Secure Authentication & Cryptography System

Team members

Temirlan Maksat - Security Engineer

Suleimenova Dilnaz - Backend Developer

Amangeldiyev Aidos - Frontend Developer

Problem Statement & Motivation

Slide content

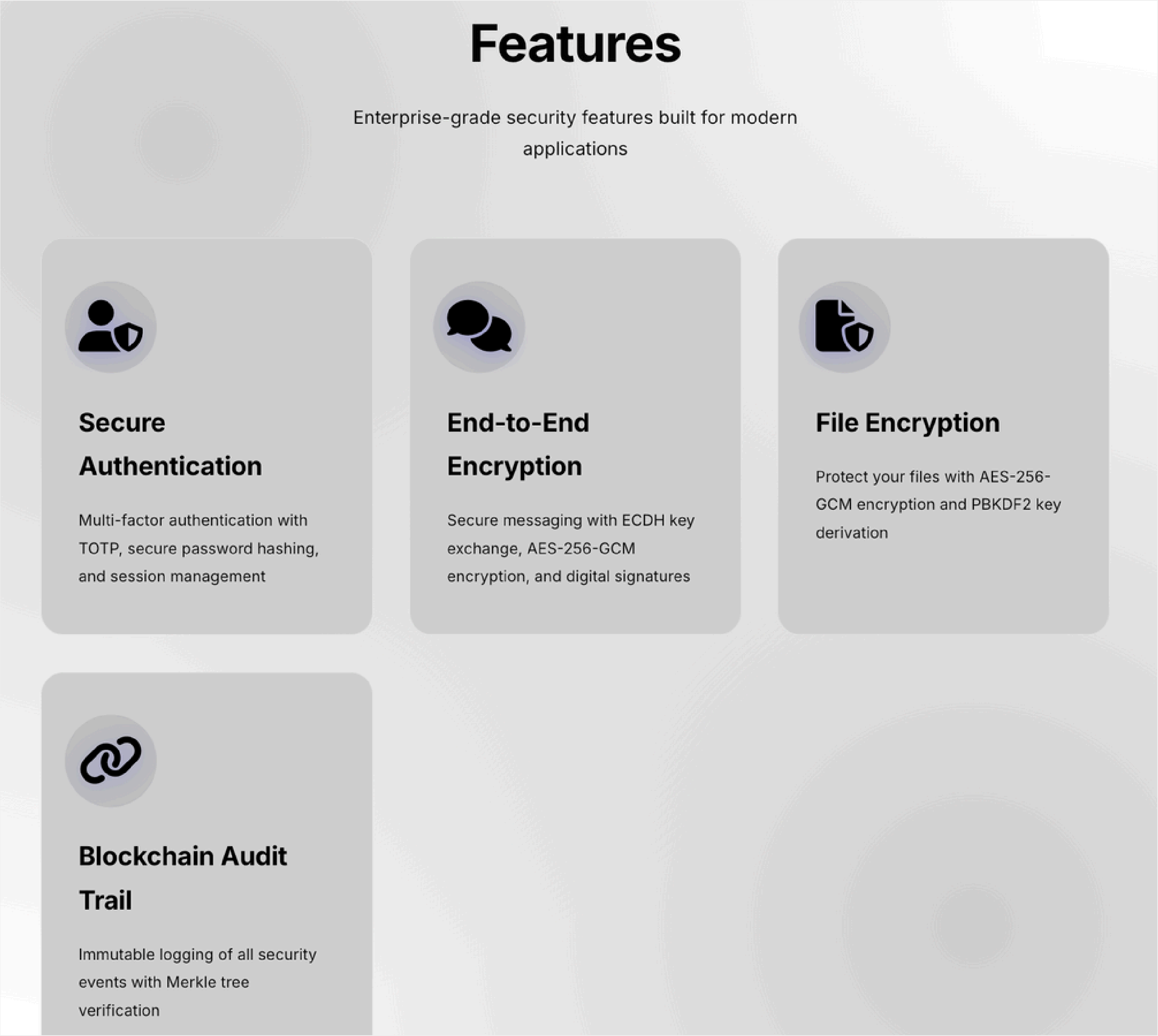
- Password leaks & weak authentication
- Data leakage in file storage
- Lack of tamper-proof audit logs

