Mesures de Securite pour l'Escrow Interne

Document Technique

1 SECURITE ARCHITECTURALE

1.1 Isolation des Donnees Sensibles

```
+---------------------------------------------+
|     BASE DE DONNEES PRINCIPALE              |
|  - Table users (id, phone, kyc_status)      |
|  - Table offers                             |
|  - Table transactions                       |
+---------------------------------------------+
|     BASE DE DONNEES SECURISEE (ISOLEE)      |
|  - Table wallets (balances, user_id)        |  <- CRITIQUE
|  - Table escrow_locks                       |  <- CRITIQUE
|  - Table audit_logs                         |  <- TOUTES modifications
+---------------------------------------------+
```

1.2 Microservices Separes

```
# Service Wallet (acces restreint)
class WalletService:
    endpoints = ["/wallet/balance", "/wallet/transfer"]
    auth_level = "HIGH"
    rate_limit = "10 req/min"

# Service Escrow (ultra-securise)
class EscrowService:
    endpoints = ["/escrow/lock", "/escrow/release"]
    auth_level = "MAXIMUM"
    rate_limit = "5 req/min"
    whitelist_only = True  # Seulement services internes
```

2 MECANISMES DE BLOCAGE DES FONDS

2.1 Double Validation des Transactions

Workflow de blocage :

1. User A1 initie echange -> demande de blocage
2. Service Wallet verifie solde >= montant
3. Service Escrow cree un LOCK avec hash unique
4. Base de donnees : UPDATE avec version locking
5. Audit : log crypte de l'operation

2.2 Implementation Technique

```
class SecureEscrow:
    def lock_funds(self, user_id, amount, currency, offer_id):
        # 1. Verifier signature numerique
        if not self.verify_digital_signature(user_id, offer_id):
            raise SecurityError("Signature invalide")

        # 2. Version locking pour eviter double-spend
        with db.transaction(isolation_level='SERIALIZABLE') as tx:
            # Recupere avec FOR UPDATE (verrouillage)
            wallet = tx.execute("""
                SELECT balance, version
                FROM wallets
                WHERE user_id = ? AND currency = ?
                FOR UPDATE
            """, (user_id, currency))

            # Verifier solde
            if wallet.balance < amount:
                tx.rollback()
                raise InsufficientFundsError()

            # 3. Creer lock avec UUID cryptographique
            lock_id = self.generate_crypto_uuid()

            # 4. Debiter ET creer lock en une seule operation
            tx.execute("""
                UPDATE wallets
                SET balance = balance - ?,
                    version = version + 1
                WHERE user_id = ? AND currency = ?
            """, (amount, user_id, currency))

            tx.execute("""
                INSERT INTO escrow_locks
                (id, user_id, amount, currency, offer_id,
                 created_at, expires_at, status, hash)
                VALUES (?, ?, ?, ?, ?, ?, ?, 'LOCKED', ?)
            """, (lock_id, user_id, amount, currency, offer_id,
                  datetime.now(), datetime.now() + timedelta(hours=24),
                  self.calculate_hash(user_id, amount, offer_id)))

            # 5. Log d'audit immuable
            self.log_to_immutable_ledger(
                action="LOCK",
                lock_id=lock_id,
                user_id=user_id,
                amount=amount
            )

        return lock_id
```

3 SURVEILLANCE EN TEMPS REEL

3.1 Systeme de Detection d'Anomalies

```
class AnomalyDetection:
    def monitor_escrow_activity(self):
        rules = {
            "RAPID_SUCCESSIVE_LOCKS": {
                "condition": ">3 locks en 60 secondes pour meme user",
                "action": "FREEZE_ACCOUNT",
                "severity": "HIGH"
            },
            "UNUSUAL_AMOUNTS": {
                "condition": "montant > 10x moyenne historique user",
                "action": "REQUIRE_2FA",
                "severity": "MEDIUM"
            },
            "ROUND_NUMBER_PATTERN": {
                "condition": "5 transactions montants ronds (1000, 2000...)",
                "action": "FLAG_FOR_REVIEW",
                "severity": "LOW"
            },
            "UNUSUAL_TIME": {
                "condition": "transaction entre 2h-5h locale user",
                "action": "SMS_VERIFICATION",
                "severity": "MEDIUM"
            }
        }
```

