

**HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG  
KHOA AN TOÀN THÔNG TIN**



**BÁO CÁO BÀI TẬP LỚN  
HỌC PHẦN: KỸ THUẬT GIẤU TIN**

**ĐỀ TÀI: STEG-CFG-BASIC**

<Mã sinh viên>      <Họ tên sinh viên>

Tên nhóm: <số nhóm>

Tên lớp: <Tên lớp>

Giảng viên hướng dẫn: <Chức danh> + <Họ tên GV>

**HÀ NỘI 3-2025**

# BÀI THỰC HÀNH GIẤU TIN VĂN BẢN CFG DỰA TRÊN TEMPLATE VÀ MÃ HÓA HUFFMAN

## 1.1 Mục đích

- Hiểu và thực hành nguyên lý giấu tin trong văn bản tự nhiên sử dụng ngữ pháp phi ngữ cảnh (CFG)
- Áp dụng kỹ thuật mã hóa thông điệp bí mật vào văn bản dựa trên cấu trúc ngữ pháp
- Phát hiện và giải mã thông tin ẩn từ văn bản đã mã hóa.
- Tự xây dựng và kiểm thử ngữ pháp tùy chỉnh để phục vụ mục đích giấu tin

## 1.2 Nội dung thực hành

### Tải bài lab

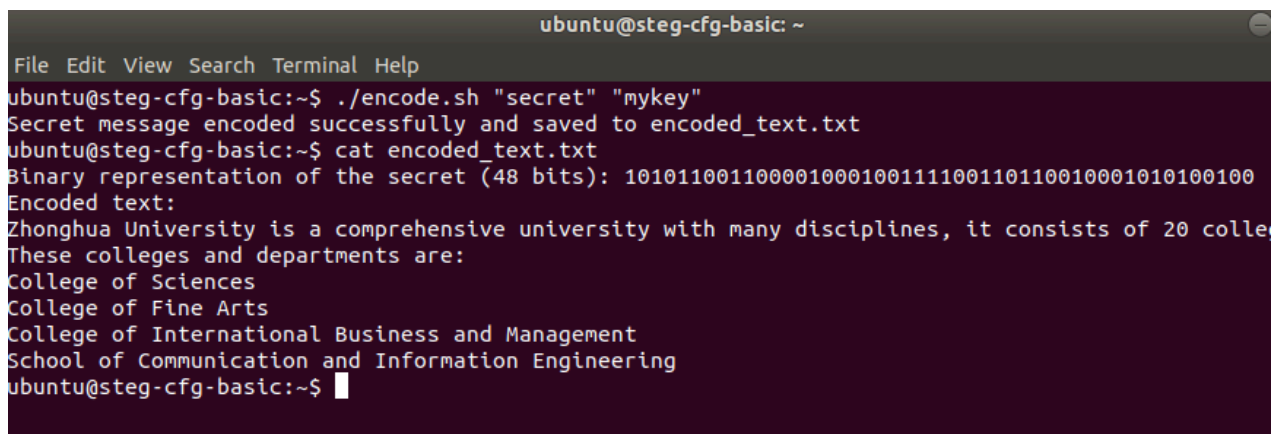
imodule <https://github.com/hieunm2025/steg-cfg-basic/raw/refs/heads/main/imodule.tar>

