LOMÉ BUSINESS SCHOOL

Département Informatique & Transformation Digitale

DOCUMENT TECHNIQUE

ARCHIPEL

Protocole P2P Chiffré et Décentralisé à Zéro-Connexion

Vendredi 27 Février 2026, 16h00 ➔ Samedi 28 Février 2026, 16h00

24 HEURES DE CODING NON-STOP

"The Geek & The Moon"

Février 2026

1. CONTEXTE & MISSION

Vous avez 24 heures pour concevoir et implémenter **Archipel** : un protocole de communication P2P fonctionnant sans Internet, sans serveur central, sans autorité de certification. Archipel doit survivre à une coupure totale d'infrastructure.

> **MISSION :** Créer un réseau local souverain, chiffré bout-en-bout, où chaque nœud est à la fois client et serveur — inspiré de BitTorrent, blindé comme Signal.

Architecture cible

Le protocole repose sur trois piliers fondamentaux :

- **Décentralisation totale :** aucune donnée ne transite par un serveur central. Pas de tracker. Pas de DNS. Rien.
- **Segmentation des données (chunking) :** fichiers et messages fragmentés en blocs pour transfert parallèle et tolérance aux pannes.
- **Chiffrement de bout en bout :** chaque paquet chiffré avant émission, jamais en clair sur le réseau local.

Contrainte absolue

> **ZÉRO CONNEXION INTERNET** Le prototype doit fonctionner sur un réseau local ad-hoc pur. Tout appel vers l'extérieur = disqualifié lors de la démo.

2. ORGANISATION DES ÉQUIPES & GITHUB

Accès au dépôt GitHub

Un dépôt GitHub a été créé pour le hackathon. Les responsables d'équipe seront ajoutés en tant qu'administrateurs sur ce repo. C'est leur responsabilité d'ajouter les autres membres de leur équipe.

Rôle	Action requise	Délai
Responsable d'équipe	Accepter l'invitation admin GitHub	Dès réception (< 30 min)
Responsable d'équipe	Ajouter les membres de l'équipe au repo	Sprint 0 (< 1h)
Tous les membres	Configurer git local + premier commit	Sprint 0 (< 1h)
Tous les membres	Créer sa branche de travail	Début Sprint 1

Workflow Git obligatoire

- **Branches :** main (stable) + develop + branches feature/xxx par sprint
- **Commits :** au minimum 1 commit toutes les 2 heures par membre actif
- **Issues :** chaque tâche technique = une issue GitHub. Les issues servent de log de progression au jury.
- **README.md :** doit être mis à jour à chaque fin de sprint avec l'état d'avancement.

3. PLAN DE SPRINTS — 24 HEURES

Les 26 heures sont découpées en 4 sprints techniques + un buffer de finalisation. Chaque sprint a des livrables précis et des critères de validation binaires (passe / ne passe pas).

Sprint	Titre	Durée	Livrable clé
0	Bootstrap & Architecture	H+0 → H+2	Repo actif, proto PKI, spec protocole
1	Couche Réseau P2P	H+2 → H+8	Découverte de pairs fonctionnelle
2	Chiffrement & Auth	H+8 → H+13	E2E chiffré, auth sans CA
3	Chunking & Transfert	H+13 → H+19	Transfert fichier 50 Mo multi-nœuds
4	Intégration & Polish	H+19 → H+23	CLI/UI démo, README complet
Buffer	Finalisation DevPost	H+23 → H+24	Soumission DevPost, démo prête

SPRINT 0 — BOOTSTRAP & ARCHITECTURE H+0 → H+2

Objectifs Sprint 0

Ces 2 premières heures sont critiques. Chaque minute perdue ici se répercute sur tous les sprints suivants. Focus absolu : environnement opérationnel + décisions d'architecture figées.

