Projet PX511 :

One round tripartite non-interactive key exchange

Sommaire

I Problématique

En cryptographie, l'échange chiffré entre deux parties peut être réalisé grâce à une clé de session partagée, établie par l'algorithme de Diffie–Hellman (DH). En revanche, avec trois parties ou plus, un émetteur devrait, en l'absence d'un protocole adapté, établir des clés bilatérales distinctes avec chaque destinataire, ce qui multiplie le nombre de clés et de négociations. Dans ce projet, nous visons d'abord un échange de clé à trois participants, permettant à Alice, Bob et Charlie de dériver la même clé secrète K sur un canal non sécurisé. Le but de ce projet est de fournir un démonstrateur qui choisit des clés aléatoires et vérifie, du point de vue de chaque participant, que la clé obtenue est identique.

II État de l'art

1) Échange de clé Diffie–Hellman à trois participants

a. Résumé du document

L'article de Chunling Liu, Yufeng Wang et Qinxi Bai [LWB11] présente une extension à trois participants du schéma Diffie–Hellman (DH) classique, permettant à Alice, Bob et Charlie d'établir une clé secrète commune K sur un canal non sécurisé en deux étapes de protocole (aucune communication lors de l'étape 2). Le schéma conserve l'ossature arithmétique de DH et vise une faible latence de négociation.

Le schéma réduit la communication à deux étapes tout en conservant la structure mathématique de Diffie–Hellman.

Soit G un groupe cyclique d'ordre premier q avec générateur g. Chaque utilisateur $U \in \{A, B, C\}$ possède:
- une **clé longue durée** $x_U \in \mathbb{Z}_q^*$ (choisie uniformément dans $\{1, ..., q-1\}$),
- une **clé publique** $y_U = g^{x_U}$ (dans G; si l'on travaille modulo un grand premier p, on note $y_U \equiv g^{x_U} (\mathrm{mod}\, p)$).

b. Présentation du protocole

(i) Étape 1 — Échanges publics

À chaque exécution, chaque partie tire un exposant éphémère $a, b, c \in \mathbb{Z}_q$, qui sert de clé privée temporaire de session. Ces valeurs sont indépendantes des clés longues x_U et assurent la confidentialité persistante.

Chaque participant calcule et envoie aux deux autres des valeurs de type Diffie–Hellman :

$$A \to B : T_{\{AB\}} = y_B^{\{ax_A\}} = g^{\{ax_Ax_B\}},$$

$$A \to C : T_{\{AC\}} = y_C^{\{ax_A\}} = g^{\{ax_Ax_C\}},$$

$$B \to A : T_{\{BA\}} = y_A^{\{bx_B\}} = g^{\{bx_Bx_A\}},$$

$$B \to C : T_{\{BC\}} = y_C^{\{bx_B\}} = g^{\{bx_Bx_C\}},$$

$$C \to A : T_{\{CA\}} = y_A^{\{cx_C\}} = g^{\{cx_Cx_A\}},$$

$$C \to B : T_{\{CB\}} = y_B^{\{cx_C\}} = g^{\{cx_Cx_B\}}.$$

Chaque message combine la clé publique du destinataire avec la clé longue durée et l'éphémère de l'émetteur.

(ii) Étape 2 — Calcul local (sans communication)
Aucune communication n'est requise à cette étape: à la fin de l'étape 1 chaque partie détient déjà toutes les informations nécessaires pour dériver K localement.

Chaque participant retire son propre exposant long à l'aide de l'inverse modulaire $x_U^{-1} \pmod q$. L'opération est bien définie car $x_U \in \mathbb{Z}_q^*$ implique $\gcd(x_U, q) = 1$, et l'inverse se calcule efficacement par l'algorithme d'Euclide étendu.

Pour Alice :

$$g^{\{ax_A\}} = y_A^a,$$

$$\left(T_{\{BA\}}\right)^{\left\{x_A^{\{-1\}}\right\}} = g^{\{bx_B\}},$$

$$\left(T_{\{CA\}}\right)^{\left\{x_A^{\{-1\}}\right\}} = g^{\{cx_C\}}$$

Ainsi, la clé de session d'Alice est :

$$K_A = g^{\{ax_A\}}, g^{\{bx_B\}}, g^{\{cx_C\}} = g^{\{ax_A + bx_B + cx_C\}}$$

Bob et Charlie effectuent les mêmes calculs et obtiennent :

$$K_B = K_C = K_A = g^{\{ax_A + bx_B + cx_C\}}$$

Ainsi, les trois utilisateurs dérivent la même clé secrète K sans aucun échange supplémentaire.

