BIT 2024/2025

Úkol 1 - Steganografie

Vytvořte v programovacím jazyce C/C++, Java nebo Python program, který bude provádět steganografii ve formě ukrytí libovolného souboru (např. obrázku, textového souboru či zvukové MP3 nahrávky) do BMP obrázku. Výsledný soubor bude korektní soubor BMP, který **nebude nikterak poškozen** a bude jej možné otevřít libovolným prohlížečem obrázků.

Archiv se zadáním obsahuje následující:

- **`weber.bmp`**
- **`out/`**
- **`validation/`**
- **`decoded/`**

V zadání je k dispozici obrázek **`weber.bmp`**, který slouží jako médium pro ukrytí. Do něj postupně pomocí metody LSB (Least Significant Bit) budete ukrývat soubory.

Ve složce **`validation/`** jsou validační soubory, na kterých je možné/nutné vaši práci odladit. Výsledné korektně zakódované obrázky uloží program do složky **`out/`**; každý soubor v této složce bude korektní BMP, který půjde otevřít libovolným prohlížečem obrázků. Program ve druhé fázi pak projde složku **`out/`**, dekóduje obrázky a výsledné extrahované soubory uloží do složky **`decoded/`**. Výsledkem správného běhu programu bude to, že soubory ve složce **`validation/`** a ve složce **`decoded/`** budou stejné a nepoškozené.

Vyhodnocení se bude provádět automaticky. Náhodná sada testovacích souborů se vloží do složky **`validation/`** a vyhodnocení se provede na nich. Cílem je ověřit, že program není “šitý na míru” validačním souborům, které máte k dispozici. Automatický validátor poté spustí program a porovná obsah složek **`out/`** a **`decoded/`**.

Pokud program vyhodnotí, že se metodou LSB celý soubor nevejde do obrázku **`weber.bmp`**, steganografii neprovádějte a soubor přeskočte. Nedělejte tedy kódování s využitím 2 nebo 4 posledních bitů, došlo by pak k viditelným změnám v obrázku.

Z podstaty algoritmu steganografie musí mít zakódované obrázky ve složce **`out/`** stejnou velikost jako **`weber.bmp`**; toto je jeden z testů, který se rovněž bude provádět.

Pro zpětné dekódování budete potřebovat znát jméno souboru, příponu a také počet bajtů, které jste ukrývali. Tato metadata zakódujte současně s daty (např. do prvních 4 bytů délku souboru a do dalších 4 koncovku).

Příklad výstupu

Uvažujme jediný soubor např. **`validation/validation1.mp3`**

- zakóduje se soubor **`validation1.mp3`** do obrázku **`weber.bmp`**
  → vytvoří se **`out/validation1___weber.bmp`**
- při zpětném dekódování nejprve přečtěte metadata. Z načtených bajtů poté vytvořte soubor **`decoded/validation1.mp3`**

Porovnáte-li tyto dva soubory, musí být naprosto stejné.
např. pomocí "**`diff validation/validation1.mp3 decoded/validation1.mp3`**"
Toto je taky jeden z testů, který se provádí.

Netvořte žádnou dokumentaci, pouze nejdůležitější body vaši práce stručně popište do souboru **`readme.txt`**

Jednotkové testy

Součástí archivu se zadáním je i složka **`test/`** a v ní skript s jednotkovými testy v Pythonu. Práci odlaďte tak, aby všechny jednotkové testy prošly. Pokud implementujete v jiném jazyku než v Pythonu, doporučujeme si sadu podobných jednotkových testů také napsat. Testy kontrolují následující výstupy:

1. **`test_steganography_img_sizes`**
   - porovná se, zda všechny obrázky po provedení steganografie ve složce **`out/`** mají stejnou velikost jako **`weber.bmp`**
2. **`test_all_imgs_after_steganography`**
   - porovná se, zda všechny obrázky ve složce **`out/`** jsou odlišné od **`weber.bmp`** a zároveň jdou korektně načíst pomocí **opencv** knihovny
3. **`test_decoded_results`**
   - porovná se, zda po provedení steganografie všechny soubory ve složce **`decoded/`** a odpovídající ve složce **`validation/`** jsou stejné

Pro detaily si prosím přečtěte soubor **`README.md`**, který je součástí archivu. Váš program společně se všemi adresáři z archivu se zadáním zabalte do ZIP souboru a pojmenujte ho podle svého osobního čísla.

Za takto vypracovanou práci získáte 6 bodů. Další 4 body lze získat bonusovou částí (viz níže).

Bonusová část [4 body]

Zajistěte, aby soubory před vložením do obrázku byly zašifrované pomocí jednoduché Vernamovy (“one-time pad”) šifry. Negenerujte náhodný šifrovací klíč, ale použijte klíč, který je odvoditelný z nějakého hesla (např. **`bit2025`**). K tomu můžete využít např. hashovací funkci (např. **`SHA-512`**). Ta transformuje jakýkoliv vstup na číslo o konstantní délce (v tomto případě 512 bitů). Pro soubory větší než 512 bitů je nutné klíč rozšířit až na velikost souboru. Toho lze docílit pomocí opakovaného použití hashovací funkce nebo jednoduše cyklicky přidávat konstantní 512-bitové bloky a poslední blok oříznout dle potřeby.

Příklad: má-li soubor, který šifruji velikost 100 bajtů, je nutné vytvořit 800 bitový klíč. Použijeme 2x hashovací funkci SHA-512 se vstupem (**`bit2025`**), výstupy spojíme dohromady a výsledné 1024 bitové číslo ořízneme na 800 bitů.

Jakmile vygenerujete klíč, proveďte šifrovaní pomocí operace XOR nad jednotlivými bajty souboru a klíče. Klíč není třeba ukládat do souboru, bude odvoditelný na základě hesla. Poté proveďte steganografii, stejným způsobem jako v první (nebonusové) části. Výsledkem bude dešifrovaný čitelný původní soubor (tzn. nezapoměnte ho opět dešifrovat pomocí operace XOR).

Zamyslete se nad vhodností tohoto řešení a srovnejte ho s přístupem, kdy generuji zcela náhodný klíč. Jaké jsou výhody a nevýhody tohoto řešení? Popiště v několika větách a zahrňte do souboru **`readme.txt`**