### 1.2.1 Khởi động bài lab

```
labtainer -r steg-cfg-basic
```

Thử chạy với mã hóa thông điệp template

```
./encode.sh "secret" "mykey"
```

A screenshot of a terminal window titled 'ubuntu@steg-cfg-basic: ~'. The terminal shows the execution of the script './encode.sh "secret" "mykey"'. The output indicates that the secret message was encoded successfully and saved to 'encoded\_text.txt'. It then shows the contents of 'encoded\_text.txt', which is a binary representation of the secret (48 bits) followed by the encoded text: 'Zhonghua University is a comprehensive university with many disciplines, it consists of 20 colleges and departments are: College of Sciences, College of Fine Arts, College of International Business and Management, School of Communication and Information Engineering'.

```
ubuntu@steg-cfg-basic: ~  
File Edit View Search Terminal Help  
ubuntu@steg-cfg-basic:~$ ./encode.sh "secret" "mykey"  
Secret message encoded successfully and saved to encoded_text.txt  
ubuntu@steg-cfg-basic:~$ cat encoded_text.txt  
Binary representation of the secret (48 bits): 101011001100001000100111100110110010001010100100  
Encoded text:  
Zhonghua University is a comprehensive university with many disciplines, it consists of 20 colleges and departments are:  
College of Sciences  
College of Fine Arts  
College of International Business and Management  
School of Communication and Information Engineering  
ubuntu@steg-cfg-basic:~$
```

```
./decode.sh grammar.cfg encoded_text.txt "mykey"
```

```
ubuntu@steg-cfg-basic: ~
File Edit View Search Terminal Help
ubuntu@steg-cfg-basic:~$ ./encode.sh "secret" "mykey"
Secret message encoded successfully and saved to encoded_text.txt
ubuntu@steg-cfg-basic:~$ cat encoded_text.txt
Binary representation of the secret (48 bits): 101011001100001000100111100110110010001010100100
Encoded text:
Zhonghua University is a comprehensive university with many disciplines, it consists of 20 colle
These colleges and departments are:
College of Sciences
College of Fine Arts
College of International Business and Management
School of Communication and Information Engineering
ubuntu@steg-cfg-basic:~$ ./decode.sh grammar.cfg encoded_text.txt "mykey"
[+] Detected potential hidden bits: 011100
[+] Detected 6 bits of hidden information

[+] Hidden information detected!

Attempting to decode with provided keys:
Extracted bits: 011100
Using key 'mykey': Potential message with key 'mykey'
ubuntu@steg-cfg-basic:~$
```

Tuy nhiên, thông điệp "**secret**" có độ dài 48 bit, nhưng quá trình giải mã mới chỉ tìm thấy **6 bit**.

Sinh viên chỉnh các file để giải mã thành công.

Điều chỉnh file grammar.cfg:

- Đảm bảo tất cả các non-terminal (trong quy tắc Start ) đều có ít nhất 2 lựa chọn để mã hóa 1 bit.
- Nếu số lượng lựa chọn là lẻ (không phải lũy thừa 2), cần bổ sung thêm để đảm bảo đủ bit

### **\*grammar.cfg**

Start → adjective1 adjective2 phrase1 noun3 location reputation number1 noun1 feature quality specialty adjective1 phrase1 noun3 status attribute extra

adjective1 → acclaimed || distinguished || renowned || prestigious || eminent || outstanding || leading || notable

adjective2 → innovative || dynamic || creative || resourceful || versatile || strategic || adaptive || flexible

phrase1 → a range of || various || an array of || multiple || several || diverse || extensive || numerous

noun3 → programs || courses || disciplines || majors || fields || tracks || concentrations || degrees

location → Beijing || Shanghai || Guangzhou || Nanjing || Wuhan || Chengdu || Hangzhou || Xian

reputation → honored || respected || reputable || celebrated || esteemed || recognized || admired || lauded

number1 → 20 || 25 || 30 || 35 || 40 || 45 || 50 || 55

noun1 → departments || faculties || schools || divisions || branches || sectors || units || institutes

feature → advanced || modern || state-of-the-art || innovative || cutting-edge || leading || premium || superior

quality → prestigious || top-ranked || world-class || elite || high-prestigious || outstanding || reputable || excellent

specialty → operating systems || VR/AR || graphics || NLP || databases || distributed systems || mobile computing || algorithms

status → leading || prominent

attribute → global || regional || national || international || provincial || city-level || district-level || municipal

extra → diverse || specialized || comprehensive || multidisciplinary

\*[generator.py](#)

Sửa hàm encode trong [generator.py](#):

- Sửa lại cách tính số bit cần thiết để chọn từ các tùy chọn:  
bits\_needed = math.ceil(math.log2(num\_options))
- Điều này đảm bảo đủ bit ngay cả khi số tùy chọn không phải là lũy thừa của 2.

```
#!/usr/bin/env python3
```

```
import sys
```

```
import re
```

```
import math
```

```
class CFGSteganography:
```

```
    def __init__(self, grammar_file):
```

```
        self.grammar = {}
```

```
        self.load_grammar(grammar_file)
```

```
    def load_grammar(self, grammar_file):
```

```
        with open(grammar_file, 'r') as f:
```

```

        lines = f.readlines()
current_symbol = None
for line in lines:
    line = line.strip()
    if not line:
        continue
    if '→' in line:
        parts = line.split('→')
        current_symbol = parts[0].strip()
        self.grammar[current_symbol] = []
        if len(parts) > 1:
            options_str = parts[1].strip()
            if '|' in options_str:
                self.grammar[current_symbol] = [opt.strip() for opt in options_str.split('|')]
            else:
                self.grammar[current_symbol] = [options_str]
        elif current_symbol:
            self.grammar[current_symbol].append(line)

def encode(self, binary_data):
    if 'Start' not in self.grammar:
        print("Error: Grammar must have a 'Start' rule", file=sys.stderr)
        sys.exit(1)

    start_rule_content = self.grammar['Start'][0]
    non_terminals_in_order = start_rule_content.split()

    calculated_total_bits = 0
    for symbol in non_terminals_in_order:
        if symbol in self.grammar:

```

```

options = self.grammar[symbol]
num_options = len(options)
if num_options >= 2:
    bits_needed = math.floor(math.log2(num_options))
    calculated_total_bits += bits_needed
else:
    print(f"Warning: Symbol '{symbol}' in Start rule not defined in grammar.",
file=sys.stderr)

if calculated_total_bits != 48:
    print(f"Warning: Grammar configuration provides {calculated_total_bits} bits, but
48 bits are expected for the message.", file=sys.stderr)
    if calculated_total_bits < 48:
        print("Error: Grammar provides insufficient bits. Please correct grammar.cfg.",
file=sys.stderr)
        sys.exit(1)

output_parts = []
bit_index = 0

for symbol in non_terminals_in_order:
    if symbol in self.grammar:
        options = self.grammar[symbol]
        num_options = len(options)

        if num_options >= 2 and bit_index < len(binary_data):
            bits_needed = math.floor(math.log2(num_options))
            if bit_index + bits_needed > len(binary_data):
                print(f"Warning: Not enough bits in message for symbol {symbol}. Using
default.", file=sys.stderr)
                selected_option = options[0]

```

```

else:
    bits_for_symbol = binary_data[bit_index : bit_index + bits_needed]
    if not bits_for_symbol:
        choice_index = 0
    else:
        choice_index = int(bits_for_symbol, 2)

    if choice_index >= num_options:
        print(f"Warning: Choice index {choice_index} out of bounds for
{symbol} (options: {num_options}). Using default.", file=sys.stderr)
        choice_index = 0
        selected_option = options[choice_index]
        bit_index += bits_needed
    else:
        selected_option = options[0] if options else symbol
        output_parts.append(selected_option)
    else:
        output_parts.append(symbol)

if bit_index < len(binary_data) and bit_index < 48 :
    print(f"Warning: Not all bits from the secret message were used. {len(binary_data)
- bit_index} bits remaining.", file=sys.stderr)

return " ".join(output_parts)

def str_to_binary(message, key):
    binary_message = "".join(format(ord(c), '08b') for c in message)
    if len(binary_message) < 48:
        return binary_message.ljust(48, '0')
    return binary_message[:48]

```

```

def main():
    if len(sys.argv) < 4:
        print(f"Usage: python {sys.argv[0]} <grammar_file> <secret_message> <key>",
file=sys.stderr)
        sys.exit(1)

    grammar_file = sys.argv[1]
    secret_message = sys.argv[2]
    # key = sys.argv[3]

    if len(secret_message) * 8 < 48 and len(secret_message) < 6 :
        print(f"Warning: Secret message '{secret_message}' is too short to produce 48 bits
directly. It will be padded.", file=sys.stderr)
    if len(secret_message) * 8 > 48 and len(secret_message) > 6:
        print(f"Warning: Secret message '{secret_message}' is too long. It will be truncated to
fit 48 bits.", file=sys.stderr)

    binary_data = str_to_binary(secret_message, "")
    print(f"Binary representation of the secret (48 bits): {binary_data}", file=sys.stderr)
    steg = CFGSteganography(grammar_file)
    encoded_text = steg.encode(binary_data)
    print(encoded_text)

if __name__ == "__main__":
    main()

```

[\\*detector.py](#)

```

#!/usr/bin/env python3

import sys
import re
import math

