

Introduction à la sécurité

Sujets de projets 2025–26

Consignes générales.

1. Vous devez constituer des groupes formés de 4 à 6 étudiant·es **avant le dimanche 28 septembre, à 23h59**.
2. La **constitution des groupes** sera envoyée par un email, envoyé à la fois à M. Rauzy et M. Lavauzelle, **avec en copie tous les membres du groupe**. Dans le corps de l'email, vous listerez les noms de tous les membres du groupe, puis par ordre de préférence l'ensemble des titres des sujets de projet (voir pages suivantes). Un projet sera assigné à votre groupe lors de la séance de cours suivante.
3. L'**évaluation** se fera *via* :
 - (a) le rendu d'un rapport (au format **.pdf**, compilé avec LaTeX);
 - (b) une soutenance, en groupe, durant laquelle vous présenterez l'ensemble de vos travaux et répondrez à nos questions.
4. Pour le **rapport**, il faudra impérativement :
 - indiquer un lien vers le dépôt **gitlab** qui entrepose votre implémentation;
 - expliquer les fonctionnalités qui ont été implémentées, ainsi que les choix d'implémentation qui ont été faits;
 - préciser **qui** a participé à l'implémentation de quoi;
 - ne pas oublier une courte introduction (quel est le but de ce projet?) et une conclusion (quels sont les défauts, qu'est ce qui aurait pu être fait en plus?).

Le rapport devra être rendu pour le **vendredi 6 décembre 2024**.

5. Concernant **dépôt** du projet sur **gitlab**, il faudra :
 - créer un dépôt sur la forge <https://code.up8.edu/>, et ajouter tous les membres du groupe comme contributeurs;
 - nous ajouter comme membre du projet (en tant que non-développeur), ou bien plus simplement rendre le dépôt public;
 - et bien sûr, alimenter régulièrement ce dépôt. Nous prendrons garde à ce que **chaque membre du groupe ait participé de manière significative** au développement de votre solution.
6. Pour la **soutenance**, le format n'est pas encore précisé, mais **tous les membres du groupe devront impérativement être présents**. Une démonstration du travail sera exigée; elle illustrera plusieurs fonctionnalités de votre implémentation, y compris la sécurité face à un utilisateur malhonnête. Les dates de soutenance ne sont pas encore fixées, mais se feront certainement durant la dernière semaine de cours de décembre 2025.

Sujet 1 : messagerie instantanée sécurisée

Objectif. Le but de ce projet est d'implanter un mini-chat (messagerie instantanée minimaliste) qui permet à un ensemble de personnes de s'échanger des messages de manière sécurisée.

Fonctionnalités minimales attendues. Les messages échangés seront **chiffrés « de bout en bout »**, c'est-à-dire qu'aucun message ne sera diffusé en clair sur le canal de transmission, ou stocké en clair sur le serveur.

1. Mise en place d'une **application client** (utilisée par un usager de la messagerie) qui permet de :
 - **initier** une conversation sécurisée avec **un** nouvel usager (échange de clé à la Diffie-Hellman), puis avec **un groupe** d'usagers fixé ;
 - pour chaque conversation :
 - **envoyer** des messages (textuels dans un premier temps) qui seront **chiffrés** et **authentifiés** par une signature ;
 - **afficher** le contenu (déchiffré, et dont l'authenticité aura été validée) de la conversation.
- Le client disposera d'une interface graphique "agréable d'utilisation" (exemple : onglets de conversations, boutons d'envoi, rafraîchissement automatique, etc.).
2. Mise en place d'une **application serveur** qui aura pour rôle de répondre aux différentes requêtes des clients (pour l'échange de clés, le stockage et l'envoi de messages chiffrés, etc.). Cette application n'aura accès à **aucune** clé privée des utilisateurs.
3. Mise en place d'une **application client malhonnête**, qui illustre qu'un participant malhonnête (par exemple, un participant n'ayant pas la bonne clé), ne peut pas briser la sécurité des (ou de certaines) fonctionnalités listées ci-dessus.

Pistes d'amélioration. Voici quelques suggestions d'amélioration. Attention, elles peuvent demander plus ou moins de travail de *refactoring* suivant la manière dont vous avez réalisé votre implantation initiale.

