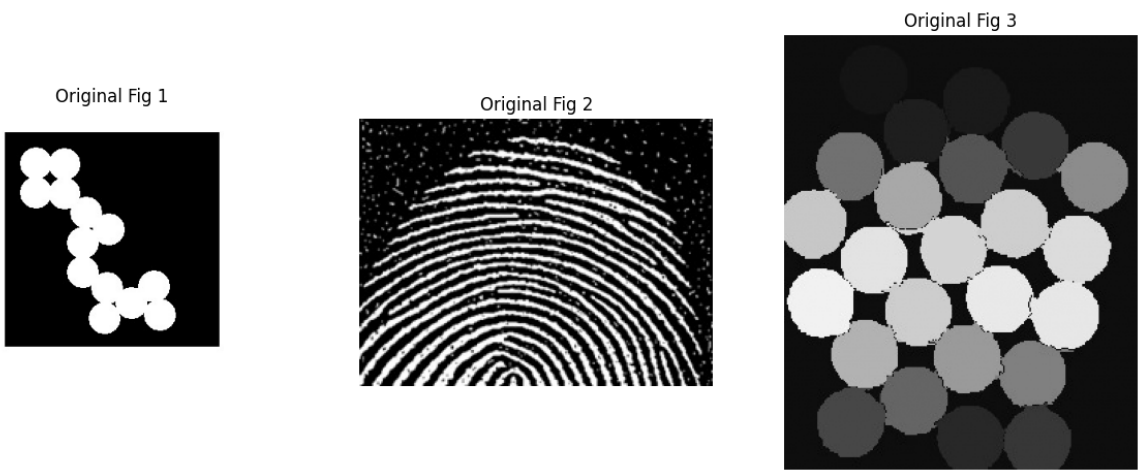
Setup

In [9]:
```
### Import required libraries
import cv2
import numpy as np
import matplotlib.pyplot as plt
from skimage import io, color, img_as_ubyte, feature
```

In [10]:
```
# Function to display images
def display_images(images, titles, rows=1, cols=3):
    """Display multiple images in a grid layout."""
    plt.figure(figsize=(15, 10))
    for i in range(len(images)):
        plt.subplot(rows, cols, i + 1)
        plt.imshow(images[i], cmap='gray')
        plt.title(titles[i])
        plt.axis('off')
    plt.show()

base_path = r'C:\Users\danis\OneDrive\Documents\Repos\CV_PE2\Practicals\P3_\data'

# Load images for the tasks
fig1 = cv2.imread( rf"{base_path}\fig1.jpg", 0)  # Replace with the path to Fig 1
fig2 = cv2.imread( rf"{base_path}\fig2.jpg", 0)  # Replace with the path to Fig 1
fig3 = cv2.imread( rf"{base_path}\fig3.jpg", 0)  # Replace with the path to Fig 1

# Display original images
display_images([fig1, fig2, fig3], ['Original Fig 1', 'Original Fig 2', 'Original F
```



Task 1: Erode and Count Balls

In [11]:
```
# Lab Task 1: Perform Erosion to separate balls in Fig 1
def task1_erosion(image, kernel_size=8):
    kernel = np.ones((kernel_size, kernel_size), np.uint8)
    eroded = cv2.erode(image, kernel, iterations=3)
    return eroded


# Optional: Count objects in the eroded image (Lab Task 1 extension)
```

```
def count_objects(eroded_image):
    _, thresh = cv2.threshold(eroded_image, 127, 255, cv2.THRESH_BINARY)
    num_labels, _ = cv2.connectedComponents(thresh)
    return num_labels - 1  # Subtract 1 for the background



# Perform operations for Lab Tasks
eroded_fig1 = task1_erosion(fig1)

# Display results
display_images([fig1, eroded_fig1], ['Original Fig 1', 'Eroded Fig 1'])
# Count objects in Fig 1 after erosion
num_balls = count_objects(eroded_fig1)
print(f"Total number of balls in Fig 1: {num_balls}")
```

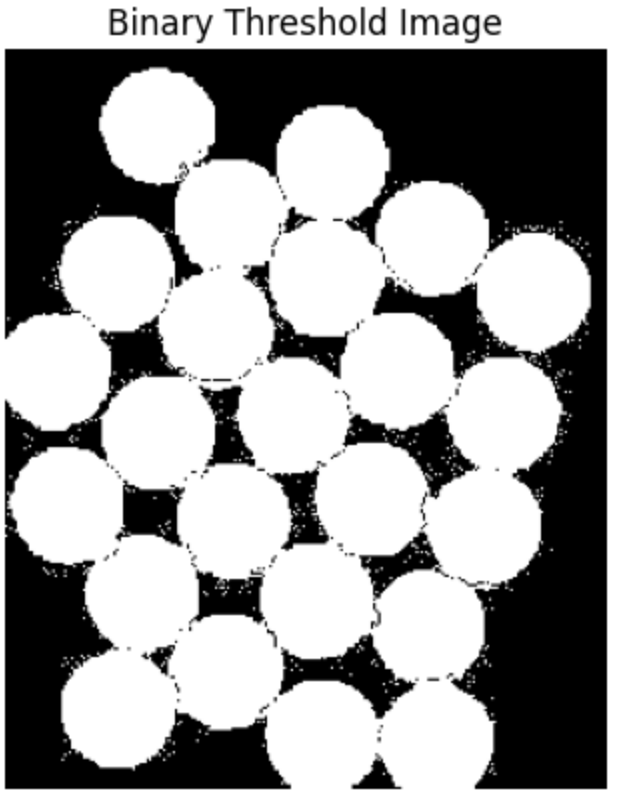

```
Total number of balls in Fig 1: 14
```

