ) with ( \gcd(e, p-1)(q-1) = 1 ). Publish ( N = pq ) and ( e ).</td>
<td></td>
</tr>
<tr>
<td>Choose document ( D ) mod ( q ). Choose random element ( 1 \leq k &lt; q ). Compute signature ( S_1 = g^k \ (\mod p) ) and ( S_2 = (D - aS_1)k^{-1} \ (\mod p-1) ).</td>
<td></td>
</tr>
</tbody>
</table>

#### Signing

<table>
<thead>
<tr>
<th><strong>Verification</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Compute ( V_1 = DS_1^{-1} \ (\mod q) ) and ( V_2 = S_1S_2^{-1} \ (\mod q) ). Verify that ( (g^{V_1}A^{V_1}) \ (\mod p) ) mod ( q = S_1 ).</td>
</tr>
</tbody>
</table>

### Table 4.3: The digital signature algorithm (DSA)