- (a) Permettre la suppression ou la modification d'un message dans une conversation.
- (b) Permettre la suppression d'un utilisateur d'un groupe.
- (c) Permettre l'ajout d'un utilisateur à un groupe¹.
- (d) Permettre la réinitialisation d'une clé (en cas de perte par exemple)¹.

Précisions. Le langage de programmation, le choix des algorithmes de chiffrement ou de signature, ainsi que les protocoles de transmission sont à votre appréciation.

L'application serveur doit être unique. Dans un premier temps, il est également conseillé de ne développer qu'une seule application client, et de rendre l'ensemble du système fonctionnel. Ensuite, et seulement ensuite, libre à vous d'en développer plusieurs.

1. cette action devrait nécessiter l'accord d'au moins un participant déjà présent dans la conversation — voire, de tous

Sujet 2 : système de vote électronique

Objectif. Le but de ce projet est d'implanter un système de vote électronique simplifié, « sécurisé » grâce au chiffrement partiellement homomorphe de Paillier.

Fonctionnalités minimales attendues.

1. Mise en place d'une **application serveur** (liée à une autorité de confiance) qui :
 - initialise le vote avec différents paramètres (nombre et nom des candidats, identifiants des électeurs, dates, etc.),
 - reçoit les bulletins, et en valide l'origine,
 - calcule de manière **homomorphe**, puis proclame, les résultats de manière authentifiée.
2. Mise en place d'une **application client** qui, associé à un électeur, permet de :
 - prendre connaissance des informations publiques liées à l'élection,
 - voter, c'est-à-dire émettre un bulletin **chiffré** et **authentifié** à destination de l'autorité de confiance,
 - accéder aux résultats du vote et vérifier l'authenticité de l'émetteur des résultats.
- Le client disposera d'une interface graphique "agréable d'utilisation".
3. Mise en place d'une **application client malhonnête**, qui illustre qu'un participant malhonnête (par exemple, un participant n'ayant pas la bonne clé), ne peut pas briser la sécurité des (ou de certaines) fonctionnalités listées ci-dessus.

Pistes d'amélioration. Voici quelques suggestions d'amélioration. Attention, elles peuvent demander plus ou moins de travail de *refactoring* suivant la manière dont vous avez réalisé votre implantation initiale.

- (a) explications et mises en place d'éventuelles attaques sur votre système de vote
- (b) mise en place d'une liste d'émargement, "validée" par chaque électeur et par l'autorité de confiance
- (c) permettre à un électeur de "mettre à jour" son bulletin (dans le temps imparti pour le vote)
- (d) pour l'autorité de confiance : vérification *zero-knowledge* de la validité d'un bulletin
- (e) pour un électeur : vérification *zero-knowledge* que son bulletin a bien été pris en compte dans le calcul de la somme

Précisions. Le langage de programmation, le choix des algorithmes de chiffrement ou de signature, ainsi que les protocoles de transmission sont à votre appréciation.

L'application serveur doit être unique. Dans un premier temps, il est également conseillé de ne développer qu'une seule application client, et de rendre l'ensemble du système fonctionnel. Ensuite, et seulement ensuite, libre à vous d'en développer plusieurs.

Référence. Pour le chiffrement de Paillier, la page wikipedia est une bonne entrée en matière :

https://fr.wikipedia.org/wiki/Cryptosystème_de_Paillier

Concernant les preuves *zero-knowledge* proposées en amélioration, des explications et des références bibliographiques vous seront fournies au moment voulu.

Sujet 3 : gestion sécurisée d'une cagnotte commune

Objectif. Le but de ce projet est d'implanter un service de gestion d'une cagnotte commune à plusieurs utilisateurs.

Fonctionnalités minimales attendues.

