Analyse automatique de protocoles cryptographiques basés sur l'exponentiation modulaire

Olivier Blazy

Résumé

Depuis leur création pour permettre une mise en accord de clé de groupe, certains protocoles GDH sont supposés garantir une authentification implicite par la clé ; cependant, il a été montré qu'il y avait un moyen de jouer avec ces protocoles pour pouvoir falsifier cette authentification. Bien que ces méthodes aient été précisées, il n'y a pas eu jusqu'à maintenant d'implémentation permettant de trouver comment attaquer un protocole de ce type...

Dans ce rapport, on montre comment créer un programme qui attaque les protocoles de façon automatisée lorsqu'ils comportent au moins quatre personnes différentes. Ensuite, on voit que parfois il reste encore valable pour trois identités, mais qu'il existe des cas où on ne peut pas conclure sur la faisabilité de l'attaque sans une étude détaillée du protocole ; on voit aussi que cette méthode n'est pas applicable à tous les protocoles à trois participants.

Table des matières

Introduction

Depuis les premiers protocoles à clés publiques l'exponentiation modulaire est utilisée pour la génération des clés. A partir d'opérations peu coûteuses en ressources, on arrive ainsi à obtenir un résultat dur à inverser (*Principe des fonctions à sens unique*).

Lors de la phase de génération de clé au sein du groupe d'utilisateurs, on attend de ces protocoles une authentification implicite via la clé, autrement dit lorsque qu'un utilisateur M_1 génère la clé avec d'autres personnes, il doit pouvoir être sûr, que si les autres arrivent au final à la même clé que lui, alors ils sont dans son groupe. Cette attente est extrêment faible, cependant on sait aisément transformer un tel protocole, en un autre avec une authentification explicite par la clé.

Ce stage a pour but d'implémenter un moyen automatisé afin de se faire passer auprès de M_1 pour un membre du groupe en question, lors de la génération de clés dans un protocole GDH. Lorsque le groupe comporte deux participants, il est, à priori, impossible de le faire ; avec trois participants le résultat n'est pas garanti, par contre à partir de quatre participants on est capable de réaliser l'attaque.

Le stage s'est découpé en deux parties : l'implémentation de l'attaque dans les cas où elle fonctionne à coup sûr, et une généralisation à trois participants avec une recherche pour tenter de prédire les cas où elle va échouer. A ceci est venu se greffer un portage pour le web, pour permettre d'exécuter l'attaque sans qu'il soit besoin d'avoir le code source du programme sur sa machine.

1 Introduction aux protocoles GDH

Les protocoles GDH [STW96] permettent à un groupe de personnes de se mettre d'accord sur une clé qui, si tout se passe bien, va leur servir à communiquer et à être certain que seule une personne de ce même groupe va pouvoir décrypter leur message.

1.1 Un rapide rappel du protocole Diffie-Hellman

Le protocole Diffie-Hellman [DH76] se réalise dans un groupe cyclique $\mathcal{G}$ d'ordre g et de générateur k où on suppose que l'hypothèse décisionelle de Diffie Hellman (DDH) est vérifiée. (Il n'est pas simple de retouver k^{ab} en connaissant k^a et k^b). On considère le sous-groupe de $\mathbb{Z}/p\mathbb{Z}^*$ de générateur g avec $p = 2g + 1$ où p et g sont premiers (ce qui permet d'avoir un sous-groupe d'ordre maximal, donc empêche une attaque de la clé par recherche exhaustive).

Si Alice et Bob veulent se mettre d'accord sur un secret via ce protocole, ils génèrent chacun une valeur aléatoire privée (notons-les respectivement a et b) comprise entre 1 et g, puis chacun calcule g^r, où r représente la valeur aléatoire générée, et rend public ce résultat. A ce moment là, Alice récupère le résultat de Bob et calcule $(g^b)^a$ et Bob, avec le résulat d'Alice, calcule $(g^a)^b$. Au final ils vont tous les deux avoir $K_{AB} = g^{ab}$.

$$A \underset{g^b}{\overset{g^a}{\rightleftarrows}} B$$

FIG. 1 – Le protocole DH

1.2 Généralisations à n participants

Il existe diverses façons de généraliser ce protocole. Cependant elles ont des points communs : elles utilisent toutes au moins un aléatoire R_i généré par chaque participant, généralement elles utilisent des clés à long terme K_{ij} sur lesquelles M_i et M_j se sont mis d'accord avant l'échange, au final la clé commune comporte les valeurs aléatoires de tous les participants.

Ci-dessous deux exemples à trois participants auxquels nous nous sommes intéressés, le premier car il est attaquable, le second car il ne l'est pas. Sur le schéma ci-dessous, la colonne i représente la succession des messages émis et reçus par l'utilisateur M_i

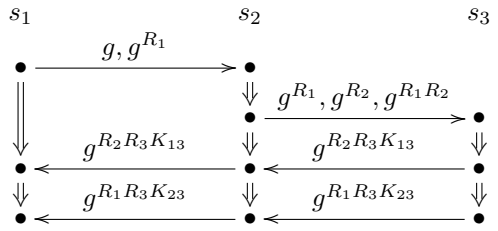


FIG. 2 – L'exemple du protocole A-GDH.2

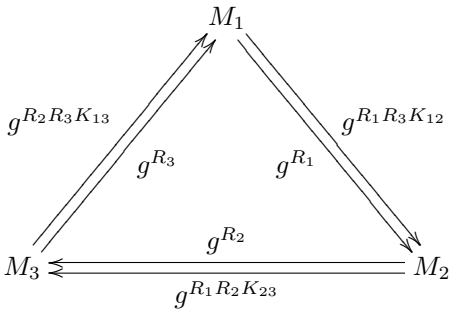


FIG. 3 – Un autre exemple : le protocole Tri-GDH

Cette représentation du protocole Tri-GDH, est particulière car on y voit deux phases où les trois participants peuvent envoyer leur message simultanément. Dans un premier temps, on réalise les échanges qui correspondent aux flèches internes, puis dans un deuxième ceux des flèches externes.