System Overview

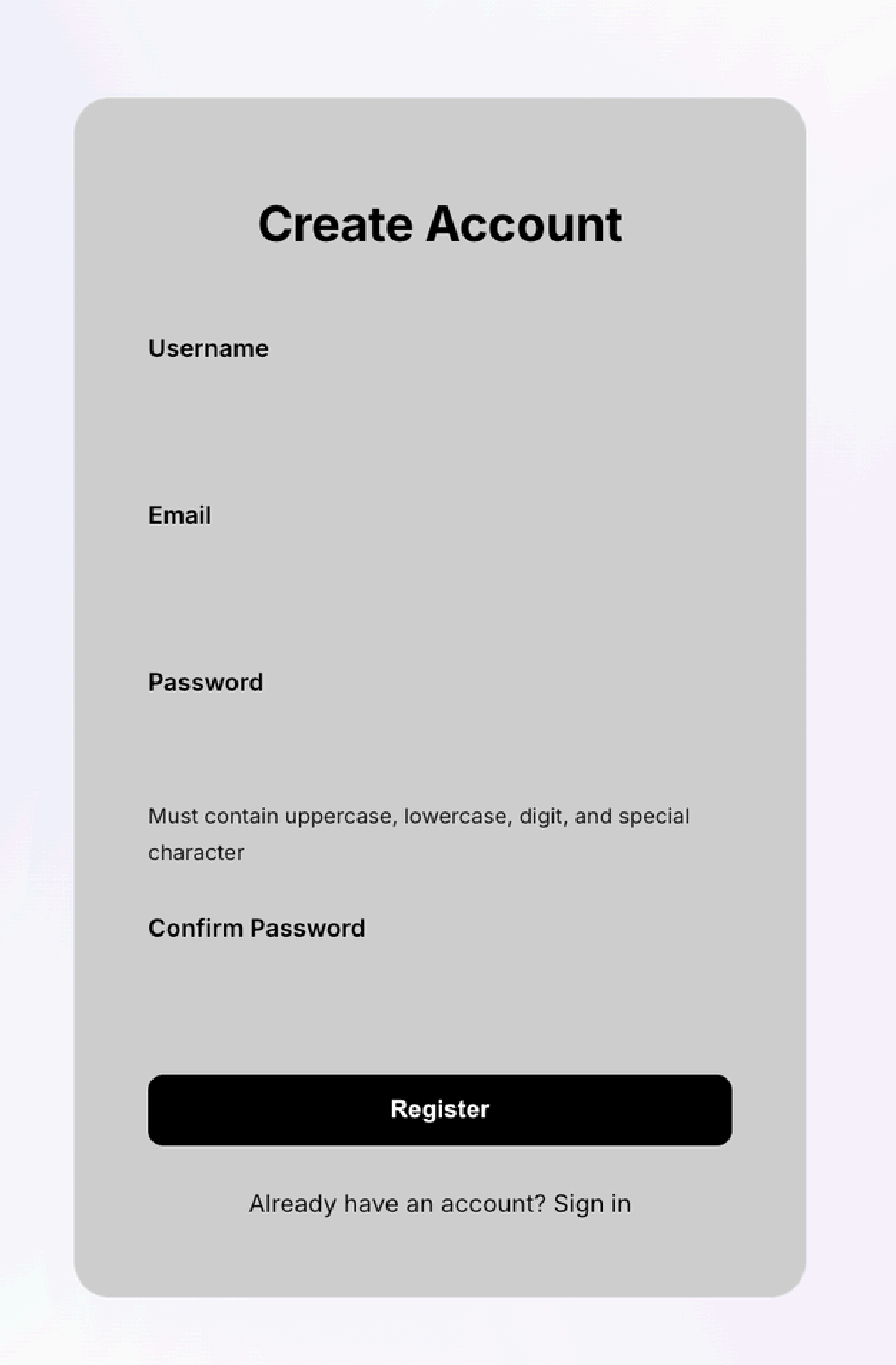
consists of several key components:

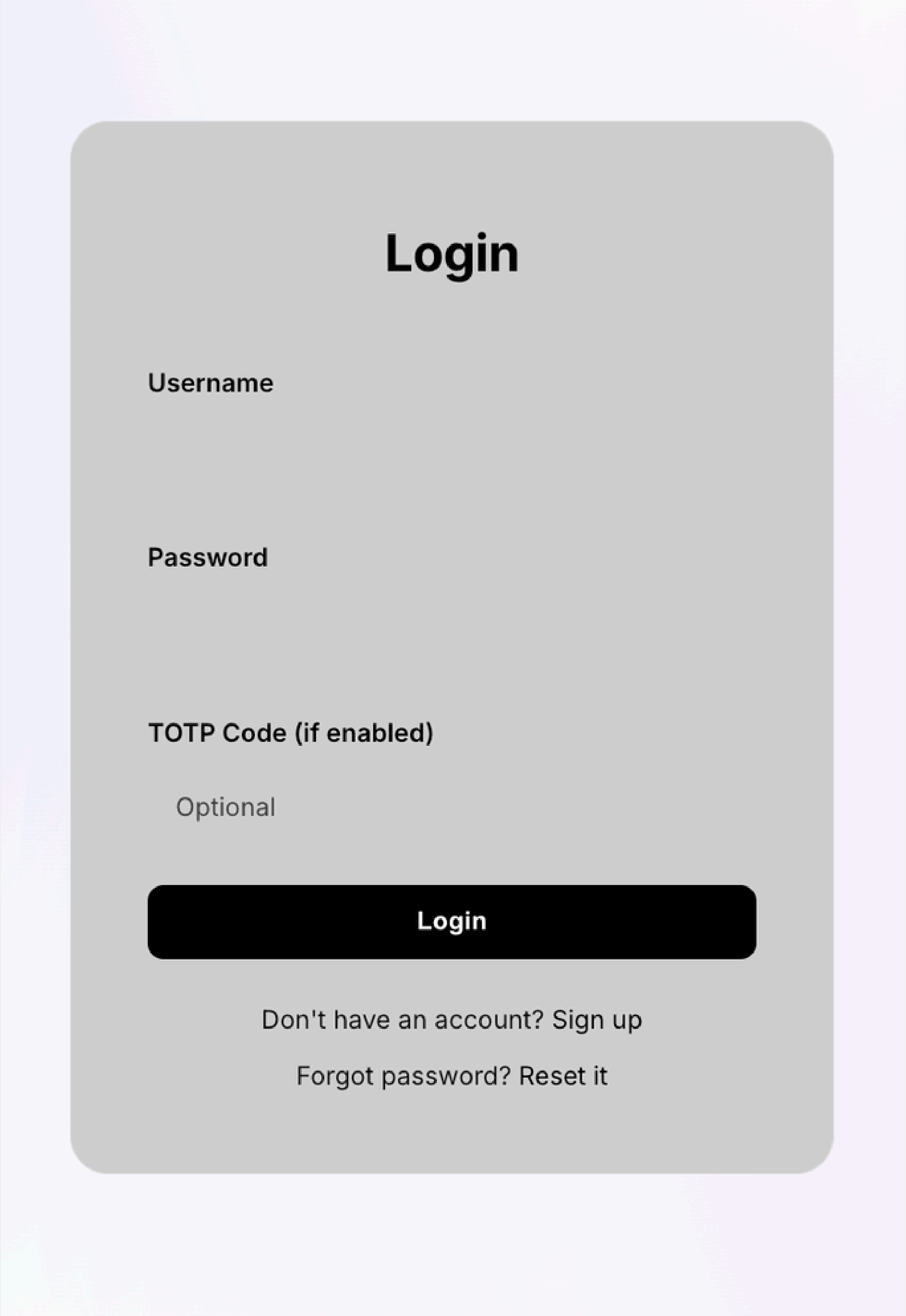
- an authentication module,
- a cryptography module,
- a secure messaging system,
- encrypted file storage,
- and a blockchain-based audit ledger.

Each module protects a specific aspect of the system, and together they create layered security, where breaking one layer is not enough to compromise the whole system.



Authentication and Multi-Factor Security





Authentication and Multi-Factor Security

```python
def generate_secret(self) -> str:
    return pyotp.random_base32()
```

```python
def generate_qr_code(self, username: str, secret: str, issuer: str = "CryptoVault") -> str:
    uri = self.get_provisioning_uri(username, secret, issuer)

    qr = qrcode.QRCode(
        version=1,
        error_correction=qrcode.constants.ERROR_CORRECT_L,
        box_size=10,
        border=4,
    )
    qr.add_data(uri)
    qr.make(fit=True)

    img = qr.make_image(fill_color="black", back_color="white")

    buffer = BytesIO()
    img.save(buffer, format='PNG')
    buffer.seek(0)

    img_base64 = base64.b64encode(buffer.getvalue()).decode()
    return f"data:image/png;base64,{img_base64}"
```

