UNIVERSITÉ DE SHERBROOKE
DÉPARTEMENT D'INFORMATIQUE

IFT606 - Sécurité et cryptographie

Travail de session

Projet AirTransit

Travail présenté à
M. Marc VERREAULT et
M. Mohammed OUENZAR

par

Hugo Boisselle-Morin (15 093 701)
Raphaël Cayouette (15 064 430)
Jean-Philippe Croteau (15 072 957)
Jonathan Martineau (15 059 124)
Félix Vigneault (15 079 576)

Le 11 avril 2018

Table des matières

Méthodologies *3*
Analyse 3
Design 3
Développement 3
Assurance qualité 3
Description de la problématique *4*
Solution proposée *5*
Explication 5
Serveur 5
Client 6
Signature des messages 7
Survol des composantes *8*
Explication du chiffrement *9*
Limitations de la solution proposée *10*
Serveur en tant qu'autorité de confiance 10
Validation de l'inscription des clients 11
Non-répudiation des messages 11
Réflexion sur la solution proposée *13*
Conclusion 13
Liste des compétences acquises *14*
Plan de travail et efforts *14*

Méthodologies

Analyse

Nous avons débuté avec l'énumération des besoins, par exemple la sécurité et la portabilité de notre application. Nous avons également pris en compte les différentes solutions déjà présentes sur le marché, comme Signal ou Facebook Messenger. Dans le cas de Signal, nous pouvions également consulter le code puisqu'il s'agit d'un logiciel libre.

Design

Nous avons discuté des différentes solutions possibles et fait le schéma de notre implémentation. Le projet a été séparé en différents morceaux qui pouvaient être développés en parallèle, donc un projet pour le serveur, le client, l'interface Windows et l'interface console. Spécification des interfaces entre ces dépendances.

Développement

Les sous-projets ont été développés en parallèle à partir de la spécification. L'architecture a été élaborée au fil du temps par itération à mesure que les besoins devenaient évidents. Par exemple, l'architecture du client a été pensée pour que les parties critiques soient facilement testables par injection des dépendances. Ainsi, il a été possible de développer le module de chiffrement avec l'approche d'écrire les tests en premier et d'implémenter par la suite.

Assurance qualité

Nous avons testé les cas d'usage typique et réitéré sur la solution à mesure que les problèmes survenaient. Plusieurs problèmes avec le chiffrement ont fait surface à cette étape.

Description de la problématique

La protection de la confidentialité est un enjeu critique majeur dans la société actuelle. Avec tous les scandales de divulgation d'informations confidentielles (tels que la brèche Equifax aux États-Unis survenue en 20171), les gens commencent à prendre conscience de la situation et se tournent vers des produits et solutions qui ont à coeur la confidentialité de leurs informations personnelles.

De plus, avec tous les messages textes envoyés sur Internet (plus de 7 milliards de conversations chaque jour sur Facebook Messenger en 20172), il n'est pas étrange de vouloir avoir une façon sécuritaire et protégée de s'envoyer de l'information. C'est d'ailleurs pourquoi des dizaines d'applications de messagerie sécurisée sont disponibles. En voici quelques-unes des plus connues: Signal3, Telegram4, et iMessage.

Pour ce projet, nous avons donc voulu également explorer le monde de la communication chiffrée en proposant une application de messagerie confidentielle en utilisant les connaissances en sécurité apprises dans ce cours.

[1] https://arstechnica.com/information-technology/2017/09/why-the-equifax-breach-is-very-possibly-the-worst-leak-of-personal-info-ever/
[2] https://fbnewsroomus.files.wordpress.com/2017/12/yearinreview-global-2.pdf
[3] https://www.signal.org/
[4] https://telegram.org/

Solution proposée

Explication

Notre but était de définir un protocole de communication utilisant le chiffrement de bout en bout et de développer une architecture complète implémentant ce protocole. En s'inspirant particulièrement de l'application Signal, nous nous demandions comment il était possible de gérer efficacement le chiffrement des messages et leur distribution afin de réussir à garantir la confidentialité et l'intégrité des messages ainsi que l'authentification des utilisateurs, tout ça en partant d'un simple numéro de téléphone.

