# TP2 : génération de secret par puzzles cryptographiques

**Durée du TP** : 4h.
**Date limite de remise du TP** : Vendredi 29 janvier 2016 à 23h59.
Le TP se fait par groupe de 2 étudiants (au maximum) et les fichiers à remettre doivent être envoyées au plus tard vendredi 29 janvier à minuit par courriel à l'adresse électronique "sgambs@irisa.fr".
**Rappel** : tout plagiat est formellement interdit et bien qu'il soit naturel que vous puissiez parfois discuter avec vos camarades oralement sur comment attaquer ou résoudre un problème, il est formellement interdit d'échanger des fichiers de code.

**Description :**
Le but de ce deuxième TP est de vous faire analyser, modifier et implémenter un protocole de génération de secret qui permet à deux entités ne partageant aucune information en commun à priori de générer un secret simplement en communiquant à travers un canal publique, donc potentiellement écouté par un espion. On suppose cependant que cet espion est un adversaire passif, c'est à dire qu'il écoute les communications mais ne triche pas activement en injectant par exemple de faux messages.

L'idée principale du protocole est que l'entité A (Alice) va préparer une série de *n* puzzles cryptographiques qu'elle va ensuite envoyer en clair à l'entité B (Bob) à travers un canal publique. Bob va ensuite choisir un puzzle au hasard parmi les *n* et chercher à le résoudre (ce qui demande un temps considéré comme constant mais non négligeable). Une fois le puzzle résolu, celui-ci révèle une clé cryptographique (du type AES) et un index de clé. Bob envoie ensuite l'index de clé en clair à Alice et il partage maintenant un secret commun (la clé cryptographique) qui pourra être utilisée pour sécuriser les communications entre eux. En plus de l'implémentation du protocole, il vous sera demandé d'analyser sa sécurité et en particulier ce que l'adversaire peut faire pour en briser la sécurité.

Le protocole de génération de secret par puzzles cryptographiques se déroule de la façon suivante.

1. Alice génère aléatoirement un ensemble de *n* paires (*pre_puzzle_key(i)*,*secret_key(i)*) où *i* est l'index de cette paire (pour *i* entre 1 et *n*) et *pre_puzzle_key(i)* et *secret_key(i)* sont des chaînes aléatoires de 128 bits.

2. Alice va ensuite générer une série de *n* puzzles cryptographiques de la manière suivante. Pour générer le puzzle *i*, Alice commence par passer de manière récursive la chaîne aléatoire *pre_puzzle_key(i)* 1000 fois de suite dans une fonction de hachage (vous pourrez pour cela utiliser par exemple SHA-1, MD5 ou toute fonction de hachage cryptographique standard) et obtenir comme résultat *puzzle_key(i)=h^1000(pre_puzzle_key(i))* qui est aussi une chaîne aléatoire de 128 bits (si nécessaire vous aurez à tronquer la sortie de la fonction de hachage pour qu'elle corresponde à cette taille de clé). Alice construit ensuite le message *m_i* comme étant la concaténation de l'index i avec la clé *secret_key(i)* générée à l'étape précédente. Ce message sera ensuite chiffré par un chiffrement symétrique tel que AES avec la clé *puzzle_key(i)* pour obtenir le *puzzle(i)* lui-même, soit *puzzle(i)* =

Enc(*secret_key(i)* II *i*, *puzzle_key(i)).*

3. Alice crée maintenant une liste de *n* paires (*pre_puzzle_key(i),puzzle(i))* qu'elle mélange en appliquant une permutation aléatoire avant d'envoyer ensuite cette liste mélangée à Bob en clair par le canal publique.

