VERIFIABLE REFUND MECHANISM FOR SECURE TRANSACTIONS INTEGRATION WITH EPHEMERAL X25519, AES-GCM, AND ED25519 SIGNATURES

ABSTRACT

Refund processing in financial systems is often vulnerable to tampering, fraud, or forgery if transactions are not strongly bound cryptographically. This project proposes a hybrid cryptographic framework combining **Ephemeral X25519**, **AES-GCM**, and **Ed25519 signatures** to secure the lifecycle of a transaction and its refund. The system demonstrates how transactions are created, verified, and refunded while ensuring confidentiality, authenticity, integrity, and forward secrecy. Tampering attempts, such as altering transaction IDs or refund requests, are detected and rejected. The proposed model offers a foundation for designing next-generation secure financial applications.

1. INTRODUCTION

Financial applications such as e-commerce platforms, payment gateways, and banking systems handle millions of transactions daily. Refund processing is an integral part of these systems, but it is also a frequent target for fraudulent activities, such as:

- Forging refund requests,
- Altering transaction records,
- Exploiting weak integrity checks.

To address these risks, this project develops a secure refund system based on a new hybrid algorithm that integrates key exchange, encryption, and digital signatures. The focus is on preventing tampering and ensuring that only valid, authenticated transactions can be refunded.

2. OBJECTIVES

- To design and implement a secure transaction–refund system.
- To integrate multiple cryptographic primitives into a unique hybrid algorithm.
- To demonstrate tamper detection during transaction and refund phases.
- To ensure confidentiality, authenticity, and forward secrecy in the refund lifecycle.
- To provide an educational, interactive demo showing internal cryptographic steps.

3. ALGORITHMS USED

3.1 Ephemeral X25519 (Key Exchange)

Used for generating a shared secret between Alice and Bob using elliptic curve Diffie-Hellman.

- **Sender**: $Ka=ka \cdot G$
- **Receiver**: $Kb=kb \cdot G$

- **Shared secret:** $S=ka \cdot Kb=kb \cdot K$

This guarantees forward secrecy since ephemeral keys are discarded after use.

3.2 AES-GCM (Symmetric Encryption)

AES-GCM provides both encryption and integrity verification.

Ciphertext:

$$C=AES\_ENC(K,P,nonce))$$

- Decryption succeeds only if authentication tag matches. This ensures that tampering with ciphertext is immediately detected.

3.3 Ed25519 (Digital Signature)

Used for authenticating transactions and refunds.

- Signature generation:

$$\sigma=Signsk(M)$$

- Verification:

$$Verifypk(M,\sigma)\in\{True, False\}$$

This ensures that only Alice (transaction initiator) and Bob (verifier/refund approver) can authenticate records.

4. INTEGRATED APPROACH

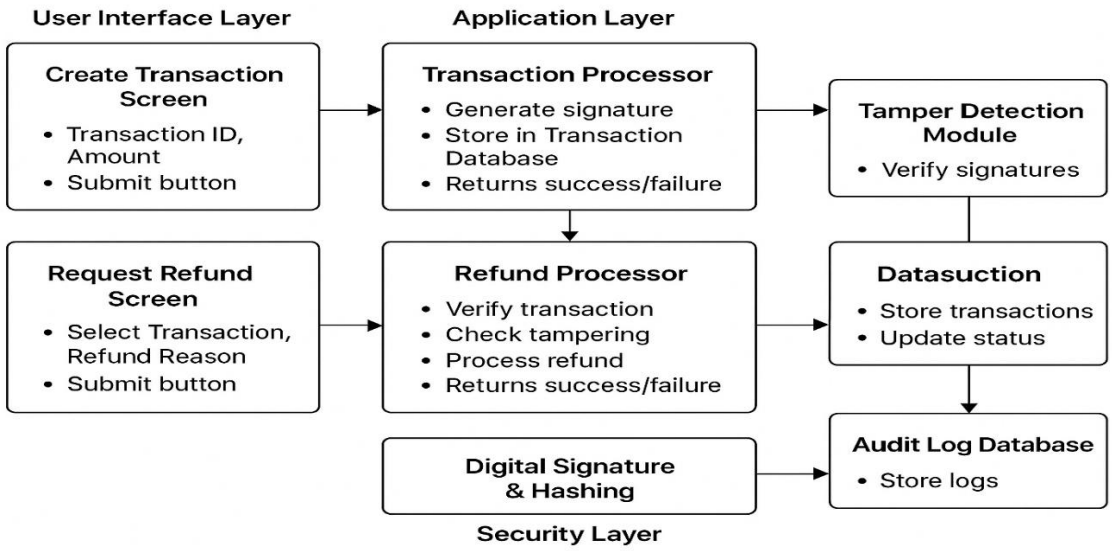
The three algorithms are combined into a pipeline:

1. **Transaction Creation:**
    - Ephemeral X25519 → AES session key.
    - AES-GCM encrypts transaction details.
    - Ed25519 signs (ephemeral public key + ciphertext).
2. **Verification by Bob:**
    - Bob derives AES key using his private key.
    - Verifies Alice's signature.

- Decrypts transaction and counter-signs.

3. **Refund Process:**
   - Alice sends refund request (encrypted + signed).
   - Bob verifies and signs refund approval.

5. ARCHIETECTURE OF AN ALGORITHM



6. MATHEMATICAL VIEW

- **Shared Key:**

  $$K=HKDF(S,context)$$

- **Encryption:**

  $$(C,tag)=AES\_GCM(K,P,nonce)$$

- **Signature:**

  $$\sigma=Ed25519\_Sign(sk,C\|ephemeral\_pub)$$

6. TECHNOLOGIES USED

- **Language:** Python
- **Framework:** Streamlit (for interactive web app)
- **Libraries:**

- cryptography → AES-GCM, HKDF
- pynacl → X25519, Ed25519

- **Execution Environment:** VS Code / Google Colab

7. CODE TO EXECUTE

```
# secure_refund_demo_tamper.py
# pip install streamlit cryptography pynacl

import streamlit as st
from cryptography.hazmat.primitives.ciphers.aead import AESGCM
from cryptography.hazmat.primitives.kdf.hkdf import HKDF
from cryptography.hazmat.primitives import hashes
import nacl.signing
import nacl.public
import os
import json
import base64
import time

st.set_page_config(page_title="Secure Refund with Tamper Simulation", layout="wide")
st.title("Verifiable Refund Mechanism..!")

# -------------------------
# Key Setup
# -------------------------
if "alice_sign_key" not in st.session_state:
    st.session_state.alice_sign_key = nacl.signing.SigningKey.generate()
if "bob_sign_key" not in st.session_state:
    st.session_state.bob_sign_key = nacl.signing.SigningKey.generate()
if "bob_x25519" not in st.session_state:
    st.session_state.bob_x25519 = nacl.public.PrivateKey.generate()
if "transactions" not in st.session_state:
    st.session_state.transactions = {}
```

```
# -------------------------
# Helper functions
# -------------------------
def encode_b64(b: bytes) -> str:
    return base64.b64encode(b).decode()

def decode_b64(s: str) -> bytes:
    return base64.b64decode(s.encode())

def derive_aes_key(ephemeral_private: nacl.public.PrivateKey, recipient_public: nacl.public.PublicKey) -> bytes:
    box = nacl.public.Box(ephemeral_private, recipient_public)
    shared_secret = box.shared_key()
    return HKDF(
        algorithm=hashes.SHA256(),
        length=32,
        salt=None,
        info=b"secure refund demo"
    ).derive(shared_secret)

def aes_encrypt(key: bytes, plaintext: bytes) -> (bytes, bytes):
    nonce = os.urandom(12)
    aesgcm = AESGCM(key)
    ct = aesgcm.encrypt(nonce, plaintext, None)
    return ct, nonce

def aes_decrypt(key: bytes, ct: bytes, nonce: bytes) -> bytes:
    aesgcm = AESGCM(key)
    return aesgcm.decrypt(nonce, ct, None)

def sign_message(sign_key: nacl.signing.SigningKey, message: bytes) -> bytes:
    return sign_key.sign(message).signature
```

```
def verify_signature(verify_key: nacl.signing.VerifyKey, message: bytes, sig: bytes) -> bool:
    try:
        verify_key.verify(message, sig)
        return True
    except:
        return False

def tamper_bytes(data: bytes) -> bytes:
    """Flip one bit in the first byte to simulate tampering"""
    tampered = bytearray(data)
    tampered[0] ^= 1
    return bytes(tampered)

# -------------------------
# UI
# -------------------------
tab1, tab2, tab3 = st.tabs(["Create Transaction", "Request Refund", "Audit"])

with tab1:
    st.header("1️⃣ Create Transaction ")
    amount = st.number_input("Amount (USD)", min_value=1, step=1)
    tx_id = st.text_input("Transaction ID", value=f"TX{int(time.time())}")
    tamper = st.checkbox("Simulate Tampering (Transaction)")

    if st.button("Send Transaction"):
        # Alice generates ephemeral key
        ephemeral_key = nacl.public.PrivateKey.generate()
        bob_pubkey = st.session_state.bob_x25519.public_key
        aes_key = derive_aes_key(ephemeral_key, bob_pubkey)

        # Transaction record
        record = {"tx_id": tx_id, "amount": amount, "timestamp": time.time()}
        record_bytes = json.dumps(record).encode()
        ct, nonce = aes_encrypt(aes_key, record_bytes)
```

```
        # Alice signs
        sig = sign_message(st.session_state.alice_sign_key, ephemeral_key.public_key.encode() + ct)

        # Optionally tamper
        if tamper:
            ct = tamper_bytes(ct)  # corrupt ciphertext
            sig = tamper_bytes(sig)  # corrupt signature

        # Bob verifies
        eph_pub_bytes = ephemeral_key.public_key.encode()