1.3 Méthodes de description d'un protocole

Lorsqu'on regarde ces schémas on peut concevoir au moins deux types d'approches pour représenter ces protocoles :

Soit on s'intéresse à ce que perçoit chaque utilisateur (les colonnes sur le schéma de l'A-GDH.2), on parle alors de *strands* (de traces), on pourrait alors décrire l'A-GDH.2 de la façon suivante (où + représente un envoi et − une réception) :

$s_1 = \left(+g, +g^{R_1}, -g^{R_2R_3K_{13}}, -g^{R_1R_3K_{23}}\right)$
$s_2 = \left(-g, -g^{R_1}, +g^{R_1}, +g^{R_2}, +g^{R_1R_2}, -g^{R_2R_3K_{13}}, -g^{R_1R_3K_{23}}\right)$
$s_3 = \left(-g^{R_1}, -g^{R_2}, -g^{R_1R_2}, +g^{R_2R_3K_{13}}, +g^{R_1R_3K_{23}}\right)$

Soit on s'intéresse à l'histoire des clés, la suite de flèches qui permet d'élaborer la clé, dans ce cas on peut décrire l'A-GDH.2 ainsi (où (s_i, j) représente le j^{eme} noeud de s_i :

$\pi_1 = ((s_1,1),(s_2,1),(s_2,2),(s_3,1),(s_3,2),(s_1,2))$
$\pi_2 = ((s_1,1),(s_2,1),(s_2,2),(s_3,1),(s_3,3),(s_2,4))$
$\pi_3 = ((s_1,1),(s_2,1),(s_2,2),(s_3,1))$

On remarque que ces deux approches se complètent bien, et permettent de retrouver toutes les informations décrites sur un schéma, relativement simple à programmer sur un ordinateur.

Comme on veut par la suite pouvoir lancer un programme pour attaquer un protocole il a fallu trouver un moyen de représenter ces protocoles en mémoire et aussi (et même surtout), un moyen simple pour l'utilisateur de les saisir. Les *strands* et les histoires semblent être une bonne solution pour la représentation en mémoire, car ils sont simples à manipuler et décrivent entièrement le protocole.

Cependant un utilisateur ne va pas taper ces deux séquences, principalement parce qu'il n'aura pas forcément calculé les différentes histoires. Mais par contre il connaît la description du protocole, donc une version améliorée des *strands* ; il connaît les *strands* avec en plus l'émetteur et le récepteur de chaque message. On va donc attendre ces informations en entrée du programme. Il a donc fallu introduire une convention pour que l'utilisateur puisse décrire ces *strands* avec des informations en plus et la perfectionner pour qu'elle prenne en compte le plus de cas possibles (En annexe A, on peut trouver une explication de ce pseudo-langage).

Pour simplifier l'analyse de la description, on ne peut pas dire qu'un utilisateurs envoie deux messages à la fois, cependant ils peuvent être représentés sans perte de spécificité par deux envois successifs. Par exemple, sur l'A-GDH.2 s_1 peut envoyer g puis g^{R_1} et non les deux à la fois.

On attend une certaine logique de la part de l'utilisateur quand il saisit le protocole ; une saisie *strand* par *strand*, ce qui correspond à une saisie de haut en bas puis de gauche à droite avec les *bundles* (schémas de représentation des protocoles munis d'une topologie adaptée), ou alors une saisie par "étage", de gauche à droite puis de haut en bas, ou enfin une saisie par histoire. Cela évite d'avoir à faire des tris hasardeux, car il se peut que l'algorithme ait des choix à faire et en faisant de mauvais choix, il risque de créer des éléments compliquant l'attaque, ou même de la rendre inadaptée puisqu'alors elle correspondrait à un autre protocole.

Il faut également faire attention quand on décrit un protocole où certains envois peuvent avoir deux antécédents possibles à spécifier quel est l'antécédent à choisir, parce qu'également le programme risque de ne pas choisir le bon, créant des complications. Sur l'A-GDH.2, s_1 envoie g au début du protocole à s_2, si on ne spécifie rien l'algorithme ne va pas savoir si par la suite s_2 va envoyer g^{R_2} en exponentiant le g reçu par R_1 ou en utilisant le g commun aux participants.

2 L'attaque

2.1 Théorie de l'attaque

Le but étant d'avoir une méthode automatisée pour se faire passer auprès d'un des intervenants comme un membre du groupe avec lequel il désire communiquer, on doit chercher un moyen de trouver le dernier calcul fait par la personne ciblée pour générer sa propre clé. Si, par exemple, on attaque M_1, et que son dernier calcul est de mettre le message reçu à la puissance $R_1 K_{12}^{-1}$ alors on doit réussir à avoir (g_1, g_2) tel que $g_2 = g_1^{R_1 K_{12}^{-1}}$. On doit alors trouver une méthode d'attaque permettant d'obtenir ce couple et trouver des conditions suffisantes pour qu'elle fonctionne.

Pour cela, on va avoir besoin de quelques notions supplémentaires sur les *strands* et les *bundles*.

Définition 2.1 *Les* contributions *notées* $C(M_i \rightarrow M_j)$ *correspondent au produit des calculs effectués sur le* strand s_i *dans l'histoire* π_j.

Dans [PQ06], il a été démontré que pour tout i, j, k,

$$\begin{aligned} p = P(\pi_i(\bar{0})) = C^{-1}(M_i \rightarrow M_i).C(M_i \rightarrow M_j) \\ .[S_j \setminus M_I : C^{-1}(M_i \rightarrow M_j).C(M_i \rightarrow M_k)] \\ . \prod_{M_l \in S_k} [S_j \setminus M_I : C^{-1}(M_l \rightarrow M_j).C(M_l \rightarrow M_k)] \\ . \prod_{M_l \in S_j} [S_k \setminus M_I : C^{-1}(M_l \rightarrow M_i).C(M_l \rightarrow M_j)] \\ . \prod_{M_l \in M} K_{Il}^{e_l} \end{aligned}$$

Avec S_j et S_k deux ensembles disjoints tels que $M_k \in S_j$, $M_j \in S_k$; S_j, S_k et $\{M_i\}$ forment une partition de M et les e_l tels qu'il n'y ait plus aucun facteur en K_{Il}, pour éliminer les clés que l'attaquant M_I a générées ; comme il les connait, il peut les enlever de l'exposant.

