```
import numpy as np
import pandas as pd
from cryptography.hazmat.primitives.asymmetric import ec
from cryptography.hazmat.primitives import serialization, hashes
from cryptography.hazmat.primitives.asymmetric.utils import decode_dss_signature
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
import time
# Simulate IoT network data (smart city devices)
np.random.seed(42)
def generate_iot_data(n_samples=1000):
    data = {
        'packet_rate': np.random.normal(100, 20, n_samples),  # Packets per second
        'data_size': np.random.normal(500, 100, n_samples),   # Bytes
        'connection_duration': np.random.normal(10, 2, n_samples),  # Seconds
        'is_attack': np.random.choice([0, 1], n_samples, p=[0.8, 0.2])  # 20% DDoS attacks
    }
    return pd.DataFrame(data)
# Enhanced ECC Implementation
def generate_ecc_keys():
    private_key = ec.generate_private_key(ec.SECP256R1())  # 256-bit curve for EECC
    public_key = private_key.public_key()
    return private_key, public_key
def encrypt_data(data, public_key):
    # Simulate encryption (hashing for simplicity)
    digest = hashes.Hash(hashes.SHA256())
    digest.update(data.encode())
    signature = private_key.sign(digest.finalize(), ec.ECDSA(hashes.SHA256()))
```

```
    return signature
def verify_data(data, signature, public_key):
    digest = hashes.Hash(hashes.SHA256())
    digest.update(data.encode())
    try:
        public_key.verify(signature, digest.finalize(), ec.ECDSA(hashes.SHA256()))
        return True
    except:
        return False
# ML Model for DDoS Detection
def train_ml_model(data):
    X = data[['packet_rate', 'data_size', 'connection_duration']]
    y = data['is_attack']
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
    model = RandomForestClassifier(n_estimators=100, random_state=42)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    cm = confusion_matrix(y_test, y_pred)
    return model, accuracy, cm
# Simulate IoT Network and Security Process
def simulate_iot_network():
    print("Simulating IoT Smart City Network...")
    data = generate_iot_data()
    # Generate ECC keys
    private_key, public_key = generate_ecc_keys()
     # Encrypt and verify sample data
    sample_data = "Smart City Sensor Data"
    signature = encrypt_data(sample_data, public_key)
    is_verified = verify_data(sample_data, signature, public_key)
    print(f"Data Verification: {'Successful' if is_verified else 'Failed'}")
```

```
    # Train ML model
    model, accuracy, cm = train_ml_model(data)
    print(f"DDoS Detection Accuracy: {accuracy*100:.2f}%")
        # Visualize Results
    plt.figure(figsize=(10, 5))
        # Confusion Matrix
    plt.subplot(1, 2, 1)
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)
    plt.title('Confusion Matrix for DDoS Detection')
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
        # Feature Importance
    plt.subplot(1, 2, 2)
    feature_importance = pd.Series(model.feature_importances_, index=['Packet Rate', 'Data Size',
'Connection Duration'])
    feature_importance.plot(kind='bar', color='green')
    plt.title('Feature Importance in DDoS Detection')
    plt.ylabel('Importance')
    plt.tight_layout()
    plt.savefig('iot_security_results.png')
    plt.show()
if __name__ == "__main__":
    simulate_iot_network()
```