Checklist technique — Sprint 0

- Repo GitHub cloné par tous les membres, branches créées
- Choix du langage principal et justification dans le README
- Choix de la technologie de transport local (voir tableau ci-dessous)
- Génération des paires de clés RSA/Ed25519 pour chaque nœud
- Définition du format de paquet binaire (header + payload + checksum)
- Maquette de l'architecture en ASCII ou schéma dans le README

Choix technologique : Transport local

Technologie	Cas d'usage	Avantage	Contrainte
UDP Multicast (LAN)	Découverte + broadcast	Simple, standard	Réseau LAN filaire requis
TCP Sockets (LAN)	Transfert de données	Fiable, contrôle flux	Connexion point-à-point
Wi-Fi Direct	Sans infrastructure réseau	No AP needed	Support OS variable
Bluetooth L2CAP	Portée courte	Ubiquitaire	Débit limité (~1-3 Mbps)

RECOMMANDATION S0 Pour un hackathon 24h : UDP Multicast pour la découverte + TCP pour le transfert. Stack éprouvée, documentation abondante, debug facile.

Format de paquet Archipel — Spécification minimale

```
┌──────────────────────────────────────────────────────┐
│  ARCHIPEL PACKET v1                                  │
├──────────┬──────────┬──────────┬─────────────────────┤
│  MAGIC   │  TYPE    │  NODE_ID │  PAYLOAD_LEN        │
│  4 bytes │  1 byte  │  32 bytes│  4 bytes (uint32_BE)│
├──────────┴──────────┴──────────┴─────────────────────┤
│  PAYLOAD (chiffré, longueur variable)                │
├──────────────────────────────────────────────────────┤
│  HMAC-SHA256 SIGNATURE  (32 bytes)                   │
└──────────────────────────────────────────────────────┘

Types de paquets :
  0x01  HELLO      — annonce de présence sur le réseau
  0x02  PEER_LIST  — réponse avec liste des nœuds connus
  0x03  MSG        — message chiffré
  0x04  CHUNK_REQ  — requête d'un bloc de fichier
  0x05  CHUNK_DATA — transfert d'un bloc de fichier
  0x06  MANIFEST   — métadonnées d'un fichier (hash, nb chunks)
  0x07  ACK        — acquittement
```

LIVRABLE S0 README.md avec : stack choisie + schéma archi + format paquet. Premier commit tagué 'sprint-0'. Clés PKI générées.

SPRINT 1 — COUCHE RÉSEAU P2P — DÉCOUVERTE & ROUTAGE

H+2 → H+8 (6h)

Objectifs Sprint 1

Implémenter la couche fondamentale du protocole : la découverte de pairs et la gestion du réseau mesh. À la fin de ce sprint, deux nœuds doivent se trouver automatiquement sur le réseau local et s'échanger leurs métadonnées.

Module 1.1 — Découverte de pairs (Node Discovery)

- **Mécanisme :** UDP Multicast sur 239.255.42.99:6000 (adresse multicast privée)
- **Paquet HELLO :** émis toutes les 30 secondes par chaque nœud actif
- **Réponse PEER_LIST :** envoi en unicast TCP de la liste des nœuds connus avec leurs ports
- **Timeout :** nœud considéré mort après 90 secondes sans HELLO

```
// Pseudocode — Boucle de découverte
function discoveryLoop() {
  socket.bind(6000);
  socket.addMembership('239.255.42.99');

  setInterval(() => {
    const hello = buildPacket(TYPE.HELLO, {
      node_id: myKeyPair.publicKey,
      tcp_port: MY_TCP_PORT,
      timestamp: Date.now()
    });
    socket.send(hello, 6000, '239.255.42.99');
  }, 30000);

  socket.on('message', (msg, rinfo) => {
    const pkt = parsePacket(msg);
    if (pkt.type === TYPE.HELLO && pkt.node_id !== myKeyPair.publicKey) {
      peerTable.upsert(pkt.node_id, { ip: rinfo.address, port: pkt.tcp_port });
      replyWithPeerList(rinfo.address, rinfo.port);
    }
  });
}
```

Module 1.2 — Table de routage P2P (Peer Table)

Chaque nœud maintient une table de pairs en mémoire (et sur disque pour persistance) :

Champ	Type	Description
node_id	bytes[32]	Clé publique Ed25519 du nœud (identifiant unique)
ip	string	Adresse IP locale du pair
tcp_port	uint16	Port TCP d'écoute pour transferts
last_seen	timestamp	Dernière réponse HELLO reçue
shared_files	hash[]	Liste des hash de fichiers disponibles chez ce pair
reputation	float	Score de fiabilité (ratio succès/échecs des chunks)

Module 1.3 — Serveur TCP d'écoute

- **Port :** configurable via .env, défaut 7777
- **Connexions simultanées :** minimum 10 connexions parallèles
- **Protocole :** TLV (Type-Length-Value) sur le stream TCP
- **Keep-alive :** ping/pong toutes les 15 secondes sur les connexions établies

TEST S1 Lancer 3 instances du nœud sur des ports différents (même machine ou machines différentes). Vérifier que chaque nœud liste les 2 autres dans sa peer table. Logguer les HELLOs.