Authentication and Multi-Factor Security

```python
def setup_totp(self, user: User) -> Tuple[str, List[str], str]:
    secret = self.generate_secret()
    backup_codes = self.generate_backup_codes()
    user.totp_secret = secret
    user.backup_codes = json.dumps(backup_codes)
    db.session.commit()

    qr_code = self.generate_qr_code(user.username, secret)

    return secret, backup_codes, qr_code

def verify_totp(self, user: User, code: str) -> bool:
    if not user.totp_secret:
        return False

    totp = pyotp.TOTP(user.totp_secret)
    return totp.verify(code, valid_window=self.time_window)
```

TOTP Code (if enabled)

Optional

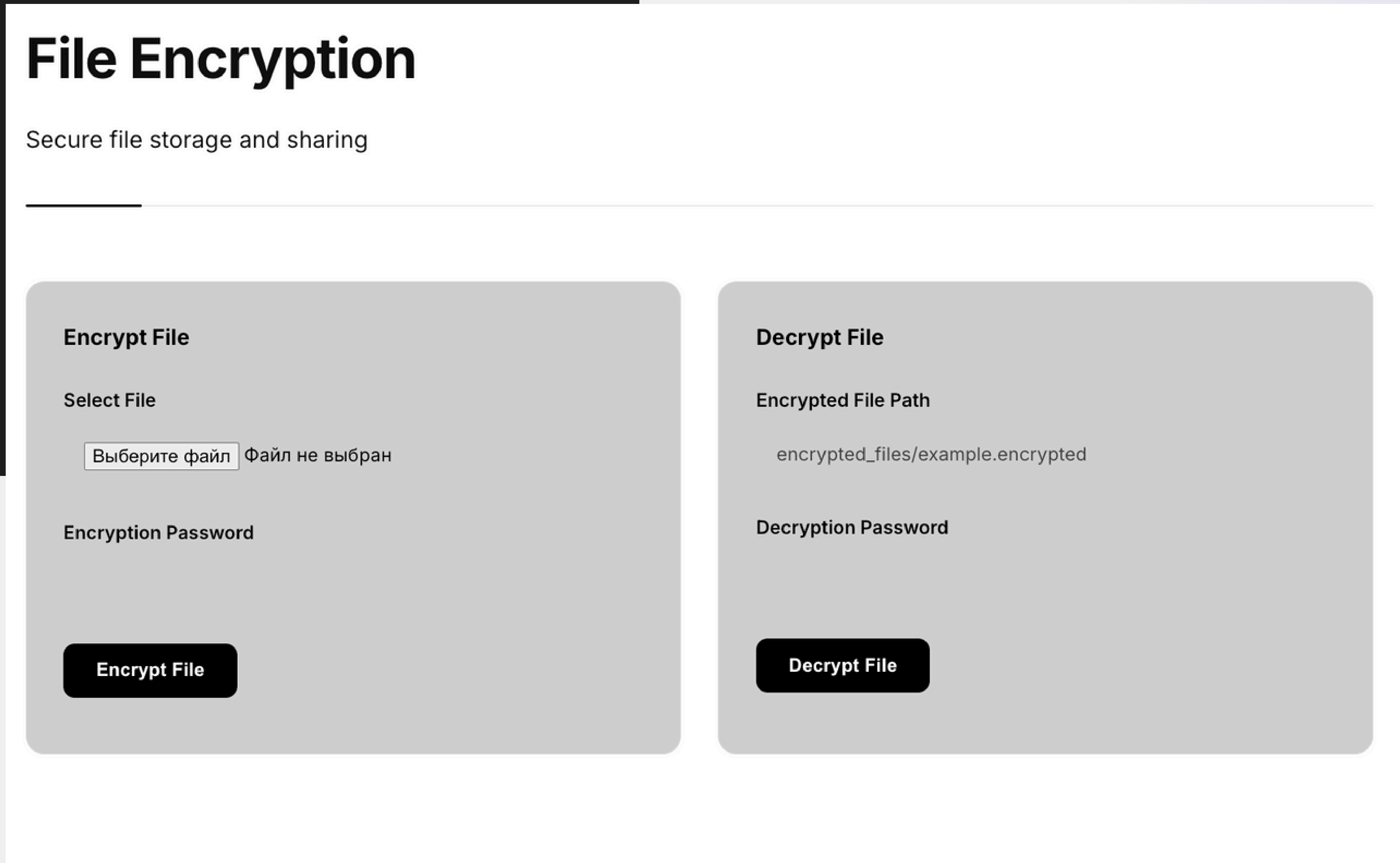
Secure File ENCRYPTION

```python
def derive_master_key(self, password: str, salt: bytes = None) -> Tuple[bytes, bytes]:
    if salt is None:
        salt = secrets.token_bytes(32)

    kdf = PBKDF2HMAC(
        algorithm=hashes.SHA256(),
        length=32,
        salt=salt,
        iterations=self.pbkdf2_iterations,
        backend=self.backend
    )

    master_key = kdf.derive(password.encode('utf-8'))
    return master_key, salt

def generate_file_encryption_key(self) -> bytes:
    return secrets.token_bytes(32)
```