(iii) Étape 3 – Confirmation de clé (optionnel)
Pour une sécurité pratique, une troisième étape peut être ajoutée pour assurer la **confirmation de la clé**. Chaque partie envoie un HMAC calculé avec K comme clé sur un transcript public. La vérification croisée confirme que tous ont dérivé la même clé et mitige les attaques actives (MITM) si combinée à une authentification des émetteurs.

Nous nous appuyons sur Liu, Wang, Bai (2011) [LWB11], qui proposent un échange de clé Diffie–Hellman à trois participants et en deux étapes (sans serveur), avec confidentialité persistante.

c. Résultat
Ci-dessous, un exemple d'exécution de la mise en œuvre Python (Annexe 1) montrant que $K_A = K_B = K_C$ et que les HMACs se vérifient mutuellement.

```
A public y_A = 1554
B public y_B = 1925
A public y_C = 1045

Clés de session :
  K_A =  777
  K_B =  777
  K_C =  777

Tags (hex):  {'A':
'4b9bf5dedb0774ffa1d50dddbc0ffca14b3e0f315901f87450ea1af5b776318b', 'B':
'eb6e7634f3b6186d13da0b65962380d280f5a2c9c9e33d129e6d04ce33d86ff3', 'C':
'cb371fd9a0508058a7f37cb30f96f8ad6d7ecbf3600e956acce9d921c9543420'}

Key Confirmation OK ? -> True
```

Ce schéma convient au cadre académique (il est simple, reproductible, et permet une implémentation "sans librairie"). Cependant, nous ne le retenons pas comme solution principale car il nécessite deux étapes et six

messages (donc une latence et un coût de bande passante supérieurs) et n'est pas non-interactif. Or, l'objectif central du projet est d'illustrer un échange "one-round / non-interactif" : pour cela, nous privilégions le NIKE tripartite basé sur couplage (type Joux [Jou04]), en gardant ce DH 2-étapes comme référence de base et comme démonstrateur "sans librairie".

2) Three-party NIKE (Non Interactive Key Exchange)

a. Courbes elliptiques

Une courbe elliptique est un cas particulier de courbe algébrique. Elles peuvent être représentées dans un plan par une équation sous la forme :

$$y^2 + a_1x + a_3y = x^3 + a_2x^2 + a_4x + a_6$$

Dans le cas de la cryptographie, les courbes elliptiques utilisées sont de la forme :

$$y^2 = x^3 + ax + b$$

Les plus courantes sont les NIST P-256 et X25519.

Dans le cas de la cryptographie à base de couplage, les courbes utilisées sont généralement, les BN (Barreto–Naehrig) qui ont été spécialement construites pour optimiser ces trois critères :

- sécurité (128 bits)
- efficacité du couplage
- facilité d'implémentation

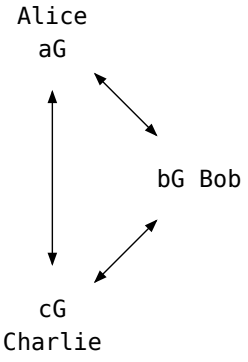
b. Cryptographie à base de couplages

Un couplage est une application $e : G_1 \times G_2 \to G_T$ qui a les propriétés suivantes:

- Bilinéarité : $\forall (x; y) \in G_1, G_2, \forall a, b \in \mathbb{Z} e(ax; by) = e(x; y)^{ab}$
- Non-dégénérescence : $e(x; y) = 1 \Leftrightarrow x = 1$ ou $y = 1$ ($e \neq 1$)

On parle de couplage symétrique lorsque $G_1 = G_2$

c. Couplage basique (Three-party)



(i) Théorie

Considérons 3 personnes, Alice, bob et Charlie qui transmettent leur clef publique aux deux autres.

Alice peut donc calculer $e(bG, cG)^a = e(G, G)^{abc}$
Bob peut donc calculer $e(aG, cG)^b = e(G, G)^{abc}$
Charlie peut donc calculer $e(aG, bG)^c = e(G, G)^{abc}$

Alice, Bob et Charlie possèdent les secrets $a, b, c \in \{\mathbb{Z}\}_q$ et publient leurs clés aG, bG, cG.

Alice reçoit bG et cG.

Elle utilise son secret a : $K_A = e(bG, cG)^a$

Par bilinéarité, $e(bG, cG) = e(G, G)^{\{bc\}}$.