Pour expliquer comment se servir de cette formule pour collecter les contributions, on va d'abord définir deux termes qui sont à l'origine de difficultés.

Définition 2.2 *Les* splitting points *ou* split *sont les points de séparations de deux histoires* π_i *et* π_j.

Les starting points *ou* start *sont les points de départ des histoires, on les note* $\pi_i(1)$. *On note* $\pi_i(\bar{i})$ *le* i^{eme} *noeud d'une histoire en partant de la fin, qui est notée* $\pi_i(\bar{0})$. *Et* $P(\pi_i(j))$ *correspond au calcul fait à ce noeud-ci de l'histoire, c'est-à-dire, l'exposant du message émis par celui du message reçu au noeud précédent de l'histoire. Pour* $\pi_i(\bar{0})$, *on quotiente l'exposant de la clé par celui du dernier message reçu. On a bien ainsi le dernier calcul fait pour avoir la clé par l'utilisateur* i.

Il y a des histoires qui peuvent ne pas avoir de *splitting point*, comme c'est le cas pour (π_1 , π_2) dans le Tri-GDH.

Supposons que l'attaquant possède un élément h du groupe généré par g et souhaite obtenir $h^{C(M_i \rightarrow M_j)}$, où $C(M_i \rightarrow M_j)$ est la contribution apportée par M_i à l'histoire π_j sur le strand s_i. Pour ce faire, il va interagir avec le strand s_i, de la manière suivante. Pour chaque noeud négatif de s_i, il va examiner si ce noeud se trouve sur π_j. Si c'est le cas, il envoie le terme h à ce noeud, sinon, il envoie un élément de son choix. Pour chaque noeud positif de s_i envoyant un terme h',

il examine à nouveau si ce noeud se trouve sur π_j. Si c'est le cas, il met à jour la valeur de h en posant $h := h'$, sinon, l'attaquant ne se soucie pas du message transmis. Lorsqu'on arrive ainsi à la fin du strand s_i, on observe que h a été successivement exponentié par M_i avec tous les éléments aléatoires et clés contenus dans $C(M_i \rightarrow M_j)$, et a donc la valeur souhaitée.

Considérons, par exemple, l'A-GDH.2. On va calculer $C(M_1 \rightarrow M_3)$, les exponentiations faites par M_1 pour l'histoire π_3. On remarque que π_3 ne contient qu'un seul noeud sur s_1, et que de celui-ci, on envoie g^{R_1}. Dès lors, $C(M_1 \rightarrow M_3) = R_1$. De cette façon, on trouve le tableau suivant :

	M_1	M_2	M_3
M_1	$R_1 K_{13}^{-1}$	R_1	R_1
M_2	R_2	$R_2 K_{23}^{-1}$	R_2
M_3	$R_3 K_{13}$	$R_3 K_{23}$	R_3

TAB. 1 – Tableau des contributions de l'A-GDH.2

On peut en déduire le protocole d'attaque ; cependant, il se peut qu'on ait à envoyer au cours de celui-ci des informations que nous ne possédons pas (ou pas encore). Il va donc falloir réussir à trouver des (i, j, k) ainsi que les ensembles correspondants pour n'avoir à envoyer que des messages avec des exposants connus ou calculables, et donc pouvoir réellement les envoyer. Ces difficultés proviennent des *splitting points* et des *starting points*. Supposons que pour une attaque de l'A-GDH.2 il faille mettre un message à la puissance R_1, comme la seule exponentiation à cette puissance se fait en début d'histoire, il va être impossible de choisir le message à exponentier. Le problème des *splitting points* est différent, il se peut qu'au cours de l'attaque il faille envoyer deux messages différents à partir du même noeud.

Définition 2.3 *On a un start$^+$, si et seulement si on a dans p : $C(M_i \rightarrow M_j)$ où start(π_j) se trouve sur s_i.*

De même on a un start$^-$ si et seulement si on a dans p : $C^{-1}(M_i \rightarrow M_j)$ avec start(π_j) sur s_i.

P contient un splitting point si et seulement si il y a $C^{-1}(M_a \rightarrow M_b).C(M_a \rightarrow M_c)$ où split(π_b, π_c) se trouve sur s_a.

Proposition 2.4 *Considérons un protocole GDH à n participants et p tel que défini précédemment. Un attaquant actif peut alors obtenir, sans les complications expliquées ci-dessus, une paire (g_1, g_2) telle que $g_2 = g_1^p$ si une des conditions suivantes est vérifiée :*

- *p contient au plus un splitting point et aucun starting point*
- *p ne contient aucun splitting point, un start$^+$ et aucun start$^-$*
- *p ne contient aucun splitting point, aucun start$^+$ et un start$^-$*
- *p ne contient aucun splitting point, un start$^+$ (en i_+), un start$^-$ (i_-), $k_{i_-} = k_{i_+}$ et $l_{i_-} = l_{i_+}$.*

Il a fallu préciser un peu cette définition pour que le théorème soit vrai. On ne prend pas en compte les *start$^+$* et *start$^-$*, quand, dans le produit $C^{-1}(M_a \rightarrow M_b).C(M_a \rightarrow M_c)$, on avait un *splitting point* entre b et c (qu'il soit ou non sur a) car sinon on excluait des cas viables.

Grâce à ces règles seules, il est toujours possible de trouver (i, j, k) et les ensembles S_j, S_k dans un protocole à au moins cinq participants.

A quatre participants, il y a neuf configurations dans lesquelles aucune de ces règles ne peut être vérifiée. Lors de mon stage, j'ai remarqué que ces configurations pouvaient êtres décrites simplement

comme étant les uniques quadruplets (s_a, s_b, s_c, s_d) avec $(a, b, c, d) \in \{1, 2, 3, 4\}^4$ tels que les indices soient tous différents deux à deux et $\forall i \in \{1, 2, 3, 4\}$, s_i n'est pas en i^{eme} position ; on place en i^{eme} position, le *strand* où se trouve *start(π_i)*.