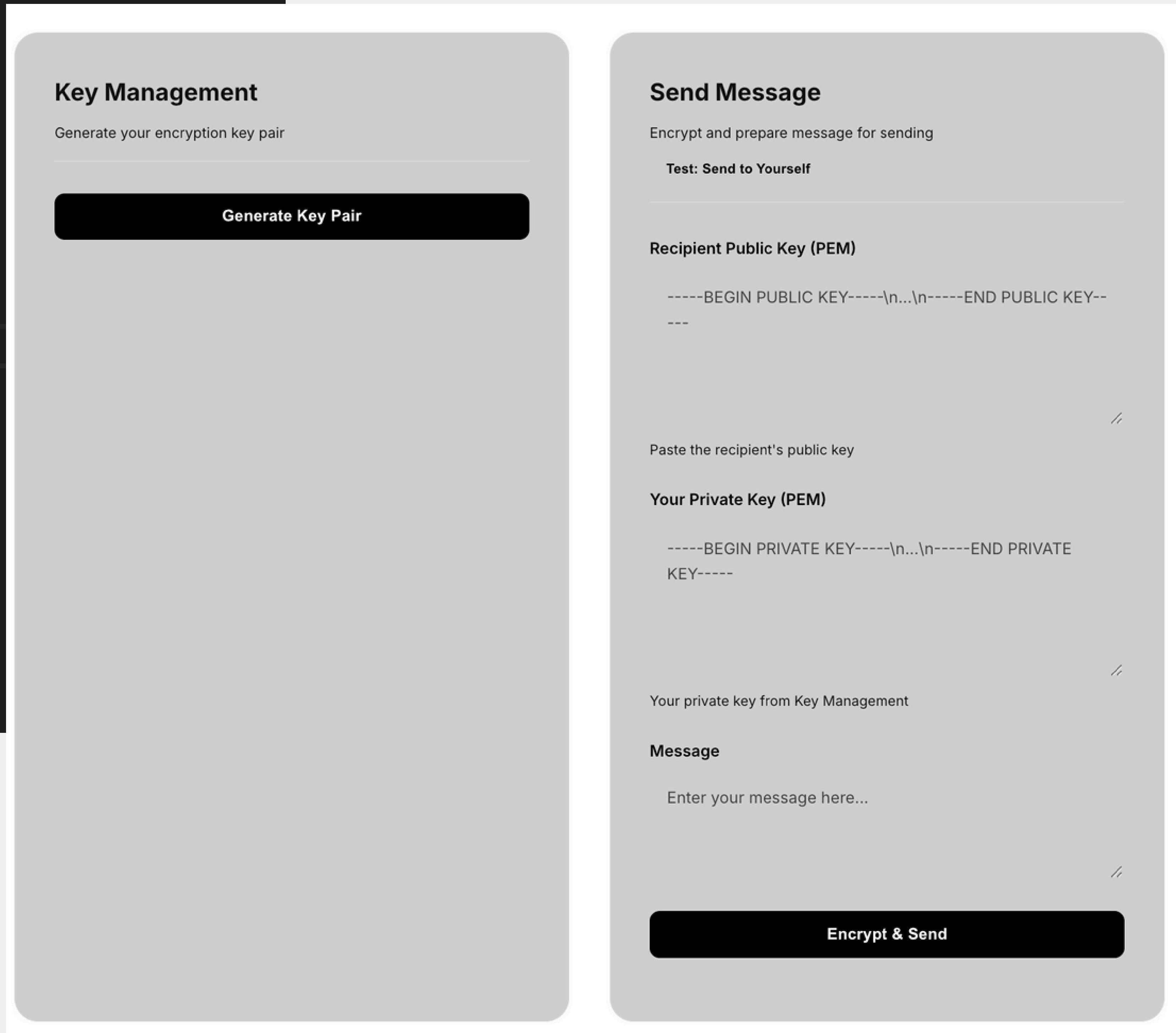


Secure File ENCRYPTION

```python
def encrypt_file(self, file_path: str, password: str, output_path: str = None) -> dict:
    if output_path is None:
        output_path = file_path + '.encrypted'
    original_hash = self.compute_file_hash(file_path)
    master_key, master_salt = self.derive_master_key(password)
    fek = self.generate_file_encryption_key()
    encrypted_fek, fek_nonce = self.encrypt_fek_with_master_key(fek, master_key)
    initial_nonce = secrets.token_bytes(12)
    nonce = initial_nonce
    aesgcm = AESGCM(fek)
    encrypted_chunks = []
    with open(file_path, 'rb') as f:
        while chunk := f.read(self.chunk_size):
            encrypted_chunk = aesgcm.encrypt(nonce, chunk, None)
            encrypted_chunks.append(encrypted_chunk)
            nonce = int.from_bytes(nonce, 'big')
            nonce = (nonce + 1) % (2**96)
            nonce = nonce.to_bytes(12, 'big')
    enc_sha = hashlib.sha256()
```

Secure Messaging

```python
def encrypt_message(self, recipient_public_key_bytes: bytes,
                    message: str, sender_private_key: ec.EllipticCurvePrivateKey) -> Dict:
    ephemeral_private, ephemeral_public_bytes = self.generate_keypair()