Donc $K_A = \left(e(G, G)^{\{bc\}}\right)^a = e(G, G)^{\{abc\}}$

(ii) Weil

Le couplage de Weil est défini sur le r-torsion complet d'une courbe elliptique E : $e_{Weil} : E[r] \times E[r] \to \mu_r \subset \mathbb{F}_{p^k}^*$. Il provient de fonctions rationnelles dont les diviseurs sont multiples de r (idée : construire une fonction f_r, P de diviseur $r(P) - r(O)$ et l'évaluer "autour" de Q). Il est bilinéaire, alterné, et non dégénéré dès que les deux arguments appartiennent à $E[r]$. En pratique, on le calcule via l'algorithme de Miller (boucle de longueur $\simeq \log r$, composée d'additions/doublages et d'évaluations de droites) suivi d'une exponentiation de normalisation.

Sur le plan performance, le Weil est historiquement important mais plus coûteux : il exige typiquement que les deux points vivent dans l'extension $\mathbb{F}_{p^k}^*$ (ou sur le tordu) et la normalisation n'est pas plus simple que pour Tate/Ate. On le rencontre donc surtout à des fins pédagogiques ou pour des preuves. Pour notre NIKE tripartite, il suffit d'avoir un couplage bilinéaire non dégénéré ; le Weil fonctionne, mais n'est pas le meilleur choix en efficacité.

(iii) Tate

Le couplage de Tate (réduit) s'écrit $e_{Tate}(P, Q) = f_{r,P}(Q)^{\frac{p_k-1}{r}}$, avec $P \in E[r]$ et $Q \in E(\mathbb{F}_{p^k})$. L'évaluation brute $f_{r,P}(Q)$ (Miller) n'est définie qu'à une puissance r —ième près ; l'exponentiation finale $\frac{p_k-1}{r}$ projette dans μ_r et rend la valeur unique. Par rapport au Weil, le Tate est souvent plus rapide et plus souple : on place P et Q dans des sous-groupes G_1, G_2 convenables (Type-3 "asymétrique"), par exemple $P \in E(F_p)$ et Q sur un tordu défini sur $F_{p^{\frac{k}{d}}}$, ce qui réduit les coûts d'arithmétique.

Côté sécurité, la bilinéarité $e(aP, bQ) = e(P, Q)^{\{ab\}}$ et la non-dégénérescence tiennent sous des conditions standard : cofactor-clearing et choix d'un degré d'immersion k tel que $rm\,\mathrm{id}(p^k - 1)$. Le couplage de Tate couvre parfaitement l'usage NIKE tripartite de Joux (2000) [Jou04] — première construction "une ronde" basée sur un couplage ; on peut prendre $K_A = e(B_1, C_2)^a$, $K_B = e(C_1, A_2)^b$, $K_C = e(A_1, B_2)^c$ (voir aussi la référence ajoutée en bibliographie). Il reste le coût de l'exponentiation finale, souvent décomposée en partie "facile" (facteurs cyclotomiques) et "difficile" (exposant résiduel).

(iv) Ate

Le couplage d'Ate accélère la phase Miller en remplaçant le paramètre r par un petit entier lié à la trace de Frobenius. Sur une courbe ordinaire (BN, BLS), si t est la trace, on pose $T = t - 1$ et on définit (schématiquement) $e_{Ate}(Q, P) = f_{T,Q}(P)^{\frac{p^k-1}{r}}$. La boucle de Miller parcourt alors $\log T$ au lieu de $\log r$, ce qui apporte un gain net quand $\mid T \mid \ll r$ (cas des courbes "pairing-friendly"). Les arguments sont "inversés" par rapport à Tate (on évalue une fonction attachée à Q sur P), mais l'objet final vit toujours dans μ_r.

Des variantes modernes — R-ate, optimal Ate, optimal-Ate sur BN/BLS12 — combinent plusieurs itérations liées au Frobenius pour raccourcir encore la boucle, tout en conservant la même exponentiation finale (optimisée par décomposition algébrique). En pratique, les bibliothèques actuelles (bplib, mcl, relic, etc.) implémentent un optimal-Ate Type-3 sur BN/BLS, qui est aujourd'hui le meilleur compromis performance/simplicité. Pour notre NIKE, l'Ate (ou optimal-Ate) est le choix recommandé : même garantie de bilinéarité/non-dégénérescence, mais temps de calcul sensiblement moindre que Tate/Weil.