3.2 Dashboard de Monitoring

```
+-----------------------------------------------+
|          DASHBOARD SECURITE ESCROW            |
+-----------------------------------------------+
| ALERTES ACTIVES (3)                           |
|   - User#4421: 5 locks/2min (suspect)         |
|   - XOF: Position nette a 8% (RISQUE)         |
|   - Serveur DB: CPU a 95%                     |
+-----------------------------------------------+
| STATISTIQUES 24H                              |
|   - Locks: 1,452                              |
|   - Releases: 1,448                           |
|   - Echecs: 4 (0.27%)                         |
|   - Temps moyen: 47ms                         |
+-----------------------------------------------+
| VERROUILLAGES ANORMAUX                        |
|   - XOF: 12 locks > 1M                        |
|   - EUR: 3 locks > 10K                        |
|   - Anomalie geographique: CI -> SN soudain   |
+-----------------------------------------------+
```

4 PROTECTION CONTRE LES ATTAQUES

4.1 Protection Double-Spend

```
class DoubleSpendProtection:
    def __init__(self):
        self.pending_locks = {}
        self.lock_timeout = 30

    def attempt_lock(self, user_id, amount, currency):
        key = f"{user_id}:{currency}"

        if self.is_processing(key):
            raise ConcurrentModificationError(
                "Transaction deja en cours pour ce wallet")

        with redis.lock(f"wallet_lock:{key}", timeout=self.lock_timeout):
            current = self.get_wallet_with_version(user_id, currency)

            locked_amount = self.get_pending_locks_amount(user_id, currency)
            available = current.balance - locked_amount

            if available < amount:
                raise InsufficientFundsError(
                    f"Solde disponible: {available}, demande: {amount}")

            return True
```

4.2 Timeouts et Rollbacks Automatiques

Schema de timeout intelligent :

- Lock standard : 24 heures maximum
- Apres 6h : Notification “Transaction en attente”
- Apres 12h : Notification aux deux parties
- Apres 18h : Alerte support
- Apres 24h : AUTO-ROLLBACK avec notification

5 JOURNALISATION ET AUDIT

5.1 Ledger Immutable

```
class ImmutableAuditLedger:
    def log_transaction(self, action, data):
        previous_hash = self.get_last_hash()
        current_hash = sha256(
            f"{previous_hash}:{json.dumps(data)}:{timestamp}"
        ).hexdigest()

        self.write_to_ledger({
            "timestamp": datetime.utcnow().isoformat(),
            "action": action,
            "data": data,
            "previous_hash": previous_hash,
            "hash": current_hash,
            "signed_by": self.get_system_signature()
        })

        self.backup_to_cold_storage(current_hash)
```

5.2 Format de Log Standardise

```
{
  "audit_id": "AUD-2024-01-15-xyz789",
  "timestamp": "2024-01-15T14:30:45.123Z",
  "event_type": "ESCROW_LOCK",
  "user_id": "user_12345",
  "wallet_id": "wal_67890",
  "amount": 100000,
  "currency": "XOF",
  "offer_id": "offer_abc123",
  "lock_id": "lock_def456",
  "previous_balance": 150000,
  "new_balance": 50000,
  "locked_balance": 100000,
  "ip_address": "192.168.1.100",
  "user_agent": "App/1.0 iOS/16.5",
  "geolocation": "Dakar, SN",
  "risk_score": 15,
  "signature": "rsa-2048:abc123def456",
  "merkle_proof": "0x123abc..."
}
```

6 RESUME DES 10 POINTS CLES

1. **Isolation** : Base de donnees wallets separee
2. **Atomicite** : Transactions tout-ou-rien avec rollback
3. **Immuabilite** : Ledger crypte pour audit
4. **Surveillance** : Detection d'anomalies en temps reel
5. **Resilience** : Sauvegardes 3-2-1-1-0
6. **Verifications** : 4 points pour chaque echange P2P
7. **Reputation** : Scores de confiance dynamiques
8. **Tests** : Audit quotidien + mensuel externe

9. **Monitoring** : Dashboard temps reel avec alertes
10. **Urgence** : Procedures documentees et testees

Quelle partie souhaitez-vous que je detaille davantage ?