Cependant, un autre résultat de [PQ06] prouve que dans les cas restants, on peut encore choisir (i, j, k) et les ensembles S_j, S_k ; pour cela on introduit une relation de précédence $\prec$ définie comme suit :

Définition 2.5 *On dit que $C(M_a \rightarrow M_b) \prec C(M_a \rightarrow M_c)$ si et seulement si le premier point de π_j sur s_i se trouve plus haut que le premier point de π_k sur s_i.*

Proposition 2.6 *La condition suivante est, elle aussi, suffisante pour avoir la même conclusion que la proposition 2.4*

- *p ne contient aucun splitting point, un start$^+$ (i_+), un start$^-$ (i_-), tels que $C(M_{j_{i_-}} \rightarrow M_{k_{i_-}}) \prec C(M_{j_{i_-}} \rightarrow M_{l_{i_-}})$ ou $C(M_{j_{i_+}} \rightarrow M_{l_{i_+}}) \prec C(M_{j_{i_+}} \rightarrow M_{k_{i_+}})$.*

Avec ces cinq règles, il est donc possible d'effectuer la tâche qui nous est assignée, lorsque le protocole initial possède au moins quatre participants.

Le cas à trois participants se révèle plus ardu. Les règles précédentes ne permettent plus de conclure. C'est ce cas-là auquel je me suis intéressé plus particulièrement.

2.2 D'un utilisateur attaquable à $n - 3$

On vient de voir que les protocoles à n participants pour n plus que grand que quatre pouvaient être attaqués. Cependant on sait seulement qu'un des participants est attaquable. Un raisonnement élémentaire permet de déduire le théorème suivant.

Théorème 2.7 *Dans un protocole GDH à n participants il est possible de choisir sa victime parmi au moins $n - 3$ d'entre eux.*

Ceci permet essentiellement de montrer que l'attaquant a un large choix de victimes pour tous les protocoles. Un autre résultat intéressant permet de dire qu'il peut attaquer $n/4$ participants simultanément.

2.3 Mise en œuvre pratique

Une fois la théorie assimilée, il a fallu l'implémenter. J'ai choisi l'Ocaml, car il pouvait faire ce dont on avait besoin (lire dans fichier, calculer, écrire dans un fichier), tout en restant assez clair pour corriger facilement les erreurs.

La production du code s'est faite par étapes.

Tout d'abord un module qui générait le *bundle* correspondant aux histoires données en remplaçant automatiquement les exposants trop grands, créant une légende à côté, et alignant les premiers et derniers points de chaque *strand*. Ce module n'est pas vraiment utile pour l'attaque en elle-même, mais permet de générer des graphiques si on ne désire pas utiliser le code LaTeX généré par le programme pour expliquer en détail la démarche de l'attaque.

Ensuite un module, qui convertissait un fichier texte décrivant le protocole en *strands* et histoires. Comme le texte à analyser était relativement simple, une grammaire n'était pas utile, il suffisait de comparer la chaîne de caractères à quelques formes de base pour s'en sortir.

Puis un module qui faisait l'attaque elle-même. Pour cela on se sert de l'algorithme décrit dans [PQ06]. Il a fallu gérer certaines notions sur les *strands* (*start, split, term, ...*) donc clarifier leur définition, pour pouvoir les faire comprendre à Ocaml.

Vint alors la phase de génération de (i, j, k) et des ensembles associés. Pour ce faire on se sert de la proposition 2.4 quand il y a cinq utilisateurs et plus ; on utilise en plus la proposition 2.6 quand il n'y en a que quatre. La détection du nombre d'utilisateurs se faisant simplement par comptage du nombre de *strands*. Dans le cas à trois utilisateurs on essaye ces deux propositions dans un premier temps, car si jamais leurs hyptohèses sont vérifiées, on est sûr de pouvoir conclure. On remarque également que si on ne teste que les cas où $S_j = \{M_k\}$ ou $S_j = M \setminus \{M_i, M_j\}$, on est sûr de trouver une solution dans les cas à quatre utilisateurs et plus, on peut alors alléger la recherche et ne pas avoir à gérer toutes les partitions possibles de M. (Comme à trois utilisateurs il n'y a pas d'autre partition possible, on est sûr de ne pas exclure des cas solubles).

L'attaque en elle-même est très courte, il y a au plus $2C_5^3$ série de tests pour déterminer les (i, j, k) et les ensembles, où une série de tests comporte moins de l tests où l est la longueur du plus grand *strand*. Donc au final, la création du protocole d'attaque est au pire linéaire par rapport au nombre d'étapes du protocole.

Pour finir, il a fallu générer le code LaTeX pour montrer ce que doit faire l'attaquant. En annexe A, on trouve un fichier de description de protocole.

3 Traitement du cas à trois participants

Par défaut le programme, quand on lui donne trois participants, tente de faire comme dans le cas à quatre participants. Cependant comme il est possible qu'il n'y arrive pas, on aimerait pouvoir prédire les cas où il échoue, et en réduire le nombre en trouvant par exemple d'autres conditions suffisantes pour avoir la même conclusion que la proposition 2.4.

Comme dans le cas à quatre participants, il y a neuf cas particuliers, il pourrait être intéressant de s'y attarder. Car si on trouve un moyen, autre qu'une recherche exhaustive, de trouver des (i, j, k), S_j, S_k convenables alors peut-être y aura-t-il des cas à trois participants où on pourra appliquer les mêmes méthodes.

3.1 Les neufs cas particuliers à quatre participants

Au cours de l'implémentation, j'avais remarqué un moyen de décrire les neuf cas problèmatiques. Ceci pourrait, à la limite, permettre de déceler certains cas particuliers à trois participants, mais sans permettre de les résoudre. La recherche d'une formule permettant de relier les solutions aux différentes configurations des cas particuliers pouvait peut-être nous permettre d'obtenir des résultats supplémentaires à trois participants...

Pour avoir plus de chances de comprendre la logique des solutions, j'ai évité de reprendre celles de [PQ06], mais j'ai essayé d'en trouver de nouvelles (disponibles en annexe B), et surtout une fois que j'en ai eu une, j'ai essayé d'en déduire des solutions d'un autre cas. Ci-dessous, on trouve la justification de la solution du deuxième cas. (Toutes les preuves sont similaires à celle-ci).