LIVRABLE S1 Démo : 3 nœuds se découvrent en < 60 secondes. Peer table affichée en console. Commit tagué 'sprint-1'.

SPRINT 2 — CHIFFREMENT E2E & AUTHENTIFICATION SANS CA

H+8 → H+13 (6h)

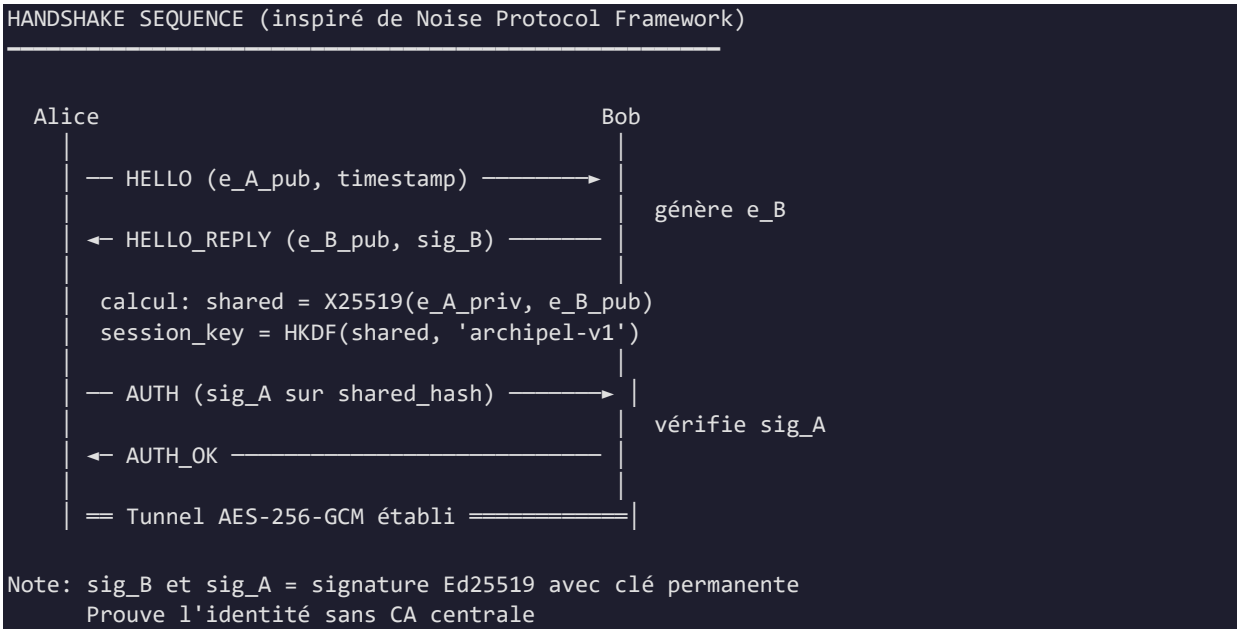
Objectifs Sprint 2

Le sprint le plus critique en termes de sécurité. Vous devez implémenter le chiffrement de bout en bout et résoudre le problème de confiance entre pairs sans autorité centrale. C'est le cœur différenciateur du projet.

Module 2.1 — Cryptographie des nœuds

- **Identité du nœud :** paire de clés Ed25519 (signing/verification)
- **Chiffrement des échanges :** X25519 (ECDH) pour l'échange de clé + AES-256-GCM pour le chiffrement des données
- **Intégrité des paquets :** HMAC-SHA256 sur chaque paquet
- **Clés éphémères :** nouvelle clé de session à chaque connexion TCP (Forward Secrecy)

Module 2.2 — Handshake Archipel



Module 2.3 — Web of Trust (Authentification sans CA)

Archipel ne peut pas utiliser une CA centrale. La solution : un modèle Web of Trust inspiré de PGP, simplifié pour le hackathon.