(v) Application

Pour illustrer le NIKE tripartite basé sur couplage, nous exécutons le prototype dont le code source est donné en (Annexe 2). Le bloc ci-dessous montre la sortie typique d'une exécution (égalité des clés dans G_T et clé de session exportée en hexadécimal) :

```
[*] Group order (bits): 254
a = 97959392386571372070014149464232683189843674598595383663734639609015029054l5
b = 10134203465698003418838719594144630324906411920066739450740846534179623762673
c = 98247287285094207950795992295402612045970855322749823744344032707683709432l0

[*] Checking equality in GT...
```

```
K_A == K_B ? True
K_B == K_C ? True
K_A == K_C ? True

[+] Success: all three parties share the same pairing-based key.

Session key (GT element, hex):

151180a480e4707008b7500415d1a1de3cde60b10c0fac459fb7c4fb8f3d906013be043c9f5f18eb9433d3
139914096bf366d71ee8846e4eaad276a33218b2dc0a42a9743ee0f618a0f3519aa19cb0033eade8aa5ed5
1dbc25baa0754101633d139cbab51138833bcdf5a28fff434d558f3944d688f197e87945407d112c4c7514
777a3873fa4cd93f89ea516d7395adae64a5b4f08eb910a2fe6d883c4833b015b4e106d090a553cbc8f32c
a0846c2750be8c21df65ce3249e83541151b72f212ff9a46051e0914648324ddf8b662de29cf419406209c
a1f9a0fc7b5a45db680ff00d378d4c0fc65a62321a6fededce0c1cfcec27bd65a2d67cca2b33f937ee2017
57c5dcf2ce7778d7e44b275a6ccb1496ea5322faa08c336a6a4b5033b2380c96c2455d4b98f1b298769ef5
25ca780f5c140ca1fe0704e767d3fdc4c776f31c2d6ac4d4e9b403b488df6438d5a9de1244dad08adff8bb
23cd3c2ec7cfd1f201f78a94801381f3a5a9d0039696074870c28cc4f96f455612bea97772ecc6a3
```

III Pistes : applications multilinéaires cryptographiques (> 3 personnes)

Dans cette troisième partie, nous verrons quelques pistes concernant un cas non obligatoire mais intéressant à étudier : l'extension du protocole à quatre participants. Ce sujet reste en dehors du cadre principal du projet, mais on pourra l'explorer si le travail principal est terminé.
En annexe (Annexe 3) on retrouve un tableau proposant plusieurs pistes pour cette partie.

1) Les variantes récentes de GGH15

Parmi les constructions héritées de la lignée GGH13–GGH15, la variante proposée par Bartusek, Guan, Ma et Zhandry (2018) constitue l'une des approches les plus prometteuses pour obtenir un schéma d'échange de clé non interactif multiparti réellement robuste. Contrairement aux versions originelles GGH13 et GGH15 — aujourd'hui considérées comme brisées en raison d'attaques "zeroizing" et de la rupture des hypothèses MDDH — cette variante introduit des mécanismes d'aléatorisation supplémentaires et s'appuie sur des hypothèses de sécurité plus structurelles, comme la Branching-Program Un-Annihilatability (BPUA). Bien que ces hypothèses soient fortes et encore peu étudiées, elles permettent d'établir des preuves de résistance contre toutes les attaques connues visant les maps multilinéaires basées lattices. En ce sens, cette famille représente la première tentative sérieuse de fournir une alternative « réparée » à GGH15, tout en conservant la capacité essentielle de réaliser un NIKE multiparti.

2) Les constructions basées sur iO

Les travaux d'Albrecht et al. (2020) basés sur l'obfuscation indistinguable (iO) constituent une autre piste prometteuse, bien que largement impraticable aujourd'hui. Cette approche ne dépend pas des mécanismes traditionnels des graded encodings et échappe donc aux vulnérabilités qui ont conduit à casser GGH13, GGH15 ou CLT13. En construisant des maps multilinéaires à partir d'iO, de NIZK dual-mode et parfois de FHE, les auteurs parviennent à obtenir une classe de maps pour lesquelles une version multilinéaire de DDH tient par construction. Ceci rend ces systèmes extrêmement robustes en théorie, sans vulnérabilités structurelles connues. Leur principal défaut réside dans les coûts prohibitifs de l'iO, ce qui limite tout usage pratique. Cependant, du point de vue de la recherche, ces modèles représentent probablement la direction la plus durable pour obtenir des NIKE multiparti pleinement prouvés et conceptuellement sûrs.

3) Les schémas plus récents et idéalistes (Salimi 2021, weak maps, etc.)

Enfin, les constructions plus récentes — telles que le GES de Salimi (2021) ou les versions à degré constant basées sur CLT13 proposées par Ma et Zhandry — méritent également l'attention. Elles cherchent principalement à contourner les attaques ayant compromis les graded encodings classiques en modifiant profondément les mécanismes de zéro-test ou en restreignant les fonctionnalités disponibles. Le schéma de Salimi élimine totalement les encodages de zéro test, souvent responsables des attaques zeroizing, et propose même un NIKE multiparti ou à identité. Cependant, aucun examen cryptanalytique poussé n'a encore été mené. Elles ne réparent donc pas réellement les failles connues : elles contournent plutôt les mécanismes vulnérables, mais au prix d'une grosse incertitude quant à leur solidité. Elles peuvent être vues comme prometteuses sur le plan conceptuel, mais ne peuvent pas encore rivaliser avec les constructions fondées sur iO en termes de garanties cryptographiques.