```



```

class StegDetector:
    def __init__(self, grammar_file):
        self.grammar = {}
        self.load_grammar(grammar_file)

    def load_grammar(self, grammar_file):
        with open(grammar_file, 'r') as f:
            lines = f.readlines()
            current_symbol = None
            for line in lines:
                line = line.strip()
                if not line:
                    continue
                if '→' in line:
                    parts = line.split('→')
                    current_symbol = parts[0].strip()
                    self.grammar[current_symbol] = []
                    if len(parts) > 1:
                        options_str = parts[1].strip()
                        if '|' in options_str:
                            self.grammar[current_symbol] = [opt.strip() for opt in options_str.split('|')]
                        else:
                            self.grammar[current_symbol] = [options_str]
                    elif current_symbol:
                        self.grammar[current_symbol].append(line)

    def detect(self, text_content):
        if 'Start' not in self.grammar:
            print("Error: Grammar must have a 'Start' rule", file=sys.stderr)
            return False, ""

```

```

start_rule_content = self.grammar['Start'][0]
non_terminals_in_order = start_rule_content.split()

# Tất cả các "từ" từ văn bản mã hóa
all_text_words = text_content.split()

extracted_bits_list = []
total_bits_expected_from_grammar = 0
current_text_word_index = 0 # Theo dõi vị trí hiện tại trong all_text_words

        print("[+] Non-terminals and expected order from grammar:",
non_terminals_in_order, file=sys.stderr)

for symbol_name in non_terminals_in_order:
    if symbol_name in self.grammar:
        options = self.grammar[symbol_name] # Các sản phẩm có thể có của
non-terminal này
        num_options = len(options)

        if num_options >= 2:
            bits_for_symbol = math.floor(math.log2(num_options))
            total_bits_expected_from_grammar += bits_for_symbol

            matched_this_symbol = False
            # Sắp xếp các options theo số lượng từ giảm dần để ưu tiên khớp dài nhất
trước

            # Điều này giúp xử lý các trường hợp như "a" vs "a range of"
            sorted_options_with_indices = sorted(
                [(idx, prod.split()) for idx, prod in enumerate(options)],
                key=lambda x: len(x[1]),

```

```

        reverse=True
    )

    for original_idx, production_as_word_list in sorted_options_with_indices:
        num_words_in_production = len(production_as_word_list)

        # Kiểm tra xem có đủ từ còn lại trong văn bản mã hóa để khớp không
        if current_text_word_index + num_words_in_production <=
len(all_text_words):
            # Lấy ra đoạn văn bản để so sánh
            text_segment_to_compare = all_text_words[current_text_word_index :
current_text_word_index + num_words_in_production]

            # So sánh (không phân biệt chữ hoa/thường)
            if [w.lower() for w in text_segment_to_compare] == [p_w.lower() for
p_w in production_as_word_list]:
                binary_representation = format(original_idx, f'0{bits_for_symbol}b')
                extracted_bits_list.append(binary_representation)

                print(f"[+] Symbol: {symbol_name}, Matched Option: '{'
'.join(production_as_word_list)}' (Index: {original_idx}), Bits: {binary_representation}",
file=sys.stderr)

                current_text_word_index += num_words_in_production
                matched_this_symbol = True

                break # Đã khớp với một production, chuyển sang non-terminal tiếp
theo

            if not matched_this_symbol:
                print(f"[!] Error: For symbol '{symbol_name}', could not match any
production with the text segment starting near: '{'
'.join(all_text_words[current_text_word_index:current_text_word_index+3])' if
current_text_word_index < len(all_text_words) else 'EOF'}.", file=sys.stderr)

                print(f"    Available options for {symbol_name}: {options}",
file=sys.stderr)

```

```

        # Xử lý lỗi: thêm bits mặc định và dừng hoặc cố gắng tiếp tục
        # Để đảm bảo độ dài bit, nhưng có thể sai:
        # binary_representation = format(0, f'0{bits_for_symbol}b')
        # extracted_bits_list.append(binary_representation)
        # print(f"    Defaulting to 0-bits for {symbol_name}.", file=sys.stderr)
        # current_text_word_index += 1 # Cố gắng bỏ qua 1 từ, nhưng rất rủi ro
        return False, "".join(extracted_bits_list) # Lỗi parsing nghiêm trọng
    else:
        print(f"Warning: Symbol '{symbol_name}' from Start rule not in grammar
definitions.", file=sys.stderr)

    final_binary_string = "".join(extracted_bits_list)

    print(f"[+] Total bits expected from grammar definition (should be 48):
{total_bits_expected_from_grammar}", file=sys.stderr)

    if current_text_word_index != len(all_text_words):
        print(f"[!] Warning: Not all words from the input text were consumed during
parsing.    Remaining:    '{'    '.join(all_text_words[current_text_word_index:])}'",
file=sys.stderr)

    if total_bits_expected_from_grammar != 48:
        print(f"[!] CRITICAL WARNING: The grammar is configured to produce
{total_bits_expected_from_grammar} bits, not 48. Decoding will likely fail or be
incorrect. PLEASE FIX grammar.cfg.", file=sys.stderr)

    print(f"[+] Detected potential hidden bits: {final_binary_string}", file=sys.stderr)
    print(f"[+] Detected {len(final_binary_string)} bits of hidden information.",
file=sys.stderr)

    if len(final_binary_string) == 48 :
        return True, final_binary_string