4. Bob choisit une paire (*pre_puzzle_key(i),puzzle(i))* au hasard parmi les *n* possibles (Bob ne connait alors pas encore l'index *i* de cette paire) et "brise" ce puzzle en calculant tout d'abord la clé correspondante du puzzle *puzzle_key(i)* en passant 1000 fois *pre_puzzle_key(i)* à travers la fonction de hachage. À partir de *puzzle_key(i),* Bob peut maintenant déchiffrer *puzzle(i)* pour récupérer le message *m_i* et donc l'index *i* du puzzle ainsi que la clé secrète correspondante *secret_key(i)*.

5. Bob envoie l'index *i* du puzzle en clair à Alice et ils peuvent maintenant communiquer de manière sécuritaire en utilisant la clé *secret_key(i)* qu'ils connaissent tous les deux (par exemple en utilisant un chiffrement du type AES ou un HMAC se basant sur cette clé).

**Travail à remettre :**

Vous devez implémenter le protocole d'implémentation décrit ci-dessus sous la forme d'une classe `User_puzzle` contenant les fonctionnalités nécessaires pour simuler un utilisateur (en Java ou dans le langage de votre choix). Il vous est aussi demandé d'écrire un court programme qui simule le protocole d'authentification décrit ci-dessus et utilise la classe `User_puzzle`. C'est ce court programme ainsi que la classe demandée que vous devrez remettre, ainsi qu'un court rapport (2 à 3 pages) qui fait une analyse de sécurité du protocole. En particulier, vous devrez répondre dans le rapport aux questions suivantes :

1. Dans l'implémentation actuelle du protocole, la quantité de travail demandé à Alice (l'entité A) est de l'ordre de *O(n*), pour *n* le nombre de puzzles cryptographiques générés. En contrepartie, quelle est la quantité de travail (en terme de calcul et d'espace) demandée à Bob dans ce protocole ainsi que la quantité de travail qui serait nécessaire à un espion éventuel pour briser la sécurité du protocole? Faites une analyse asymptotique du temps de calcul et de l'espace demandées à Bob et l'espion en n'oubliant surtout pas d'expliciter et d'argumenter vos réponses.

2. Supposons qu'on fait maintenant une modification à l'étape 3 du protocole en faisant deux listes séparées, une contenant les *pre_puzzle_key(i)* et l'autre les *puzzle(i)* (les deux listes sont de taille *n*) qu'Alice mélange aléatoirement mais de manière différente (c'est-à-dire en appliquant une permutation aléatoire différente pour chacune des deux listes) avant de les envoyer ensuite séparément à Bob. Bob devra ensuite à l'étape 4 calculer tous les *n* différents *puzzle_key* puis choisir ensuite au hasard un puzzle et essayer de le déchiffrer en utilisant toutes les clés *puzzle_key* jusqu'à trouver la bonne (celle qui déchiffre ce puzzle particulier).

Faites une analyse asymptotique de la quantité de travail (toujours en terme de calcul et d'espace) demandée à Bob et à l'espion pour cette version modifiée du protocole en n'oubliant surtout pas d'expliciter et d'argumenter vos réponses.

Pourquoi cette implémentation semble t'elle plus intéressante au niveau sécurité que l'implémentation initiale?

3. Utiliser le programme que vous avez écrit pour simuler le temps de calcul nécessaire à Alice et Bob pour réaliser le protocole dans sa version originale et sa version modifiée pour les valeurs de *n* suivantes : 100, 1000 et 10 000. Ecrivez aussi un court programme qui simule un espion essayant de briser la sécurité du protocole pour les deux protocoles et pour les mêmes valeurs de *n* que pour Alice et Bob. Discutez les résultats que vous avez obtenu en terme de temps de calcul nécessaire si possible en les comparant dans un graphique. Est ce que les résultats que vous observez sont en accord avec l'analyse asymptotique que vous aviez réalisé? Enfin, quels sont les paramètres que vous estimez apporter une sécurité raisonnable?

4. Pourquoi est qu'il est nécessaire pour que le protocole soit sécuritaire que l'espion soit passif et non pas actif (c'est à dire qu'il écoute simplement les messages échangés par Alice et Bob mais ne triche pas activement en injectant de faux messages)?