```python
import requests
import validators # type: ignore
from urllib.parse import urlparse

# 1. Check if a URL is valid
def is_valid_url(url):
    return validators.url(url)

# 2. Check if URL uses HTTPS
def uses_https(url):
    return url.startswith("https://")

# 3. Check for suspicious keywords in URL
def contains_suspicious_keywords(url):
    suspicious_keywords = ['login', 'account', 'secure', 'verify', 'bank', 'pay', 'update', 'signin']
    for keyword in suspicious_keywords:
        if keyword in url.lower():
            return True
    return False

# 4. Check if domain is a known suspicious domain (using a dummy blacklist here for demonstration)
def is_suspicious_domain(url):
    suspicious_domains = ["phishing.com", "malicioussite.com"]  # Example, replace with a real blacklist
    domain = urlparse(url).netloc
    return domain in suspicious_domains

# 5. Check if the URL contains URL shorteners
def contains_url_shortener(url):
    url_shorteners = ['bit.ly', 'goo.gl', 't.co', 'tinyurl.com']
    for shortener in url_shorteners:
        if shortener in url:
            return True
    return False

# 6. Perform basic checks on the URL
def scan_url(url):
    if not is_valid_url(url):
        return "Invalid URL format."

    # 6.1. Check for HTTPS
    if not uses_https(url):
        return "Warning: URL does not use HTTPS."

    # 6.2. Check for suspicious keywords
    if contains_suspicious_keywords(url):
        return "Warning: URL contains suspicious keywords."

    # 6.3. Check for suspicious domain
    if is_suspicious_domain(url):
        return "Warning: URL belongs to a suspicious domain."

    # 6.4. Check for URL shorteners
    if contains_url_shortener(url):
        return "Warning: URL uses a URL shortener."

    return "URL looks safe."

# 7. Check URL against Google Safe Browsing API (Optional, requires API key)
def check_with_google_safe_browsing(url):
    api_key = 'YOUR_GOOGLE_API_KEY'
    safe_browsing_url = 'https://safebrowsing.googleapis.com/v4/threatMatches:find'
    payload = {
        "client": {
            "clientId": "your-client-id",
            "clientVersion": "1.0"
        },
        "threatInfo": {
            "threatTypes": ["MALWARE", "SOCIAL_ENGINEERING"],
            "platformTypes": ["ANY_PLATFORM"],
            "threatEntryTypes": ["URL"],
            "threatEntries": [{"url": url}]
        }
    }

    headers = {
        'Content-Type': 'application/json',
    }

    try:
        response = requests.post(safe_browsing_url, json=payload, headers=headers, params={'key': api_key})
        result = response.json()
        if 'matches' in result:
            return "Warning: Google Safe Browsing detected the URL as malicious."
        return "URL is clean according to Google Safe Browsing."
    except requests.exceptions.RequestException as e:
        return f"Error checking with Google Safe Browsing: {e}"

# 8. Main function to run the scanner
def phishing_link_scanner(url):
    result = scan_url(url)
    if result == "URL looks safe.":
        # Optionally, check with Google Safe Browsing API if needed
        google_safe_result = check_with_google_safe_browsing(url)
        return result + " " + google_safe_result
    return result

# Test URLs
test_urls = [
    "https://www.example.com/login",
    "http://phishing.com/secure-account",
    "https://bit.ly/123456",
    "http://malicioussite.com/login",
    "https://safe.com/update-account"
]

for test_url in test_urls:
    print(f"URL: {test_url}")
    print(f"Result: {phishing_link_scanner(test_url)}\n")
```