On a π_1 qui commence en s_2, π_2 en s_3, π_3 en s_4 et π_4 en s_1. Lorsque $(i, j, k) = (1, 3, 4)$, on a un $start^+$ avec le couple $C^{-1}(M_3 \rightarrow M_1).C(M_3 \rightarrow M_4)$ et un $start^-$ avec le couple $C^{-1}(M_4 \rightarrow M_1)$ $.C(M_4 \rightarrow M_3)$. A ce moment-là, soit il y a une relation de précédence comme dans la proposition 2.6 et c'est bon, soit il n'y en a pas et alors d'après la définition des chemins, des "non-précédences" trouvées et du fait que $C(M_4 \rightarrow M_1)$ contienne un *start*, on a

$$\pi_3(1) \prec C(M_4 \rightarrow M_2) \preceq \pi_1(1).$$

Ensuite on utilise un raisonnement similaire avec la deuxième solution proposée $(i, j, k) = (3, 1, 2)$, on a un $start^+$ avec le couple $C^{-1}(M_1 \rightarrow M_3).C(M_1 \rightarrow M_2)$ et un $start^-$ avec le couple $C^{-1}(M_2 \rightarrow M_3).C(M_2 \rightarrow M_1)$. A ce moment-là, soit il y a une relation de précédence comme dans la proposition 2.6 et c'est bon, soit il n'y en a pas et alors d'après la définition des chemins, des "non-précédences" trouvées et du fait que $C(M_2 \rightarrow M_3)$ contienne un *start*, on a

$$\pi_1(1) \prec C(M_2 \rightarrow M_1) \preceq \pi_3(1).$$

Ce qui bien entendu est absurde car on ne peut avoir simultanément :

$$\pi_1(1) \prec \pi_3(1),$$
$$\pi_3(1) \prec \pi_1(1).$$

On est donc certain que l'une des deux solutions proposées vérifie les hypothèses de la proposition 2.6, puisque les seules suppositions faites étaient l'absence de relations de précédences ; ce qui permet de conclure que les (i, j, k) choisis et les ensembles associés conviennent.

Les solutions trouvées font intervenir à chaque fois les quatre utilisateurs, on ne va donc pas pouvoir les adapter simplement au cas à trois utilisateurs. Cependant la preuve nous permet d'avoir une autre approche, et peut nous inciter à rechercher des contradictions dans les relations de précédences.

3.2 Les 27 cas à trois participants

L'étude précédente n'est cependant pas totalement inutile, car elle permet d'avoir une idée de la manière de classer les protocoles à trois participants. On va donc les classer en fonction des *strands* de départ des différentes histoires. Il va y avoir trois catégories, les cas où il n'y aura pas de *splitting points*, on en dénombre six ; les cas où il y en a un unique, on en dénombre 18 (quand on traitera ces cas il faudra à chaque fois les découper en trois selon le *strand* où se trouve le *splitting point*), et enfin les trois cas où il y a trois *splitting points*, qui eux se redécouperont en 21 sous-cas...

En se contentant d'appliquer les règles trouvées lors des études précédentes, je suis parvenu à résoudre sept cas sur les 27 présents (leurs solutions se trouvent en annexe C), tous sauf un possèdent un unique *splitting point*, les six cas solubles avec un *splitting point* sont d'ailleurs les cas où il existe m et n, $m \neq n$ tels que π_m commence sur s_m et π_n commence sur s_n. La règle trouvée pour les cas particuliers du cas à quatre participants ne permet pas de conclure directement sur certains cas dépourvus de *splitting point* ; on a cependant deux triplets (i, j, k) qui permettront d'avoir des relations de précédences qui, si elles ne sont pas vérifiées par le protocole, permettront de conclure à l'existence d'une attaque.

En s'inspirant de la règle supplémentaire on arrive à traiter les cas $\{(1,3,2);(2,1,3);(3,2,1)\}$, les cas sans *splitting point* où il existe k tel que π_k commence sur s_k. On arrive à trouver à chaque fois deux triplets (i,k,j) et (j,k,i) qui nous apprennent que soit on a une configuration attaquable, soit on a :

$$\pi_k(1) \prec \pi_i(1),$$

$$\pi_k(1) \prec \pi_j(1).$$

Dès lors, il suffit de regarder la relation de précédence entre $\pi_i(1)$ et $\pi_j(1)$, on sait qu'il n'y a pas égalité car il n'y a pas de *splitting point*. Une fois trouvée, on peut les ordonner, ce qui nous permet de savoir dans quel ordre calculer les contributions.

Ensuite les cas restants peuvent se regrouper en diverses familles de telle façon que si on sait résoudre un des cas de cette famille, alors il est possible de résoudre tous les cas de la famille.

On démontre qu'il y a une famille dans les cas à trois *splitting points*. On sait résoudre 15 sous-cas sur 21 pour chaque élément de la famille.

Il y a deux familles dans les cas à un *splitting point*, la famille composée des configuration où il existe m tel que π_m commence sur s_m, elle comporte six éléments qu'on sait résoudre dans deux cas sur trois (annexe D) ; on ne sait rien faire avec les éléments de l'autre famille.

Il reste une famille dans les cas sans *splitting point*, la famille qui contient les deux cas $\{(3,1,2);(2,3,1)\}$ correspond aux cas particuliers lors de l'étude à quatre participants, on sait qu'il existe des protocoles dans cette famille qui ne sont pas attaquables (le Tri-GDH, et son miroir par exemple), donc à priori l'étude de cette famille va être complexe.

3.3 Si on s'éloigne des contributions

Cette méthode permet certes de traiter certains cas, cependant on bloque assez rapidement. De plus si on considère le Tri-GDH et un protocole identique avec une phase supplémentaire au début où chacun envoie le générateur, on a deux protocoles avec les mêmes positions pour les *starting points* mais le Tri-GDH n'est pas attaquable alors que l'autre l'est.

Il a donc fallu trouver une autre solution.

Pour cela, on va oublier les contributions, celles-ci sont très pratiques mais cependant masquent certains aspects des protocoles. On va choisir une personne dans le protocole. On va regarder quelles

transformations elle est supposé faire après avoir reçu le dernier message pour pouvoir obtenir la clé.

Par exemple dans l'A-GDH.2 M_3 doit mettre le message reçu à la puissance R_3.