IV Objectifs pour la suite

Pour la suite du travail, plusieurs axes d'avancement sont prévus afin d'approfondir la compréhension et d'étendre les résultats obtenus jusqu'à présent.

- Analyser en profondeur les librairies bplib et petlib, actuellement utilisées comme boîtes noires, afin de comprendre leur fonctionnement interne et d'être capable de reproduire, voire de réimplémenter, leurs mécanismes de manière autonome.
- Étudier plus en détail les couplages de Weil, Tate et Ate et en approfondir la compréhension mathématique, qui nous permettront de coder le couplage et le pairing nous-même dans le démonstrateur.
- S'il reste du temps avant la fin du projet, envisager une extension du protocole à plus de trois participants (≥ 4) afin d'évaluer la généricité du schéma dans des scénarios multiparticipants.

V Gantt

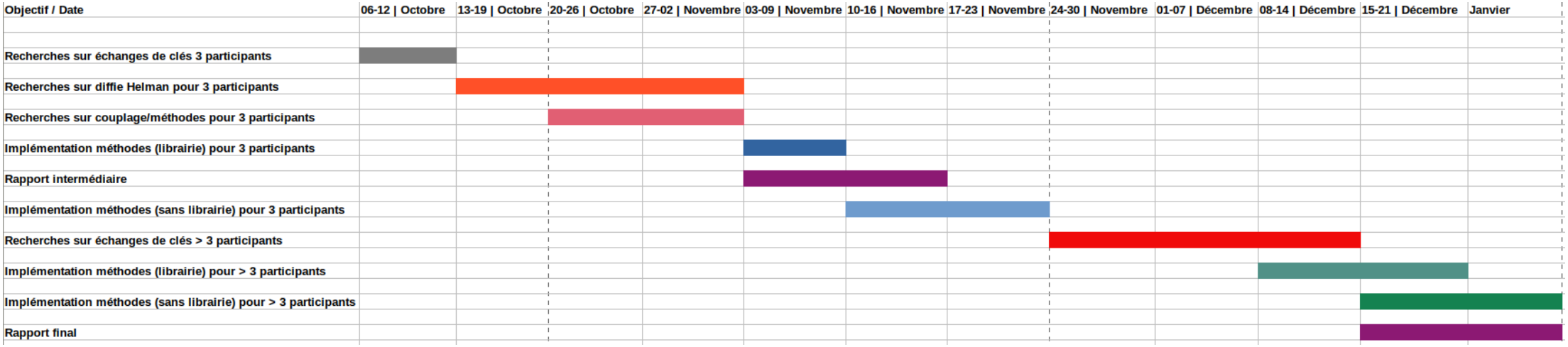


Fig. 1. – Le diagramme de GANT prévu pour notre projet.

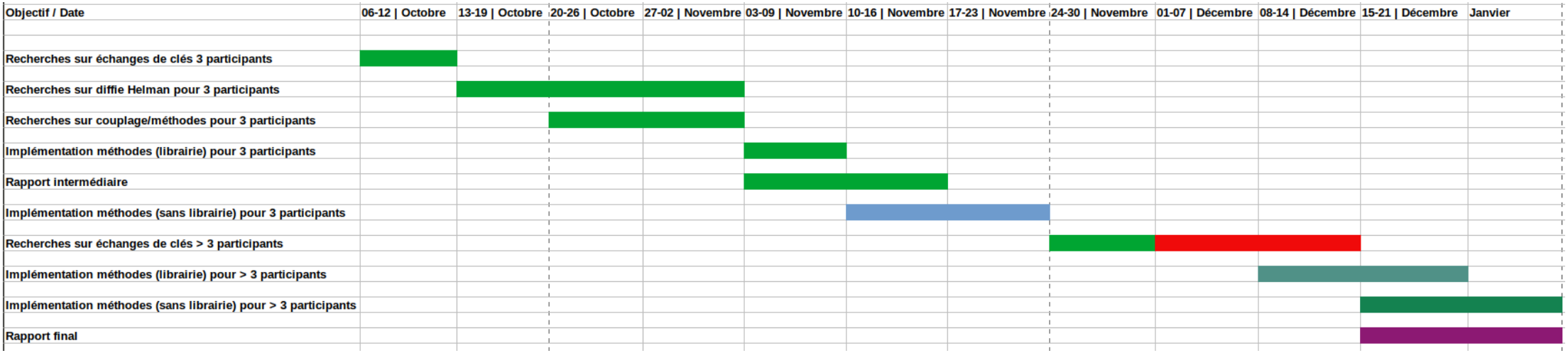


Fig. 2. – Le diagramme de GANT avec les taches réalisées en vert au jour du rapport intermédiaire.

