APPENDICES

Appendix 1. Biography

Gabriel Nathanael Purba was born in Jakarta on December 8, 2004. The author was born to Mr. Hotben Purba and Mrs. Nurhaida Silalahi. The author is an Indonesian citizen and a Protestant Christian. The author currently resides at Jalan Bandung Blok J5 Number 11 RT 10 RW 7, Tegal Alur Village, Kalideres District, West Jakarta. The author completed primary education at SD Yadika 1 Tegal Alur. The author then continued to SMP Yadika 10 Kosambi for secondary education. Subsequently, the author completed senior high school education at SMA Negeri 56 Jakarta and continued to the Bachelor of Computer Science Program at Universitas Pendidikan Ganesha. At the time of writing this thesis, the author is still registered as a student of the Bachelor of Computer Science Program at Universitas Pendidikan Ganesha

Appendix 2. AES-CBC Encryption and Decryption Implementation

The following pseudocode presents the AES-CBC encryption and decryption functions used in the system. A random 16-byte Initialization Vector (IV) is generated for each encryption operation to ensure ciphertext uniqueness. The IV is prepended to the ciphertext before Base64 encoding.

Pseudocode: encrypt_aes_cbc

```
FUNCTION encrypt_aes_cbc(plaintext, key):
    iv = generate_random_bytes(16)
    cipher = AES_CBC_init(key, iv)
    padded = apply_PKCS7_padding(plaintext)
    ciphertext = cipher.encrypt(padded)
    return base64_encode(iv + ciphertext)
END FUNCTION
```

Pseudocode: decrypt_aes_cbc

```
FUNCTION decrypt_aes_cbc(ciphertext_b64, key):
    raw = base64_decode(ciphertext_b64)
    iv = raw[0:16]
    ct = raw[16:]
    cipher = AES_CBC_init(key, iv)
    plaintext = remove_PKCS7_padding(cipher.decrypt(ct))
    return plaintext
END FUNCTION
```

Appendix 3. LSB Embedding and Extraction Implementation

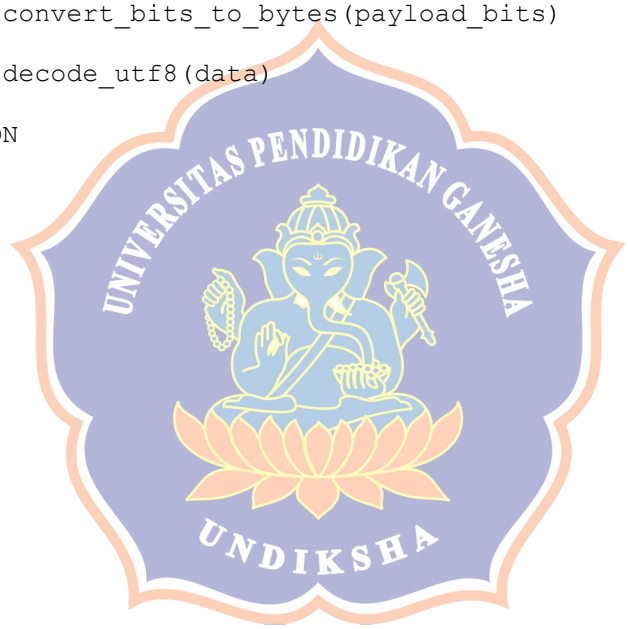
The following pseudocode presents the LSB steganography functions. The embedding function inserts the ciphertext bit-by-bit into the least significant bit of each RGB channel of the cover image pixels. A 4-byte length header is prepended to the payload to support accurate extraction. The extraction function reverses this process to recover the original ciphertext.

Pseudocode: lsb_embed

```
FUNCTION lsb_embed(ciphertext, cover_path, stego_path):
    data ← encode_utf8(ciphertext)
    length ← byte_length(data)
    payload ← length_as_4_bytes(length) + data
    bits ← convert_each_byte_to_8_bit_string(payload)
    image ← open_image_as_RGB(cover_path)
    IF length(bits) > width * height * 3:
        RAISE error 'Ciphertext too large for cover image'
    bit_index ← 0
    FOR each pixel (x, y) in image:
        r, g, b ← get_pixel(x, y)
        r ← replace_LSB(r, bits[bit_index]); bit_index += 1
        g ← replace_LSB(g, bits[bit_index]); bit_index += 1
        b ← replace_LSB(b, bits[bit_index]); bit_index += 1
        set_pixel(x, y, r, g, b)
        IF bit_index >= length(bits): BREAK
    save_image(image, stego_path)
END FUNCTION
```

Pseudocode: lsb_extract

```
FUNCTION lsb_extract(stego_path):
    image ← open_image_as_RGB(stego_path)
    bits ← empty string
    FOR each pixel (x, y) in image:
        r, g, b ← get_pixel(x, y)
        bits ← bits + LSB(r) + LSB(g) + LSB(b)
    length ← convert_bits_to_integer(bits[0:32])
    payload_bits ← bits[32 : 32 + (length * 8)]
    data ← convert_bits_to_bytes(payload_bits)
    return decode_utf8(data)
END FUNCTION
```



Appendix 4. Distribution Transforming Encoder (DTE) Implementation

The Distribution Transforming Encoder (DTE) module implements the core Honey Encryption mechanism. Two separate JSON-based mappings are maintained: one for original (real) documents and one for decoy documents. Each document is converted to Base64, assigned a unique 10-character alphanumeric seed, and stored directly within the database mapping.