Notre solution se base sur le chiffrement asymétrique de messages entiers par RSA. Chaque client a une clé privée et une clé publique et le serveur agit en tant que registraire des clés.

Serveur

Plus précisément, le serveur a deux rôles : transmettre les messages aux destinataires qui en font la demande et gérer l'inscription et l'identification des clients. Il contient donc très peu de logique puisque la validation des messages se fait du côté client. Le serveur administre une base de données simple qui contient uniquement deux tables. La première sert de registre des clés publiques et est exposée publiquement en lecture via des appels d'API. Ces informations servent à identifier la clé de chiffrement à utiliser pour communiquer avec un individu. Le serveur a donc la responsabilité de s'assurer que cette table soit valide et que les clés publiques appartiennent bien à celles des clients. La seconde table est simplement un dépôt pour les messages. Elle contient donc tous les messages chiffrés associés à leur destinataire. Lorsqu'un client veut récupérer ses messages, le serveur applique un filtre sur la base de données pour retrouver ses messages et lui envoyer. Le client peut aussi supprimer ses messages de la base de données centrale sur demande.

Le serveur gère aussi l'authentification des clients qui font des demandes d'accès à la table des messages. L'authentification est réalisée à l'aide d'une signature

lors des appels d'API. Le client utilise sa clé privée pour chiffrer son numéro de téléphone, ce qui génère sa signature. Il joint alors cette signature à ses demandes de récupération et de suppression de messages. Le serveur est alors en mesure de déchiffrer le numéro de téléphone avec la clé publique pour valider avec qui il communique réellement. Cette technique fonctionne sur la base que la communication entre le client et le serveur est chiffrée sous HTTPS. En effet, si la requête en clair peut être interceptée par un attaquant, il pourrait copier la signature et imiter le client initial en apposant la signature copiée sur des requêtes malicieuses.

Client

Le client dans notre application possède beaucoup plus de responsabilités qu'un client de messagerie habituel. Il est, entre autres, responsable du chiffrement et du déchiffrement des messages. Puisque le serveur sert uniquement de tampon pour les messages, le client doit aussi gérer la validation des messages afin de s'assurer qu'il reçoit des messages valides provenant du bon destinataire.

Le client, lors de sa première utilisation du service, génère sa paire de clés et enregistre la clé privée localement. Il initie ensuite une procédure d'inscription auprès du serveur en lui transmettant son numéro de téléphone ainsi que sa clé publique. Une fois cette étape franchie, le client est considéré comme inscrit et les autres clients peuvent alors lui envoyer des messages.

Le scénario habituel d'échange d'un message est assez simple. Le client source prépare son message et indique l'heure d'envoi ainsi que son propre numéro de téléphone. Pour valider que c'est bien lui qui a écrit ce message, il joint aussi une signature. Cette signature est simplement le numéro de téléphone du destinataire, chiffrée avec la clé privée du client source (voir la section suivante pour une explication complète de la sécurité qu'ajoute cette signature dans l'échange de messages). Une fois le message complètement formé, le client source récupère la clé publique du destinataire auprès du serveur, puis utilise cette clé pour chiffrer le message en entier. Il transmet alors le résultat au serveur en indiquant le numéro de téléphone de destination. Le serveur enregistre le message chiffré dans sa base de données en l'associant au numéro de téléphone fourni.

Lorsque le destinataire récupère ses messages, en signant sa demande avec son numéro de téléphone chiffré par sa clé privée, le serveur lui transmet le message reçu. Le destinataire commence par déchiffrer le message puis passe à l'étape de validation. La validation consiste à vérifier la signature du message. Puisque cette dernière doit être déchiffrée par la clé publique du client source, le destinataire utilise le numéro de téléphone source pour récupérer auprès du serveur la clé publique du client source. Le destinataire déchiffre ensuite la signature avec cette clé publique et valide que le résultat est bel et bien son numéro de téléphone personnel. Le message est alors considéré comme valide. Si une des étapes échoue, on considère le message invalide et on le rejette.