Dès lors on va avoir certains exposants à trouver, on va les placer dans une liste de contraintes. Notre but sera de la vider. On va alors chercher dans le protocole s'il y a des transformations qui permettent d'exponentier un message avec des puissances de la liste de contraintes. Généralement, on va en trouver plusieurs qui, en plus, seront accompagnées d'autres exposants, on va donc en choisir une, ajouter les exposants à la liste de contraintes et recommencer.

Dans l'A-GDH.2 pour éliminer R_3 on a soit le calcul qui met à la puissance R_3K_{13} soit R_3K_{23}. On a donc 2 sous-cas soit on doit éliminer K_{13} soit K_{23}, aucune des solutions n'est possible donc l'attaque échoue, donc l'utilisateur M_3 n'est pas attaquable.

Cette méthode possède un avantage si elle n'arrive pas à détecter une attaque alors le protocole n'est pas attaquable du tout. Comme pour l'utilisateur M_3 dans le cas de l'A-GDH.2. Par contre elle possède un inconvénient majeur elle peut se mettre à boucler car il se peut qu'il y ait des endroits avec plusieurs choix possibles, et il n'y a aucun moyen de déterminer le bon choix : celui qui ne fera pas boucler. Il n'y a pas de moyen de limiter le nombre de boucles car aucun résultat actuel ne dit que si un protocole n'est pas attaquable avec n sessions simultanées alors il n'est pas attaquable du tout. Donc on risque même de ne pas pouvoir attaquer certains protocoles à cause d'une saturation de la mémoire.

Conclusion

Ce stage a permis d'implémenter un moyen automatisé d'attaquer les protocoles GDH à quatre participants ou plus. Pour pouvoir simplifier l'utilisation de l'attaque au maximum, il a fallu développer un moyen simple de décrire les protocoles, et ensuite un moyen d'exporter le résultat ; générer un code LaTeX s'est rapidement présenté comme étant une bonne solution car il est facilement exportable, tout en restant simple à créer.

Il a permis, de plus, de mettre en avant le fait quand dans tous les protoocoles GDH à n participants, $n-3$ au moins étaient attaquables.

Une fois la méthode d'attaque comprise, la programmation a été simple ; la vision globale du problème permettant de bien comprendre les spécificités des différents modules.

L'étude du cas à trois participants s'est révélée intéressante, il a fallu reprendre tout ce qui avait été fait précédemment, inspecter chaque démonstration, pour voir où étaient les problèmes, et repréciser certaines notions pour pouvoir effacer certaines difficultés.

Ce qui s'est révélé particulièrement intéressant ici, n'était pas tant le coté "attaquer un protocole", mais plutôt l'idée de concevoir un moyen de permettre une attaque automatique, autrement dit de concevoir tout ce qui permettra à l'ordinateur de se débrouiller seul pour trouver ce qu'il doit faire. Au lieu d'appliquer un moyen d'attaquer le protocole, on attend de la machine qu'elle le génère.

Il serait bien de voir s'il ne reste pas des moyens de diminuer le nombre de cas non résolus à trois participants, soit en trouvant de nouvelles règles permettant de les résoudre, soit en arrivant à démontrer qu'il est impossible de le faire.

Remerciements

Ce stage a été effectué au département de Microélectronique de l'Université catholique de Louvain dans l'unité DICE sous la direction de M. Olivier Pereira et M. Jean-Jacques Quisquater que je remercie.

Le groupe crypto fait partie de l'unité DICE (Dispositifs Intégrés et Circuits Electroniques), il s'intéresse donc aussi à des approches hardware de la cryptographie, ce qui m'a permis de découvrir une facette de celle-ci.

Références

[AST00] Giuseppe Ateniese, Michael Steiner, and Gene Tsudik. New Multiparty Authentication Services and Key Agreement Protocols. *IEEE Journal of Selected Areas in Communications*, 18(4) :628–639, 2000.

[DH76] W. Diffie and M.E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22 :644 – 654, 1976.

[PQ06] Olivier Pereira and Jean-Jacques Quisquater. On the Impossibility of Building Secure Cliques-type Authenticated Group Key Agreement Protocols. *Journal of Computer Security*, 14(2) :197 – 246, 2006.

[STW96] Michael Steiner, Gene Tsudik, and Michael Waidner. Diffie - hellman key distribution extended to group communication. In *ACM Conference on Computer and Communications Security*, pages 31–37, New York, NY, USA, 1996. ACM Press.

[THG99] J. Thayer, J. Herzog, and J. Guttman. Strand spaces : Proving security protocols correct. *Journal of Computer Security*, 7(2/3) :191–230, 1999.

A Convention de représentation des protocoles

Pour décrire cette convention on va s'intéresser à la représentation de l'Ex-GDH dans celle-ci :

FIG. 4 – Représentation de l'Ex-GDH dans le pseudo-langage

```
 % Exemple de description Ex-GDH
1 - R1,2 > 2
% 1 envoie à 2 g à la puissance R1,2
1 - R1 > 2
% 1 envoie à 2 : g à la puissance R1
2 - R1,2R2K2|3 > 3
% 2 envoie à 3 g à la puissance RR1*R2* K23
% La barre Ki|j permet au programme (s'il y a plus de neuf personnes) de
% séparer correctement K121 K12|1 ou K1|21
% N'importe quel symbole hors chiffre et espace peut être employé à la place de |
2 - R1K2|3 > 3
2 - R1R2,1 > 3
3 - R1,2R2R3K1|3 > 1
3 - R1R3K2|3K2|3 > 2
```

Cet exemple illustre les diverses possibilités du pseudo langage. On a déjà la base, pour décrire M_1 envoie g^e à M_2, il suffit d'écrire 1 - e > 2, g n'étant pas utile dans l'attaque, le programme n'a pas à le connaitre (Par la suite, il a fallu faire attention à ne pas faire de multiplications d'exposants). Les espaces sont là pour aérer le code, une fonction les supprime automatiquement lors de l'analyse du code.

Il peut arriver que lors de la création des histoires, le programme ait plusieurs choix possibles, si on veut lui en forcer un, il est possible d'ajouter à coté de l'expéditeur, le numéro du noeud du *strand* en cours qui a reçu l'information dont il se sert. Par exemple M_2 veut envoyer $g_1^R R_2 K_{12}$ à M_3 sur son troisième noeud ; au noeud 1 de son *strand* il a reçu de la part de M_1 g_1^R et au noeud 2 $g^{R_1 K_{12}}$. Il a donc deux possibilités d'antécédent, ainsi si l'utilisateur indique 2(1) le programme va choisir le premier noeud comme antécédent, ce qui peut permettre d'éviter des *splitting points*. Par défaut il choisit le noeud le plus bas sur le *strand*.