```

```

else:
    print(f"[!] Error: Final extracted bit count ({len(final_binary_string)}) does not
match expected 48 bits. Check parsing logic and grammar.cfg.", file=sys.stderr)
    return False, final_binary_string

def binary_to_message(self, binary_string):
    if len(binary_string) != 48:
        return f"Failed to recover message: extracted {len(binary_string)} bits, expected
48"

    message = ""
    try:
        for i in range(0, 48, 8):
            byte = binary_string[i:i+8]
            if len(byte) == 8:
                message += chr(int(byte, 2))
            else:
                print(f"Warning: Incomplete byte at end of bit string: {byte}", file=sys.stderr)
                break

        return f"Recovered message: {message}"
    except ValueError:
        return "Failed to recover message: Invalid character in binary string"
    except Exception as e:
        return f"Failed to recover message: {str(e)}"

def try_decode_with_key_placeholder(self, text_content, key_placeholder):
    detected, binary_data = self.detect(text_content)
    if not detected:
        print("[ - ] No hidden information reliably decoded due to previous errors.",
file=sys.stdout)
        return

```

```

    print(f"Extracted bits for decoding: {binary_data}", file=sys.stdout)
    possible_message = self.binary_to_message(binary_data)
    print(f"Using key (placeholder) '{key_placeholder}': {possible_message}",
file=sys.stdout)

def main():
    if len(sys.argv) < 4:
        print(f"Usage: python {sys.argv[0]} <grammar_file> <text_file> <key>",
file=sys.stderr)
        sys.exit(1)

    grammar_file = sys.argv[1]
    text_file = sys.argv[2]
    key = sys.argv[3]

    try:
        with open(text_file, 'r') as f:
            text_content = f.read().strip()
    except FileNotFoundError:
        print(f"Error: Text file '{text_file}' not found.", file=sys.stderr)
        sys.exit(1)

    if not text_content:
        print(f"Error: Text file '{text_file}' is empty.", file=sys.stderr)
        sys.exit(1)

    detector = StegDetector(grammar_file)
    detector.try_decode_with_key_placeholder(text_content, key)

if __name__ == "__main__":

```

main()