Signature des messages

La signature jointe à chaque message, d'apparence assez banale, remplit plusieurs rôles. D'abord, elle sert à prouver auprès de la destination que le client source est bien celui qui a écrit le message, puisque seul ce client avait accès à la clé privée. Il est d'ailleurs facile de valider que c'est cette clé qui fut utilisée puisque le numéro de téléphone source est joint au message et on peut alors récupérer la clé publique associée à partir du serveur. La clé publique déchiffre la signature afin que l'on puisse valider que l'information recueillie est bien notre numéro de téléphone.

Une autre propriété intéressante de cette signature est qu'elle est unique pour chaque conversation. Cela signifie qu'on ne peut pas simplement la copier et l'utiliser pour imiter le client source auprès d'un autre client destinataire. Étant donné qu'on chiffre le numéro de téléphone de destination, on est certain que seul ce destinataire acceptera le message puisqu'il connaît son numéro de téléphone. Initialement, on voulait chiffrer le numéro de téléphone source, mais cette méthode rendait la signature copiable pour plusieurs destinataires.

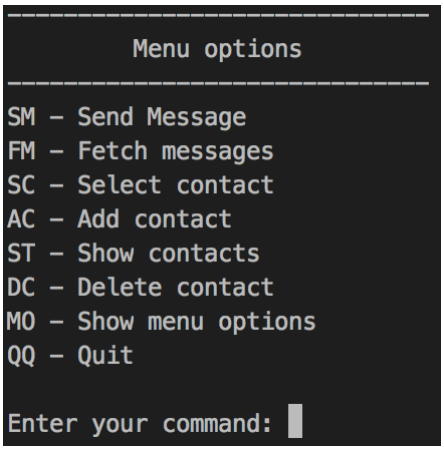
La signature est donc identique pour tous les messages envoyés à un destinataire précis, et ce pour toujours. C'est toutefois sécuritaire puisque la signature est jointe au message avant l'étape de chiffrement avec la clé publique du destinataire. La signature est donc à nouveau chiffrée avec la clé publique du destinataire et seul ce dernier pourra la déchiffrer et la consulter. Cette propriété, combinée avec la

précédente, fait en sorte que seule la personne visée pourra voir la signature et en déceler un sens, ce qui empêche complètement la copie de la signature à des fins malicieuses.

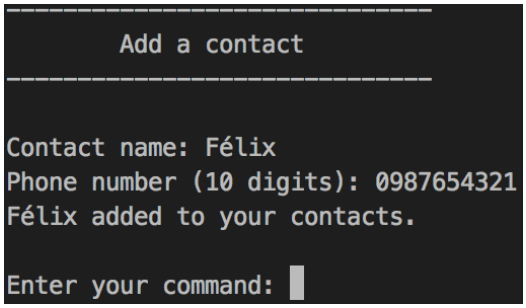
Survol des composantes

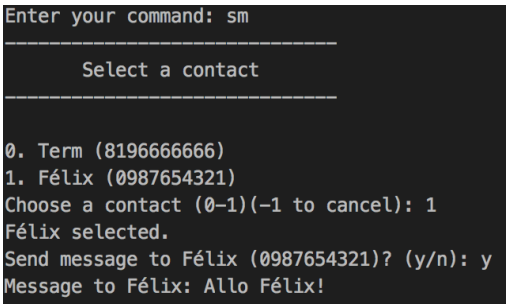
Notre application possède quatre composantes principales. Le serveur en .NET Core 2.0, supporté par une base de données SQL Lite Entity Framework, expose une interface de programmation applicative (API) Swagger permettant d'envoyer des messages, obtenir les messages reçus, ajouter des contacts au bottin téléphonique de clés publiques, et les retirer. Toute l'information textuelle enregistrée sur le serveur est chiffrée par le code du coeur de l'application avant l'envoi.

