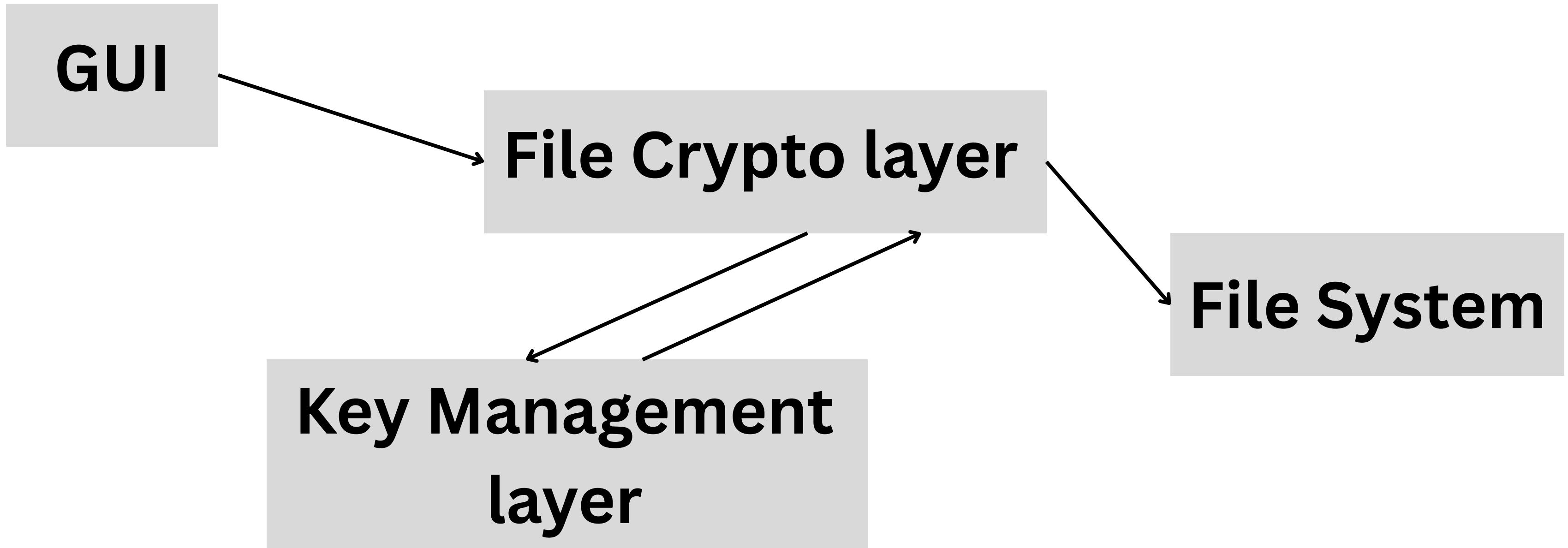


Secure Folder Encryption Tool

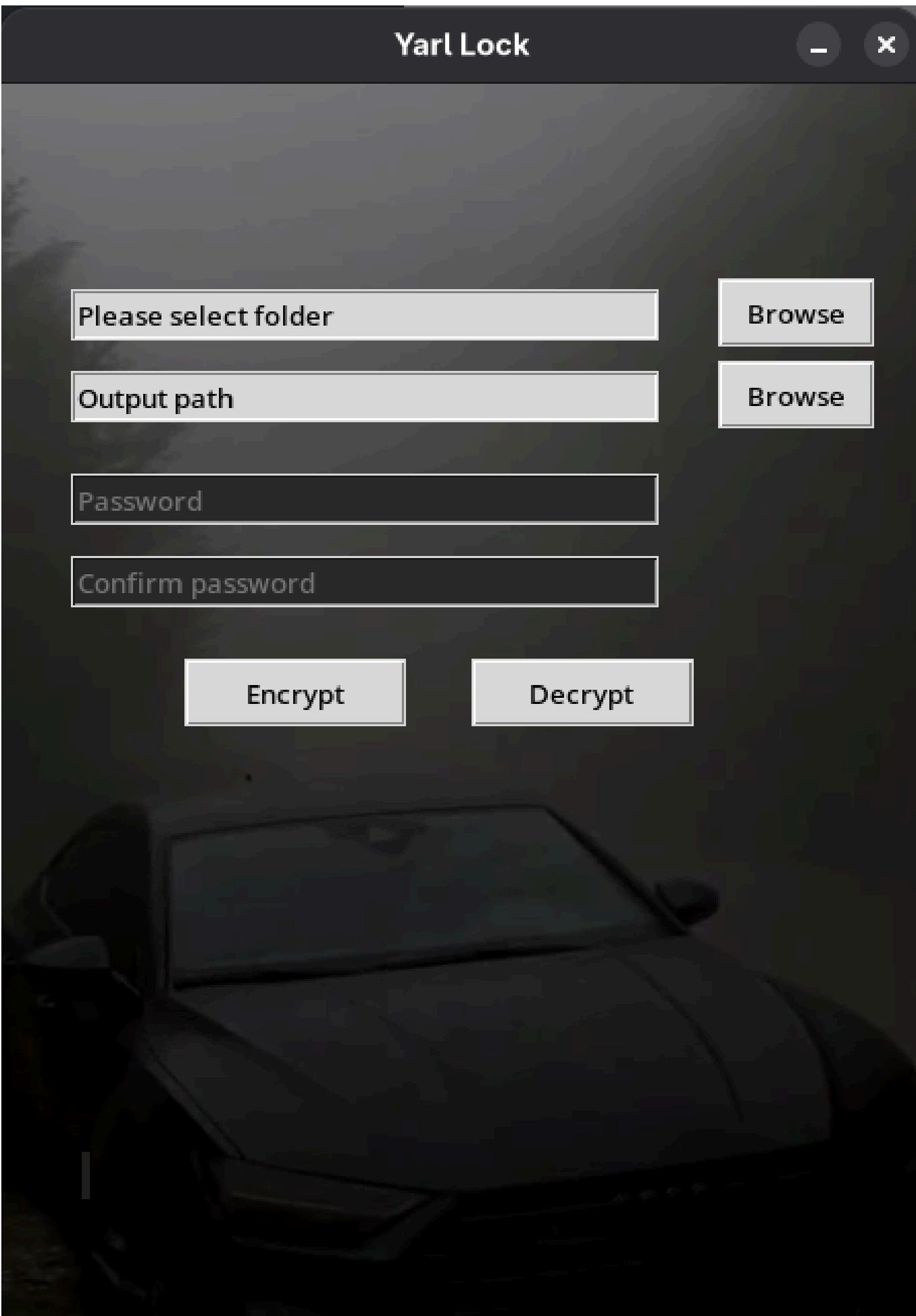
Yernar Askarov

Architecture

GUI collects paths + password
Crypto layer does AES-GCM + signatures
Key management stores RSA keys securely



GUI



The GUI checks:

- All fields are filled.
- Password and confirm password match.
- If anything is missing or mismatched, it shows an error and stops.