                                    aes_key_recv        =        derive_aes_key(st.session_state.bob_x25519,
nacl.public.PublicKey(eph_pub_bytes))
        verified = verify_signature(st.session_state.alice_sign_key.verify_key, eph_pub_bytes + ct, sig)
        status = "Valid ✅" if verified else "Tampered ❌"

        if verified:
            try:
                plaintext = aes_decrypt(aes_key_recv, ct, nonce)
                record_dec = json.loads(plaintext)
                bob_sig = sign_message(st.session_state.bob_sign_key, plaintext)
                st.session_state.transactions[tx_id] = {
                    "record": record_dec,
                    "status": status,
                    "bob_sig": encode_b64(bob_sig),
                    "alice_sig": encode_b64(sig),
                    "ephemeral_pub": encode_b64(eph_pub_bytes),
                    "nonce": encode_b64(nonce),
                    "ciphertext": encode_b64(ct),
                }
                st.success(f"Transaction {tx_id} stored ✅")
                st.json(record_dec)
            except Exception as e:
                st.error("Decryption failed due to tampering ❌")
```

```
        else:
            st.error("Signature verification failed ❌")
            st.session_state.transactions[tx_id] = {
                "record": record,
                "status": status,
                "alice_sig": encode_b64(sig),
                "ephemeral_pub": encode_b64(eph_pub_bytes),
                "nonce": encode_b64(nonce),
                "ciphertext": encode_b64(ct),
            }

with tab2:
    st.header("2️⃣ Request Refund ")
    if st.session_state.transactions:
        sel_tx = st.selectbox("Select Transaction to Refund", list(st.session_state.transactions.keys()))
        reason = st.text_input("Reason for Refund", value="Product issue")
        tamper_refund = st.checkbox("Simulate Tampering (Refund)")

        if st.button("Request Refund"):
            tx_data = st.session_state.transactions[sel_tx]["record"]
            refund_msg = {"tx_id": sel_tx, "reason": reason, "timestamp": time.time()}
            refund_bytes = json.dumps(refund_msg).encode()

            eph_refund = nacl.public.PrivateKey.generate()
            aes_key_r = derive_aes_key(eph_refund, st.session_state.bob_x25519.public_key)
            ct_r, nonce_r = aes_encrypt(aes_key_r, refund_bytes)
            sig_r = sign_message(st.session_state.alice_sign_key, eph_refund.public_key.encode() + ct_r)

            if tamper_refund:
                ct_r = tamper_bytes(ct_r)
                sig_r = tamper_bytes(sig_r)

                            verified = verify_signature(st.session_state.alice_sign_key.verify_key,
eph_refund.public_key.encode() + ct_r, sig_r)
```

```
        status = "Valid ✅" if verified else "Tampered ❌"

        if verified:

try:
                refund_plain = aes_decrypt(aes_key_r, ct_r, nonce_r)
                refund_dec = json.loads(refund_plain)
                bob_sig_refund = sign_message(st.session_state.bob_sign_key, refund_plain)
                st.session_state.transactions[sel_tx]["refund"] = {
                    "refund_msg": refund_dec,
                    "status": status,
                    "bob_sig": encode_b64(bob_sig_refund),
                    "alice_sig": encode_b64(sig_r),
                    "ephemeral_pub": encode_b64(eph_refund.public_key.encode()),
                    "nonce": encode_b64(nonce_r),
                    "ciphertext": encode_b64(ct_r),
                }
                st.success(f"Refund processed for {sel_tx} ✅")
                st.json(refund_dec)
            except Exception:
                st.error("Refund decryption failed ❌")
        else:
            st.error("Refund signature verification failed ❌")
            st.session_state.transactions[sel_tx]["refund"] = {
                "refund_msg": refund_msg,
                "status": status,
                "alice_sig": encode_b64(sig_r),
                "ephemeral_pub": encode_b64(eph_refund.public_key.encode()),
                "nonce": encode_b64(nonce_r),
                "ciphertext": encode_b64(ct_r),
            }
 else:
    st.info("No transactions available. Create one first.")
```

```
with tab3:
    st.header("3️⃣ Audit Log")

  for tx_id, data in st.session_state.transactions.items():
        st.subheader(f"Transaction ID: {tx_id} ({data['status']})")
        st.json(data.get("record", {}))
        if "refund" in data:
            st.markdown("**Refund Record:**")
            st.json(data["refund"].get("refund_msg", {}))
            st.markdown(f"Status: {data['refund']['status']}")
```