VI Annexes

1) CODE - Échange de clé Diffie–Hellman à trois participants

```py
import secrets
from math import gcd
import hmac, hashlib

# Paramètres de démonstration (en pratique choisir un p grand et sûr)
p = 2089      # p premier -> (Z/pZ)* est cyclique d'ordre q = p-1
g = 2         # générateur (démonstration)
q = p - 1     # ordre du groupe multiplicatif - l'arithmétique des exposants = mod q

def rand_nonzero(modulus: int) -> int:
    """Tire uniformément un entier dans {1,2,...,modulus-1} (non nul)"""
    while True:
        x = secrets.randbelow(modulus)
        if x != 0:
            return x

def rand_coprime(modulus: int) -> int:
    """Tire un entier 1..modulus-1 premier avec 'modulus' (pour garantir l'inverse modulaire)"""
    while True:
        x = rand_nonzero(modulus)
        if gcd(x, modulus) == 1:
            return x

def modinv(a: int, m: int) -> int:
    """Inverse modulaire de a (mod m). Lève ValueError s'il n'existe pas"""
    return pow(a, -1, m)

def int_to_bytes(x: int) -> bytes:
    """Conversion d'un entier non négatif vers une chaîne d'octets big-endian minimale"""
    if x == 0:
        return b"\x00"
    return x.to_bytes((x.bit_length() + 7)//8, "big")

class User:
    def __init__(self, name: str, x: int, y: int):
        self.name = name   # identifiant textuel (utilisé dans la balise HMAC)
        self.x = x         # clé longue durée (privée)
        self.y = y         # clé publique y = g^x mod p

def setup_user(name: str) -> User:
    """
    Génère une paire de clés longue durée
    Exige x copremier avec q pour assurer l'existence de x^{-1} (mod q)
    """
    x = rand_coprime(q)
    y = pow(g, x % q, p)
    return User(name, x, y)

# HMAC-SHA256 - confirmation
def compute_tag(name: str, K: int, transcript: bytes) -> str:
    """
```

```
    Calcule une balise d'authentification: tag = HMAC-SHA256( key = K (en octets), msg = name | transcript )
    Le 'transcript' regroupe les éléments publics échangés pendant l'étape 1
    """
    key = int_to_bytes(K % p)
    msg = name.encode("utf-8") + b"|" + transcript
    return hmac.new(key, msg=msg, digestmod=hashlib.sha256).hexdigest()

def verify_tag(name: str, K: int, transcript: bytes, tag_hex: str) -> bool:
    """Vérifie une balise HMAC attendue pour (name, K, transcript)."""
    exp = compute_tag(name, K, transcript)
    return hmac.compare_digest(exp, tag_hex)

def three_party_dh_with_confirmation(A: User, B: User, C: User):
    """
    Étape 1 : chaque partie choisit un exposant éphémère (a,b,c) et envoie ses T_..
    Étape 2 : chacun calcule localement la clé K
    Confirmation : HMAC-SHA256 avec K sur le transcript public
    Retourne : (K_A, K_B, K_C, tags, all_ok)
    """
    # Étape 1
    a = rand_nonzero(q)  # exposants éphémères (secrets temporaires)
    b = rand_nonzero(q)
    c = rand_nonzero(q)

    # Messages
    # A -> B, A -> C
    T_AB = pow(B.y, (a * A.x) % q, p)
    T_AC = pow(C.y, (a * A.x) % q, p)
    # B -> A, B -> C
    T_BA = pow(A.y, (b * B.x) % q, p)
    T_BC = pow(C.y, (b * B.x) % q, p)
    # C -> A, C -> B
    T_CA = pow(A.y, (c * C.x) % q, p)
    T_CB = pow(B.y, (c * C.x) % q, p)

    # Étape 2
    inv_xA = modinv(A.x % q, q)
    inv_xB = modinv(B.x % q, q)
    inv_xC = modinv(C.x % q, q)

    # Alice :
    g_axA_A = pow(A.y, a % q, p)        # g^{a x_A}
    g_bxB_A = pow(T_BA, inv_xA, p)      # (g^{b x_B x_A})^{x_A^{-1}} = g^{b x_B}
    g_cxC_A = pow(T_CA, inv_xA, p)      # g^{c x_C}
    K_A = (g_axA_A * g_bxB_A * g_cxC_A) % p

    # Bob :
    g_axA_B = pow(T_AB, inv_xB, p)
    g_bxB_B = pow(B.y, b % q, p)
    g_cxC_B = pow(T_CB, inv_xB, p)
    K_B = (g_axA_B * g_bxB_B * g_cxC_B) % p

    # Charlie :
    g_axA_C = pow(T_AC, inv_xC, p)
```

```
    g_bxB_C = pow(T_BC, inv_xC, p)
    g_cxC_C = pow(C.y, c % q, p)
    K_C = (g_axA_C * g_bxB_C * g_cxC_C) % p

    # Confirmation de clé
    transcript = (
        f"p={p},g={g},q={q},yA={A.y},yB={B.y},yC={C.y},"
        f"T_AB={T_AB},T_AC={T_AC},T_BA={T_BA},T_BC={T_BC},T_CA={T_CA},T_CB={T_CB}"
    ).encode("utf-8")

    tag_A = compute_tag(A.name, K_A, transcript)
    tag_B = compute_tag(B.name, K_B, transcript)
    tag_C = compute_tag(C.name, K_C, transcript)

    # Vérifications croisées
    A_ok = verify_tag(B.name, K_A, transcript, tag_B) and verify_tag(C.name, K_A, transcript, tag_C)
    B_ok = verify_tag(A.name, K_B, transcript, tag_A) and verify_tag(C.name, K_B, transcript, tag_C)
    C_ok = verify_tag(A.name, K_C, transcript, tag_A) and verify_tag(B.name, K_C, transcript, tag_B)
    all_ok = A_ok and B_ok and C_ok

    tags = {"A": tag_A, "B": tag_B, "C": tag_C}
    return K_A, K_B, K_C, tags, all_ok

if __name__ == "__main__":
    # Clés longue durée
    A = setup_user("Alice")
    B = setup_user("Bob")
    C = setup_user("Charlie")

    K_A, K_B, K_C, tags, ok = three_party_dh_with_confirmation(A, B, C)

    print("A public y_A =", A.y)
    print("B public y_B =", B.y)
    print("A public y_C =", C.y)
    print()
    print("Clés de session :")
    print("\tK_A = ", K_A)
    print("\tK_B = ", K_B)
    print("\tK_C = ", K_C)
    print()
    print("Tags (hex): ", tags)
    print()
    print("Key Confirmation OK ? ->", ok)
```