Comme le nombre de participants n'est pas limité (l'algorithme devant fonctionner à coup sûr pour quatre personnes et plus) il a fallu introduire un moyen de permettre au programme de savoir si K_{123} est la clé à long terme de M_{12} et de M_3 ou de M_1 et M_{23}, d'où la présence du symbole | dans le programme ci-dessus. En fait tout caractère autre qu'un chiffre, une espace, un % et > convient.

L'algorithme se voulant le plus général possible, il a fallu aussi traiter les cas où certains utilisateurs utilisent plus d'une variable aléatoire d'où la virgule. Là encore tout caractère autre qu'un chiffre, une espace, %, - et > convient, de plus comme l'utilisateur risque d'avoir de nombreuses variables aléatoires à saisir, il est possible de mettre Rx au lieu de Rx,1, ce qui permet de gagner en temps et en lisibilité.

Ensuite on peut imaginer un protocole assez long ; dès lors l'utilisateur a la possibilité de saisir des lignes de commentaires en les faisant commencer simplement par un % pour pouvoir se retrouver dans son code, un % permet aussi de commenter la fin d'une ligne.

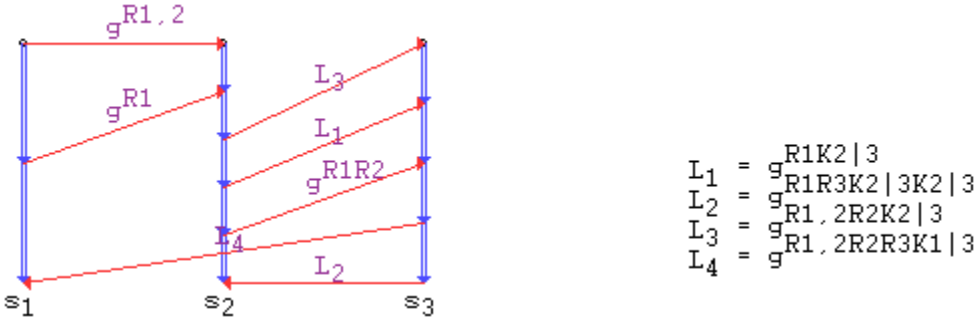
La ligne ==EOF== permet de dire à Ocaml d'arrêter l'analyse du fichier à cet endroit et ainsi laisse l'utilisateur libre de mettre des commentaires supplémentaires en dessous, en cas d'absence

de cette ligne, il y aura un arrêt normal à la fin du document.

Il est tout à fait possible de sauter des lignes pour aérer son code, ou de mettres des espaces en trop. Toutes les espaces présentes dans le document sont ignorées lors de la lecture, donc l'utilisateur peut mettre en forme son fichier de protocole s'il a besoin de l'exposer.

Une écriture plus standard de la forme Mi -> Mj : g ^ R1.R2 est aussi gérée.

La description ci-dessus fait générer le schéma suivant :



Ainsi que les schémas suivants (décrivant les échanges à faire) :

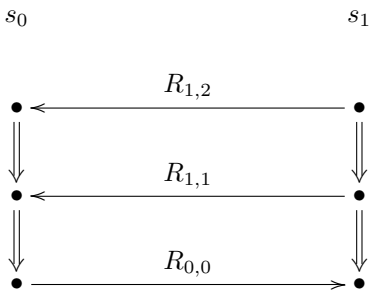


FIG. 5 – L'attaquant communique avec 1 en se faisant passer pour 2

$C^{-1}(M_1 \rightarrow M_2).C(M_1 \rightarrow M_3) = 1$

Ce message signifie qu'une des collectes de contributions est inutile, c'est pour celà qu'il n'y a que trois schémas d'échanges

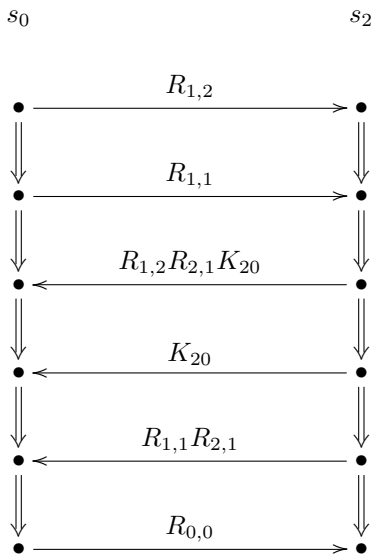


FIG. 6 – L'attaquant communique avec 2 en se faisant passer pour 3

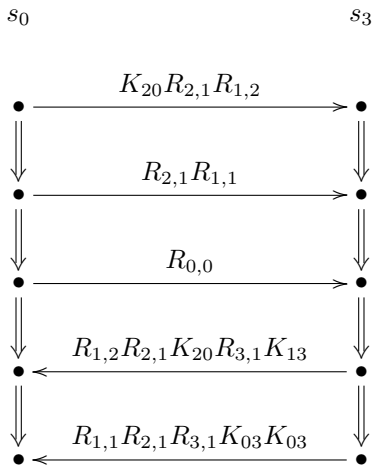


FIG. 7 – L'attaquant communique avec 3 en se faisant passer pour 2

La clé de session sera $R_{3,1}R_{2,1}R_{1,1}$

B Solutions des neuf cas particuliers à quatre participants

Le tableau ci-dessous indique les choix des i, j, k, S_j et S_k pour les neufs cas problématiques à quatre participants. Pour chaque cas, une des deux solutions proposées va permettre de conclure. La première partie indique le numéro du *strand* de départ de chaque histoire.