1. Mise en place d'une **application serveur** qui gère l'état de la cagnotte.
 - Initialiser une cagnotte qui consiste en un montant et un ensemble d'utilisateurs authentifiables.
 - Implémenter les opérations "fonctionnelles" courantes : accès au montant, crédit, débit (sans découvert).
 - La valeur de la cagnotte peut être stockée en clair, mais la transmission de cette valeur à des utilisateurs doit se faire de manière sécurisée (**chiffrée** et **authentifiée**).
2. Mise en place d'une **application client** qui gère les interactions entre les utilisateurs et la cagnotte.
 - **Accès au montant** de la cagnotte : après une authentification, réception du montant chiffré, déchiffrement, et vérification de l'authenticité.
 - **Crédit** : transmettre un montant chiffré et authentifié, à ajouter à la valeur courante de la cagnotte (implémenter les opérations correspondantes côté serveur). Chaque utilisateur peut effectuer un crédit de manière indépendante, sans l'accord d'autres utilisateurs.
 - **Débit** : le débit de la cagnotte doit se faire avec l'accord d'autres utilisateurs.
 - Pour cela, utiliser un schéma de partage de secret à seuil (exemple : Shamir), qui permet l'accès à un token temporaire autorisant le débit.
 - Comme pour le crédit, le montant sera envoyé de manière chiffrée et authentifiée.
 - Également, la demande d'autorisation de dépense d'un participant doit être transmise à l'ensemble des autres participants de manière sécurisée (mais pas forcément leurs réponses).

Le client disposera d'une interface graphique "agréable d'utilisation".

3. Mise en place d'une **application client malhonnête**, qui illustre qu'un participant malhonnête (par exemple, un participant n'ayant pas la bonne clé), ne peut pas briser la sécurité des (ou de certaines) fonctionnalités listées ci-dessus.

Pistes d'amélioration. Voici quelques suggestions d'amélioration. Attention, elles peuvent demander plus ou moins de travail de *refactoring* suivant la manière dont vous avez réalisé votre implantation initiale.

- (a) Ajout de nouveaux utilisateurs à la cagnotte.
- (b) Un utilisateur peut signer l'état courant de la cagnotte, et cela **au nom de tout le monde** (notion de signature de cercle, de signature de groupe).
- (c) Avancé : gestion inter-cagnottes.

Précisions. Le langage de programmation, le choix des algorithmes de chiffrement ou de signature, ainsi que les protocoles de transmission sont à votre appréciation.

L'application serveur doit être unique. Dans un premier temps, il est également conseillé de ne développer qu'une seule application client, et de rendre l'ensemble du système fonctionnel. Ensuite, et seulement ensuite, libre à vous d'en développer plusieurs.

Sujet 4 : plateforme de vente/achat par enchères

Objectif. Le but de ce projet est d'implanter un service de vente aux enchères sécurisé.

Fonctionnalités minimales attendues.

1. Mise en place d'une **application serveur** qui gère la mise aux enchères de **plusieurs** objets.
 - Initialisation d'une mise en vente aux enchères "publique" d'un objet : les informations transmises du service vers les acheteurs **doivent** être authentifiées, et **peuvent** être chiffrées.
 - Récupération des différentes propositions des acheteurs pour un objet donné : elles **devront** être chiffrées, authentifiées et associées à l'objet en vente. Puis, mise à jour et transmission du prix de vente aux acheteurs.
 - **Important :** on souhaite que les acheteurs ne puissent pas connaître l'actuel *leader* de la mise aux enchères.
 - Il y aura un intervalle de temps durant lequel l'enchère sera disponible. Par ailleurs, l'enchère doit être supérieure au prix. La validité d'une proposition d'enchère devra donc être vérifiée (mettre en place un protocole pour cela).
 - Proclamation publique et authentifiée du résultat, ainsi que du nom de l'acquéreur (qui ne devra pas être publié/accessible avant).
2. Mise en place d'une **application client** qui permet aux acheteurs de se renseigner sur les différents objets mis en vente, et de proposer des "promesses d'achat".
 - Transmission d'une proposition d'achat **chiffrée** et **authentifiée**.
 - Actualisation du prix de vente, en vérifiant bien l'authenticité de cette mise à jour.
 - Pareil pour la réception du résultat.
- Le client disposera d'une interface graphique "agréable d'utilisation".
3. Mise en place d'une **application client malhonnête**, qui illustre qu'un participant malhonnête (par exemple, un participant n'ayant pas la bonne clé), ne peut pas briser la sécurité des (ou de certaines) fonctionnalités listées ci-dessus.

Pistes d'amélioration. Voici quelques suggestions d'amélioration. Attention, elles peuvent demander plus ou moins de travail de *refactoring* suivant la manière dont vous avez réalisé votre implantation initiale.