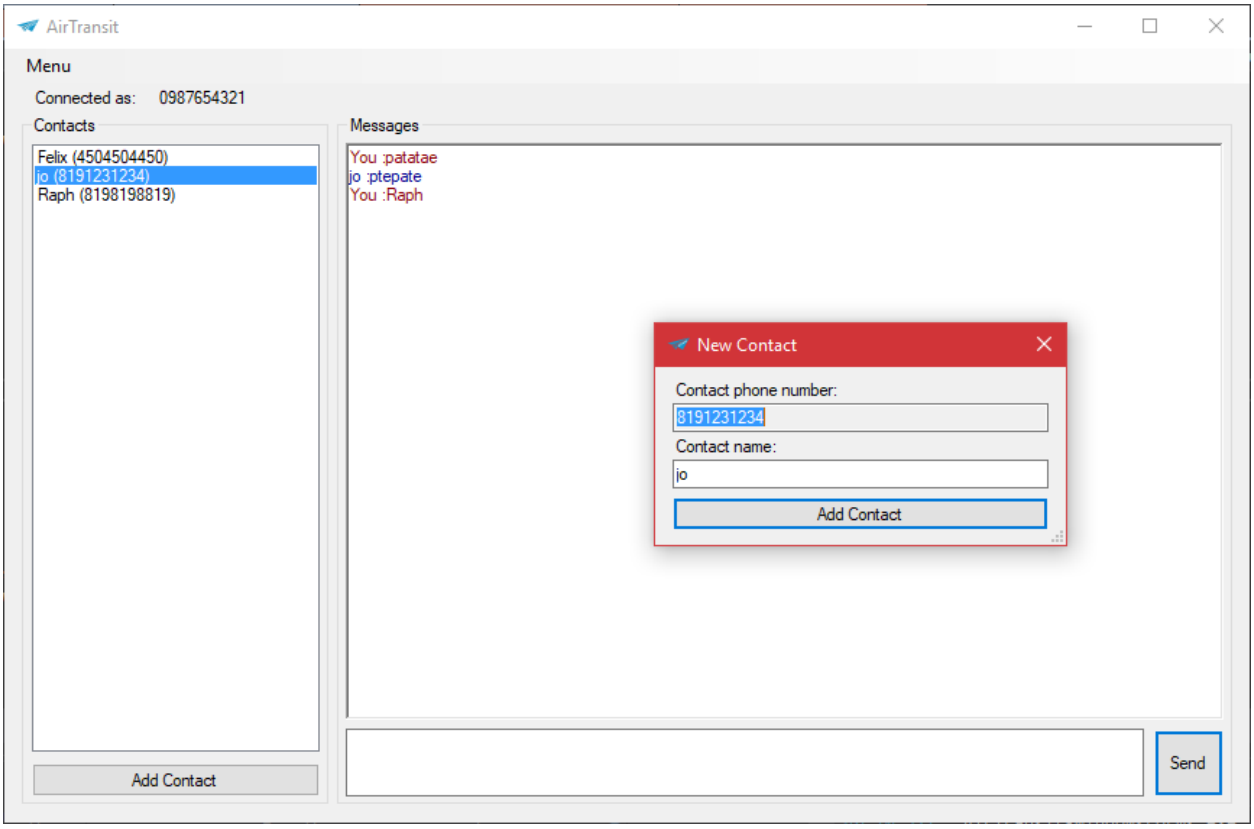
La deuxième composante principale est le « Core » (ou coeur) de notre application. Développé en C# sur la plateforme .NET, c'est lui qui effectue le traitement du chiffrement et déchiffrement asymétrique lors des envois de messages. Il expose également plusieurs interfaces qui permettent aux différentes applications clientes d'effectuer aisément des requêtes en direction du serveur. Un récupérateur de messages permet également de vérifier périodiquement de façon autonome si des nouveaux messages sont entrés en notre destination.