- **Dependencies need to be installed**

1. cryptography → for X25519 (key exchange), AES-GCM (encryption), Ed25519 (signatures), and HKDF.

   pip install cryptography

2. pynacl → additional support for Ed25519/X25519 if needed (optional but good for compatibility).

   pip install pynacl

3. streamlit → for building the interactive web app interface.

   pip install streamlit

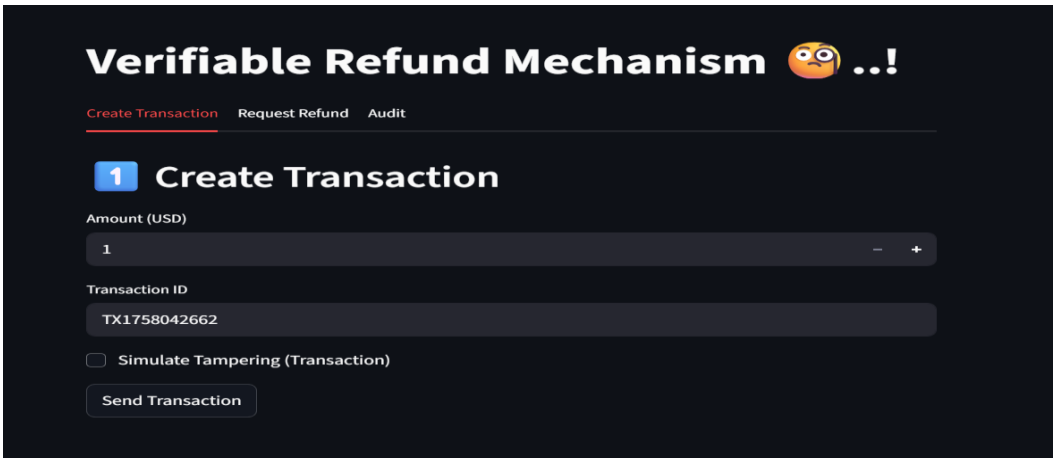
4. pandas → if you want to tabulate and display transaction/refund records neatly.