    shared_secret = self.derive_shared_secret(ephemeral_private, recipient_public_key_bytes)
    salt = secrets.token_bytes(32)
    aes_key = self.derive_aes_key(shared_secret, salt)
    message_bytes = message.encode('utf-8')
    nonce = secrets.token_bytes(12)
    aesgcm = AESGCM(aes_key)
    ciphertext = aesgcm.encrypt(nonce, message_bytes, None)
    auth_tag = ciphertext[-16:]
    encrypted_data = ciphertext[:-16]
    signature = self.sign_message(sender_private_key, ciphertext)
    return {
        'nonce': nonce.hex(),
        'ciphertext': encrypted_data.hex(),
        'auth_tag': auth_tag.hex(),
        'ephemeral_pubkey': ephemeral_public_bytes.decode('utf-8'),
        'signature': signature.hex(),
        'salt': salt.hex()
    }
```



Blockchain Audit Logging

```python
class Transaction:
    type: str
    data: dict
    timestamp: float

    def to_dict(self) -> dict:
        return {
            'type': self.type,
            'data': self.data,
            'timestamp': self.timestamp
        }

    def hash(self) -> str:
        tx_str = json.dumps(self.to_dict(), sort_keys=True)
        return hashlib.sha256(tx_str.encode()).hexdigest()
```

Block #1 00003ca53d09e093...

Previous Hash: 00000cb8f3db5e89...

Merkle Root: 8b31f56e115bc447...