- **Trust Bootstrap :** premier contact = TOFU (Trust On First Use). La clé publique du pair est stockée localement avec son empreinte.
- **Vérification :** à chaque reconnexion, vérification que la clé publique du pair correspond à celle mémorisée (détection MITM).
- **Trust propagation :** un nœud peut signer la clé d'un autre pair pour l'introduire au réseau. Score de confiance propagé.
- **Révocation :** un nœud peut broadcast un message signé de révocation de sa propre clé (ex: compromission).
-

Module 2.4 — Chiffrement des messages

```
// Envoi d'un message chiffré
function sendMessage(recipientPubKey, plaintext) {
  // 1. Récupérer/créer la session avec ce pair
  const session = getOrCreateSession(recipientPubKey);

  // 2. Chiffrer avec la clé de session AES-256-GCM
  const nonce = crypto.randomBytes(12);  // 96-bit nonce
  const cipher = createCipheriv('aes-256-gcm', session.key, nonce);
  const encrypted = Buffer.concat([cipher.update(plaintext), cipher.final()]);
  const tag = cipher.getAuthTag();  // 128-bit auth tag

  // 3. Construire le paquet TYPE.MSG
  const pkt = buildPacket(TYPE.MSG, {
    nonce,          // 12 bytes
    ciphertext: encrypted,
    auth_tag: tag,  // 16 bytes
    sender_id: myKeyPair.publicKey
  });

  // 4. Signer le paquet entier avec la clé Ed25519 permanente
  pkt.signature = ed25519.sign(pkt.hash(), myKeyPair.privateKey);
  return pkt;
}
```

ANTI-PATTERN Ne jamais réutiliser un nonce avec la même clé AES-GCM. Jamais. Utiliser un compteur de séquence + aléatoire ou un nonce 96-bit full random.

Primitive	Usage	Bibliothèque recommandée
Ed25519	Signature / identité nœud	libsodium / tweetnacl / cryptography (Python)
X25519	Échange de clé Diffie-Hellman	libsodium / PyNaCl
AES-256-GCM	Chiffrement symétrique des données	Node crypto / PyCryptodome
HKDF-SHA256	Dérivation de clé de session	hkdf / hashlib
HMAC-SHA256	Intégrité des paquets	Built-in dans tous les langages
SHA-256	Hash des fichiers / chunks	Built-in

LIVRABLE S2 Démo : Alice envoie un message chiffré à Bob. Capture réseau (Wireshark) montre uniquement des octets chiffrés. Commit tagué 'sprint-2'.

SPRINT 3 — CHUNKING & TRANSFERT DE FICHIERS MULTI-NŒUDS

H+13 → H+19 (6h)

Objectifs Sprint 3

Implémenter le transfert de fichiers à la manière BitTorrent : segmentation en chunks, distribution multi-sources, vérification d'intégrité, réassemblage. Le fichier de démo : 50 Mo minimum.

Module 3.1 — Manifest de fichier

Avant tout transfert, l'émetteur génère et broadcast un Manifest :

```
MANIFEST structure (JSON encodé puis chiffré dans paquet TYPE.MANIFEST)
{
  "file_id"    : "sha256 du fichier entier",
  "filename"   : "rapport_confidentiel.pdf",
  "size"       : 52428800,
  "chunk_size" : 524288,   // 512 KB par chunk
  "nb_chunks"  : 100,
  "chunks"     : [
    { "index": 0, "hash": "sha256_chunk_0", "size": 524288 },
    { "index": 1, "hash": "sha256_chunk_1", "size": 524288 },
    // ...
    { "index": 99, "hash": "sha256_chunk_99", "size": 524288 }
  ],
  "sender_id" : "ed25519_public_key_hex",
  "signature" : "sig_ed25519_sur_manifest_hash"
}
```

Module 3.2 — Stratégie de téléchargement

- **Rarest First :** télécharger en priorité les chunks les plus rares sur le réseau (maximise la disponibilité globale).
- **Pipeline parallèle :** télécharger simultanément depuis plusieurs nœuds (minimum 3 connexions parallèles).
- **Vérification :** après réception de chaque chunk, vérifier son SHA-256. Chunk corrompu = re-demande à un autre pair.
- **Fallback :** si un pair devient injoignable en cours de transfert, reprendre les chunks manquants auprès d'un autre.