Walk folders

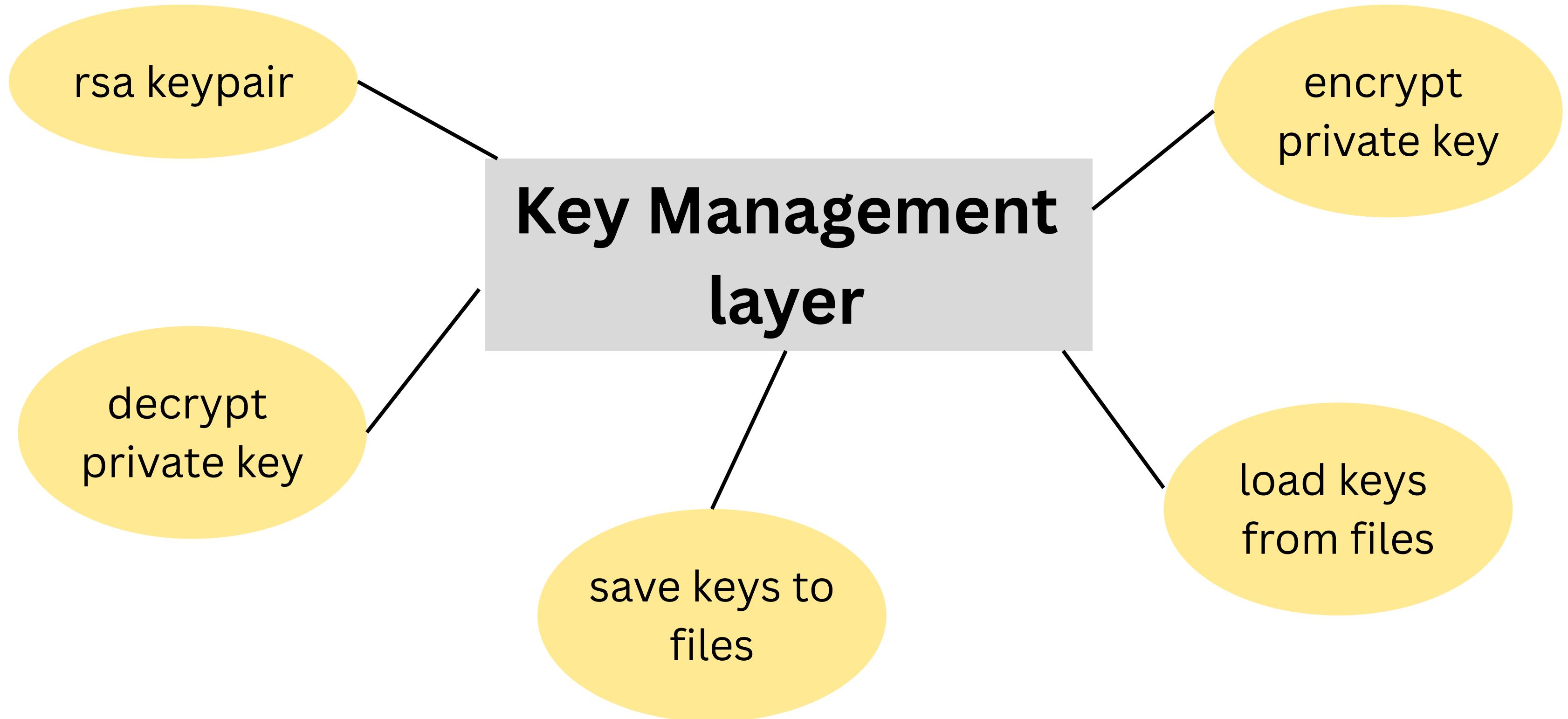
File Crypto layer

verify then
decrypt

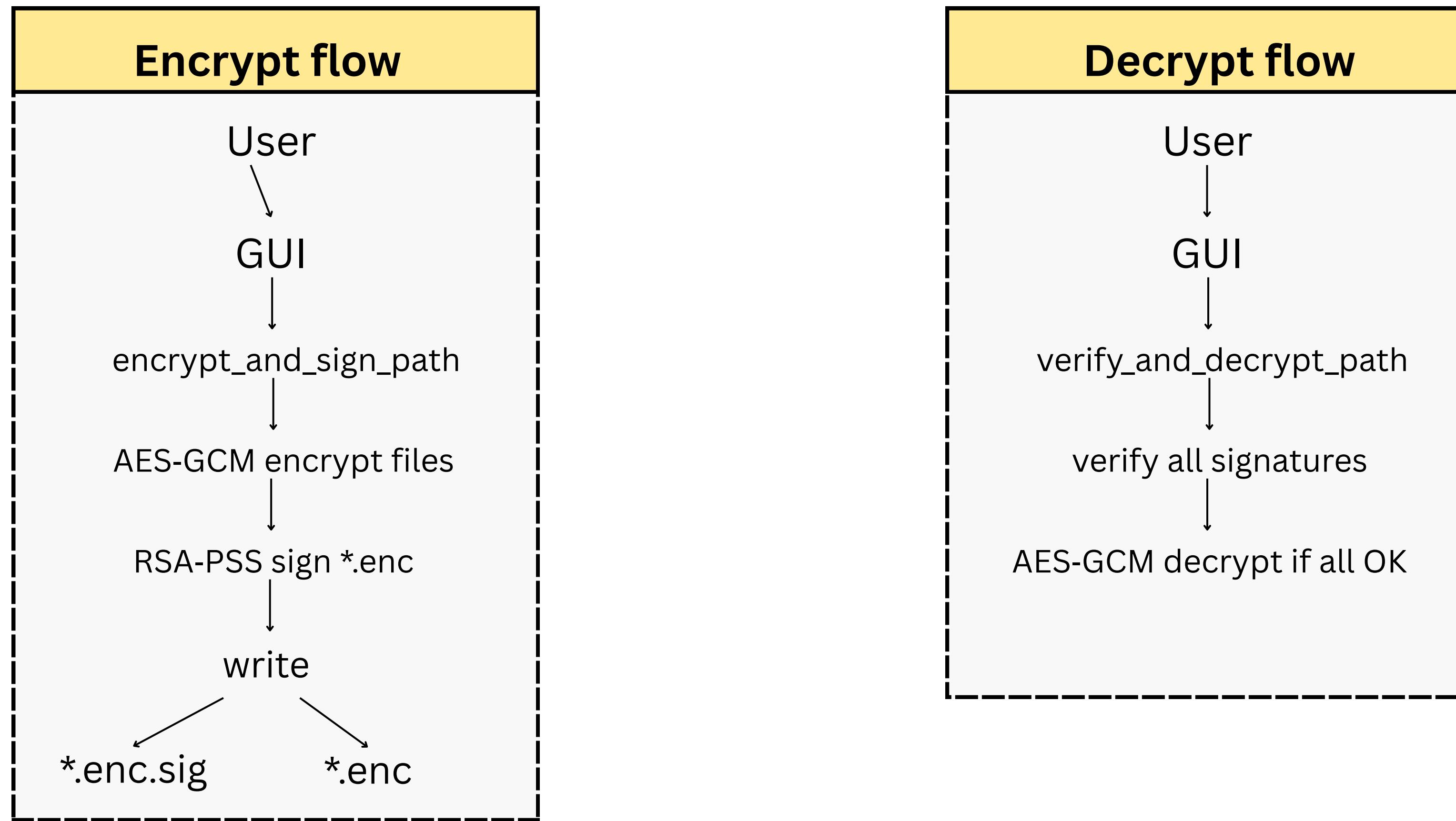
encrypt files

sign .enc files

```
105
110     def encrypt_and_sign_path(input_path: str, output_path: str, password: str, keys_dir: str | Path) -> None:
111         encrypt_path(input_path, output_path, password)
112         out = Path(output_path)
113         for p in out.rglob("*.enc"):
114             sign_encrypted_path(p, password, keys_dir)
115
116
117     def verify_and_decrypt_path(input_path: str, output_path: str, password: str, keys_dir: str | Path) -> None:
118         inp = Path(input_path)
119         # verify all .enc files first
120         for p in inp.rglob("*.enc"):
121             if not verify_encrypted_path(p, keys_dir):
122                 raise ValueError(f"Signature verification failed for {p}")
123         decrypt_path(input_path, output_path, password)
124
```



Data Flow Diagram



Algorithms Overview

Symmetric
Password
Asymmetric
Signatures
Hash
RNG

AES-256-GCM
KDF: PBKDF2
RSA-2048
RSA-PSS (SHA-256)
SHA-256
os.urandom

AES-GCM + PBKDF2

Password → PBKDF2(salt) → AES key

For each file:

Generate random salt + nonce

Store: salt || nonce || ciphertext_with_tag

GCM (integrity + confidentiality)

PBKDF2 (slow down brute-force)

RSA and Key Management

One RSA-2048 keypair:

public_key.pem – public verification key

private_key.enc – RSA private key, encrypted with AES-GCM under password-derived key.

Private key never stored in plaintext on disk

Digital Signatures on Encrypted Files

- For each file.enc:
 - Decrypt private key in memory.
 - Sign ciphertext with RSA-PSS → file.enc.sig.
- On decrypt:
 - Verify file.enc with public_key.pem and .sig.