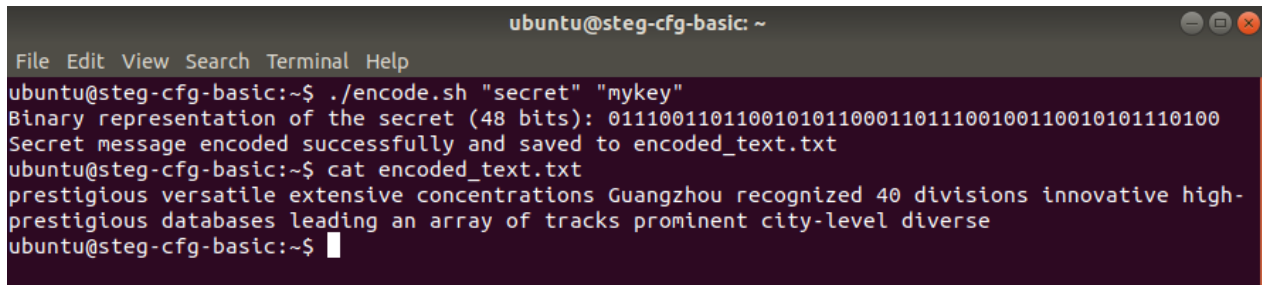
### 1.2.2 Mã hóa thông điệp bí mật

- Chạy script mã hóa:

```
./encode.sh "secret" "mykey"
```

- Kiểm tra kết quả:

```
cat encoded_text.txt
```



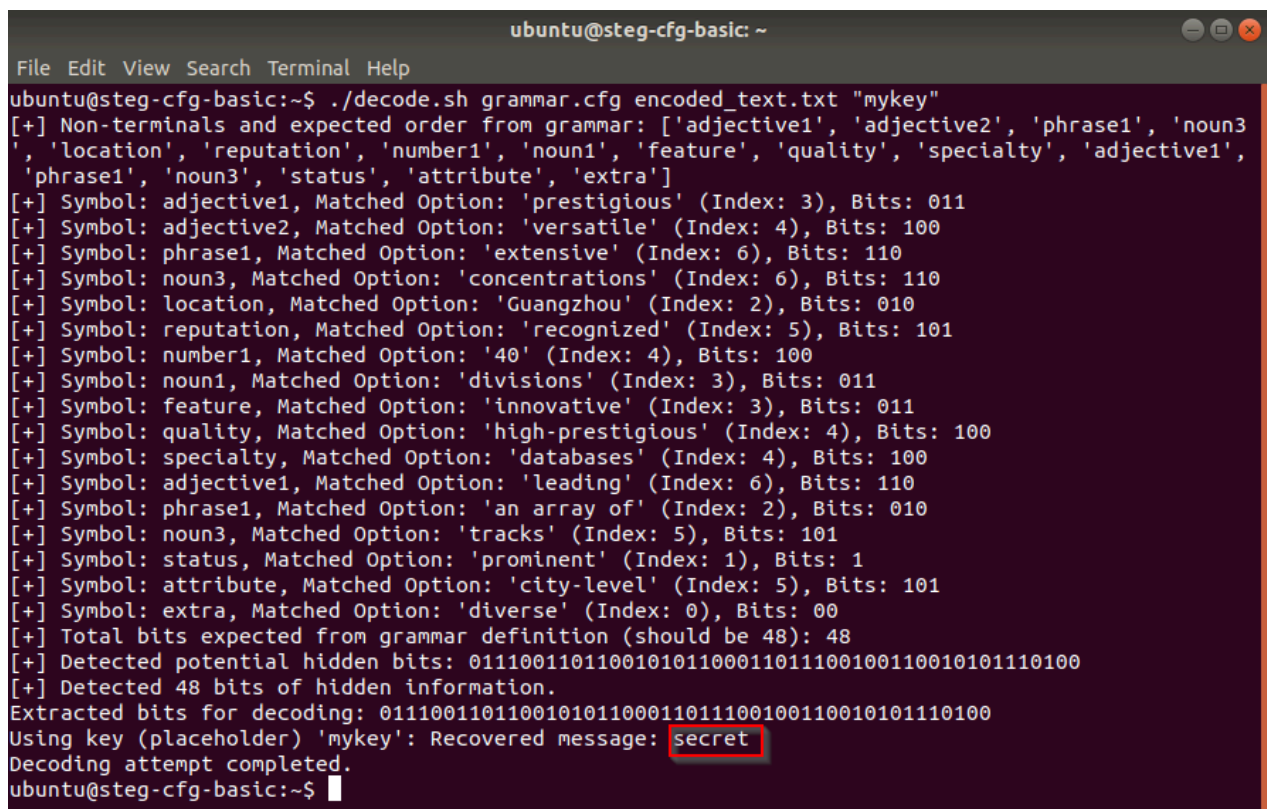
```
ubuntu@steg-cfg-basic: ~  
File Edit View Search Terminal Help  
ubuntu@steg-cfg-basic:~$ ./encode.sh "secret" "mykey"  
Binary representation of the secret (48 bits): 011100110110010101100011011100100110010101110100  
Secret message encoded successfully and saved to encoded_text.txt  
ubuntu@steg-cfg-basic:~$ cat encoded_text.txt  
prestigious versatile extensive concentrations Guangzhou recognized 40 divisions innovative high-  
prestigious databases leading an array of tracks prominent city-level diverse  
ubuntu@steg-cfg-basic:~$
```