Module 3.3 — Protocole de transfert de chunk

```
CHUNK_REQ paquet:
  file_id    : bytes[32]   // hash SHA-256 du fichier
  chunk_idx  : uint32      // index du chunk demandé
  requester  : bytes[32]   // node_id du demandeur

CHUNK_DATA paquet (réponse, chiffré avec session key):
  file_id    : bytes[32]
  chunk_idx  : uint32
  data       : bytes[]     // données du chunk chiffrées
  chunk_hash : bytes[32]   // SHA-256 pour vérification
  signature  : bytes[64]   // signature Ed25519 du fournisseur

ACK paquet:
  chunk_idx  : uint32
  status     : 0x00 (OK) | 0x01 (HASH_MISMATCH) | 0x02 (NOT_FOUND)
```

Module 3.4 — Stockage et réplication

- **Stockage local :** chaque nœud conserve les chunks qu'il a téléchargés et les partage automatiquement.
- **Index local :** fichier .archipel/index.db (SQLite ou JSON) listant les chunks disponibles localement avec leurs métadonnées.
- **Réplication passive :** option de replication factor configurable. Un nœud peut s'engager à héberger N copies d'un fichier.

Scénario de test	Attendu	Critère de validation
Transfert 50 Mo sur 2 nœuds	< 120 secondes sur LAN 100 Mbps	Fichier SHA-256 identique à source
1 nœud déconnecté en cours	Transfert continue via autre pair	Aucune corruption de données
3 nœuds, 1 source → 2 receveurs	Transfert simultané fonctionnel	Les 2 fichiers arrivent intacts
Chunk corrompu simulé	Détection + re-téléchargement	Erreur loggée, fichier final correct

LIVRABLE S3 Transfert d'un fichier 50 Mo entre 3 nœuds. Vérification SHA-256 en live. Simulation de déconnexion d'un nœud. Commit tagué 'sprint-3'.

SPRINT 4 — INTÉGRATION, INTERFACE & ASSISTANT GEMINI

H+19 → H+23 (4h)

Objectifs Sprint 4

Assembler toutes les pièces en un prototype cohérent. Ajouter l'interface utilisateur minimale pour la démo et intégrer l'assistant IA Gemini en mode contextuel.

Module 4.1 — Interface de démo (CLI ou UI)

Choisir et implémenter une interface qui permette de démontrer l'ensemble des fonctionnalités devant le jury :

- **CLI interactif (recommandé en 4h) :** commandes simples, output clair, pas de dépendance lourde
- **UI Web légère (ambitieux) :** serveur local HTTP sur le nœud, React ou HTML vanilla

```
// Commandes CLI minimales requises pour la démo

archipel start --port 7777          // Démarrer le nœud
archipel peers                       // Lister les pairs découverts
archipel msg <node_id> 'Hello!'      // Envoyer un message chiffré
archipel send <node_id> <filepath>   // Envoyer un fichier
archipel receive                     // Voir les fichiers disponibles
archipel download <file_id>          // Télécharger un fichier
archipel status                      // État du nœud + stats réseau
archipel trust <node_id>             // Approuver un pair (Web of Trust)
```

Module 4.2 — Intégration Gemini API

IMPORTANT Gemini est la seule connexion externe autorisée. Elle doit être clairement isolée dans le code et désactivable via flag --no-ai pour les tests offline.