2) CODE - Three-party NIKE

```py
from bplib import bp
from petlib.bn import Bn

def random_scalar(order: Bn) -> Bn:
    """Return a random scalar in [1, order-1]."""
    rnd = order.random()
    if rnd == 0:
        return random_scalar(order)
    return rnd
```

```python

def main():
    # --- Setup bilinear pairing group ---
    G = bp.BpGroup()
    g1, g2 = G.gen1(), G.gen2()
    order = G.order()

    print("[*] Group order (bits):", order.num_bits())

    # Secrets for Alice, Bob, Charlie (Bn)
    a = random_scalar(order)
    b = random_scalar(order)
    c = random_scalar(order)

    print(f"a = {int(a)}")
    print(f"b = {int(b)}")
    print(f"c = {int(c)}")

    # Public points
    A1 = g1 * a
    B1 = g1 * b
    C1 = g1 * c

    A2 = g2 * a
    B2 = g2 * b
    C2 = g2 * c

    # Alice: K_A = e(B1, C2)^a
    e_BC = G.pair(B1, C2)
    K_A = e_BC ** a

    # Bob: K_B = e(C1, A2)^b
    e_CA = G.pair(C1, A2)
    K_B = e_CA ** b

    # Charlie: K_C = e(A1, B2)^c
    e_AB = G.pair(A1, B2)
    K_C = e_AB ** c

    print("\n[*] Checking equality in GT...")
    print("K_A == K_B ?", K_A == K_B)
    print("K_B == K_C ?", K_B == K_C)
    print("K_A == K_C ?", K_A == K_C)

    if K_A == K_B == K_C:
        print("\n[+] Success: all three parties share the same pairing-based key.")
    else:
        print("\n[-] Keys do not match. Something went wrong.")

    key_bytes = K_A.export()
    print("\nSession key (GT element, hex):")
    print(key_bytes.hex())

if __name__ == "__main__":
    main()
```

3) Tableau : Applications multilinéaires cryptographiques (> 3 personnes)