Pseudocode: encode_document

```
FUNCTION encode_document(pdf_path):
    b64_doc = pdf_to_base64(pdf_path)
    doc_hash = sha256(b64_doc)

    IF doc_hash EXISTS in hash_map:
        return hash_map[doc_hash]

    seed = generate_unique_alphanumeric_seed(length=10)

    real_map[seed] = { "b64_doc": b64_doc }
    hash_map[doc_hash] = seed

    save(real_map)
    save(hash_map)
    return seed
END FUNCTION
```

Pseudocode: decode_real_seed

```
FUNCTION decode_real_seed(seed, output_pdf_path):
    IF seed NOT IN real_map:
```

```
        return FALSE

    entry = real_map[seed]

    b64_doc = entry.b64_doc
    base64_to_pdf(b64_doc, output_pdf_path)
    return TRUE
END FUNCTION
```

Pseudocode: get_random_decoy

```
FUNCTION get_random_decoy(output_pdf_path):
    IF decoy_map IS EMPTY:
        RAISE error 'No decoy documents available'

    seed = random_choice(decoy_map.keys())
    entry = decoy_map[seed]

    b64_doc = entry.b64_doc
    base64_to_pdf(b64_doc, output_pdf_path)
END FUNCTION
```

Appendix 5. Encryption Flow Implementation

The following pseudocode presents the encryption flow that orchestrates the full encryption and embedding process. The function receives a PDF document path, a cover image path, a user key, and an output stego path, then produces a stego image containing the encrypted seed.

Pseudocode: encrypt_flow

```
FUNCTION encrypt_flow(pdf_path, cover_image_path, key,
output_stego_path):
    seed ← encode_document(pdf_path)
    // encode_document maps the PDF to a unique seed via DTE
    ciphertext ← encrypt_aes_cbc(seed, key)
    // encrypt seed using user-provided key
    lsb_embed(ciphertext, cover_image_path, output_stego_path)
    // embed ciphertext into cover image using LSB method
    return output_stego_path
END FUNCTION
```

Appendix 6. Decryption Flow Implementation

The following pseudocode presents the decryption flow that orchestrates the full extraction and decryption process. If the key is incorrect or the decrypted seed does not match any real document, a randomly selected decoy document is returned instead. This is the core deception behavior of the Honey Encryption mechanism.

Pseudocode: decrypt_flow

```
FUNCTION decrypt_flow(stego_path, key, real_output_path,
decoy_output_path):
    ciphertext ← lsb_extract(stego_path)
    // extract embedded ciphertext from stego image
    TRY:
        seed ← decrypt_aes_cbc(ciphertext, key)
    EXCEPT decryption_error:
        seed ← NULL
    IF seed IS NOT NULL:
        success ← decode_real_seed(seed, real_output_path)
        IF success:
            return ('real', real_output_path)
    // fallback: seed invalid or not found → return decoy
    get_random_decoy(decoy_output_path)
    return ('decoy', decoy_output_path)
END FUNCTION
```

Appendix 7. Application Interface Implementation

The following pseudocode presents the application interface implemented using the Streamlit framework. The interface consists of two tabs: the Encrypt tab allows users to upload a PDF document and PNG cover image along with a secret key to produce a downloadable stego image; the Decrypt tab allows users to upload a stego image and enter a key to retrieve the original or decoy document.

Pseudocode: Application Interface

```
INITIALIZE app with title 'Honey Encryption + Steganography'
CREATE tabs: [Encrypt, Decrypt]
── ENCRYPT TAB ──────────────────────────────────────────
INPUT: pdf_file, cover_image, secret_key
ON button_click('Encrypt'):
    IF any input is missing: display error and STOP
    save pdf_file → TMP_DIR/pdf_file.name
    save cover_image → TMP_DIR/cover_image.name
    stego_path ← STEGO_DIR/stego_{timestamp}_{cover_image.name}
    call encrypt_flow(pdf_path, cover_path, secret_key,
stego_path)
    IF stego_path EXISTS:
        display success message
        display download button for stego image
    ELSE:
```

```
        display error

── DECRYPT TAB ──────────────────────────────────────────

INPUT: stego_image, secret_key

ON button_click('Decrypt'):

    IF any input is missing: display error and STOP

    save stego_image → TMP_DIR/stego_image.name

    result_type, out_path ← decrypt_flow(stego_path, secret_key,
real_output_path, decoy_output_path)

    display result_type (REAL or DECOY)

    display download button for output PDF
```