Nonce: 160588

Transactions: 1

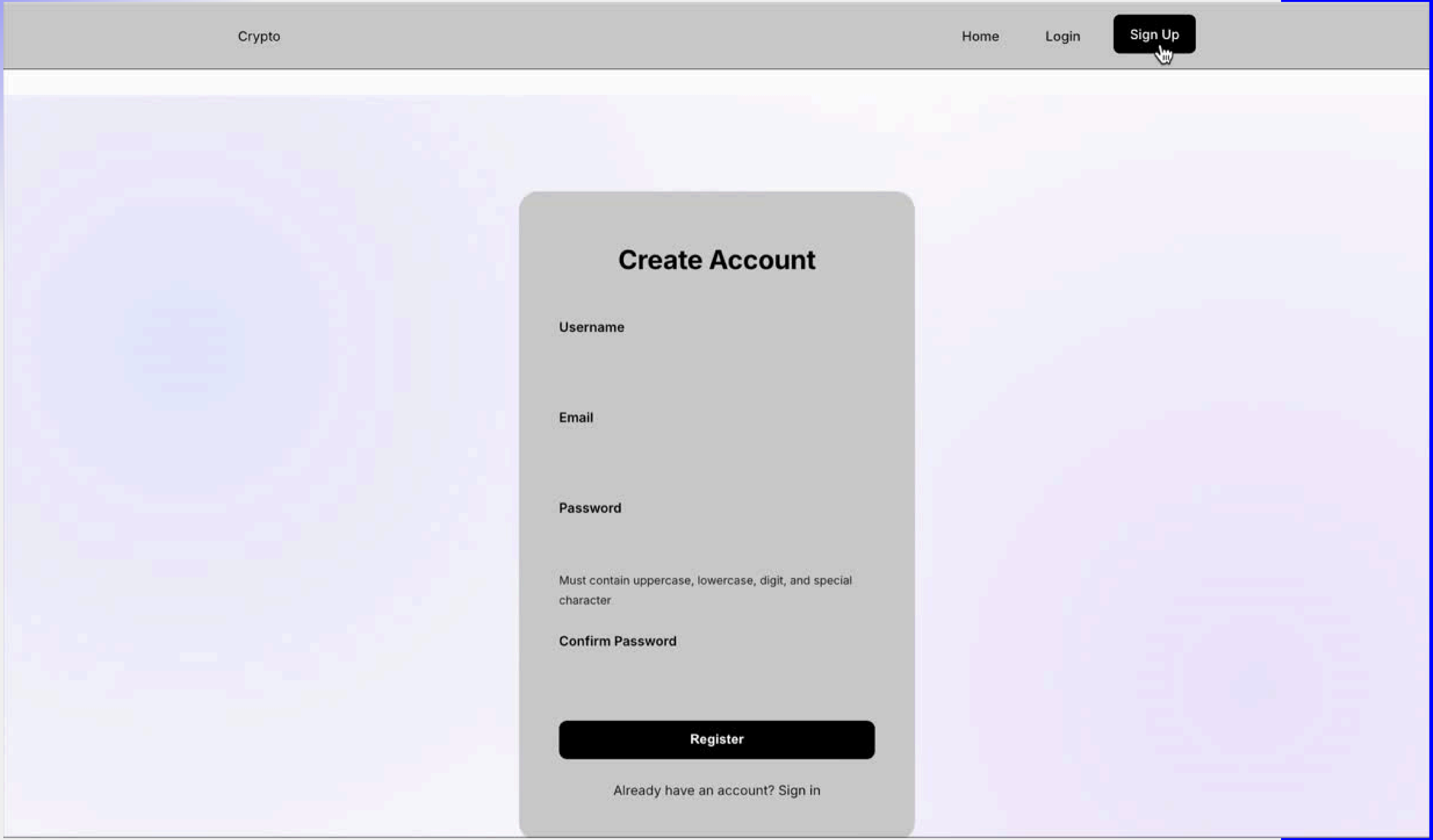
AUTH_LOGIN


```
{
  "ip_hash": "12ca17b49af2289436f303e0166030a21e525d266e209267433801a8fd4071a0",
  "success": false,
  "timestamp": 1766572134.4511945,
  "user_hash": "b6a3ac7cd7172b5f75c630f082694c269138be614a2bc347225dfc5409bfbd0e"
}
```


```python
class Block:
    index: int
    previous_hash: str
    transactions: List[Transaction]
    timestamp: float
    nonce: int
    merkle_root: str
    hash: str = None
```

```python
    def compute_hash(self) -> str:
        block_string = json.dumps({
            'index': self.index,
            'previous_hash': self.previous_hash,
            'merkle_root': self.merkle_root,
            'timestamp': self.timestamp,
            'nonce': self.nonce
        }, sort_keys=True)
        return hashlib.sha256(block_string.encode()).hexdigest()
```

Demo


Crypto
Home
Login
Sign Up
Create Account
Username
Email
Password
Must contain uppercase, lowercase, digit, and special character
Confirm Password
Register
Already have an account? Sign in

Threat Mitigation Summary

Each threat is addressed directly:

- Password theft → MFA
- Data theft → Encryption
- Insider attacks → Blockchain logs
- This creates defense in depth.

CONCLUSION

To conclude, project demonstrates how:

- Cryptography
- Authentication
- Blockchain

work together in a real security system.

This project shows not just theory, but practical security implementation.

Thank You