Approche	Année	Auteurs	Hypothèses / Fondements	Forces	Faiblesses	NIKE
CLT13 (Coron et al.)	2013	Coron, Lepoint, Tibouchi	Encodages d'entiers à ordre composite (style DGHV) ; difficulté supposée de type "DLIN multilinéaire".	Pratique : démonstration d'un DH à 7 participants en ≈40 s. Conception à base d'entiers.	Cassé : l'hypothèse DDH multilinéaire (MDDH) a été rompue par Cheon **et al.** (2015). La version CLT15 a également été cassée.	Oui (conçu pour du DH multiparti)
GGH13 (Garg–Gentry–Halevi)	2013	Garg, Gentry, Halevi	Dureté de réseaux idéaux (style NTRU) : encodages gradués basés lattices.	Premier schéma de "graded map" à base de lattices ; conceptuellement général ; un exemple d'échange de clé existe.	Cassé : la dureté MDDH rompue par Hu & Jia (2016) ; autres attaques "zeroizing" (Albrecht **et al.**, Cheon **et al.**).	Oui (le schéma supportait le DH multiparti)
GGH15 (Garg–Gentry–Halevi)	2015	Garg, Gentry, Halevi	Encodages multilinéaires induits par des graphes sur des instances LWE.	Nouveau schéma asymétrique basé LWE ; structure plus riche pour l'obfuscation et l'ABE.	Cassé : la dureté MDDH rompue par Albrecht **et al.** (CLLT16).	Oui (support multiparty DH, similaire à GGH13)
Variante GGH15 de Bartusek et al.	2018	Bartusek, Guan, Ma, Zhandry	Variante de GGH15 avec davantage d'aléa ; prouvé sûr sous les hypothèses Branching-Program Un-Annihilatability et hypothèses associées.	Résiste de manière prouvée à toutes les attaques "zeroizing" publiées (analyse dans un nouveau modèle).	Repose sur des hypothèses fortes/non établies (BPUA, PRF dans NC^1, etc.) ; sécurité garantie seulement dans un modèle idéalisé.	Oui (map utilisable pour l'échange de clé, même si le focus est l'obfuscation)
Salimi 2021 GES	2021	Majid Salimi	Nouveau schéma d'encodages gradués où les utilisateurs encodent des éléments secrets aléatoires.	Élimine les encodages de zéro test publiés ; construction efficace et un MP-NIKE (même ID-based).	Tout récent, conception heuristique ; aucune preuve formelle ni réduction connue (non vérifié).	Oui (schéma MP-NIKE / ID-NIKE explicite proposé)
CLT13 degré constant (Ma–Zhandry)	2018	Fermi Ma, Mark Zhandry	Basé sur CLT13 dans un modèle de "weak map", avec de fortes hypothèses sur la complexité algébrique.	Premier encodage gradué de degré constant avec sécurité idéale dans le modèle ; aucune attaque connue sur sa fonctionnalité limitée.	Fonctionnalité limitée (pas de réutilisation avec rerandomisation complète) ; repose sur des hypothèses algébriques non prouvées.	Oui (extension de CLT13, donc supporte le DH multiparti)
Maps basées iO (Albrecht et al.)	2020	Albrecht, Farshim, Han, Hofheinz, Larraia, Paterson	Construit des maps multilinéaires à partir d'iO probabiliste, NIZK dual-mode, FHE, etc.	Construction complètement prouvée ; les analogues DDH multilinéaires tiennent par conception	Impraticable : nécessite des primitives très lourdes (iO, NIZK, FHE) ; κ doit être choisi à l'avance.	Oui (κ-DDH multilinéaire, permettant le NIKE classique)

Bibliographie

[LWB11] C. Liu, Y. Wang, et Q. Bai, « A New Three-party Key Exchange Protocol Based on Diffie-Hellman », *International Journal of Wireless and Microwave Technologies*, vol. 1, p. 65-69, 2011, doi: 10.5815/ijwmt.2011.04.09.

[Jou04] A. Joux, « A One Round Protocol for Tripartite Diffie–Hellman », *Journal of Cryptology*, vol. 17, nº 4, p. 263-276, sept. 2004, doi: 10.1007/s00145-004-0312-y.

[3] J. Bartusek, J. Guan, F. Ma, et M. Zhandry, « Return of GGH15: Provable Security Against Zeroizing Attacks », in *Theory of Cryptography*, A. Beimel et S. Dziembowski, Éd., Cham: Springer International Publishing, 2018, p. 544-574.

[4] A. Joux, « The Weil and Tate Pairings as Building Blocks for Public Key Cryptosystems », in *Algorithmic Number Theory*, C. Fieker et D. R. Kohel, Éd., Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, p. 20-32.

[5] M. Abou Hashish, « Applications trilinéaires alternées et courbes cubiques elliptiques généralisées. Classification et utilisations cryptographiques », Doctoral dissertation, 2003. [En ligne]. Disponible sur: http://www.theses.fr/2003ISAT0004

[6] apgoucher, « Barreto-Naehrig curves and cryptographic pairings », déc. 2020, [En ligne]. Disponible sur: https://cp4space.hatsya.com/2020/12/27/barreto-naehrig-curves-and-cryptographic-pairings/