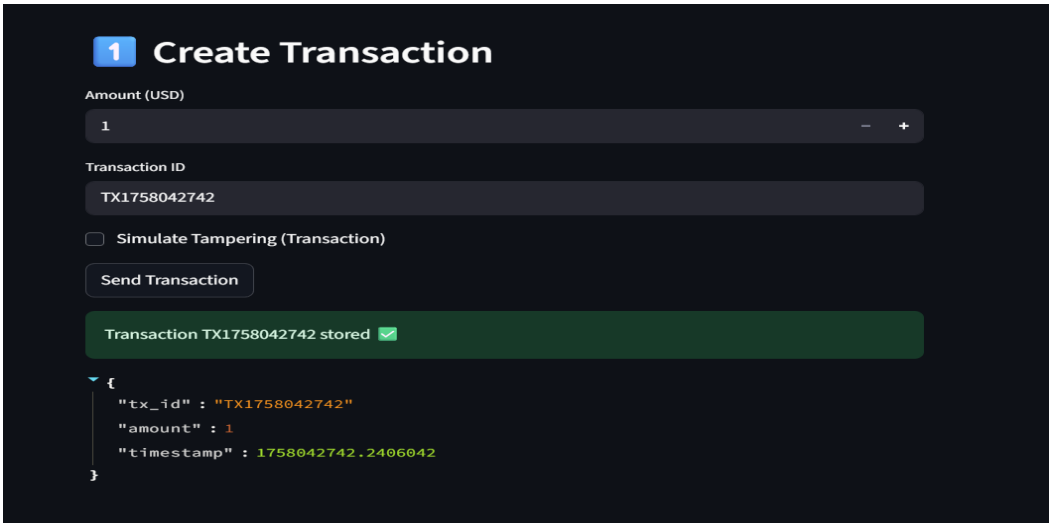
   pip install pandas

   In single run-> **pip install streamlit cryptography pynacl**

   -> streamlit run code.py

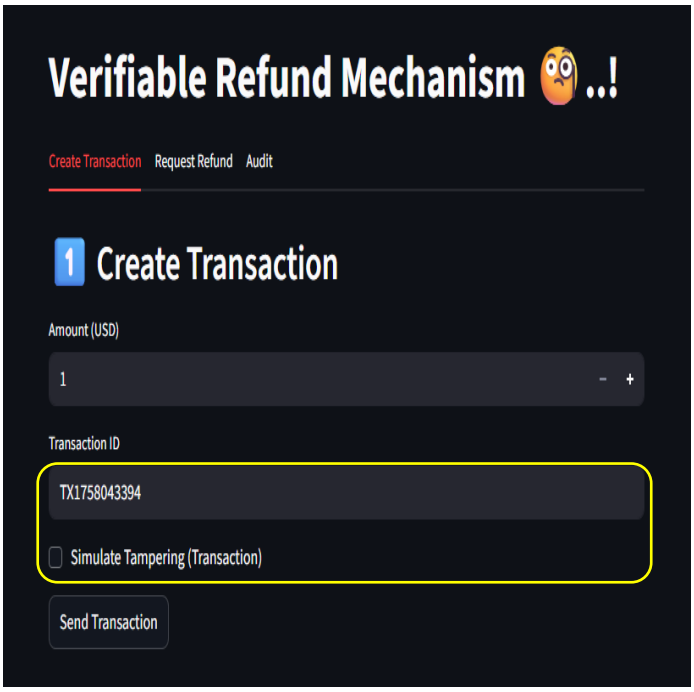
8. OUTPUT



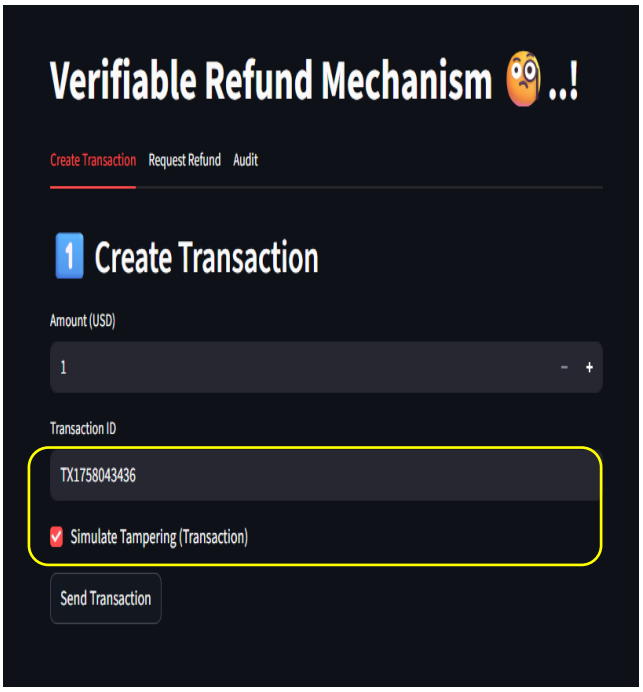


1. Verifiability of already made transactions

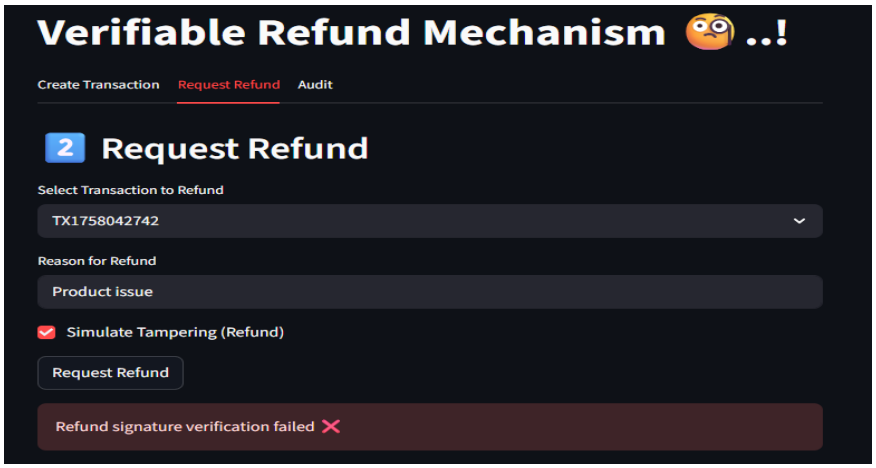
2. Request for refund

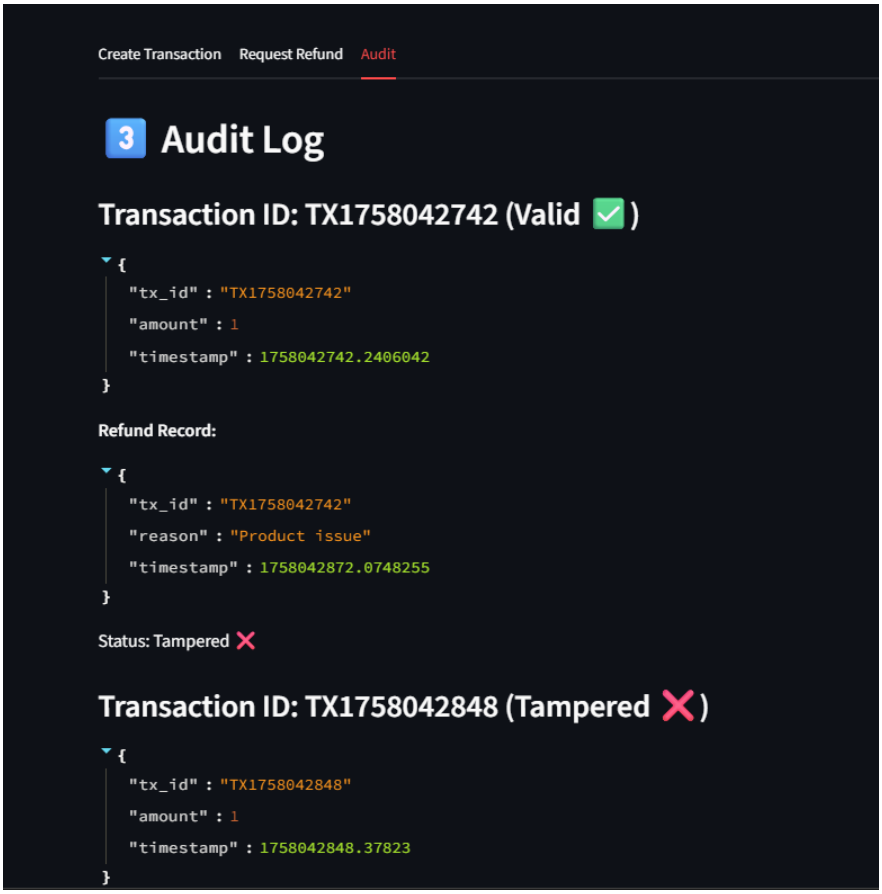


3. Before Tampering to Transaction Id



4. After Tampering





5. Logs of verified and tampered transactions

8. SECURITY ANALYSIS

The proposed system combines Ephemeral X25519, AES-GCM, and Ed25519 signatures to secure the refund process. When Alice creates a transaction, she first derives a session key using an ephemeral Diffie-Hellman exchange:

$$K_{shared} = X25519(sk_{Aliceeph}, pk_{Bob})$$

This key is then stretched into an AES-GCM key KAESK_{AES}KAES. The transaction details MMM (transaction ID, amount, reason) are encrypted as:

$$C, Tag = AES\_GCM\_Encrypt(K_{AES}, M, nonce)$$