 - Only then decrypt.

```
def sign_encrypted_path(enc_path: str | Path, password: str, keys_dir: str | Path) -> Path:  
    enc_path = Path(enc_path)  
    keys_dir = Path(keys_dir)  
  
    #load keys  
    public_pem, encrypted_private = load_keys_from_files(keys_dir)  
    if public_pem is None or encrypted_private is None:  
        raise ValueError("Keys not found in keys_dir")  
  
    #decrypt private key  
    private_pem = decrypt_private_key(encrypted_private, password)  
  
    #sign the file  
    data = enc_path.read_bytes()  
    signature = sign_bytes(private_pem, data)  
  
    #store next to encrypted file  
    sig_path = enc_path.with_suffix(enc_path.suffix + ".sig")  
    sig_path.write_bytes(signature)  
    return sig_path
```

```
def verify_encrypted_path(enc_path: str | Path, keys_dir: str | Path) -> bool:  
    enc_path = Path(enc_path)  
    keys_dir = Path(keys_dir)  
  
    public_pem, _ = load_keys_from_files(keys_dir)  
    if public_pem is None:  
        raise ValueError("Public key not found in keys_dir")  
  
    sig_path = enc_path.with_suffix(enc_path.suffix + ".sig")  
    if not sig_path.exists():  
        return False  
  
    data = enc_path.read_bytes()  
    signature = sig_path.read_bytes()  
  
    return verify_signature(public_pem, data, signature)
```

Security Analysis

Attackers can

- Read/copy .enc and .enc.sig.
- Try to modify/replace encrypted files.
- Run offline password guessing

We assume

- Strong password.
- No malware/keylogger.
- Correct cryptography library.

Mitigations

Signature verification before decryption

PBKDF2 to slow guesses

Encrypted private key (AES-GCM)

Explain limitations honestly

thanks