Đầu ra là văn bản tự nhiên đã được mã hóa chứa thông điệp bí mật

### 1.2.3 Phát hiện và giải mã thông tin ẩn

- Chạy giải mã

```
./decode.sh grammar.cfg encoded_text.txt "mykey"
```



```
ubuntu@steg-cfg-basic: ~  
File Edit View Search Terminal Help  
ubuntu@steg-cfg-basic:~$ ./decode.sh grammar.cfg encoded_text.txt "mykey"  
[+] Non-terminals and expected order from grammar: ['adjective1', 'adjective2', 'phrase1', 'noun3',  
'location', 'reputation', 'number1', 'noun1', 'feature', 'quality', 'specialty', 'adjective1',  
'phrase1', 'noun3', 'status', 'attribute', 'extra']  
[+] Symbol: adjective1, Matched Option: 'prestigious' (Index: 3), Bits: 011  
[+] Symbol: adjective2, Matched Option: 'versatile' (Index: 4), Bits: 100  
[+] Symbol: phrase1, Matched Option: 'extensive' (Index: 6), Bits: 110  
[+] Symbol: noun3, Matched Option: 'concentrations' (Index: 6), Bits: 110  
[+] Symbol: location, Matched Option: 'Guangzhou' (Index: 2), Bits: 010  
[+] Symbol: reputation, Matched Option: 'recognized' (Index: 5), Bits: 101  
[+] Symbol: number1, Matched Option: '40' (Index: 4), Bits: 100  
[+] Symbol: noun1, Matched Option: 'divisions' (Index: 3), Bits: 011  
[+] Symbol: feature, Matched Option: 'innovative' (Index: 3), Bits: 011  
[+] Symbol: quality, Matched Option: 'high-prestigious' (Index: 4), Bits: 100  
[+] Symbol: specialty, Matched Option: 'databases' (Index: 4), Bits: 100  
[+] Symbol: adjective1, Matched Option: 'leading' (Index: 6), Bits: 110  
[+] Symbol: phrase1, Matched Option: 'an array of' (Index: 2), Bits: 010  
[+] Symbol: noun3, Matched Option: 'tracks' (Index: 5), Bits: 101  
[+] Symbol: status, Matched Option: 'prominent' (Index: 1), Bits: 1  
[+] Symbol: attribute, Matched Option: 'city-level' (Index: 5), Bits: 101  
[+] Symbol: extra, Matched Option: 'diverse' (Index: 0), Bits: 00  
[+] Total bits expected from grammar definition (should be 48): 48  
[+] Detected potential hidden bits: 011100110110010101100011011100100110010101110100  
[+] Detected 48 bits of hidden information.  
Extracted bits for decoding: 011100110110010101100011011100100110010101110100  
Using key (placeholder) 'mykey': Recovered message: secret  
Decoding attempt completed.  
ubuntu@steg-cfg-basic:~$
```

=> Giải mã thành công thông điệp secret

### 1.2.4 Tạo và sử dụng ngữ pháp tùy chỉnh

- Sinh viên tự tạo file ngữ pháp để thực hiện kỹ thuật giấu và tách tin

cp grammar.cfg my\_grammar.cfg

Chỉnh sửa file ngữ pháp mới với nội dung sau:

nano my\_grammar.cfg

Start → My trip to destination1 was adjective1, I visited place1.

destination1 → Hanoi || Paris

adjective1 → unforgettable || exciting

place1 → many attractions || famous landmarks

- Lưu mã hóa với ngữ pháp mới**

```
python3 generator.py my_grammar.cfg "secret" "mykey" > my_encoded.txt
```

```
cat my_encoded.txt
```

- Phát hiện thông tin ẩn:**

```
./decode.sh my_grammar.cfg my_encoded.txt "mykey"
```

### Kiểm tra kết quả bài lab:

```
student@LabtainerVMware:~/labtainer/trunk/scripts/labtainer-student$ checkwork steg-cfg-basic
Results stored in directory: /home/student/labtainer_xfer/steg-cfg-basic
Successfully copied 51.2kB to steg-cfg-basic-igrader:/home/instructor/B21DCAT090.steg-cfg-basic.lab
Successfully copied 2.05kB to /home/student/labtainer_xfer/steg-cfg-basic
Labname steg-cfg-basic

Student          | ran_generator | created_encoded | ran_detector | created_grammar | used_new_grammar |
=====|=====|=====|=====|=====|=====|
B21DCAT090       | Y             | Y              | Y           | Y              | Y              |
What is automatically assessed for this lab:
```

Trên terminal đầu tiên sử dụng câu lệnh sau để kết thúc bài lab: stoplab

Khi bài lab kết thúc, một tệp zip lưu kết quả được tạo và lưu vào một vị trí được hiển thị bên dưới stoplab.