Les deux dernières composantes principales de notre application sont les interfaces qui permettent d'interagir avec les fonctionnalités. Une première interface, développée en « Windows Forms » du cadriciel .NET, permet une interaction graphique avec notre programme. La deuxième interface est une application console C# .NET Core permettant les mêmes interactions, mais de manière textuelle dans une console.







Client Windows Form avec exemple de changement de nom d'un contact

Explication du chiffrement

Le chiffrement asymétrique utilisé par notre application se sert d'une implémentation .NET de RSA. Afin d'uniformiser le plus possible l'échange entre les clients, nous avons standardisé les clés à une grosseur de 2048 pour tous les clients. Ce standard nous permet d'éviter d'implanter un protocole d'échange des paramètres de chiffrement entre les clients. Une clé de 2048 bits nous assure aussi une sécurité très élevée et nos tests de temps d'exécution ont démontré que les délais n'étaient pas trop élevés.

Nous avons aussi un procédé de signature qui utilise la clé privée du client afin de générer des signatures pour les messages, mais aussi pour l'authentification des requêtes au serveur. Ces signatures sont créées simplement en chiffrant un numéro de téléphone, celui de la destination lors de la signature d'un message et celui de la source lors de l'authentification auprès du serveur. La signature utilise l'algorithme de hachage SHA-256.

Pour le chiffrement et le déchiffrement des messages, nous avons dû séparer nos messages complets en plusieurs paquets. Cela est dû à la limitation de RSA qui peut simplement chiffrer des messages plus petits que le modulo de la clé (pour une clé 2048 bits, cette limite est de 256 bits). Un remplissage PKCS 1 est aussi ajouté à chaque paquet. On se retrouve donc avec des paquets de 256 bits qui contiennent au maximum 245 bits de données. Cette valeur est statique et connue de tous les clients puisque nous n'avons pas de protocole d'échange de paramètres.

Limitations de la solution proposée

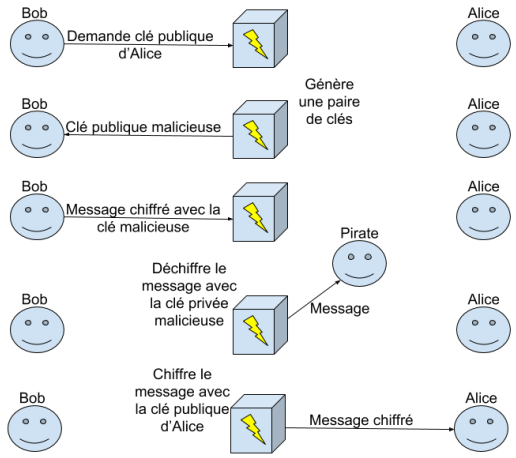
Serveur en tant qu'autorité de confiance

Avec cette solution, nous devons faire un compromis par rapport à nos garanties. En effet, en agissant comme registraire des clés, le serveur a le rôle d'autorité de confiance. Il peut, s'il le désire, se mettre à distribuer de fausses clés et agir en tant qu'homme du milieu entre les conversations de la victime et de ses destinataires. Il n'a qu'à distribuer une fausse clé pour le destinataire qu'une victime souhaite rejoindre, ce qui forcera la victime à chiffrer son message avec cette clé. Une fois le message transmis au serveur malicieux, ce dernier peut déchiffrer le message et le chiffrer à nouveau avec la véritable clé publique du destinataire avant de lui transmettre. L'écoute complète des messages est donc possible.

Ce problème est cependant mitigé par la relation de confiance entre les clients et le serveur, qui est garantie par le lien HTTPS qui les lie lors de leurs échanges. Donc un imposteur tentant de se faire passer pour un faux serveur ne sera pas capable de reproduire ce scénario puisque les clients le rejetteront en tant que serveur invalide.