π_1	π_2	π_3	π_4	i	j	k	S_j	S_k
s_2	s_1	s_4	s_3	M_1	M_3	M_2	$\{M_2\}$	$\{M_3,M_4\}$
				M_3	M_1	M_4	$\{M_4\}$	$\{M_1,M_2\}$
s_2	s_3	s_4	s_1	M_3	M_1	M_2	$\{M_2\}$	$\{M_1,M_4\}$
				M_1	M_3	M_4	$\{M_4\}$	$\{M_2,M_3\}$
s_2	s_4	s_1	s_3	M_2	M_3	M_4	$\{M_4\}$	$\{M_1,M_3\}$
				M_3	M_2	M_1	$\{M_1\}$	$\{M_2,M_4\}$
s_3	s_1	s_4	s_2	M_3	M_2	M_4	$\{M_4\}$	$\{M_1,M_2\}$
				M_2	M_3	M_1	$\{M_1\}$	$\{M_3,M_4\}$
s_3	s_4	s_1	s_2	M_1	M_2	M_3	$\{M_3\}$	$\{M_2,M_4\}$
				M_2	M_1	M_4	$\{M_4\}$	$\{M_1,M_3\}$
s_3	s_4	s_2	s_1	M_1	M_2	M_4	$\{M_4\}$	$\{M_2,M_3\}$
				M_2	M_1	M_3	$\{M_3\}$	$\{M_1,M_4\}$
s_4	s_1	s_2	s_3	M_2	M_4	M_3	$\{M_3\}$	$\{M_1,M_4\}$
				M_4	M_2	M_1	$\{M_1\}$	$\{M_2,M_3\}$
s_4	s_3	s_1	s_2	M_2	M_1	M_4	$\{M_4\}$	$\{M_1,M_3\}$
				M_1	M_2	M_3	$\{M_3\}$	$\{M_2,M_4\}$
s_4	s_3	s_2	s_1	M_1	M_2	M_4	$\{M_4\}$	$\{M_2,M_3\}$
				M_2	M_1	M_3	$\{M_3\}$	$\{M_1,M_4\}$

Tab. 2 – Les neuf cas particuliers et des couples de solutions

C Sept cas pour lesquels l'attaque fonctionne toujours

Le tableau ci-dessous indique les choix des (i, j, k) pour les sept cas simples à trois participants. Comme précédemment on trouve dans la première colonne les *strands* de départ des histoires, dans la deuxième on trouve un triplet (i, j, k) convenant si le *split* se fait sur le *strand* s_l, sauf pour le premier cas où il n'y a aucun *split).*

π_1	π_2	π_3	s_1	s_2	s_3
s_1	s_2	s_3		(1,2,3)	
s_1	s_1	s_3	(2,3,1)	(2,3,1)	(2,1,3)
s_1	s_2	s_1	(3,2,1)	(1,3,2)	(3,2,1)
s_1	s_2	s_2	(2,3,1)	(3,1,2)	(3,1,2)
s_1	s_3	s_3	(3,2,1)	(2,1,3)	(2,1,3)
s_2	s_2	s_3	(1,3,2)	(1,3,2)	(2,1,3)
s_3	s_2	s_3	(1,2,3)	(1,2,3)	(1,2,3)

TAB. 3 – Solution des sept cas simples

D Trois cas particuliers

Le tableau ci-dessous indique les choix des (i, j, k) pour les trois cas particuliers à trois participants. Comme précédemment on trouve dans la première colonne les *strands* de départ des histoires, dans la deuxième on trouve les triplets (i, j, k) et (k, j, i) utiles.

π_1	π_2	π_3	(i, j, k)	(k, j, i)
s_1	s_3	s_2	(2,1,3)	(3,1,2)
s_2	s_1	s_3	(1,3,2)	(2,3,1)
s_3	s_2	s_1	(3,2,1)	(1,2,3)

TAB. 4 – Solutions des trois cas particuliers à trois participants

On peut remarquer qu'à chaque fois le l tel que π_l commence en s_l se trouve en deuxième position dans les deux triplets de solutions...

E Six cas où l'attaque fonctionne presque toujours

Le tableau 4 (ci-dessous) indique les choix des (i, j, k) pour les six cas particuliers à trois participants. Comme précédemment on trouve dans la première colonne les *strands* de départ des histoires, dans la deuxième on trouve un triplet (i, j, k) convenant si le *split* se fait sur le *strand* s_l. Les cas indiqués par ? ne peuvent être résolus avec les règles actuelles.

π_1	π_2	π_3	s_1	s_2	s_3
s_1	s_1	s_2	(3,2,1)	(3,2,1)	?
s_1	s_3	s_1	(3,2,1)	?	(3,2,1)
s_2	s_2	s_1	(1,3,2)	(1,3,2)	?
s_2	s_3	s_3	?	(1,3,2)	(1,3,2)
s_3	s_1	s_3	(1,2,3)	?	(1,2,3)
s_3	s_2	s_2	?	(1,2,3)	(1,2,3)

TAB. 5 – Morceaux de solutions des six cas particuliers

F Les trois histoires commencent au même endroit

Le tableau 5 (ci-dessous) indique les choix des (i, j, k) pour les cas où les trois histoires commencent sur le même *strand*. En première colonne, on indique les lieux des *splitting points*, dans la deuxième, on a les (i, j, k) comme précédemment. Le tableau est identique pour les cas où toutes les histoires commencent sur s_1, sur s_2 ou sur s_3. Trivialement, on montre que les trois *splits* se font au plus sur deux *strands* différents, les ✠ représentent les *splits* impossibles et les ? représentent les cas où les régles actuelles ne permettent pas de conclure ...

split$(1, 2)$	split$(1, 3)$	s_1	s_2	s_3
s_1	s_1	(3,1,2)	(3,1,2)	(3,1,2)
s_1	s_2	?	(3,1,2)	✠
s_1	s_3	(2,1,3)	✠	(2,1,3)
s_2	s_1	(3,2,1)	(3,2,1)	✠
s_2	s_2	?	(3,2,1)	(3,2,1)
s_2	s_3	✠	(1,2,3)	(1,2,3)
s_3	s_1	?	✠	(2,1,3)
s_3	s_2	✠	?	?
s_3	s_3	?	(1,2,3)	(1,2,3)

TAB. 6 – Quand les trois histoires commencent au même endroit

Il y a une certaine symétrie. On trouve, à chaque fois, quatre cas non soluble quand $split(a, b)$ se fait sur c. Il y a bien sûr recouvrement des cas...