- **Déclenchement :** intégration dans le chat par tag @archipel-ai ou /ask
- **Contexte :** les derniers N messages du fil de discussion sont envoyés comme contexte à Gemini
- **Fallback :** si Gemini inaccessible (mode offline strict), afficher message d'erreur gracieux sans crash

```
// Exemple d'intégration Gemini
async function queryGemini(conversationContext, userQuery) {
  const response = await fetch(
    `https://generativelanguage.googleapis.com/v1beta/models/gemini-
pro:generateContent?key=${API_KEY}`,
    {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({
        contents: [
          { role: 'user', parts: [{ text: buildPrompt(conversationContext, userQuery) }] }
        ]
      })
    }
  );
  return response.json();
}
```

Module 4.3 — README.md final

Le README est noté par le jury. Il doit contenir obligatoirement :

- Description du protocole Archipel implémenté (architecture, choix techniques)
- Schéma d'architecture (ASCII art ou image)
- Instructions d'installation et de lancement (step-by-step, reproductibles)
- Guide de la démo (commandes exactes pour reproduire les 3 cas d'usage)
- Description des primitives cryptographiques utilisées et justification
- Limitations connues et pistes d'amélioration
- Membres de l'équipe et contributions respectives

> **LIVRABLE S4** Démo end-to-end en < 5 minutes devant le jury. CLI fonctionnel. README complet. Commit tagué 'sprint-4'.

SPRINT BUFFER — FINALISATION & SOUMISSION DEVPOST

H+23 → H+24 (2h)

Lien de soumission du projet sur Devpost : https://hack-days-build-in-public.devpost.com/

La soumission devra obligatoirement inclure :

- Le lien direct vers leur dépôt GitHub (**dépôt public**), servant de preuve du travail réalisé.
- Une brève description (300 mots maximum) du protocole Archipel implémenté.

Un fichier `README.md` complet et bien structuré dans le dépôt GitHub, détaillant l'architecture du protocole, les choix technologiques et les instructions claires pour son exécution et sa démonstration.

Checklist de soumission

- Vérification que tous les tests passent sur une machine fraîche
- README.md relu et corrigé par tous les membres
- Dernier commit tagué 'final-submission'
- Soumission DevPost complète avec lien GitHub + description 300 mots
- Préparer la démo jury : scénario clair, backup en cas de panne réseau

Pièges à éviter absolument

ANTI-PATTERN 1 Stocker des clés privées en clair dans le code ou dans le repo Git. Utiliser des variables d'environnement ou un keystore local.

ANTI-PATTERN 2 Implémenter son propre algorithme de chiffrement. Utiliser UNIQUEMENT des primitives éprouvées (libsodium, OpenSSL, etc.).

ANTI-PATTERN 3 Réutiliser des nonces ou des clés de session. Chaque session = nouvelles clés éphémères. Chaque message = nouveau nonce.

ANTI-PATTERN 4 Zéro commit GitHub avant la soumission. Le jury vérifie l'historique des commits. Commitez régulièrement.

5. RESSOURCES TECHNIQUES RECOMMANDÉES

Domaine	Ressource	URL / Package
Cryptographie (Node.js)	libsodium-wrappers	npm install libsodium-wrappers
Cryptographie (Python)	PyNaCl + PyCryptodome	pip install pynacl pycryptodome
Cryptographie (Go)	golang.org/x/crypto	go get golang.org/x/crypto
Réseau (Node.js)	dgram (UDP) + net (TCP)	Built-in Node.js
Réseau (Python)	socket + asyncio	Built-in Python
Noise Protocol	Spec de référence	noiseprotocol.org
BitTorrent BEP	BitTorrent Enhancement Proposals	bittorrent.org/beps/bep_0000.html
Gemini API	Google AI Studio	ai.google.dev

Debug réseau	Wireshark	wireshark.org

Structure de repo suggérée

```
Archipel_Team_Name/
├── README.md              ← CRITIQUE pour la note
├── .env.example           ← Variables d'environnement (jamais .env dans git)
├── .gitignore             ← node_modules, .env, *.key, *.archipel
├── src/
│   ├── crypto/            ← Modules PKI, E2E, handshake
│   ├── network/           ← Découverte, TCP serveur/client, peer table
│   ├── transfer/          ← Chunking, manifest, download manager
│   ├── messaging/         ← Chat, intégration Gemini
│   └── cli/               ← Interface utilisateur
├── tests/                 ← Tests unitaires (bonus)
├── docs/
│   ├── protocol-spec.md   ← Spec technique du protocole
│   └── architecture.md    ← Schéma d'architecture
└── demo/                  ← Scripts de démo pour le jury
```

Bonne chance. Construisez quelque chose qui mérite de survivre.

Hackathon Archipel 2026 · 24 heures · Document Technique Participants