Le point de fuite reste toutefois le véritable serveur en tant que tel, qui n'est pas protégé contre le piratage ou la malhonnêteté de ses propriétaires. En effet, si le serveur se fait pirater, le pirate passera pour le vrai serveur et HTTPS n'offrira plus de protection. De plus, si le propriétaire du serveur décide du jour au lendemain qu'il ne souhaite plus offrir de sécurité dans son application de messagerie, il peut décider de

fournir sa propre clé publique pour le chiffrement de tous les messages, comme ce serait le cas dans une attaque classique de type man-in-the-middle. Il pourrait ainsi déchiffrer tous les messages avec sa propre clé privée.



Validation de l'inscription des clients

Nous n'avons pas implémenté de procédure pour la validation de l'inscription des clients. Lors de l'inscription, les clients fournissent leur numéro de téléphone ainsi que la clé publique qu'ils ont générée. Le serveur peut ensuite distribuer cette clé publique aux autres clients afin qu'ils puissent chiffrer des messages pour ce destinataire. Il est donc primordial que cette clé publique appartienne réellement au client possédant ce numéro de téléphone. Pour le valider, il faudrait envoyer un jeton d'inscription au client via SMS et demander qu'il confirme son inscription à l'aide du jeton avant de commencer la distribution de sa clé publique aux autres clients. L'implantation de cette procédure ne représente pas un défi technique incroyable, mais elle demande de l'investissement dans une architecture d'envoi de SMS. C'est pourquoi nous avons accepté cette limitation dans notre projet.

Non-répudiation des messages

Un des enjeux de la cryptographie est la non-répudiation, mais nous n'avons pas implanté de mécanisme dans le protocole pour offrir cette garantie. En effet, nos messages envoyés sont chiffrés par la clé publique du destinataire. La seule chose qui prouve qu'un client source est bien celui qui a envoyé le message est la signature qui est contenue dans le message et a été générée à partir de sa clé privée. Toutefois, du moment qu'un destinataire a reçu un message du client source, il connaît la signature qui sera utilisée lors de tous les prochains messages qui lui sont envoyés. Il peut donc simplement créer de faux messages et y ajouter la même signature. Un utilisateur peut

donc s'envoyer des messages à lui-même qui semblent être de la part de quelqu'un d'autre, et ce, sans pouvoir prouver que le message n'a pas réellement été envoyé par cette autre personne.

Nous avons tout de même identifié une solution pour assurer la non-répudiation partielle des messages, mais elle a des lacunes. Pour assurer l'intention d'envoi d'un message particulier par un client source, il faudrait que le client source inclue l'entièreté du message dans sa signature. Une procédure possible pourrait inclure de générer un SHA256 du message et de signer ce hash en même temps que le numéro de téléphone du destinataire. De cette façon, le destinataire peut valider, s'il le souhaite, le hash en le répétant de son côté après avoir déchiffré la signature. La lacune principale de cette solution est qu'elle ajoute encore plus de traitement à un processus cryptographique déjà assez lourd.

Une autre lacune est la falsification des dates d'envoi. En effet, il est quasi impossible de s'assurer que la date d'envoi indiquée dans le message est valide. Le client a le contrôle et peut mettre ce qu'il veut, que ce soit une date dans le passé ou dans le futur. Cela permet donc de falsifier une conversation en insérant des messages entre les messages existants, ce qui permet de changer le sens des messages dans la conversation. Aucune garantie ne peut donc être émise sur l'ordonnancement des messages, seulement sur un message individuel. Une solution identifiée pour ce problème passe par l'intervention du serveur dans l'ordonnancement des messages. Il peut noter les heures de réception des messages pour aider le destinataire à décider si le message est valide ou non, mais beaucoup de scénarios pourraient faire en sorte qu'un message aie une date de création valide, mais être envoyé avec succès au serveur seulement plusieurs heures, voir plusieurs jours plus tard (par exemple une connexion internet qui lâche). Il serait donc néfaste d'appliquer un filtre trop agressif sur les messages dont les dates sont louches. Cette solution a aussi comme désavantage le fait qu'il faut faire confiance au serveur pour fournir des dates véridiques, ce qui augmente encore le lien de confiance avec l'autorité, ce qu'on essaie d'éviter le plus possible.