- (a) Créer une fonctionnalité côté client, qui permet de **proposer** une mise aux enchères ("client vendeur").
- (b) Possibilité pour un acheteur de se rétracter pendant un petit intervalle de temps (disons, 10 secondes).
- (c) Associer à chaque client une valeur de cagnotte qu'il peut créditer (directement, ou via le résultat d'un mise aux enchères, voir (a)), ou débiter pour une proposition d'achat.

Précisions. Le langage de programmation, le choix des algorithmes de chiffrement ou de signature, ainsi que les protocoles de transmission sont à votre appréciation.

L'application serveur doit être unique. Dans un premier temps, il est également conseillé de ne développer qu'une seule application client, et de rendre l'ensemble du système fonctionnel. Ensuite, et seulement ensuite, libre à vous d'en développer plusieurs.

Sujet 5 : tirage au sort distribué et sécurisé

Objectif. Le but de ce projet est d'implémenter un système de tirage au sort distribué et sécurisé. Plus précisément, un ensemble de personnes souhaite choisir aléatoirement et uniformément l'une d'entre elles, sans déporter le tirage vers un tiers de confiance, et sans faire confiance aux autres participants.

Fonctionnalités minimales attendues.

1. Mise en place d'une **application serveur** qui gère la communication entre les participants et orchestre le protocole de tirage au sort.
 - Initialisation d'un tirage au sort : création d'une **session** de tirage.
 - Connexion des participant·es à une session, puis lancement par l'initiateur·ice de la session du tirage.
 - Le tirage est constitué d'un certain nombre de tours, dont l'objectif est d'« éliminer » petit à petit des participants du tirage, jusqu'à ce qu'il n'en reste plus qu'un. Les détails du protocole sont décrits plus bas.
2. Mise en place d'une **application client** qui permet de participer à un tirage au sort.
 - Demande de création d'une session de tirage au sort au serveur.
 - Connexion à une session déjà créée.
 - Tirage au sort de 0 ou 1, puis création et envoi de l'engagement sous forme de condensat cryptographique (= haché) salé au serveur.
 - Création et envoi de la preuve son engagement.
 - Vérification de l'engagement d'un·e participant·e.
3. Mise en place d'une **application client malhonnête** qui permettrait à deux (ou plus) personnes de biaiser le tirage au sort en se concertant.

Pistes d'amélioration.

1. Que se passe-t-il si on autorise deux participant·es à coopérer pour tricher ? Comment se prévenir de ce problème ? Et pour trois et plus ?
2. Le serveur a encore un rôle d'intermédiaire entre les participants. Essayer de se passer complètement de lui en faisant directement du pair-à-pair.

Précisions. Le langage de programmation, le choix des algorithmes de chiffrement ou de signature, ainsi que les protocoles de transmission sont à votre appréciation.

L'application serveur doit être unique. Dans un premier temps, il est également conseillé de ne développer qu'une seule application client, et de rendre l'ensemble du système fonctionnel. Ensuite, et seulement ensuite, libre à vous d'en développer plusieurs.

Description d'un tour de protocole de tirage au sort.

Hypothèse. Il y a encore $n \geq 2$ participant·es dans ce tour.

1. Le serveur choisit arbitrairement (cela n'a pas d'importance) parmi les n participant·es deux participant·es X et Y qui procéderont au tirage du tour.
2. Chaque participant·e P_i produit un **condensat cryptographique** (= haché) d'un bit $b_i \in \{0, 1\}$ choisi aléatoirement, « salé » par une valeur aléatoire. Ces condensats sont appelés des engagements, et sont transmis de manière sécurisée aux différents participant·es par l'intermédiaire du serveur.
3. Les participant·es X et Y produisent et transmettent indépendamment deux autres condensats cryptographiques de bits x_i et y_i tirés aléatoirement.
4. Tou·te·s les participant·es fournissent une **preuve** des bits sur lesquels ils se sont engagés (c'est-à-dire, ils envoient le bit et sel correspondant au haché). Ces preuves pourront être vérifiées si besoin par les autres participants.
5. Ne restent au tour suivant que les participant·es dont le bit b_i vaut $x_i \oplus y_i$. Si le nombre de participant·es est 0, on recommence le tour.