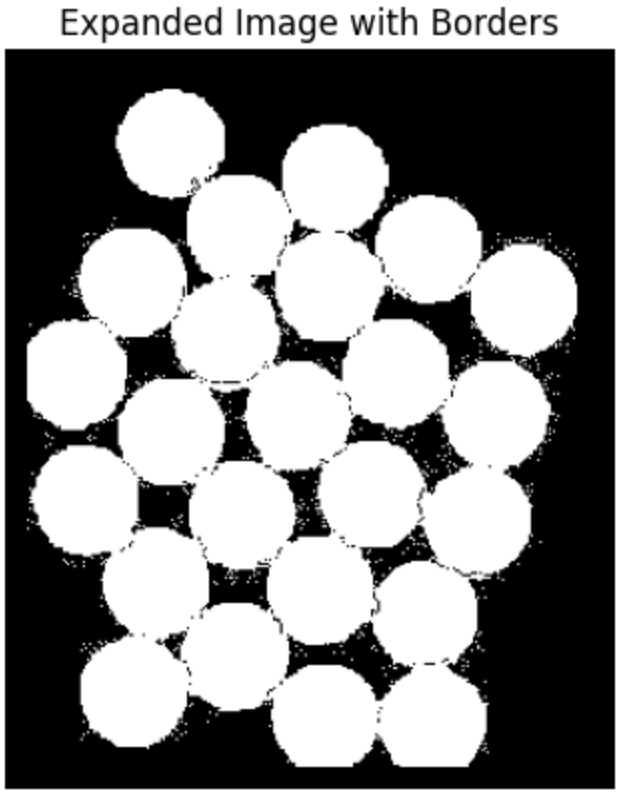
Task 2: Fingerprint

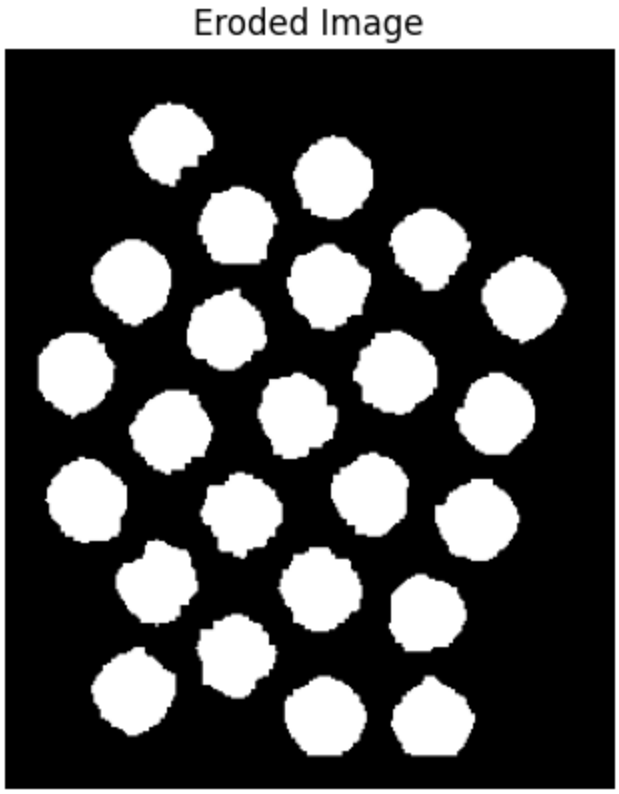
In [12]:
```
# Convert to binary image (if it's not already binary)
_, binary_fig2 = cv2.threshold(fig2, 127, 255, cv2.THRESH_BINARY)

# Define a 3x3 kernel for morphological operations
kernel1 = np.ones((2, 2), np.uint8)
kernel2 = np.ones((1, 1), np.uint8)

# Apply Morphological Opening to remove noise (small points)
opened_fig2 = cv2.morphologyEx(binary_fig2, cv2.MORPH_OPEN, kernel1)

# Apply Morphological Closing to fill gaps between ridges (or thumb impressions)
closed_fig2 = cv2.morphologyEx(opened_fig2, cv2.MORPH_CLOSE, kernel2)


# Display the images: Original, Opened, and Closed
display_images([fig2, opened_fig2, closed_fig2], ['Original Fig 2 (Fingerprint)', '
```

Morphological Gradient

In [13]:
```
# Apply binary threshold
ret, binary_thresh = cv2.threshold(fig3, 18, 255, cv2.THRESH_BINARY)

# Display the binary threshold image
plt.imshow(binary_thresh, cmap='gray')
plt.title('Binary Threshold Image')
plt.axis('off')
plt.show()


# Expand the borders of the image by 10 pixels on each side and fill with 0s
expanded_image = cv2.copyMakeBorder(binary_thresh, 10, 10, 10, 10, cv2.BORDER_CONST

# Display the expanded image
plt.imshow(expanded_image, cmap='gray')
plt.title('Expanded Image with Borders')
plt.axis('off')
plt.show()


# Define the kernel for erosion
kernel = np.ones((11, 11), np.uint8)

# Apply erosion
eroded = cv2.erode(expanded_image, kernel, iterations=1)

# Display the eroded image
plt.imshow(eroded, cmap='gray')
plt.title('Eroded Image')
plt.axis('off')
plt.show()



# Find contours in the eroded image
contours, _ = cv2.findContours(eroded, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

# Count the number of contours
ball_count = len(contours)

print(f"Number of balls: {ball_count}")
```


Binary Threshold Image


Expanded Image with Borders




```
Number of balls: 24
```