En résumé, le problème de non-répudiation est très compliqué à gérer et possède des limitations quant à l'ordonnancement des messages. Les messages de

notre application ont donc une pertinence limitée comme preuve, mais ce n'était pas de toute façon notre priorité avec ce projet.

Réflexion sur la solution proposée

Conclusion

Nous avons passé la dernière session à élaborer et implanter un protocole d'échange de messages chiffrés de bout à bout. Notre première conclusion est qu'il est très difficile de protéger toutes les failles. Les premières semaines ont été consacrées à l'élaboration sur papier du protocole et c'est l'étape que nous avons trouvé la plus difficile. On tentait de raisonner sur les sécurités offertes par notre protocole et se mettre dans la peau des pirates qui pourraient vouloir s'y attaquer.

Plusieurs itérations ont été nécessaires pour arriver à offrir les garanties de bases de confidentialité, d'intégrité et d'authentification. Il y avait toujours des failles dans notre protocole, mais chaque itération apportait des corrections et à nouveau d'autres limitations. Notre version actuelle est celle offrant le moins de limitations, mais elle n'est pas parfaite. Il reste encore des points de faille, comme l'attaque de l'homme du milieu sur les serveurs.

La seconde leçon que nous avons retenue est par rapport à la difficulté de se débarrasser d'une certaine forme d'autorité de confiance dans les échanges cryptographiques. Dans notre cas, le serveur agit comme autorité de confiance dans la distribution des clés, ce qui ajoute un risque pour les clients. Si ce serveur se fait pirater, les échanges de tous les clients sont compromis. Nous n'avons pas réussi à élaborer une stratégie de distribution des clés qui ne se basait pas sur une autorité de confiance. Le défi est plus imposant que nous pensions et nous sommes à présent mieux équipés pour comprendre les raisons ayant motivé le SSL à se baser sur des autorités de confiance.

La dernière leçon importante concerne le chiffrement, plus précisément quant à la distinction entre le chiffrement asymétrique et le chiffrement symétrique. Notre hypothèse initiale était que le chiffrement asymétrique pouvait être utilisé facilement

pour le chiffrement de message entre utilisateurs. L'école de pensée autour du chiffrement asymétrique dicte qu'il ne devrait être utilisé que pour échanger une clé de chiffrement symétrique, puisque le traitement à chaque envoi est coûteux. Notre situation, soit l'échange de messages, ne requiert pas une grande rapidité et nous avions comme hypothèse que cette propriété de notre problème pouvait simplifier notre protocole. Après des tests de performance, nous avons conclu que notre hypothèse était exacte. Les messages textes n'ont pas une longueur élevée en moyenne et leur envoi n'a pas besoin d'être instantané. La perception humaine ne différencie pas une différence de 200 millisecondes sur la réception d'un message. En fait, le plus grand facteur de délai reste encore le réseau.

Liste des compétences acquises

- Utilisation des différentes versions de point Net (« Framework .Net », « .Net Core », « .Net Standard »)
- Cadriciel Swagger pour la création d'interfaces de programmation applicatives
- Utilisation de la bibliothèque SQLite pour la conservation d'informations hors-ligne
- Cryptographie asymétrique de façon appliquée
- Utilisation de l'algorithme RSA en C# et limitations de l'algorithme
- Création d'une application utilisant plusieurs interfaces différentes sur plusieurs technologies différentes

Plan de travail et efforts

Nous n'avons pas effectué de plan de travail avant de démarrer le projet. Au total, environ 225 heures à 5 personnes ont été mises sur ce projet, dont une vingtaine d'heures sur le rapport et la présentation. Notre code, en plus de la remise, sera disponible à code source ouvert sur GitHub à la fin de la présentation sur ces deux répertoires:

https://github.com/Wingjam/AirTransitClient--E2E-Encrypted-Chat

https://github.com/Wingjam/AirTransitServer--E2E-Encrypted-Chat