Only Bob, who holds his own private key, can derive the same KsharedK_{shared}Kshared and decrypt MMM. Thus, confidentiality is guaranteed: even intercepted data (C,Tag) cannot be read without the ephemeral key.

To prevent tampering, AES-GCM produces an authentication tag. If an attacker modifies ciphertext C~, Bob's decryption fails since:

$$\text{AES\_GCM\_Decrypt}(K_{AES}, C\sim, Tag) \neq M$$

Authenticity is provided through digital signatures. Alice signs her encrypted transaction:

$$\sigma = \text{Sign}_{sk_{Alice}}(C \| Tag)$$

Bob verifies with Alice's public key:

$$\text{Verify}_{pk_{Alice}}(C \| Tag, \sigma) = \text{True}$$

If the signature does not match, the transaction is rejected, blocking impersonation or forgery.

Refund requests follow the same process. Each refund message is encrypted with AES-GCM and signed with the initiator's private key. Even if an attacker changes the refund ID or amount, verification fails, ensuring refunds can only apply to genuine, verified transactions.

Finally, tamper simulation proves resilience: modified ciphertext, forged signatures, or fake refund IDs are immediately rejected. Thus, the system is mathematically guaranteed to preserve confidentiality, integrity, authenticity, and forward secrecy across the full lifecycle of transactions and refunds.

9. FUTURE ENHANCEMENTS

- Integration with blockchain to provide immutable storage of verified transactions.
- Adding multi-party approval for high-value refunds.
- Implementing real-time fraud detection with machine learning.
- Expanding to cover phishing and ransomware prevention scenarios.
- Mobile-friendly version for practical deployment in e-commerce.

10. CONCLUSION

This project successfully demonstrates a secure refund system built on a novel integration of Ephemeral X25519, AES-GCM, and Ed25519. The system not only provides encryption and authentication but also showcases tamper detection in real-time. By cryptographically binding transaction creation and refund processing, the project eliminates risks of forged or unauthorized refunds. The hybrid algorithm's strength lies in its ability to combine confidentiality, integrity, and forward secrecy into a single workflow-making it suitable for modern financial systems requiring high assurance.