DEA IMA
Algorithmique, complexité et cryptographie

POINTCHEVAL David

Rapport de stage

SCHÉMAS D'AUTHENTIFICATION A CLÉ PUBLIQUE EXIGEANT PEU DE RESSOURCES

mars – juin 1993

LAIAC
Laboratoire d'Algorithmique
et d'Intelligence Artificielle de Caen
Université de Caen et ISMRA

Responsable :
Brigitte Vallée

Dans le cadre d'un contrat avec le SEPT
(Service d'Etudes communes de la Poste et de France Télécom)

Remerciements

Je tiens à remercier :

- Brigitte Vallée pour l'encadrement très présent qu'elle a accepté d'assumer. Elle m'a aidé tout au long de ce stage, notamment pour rédiger ce rapport (et particulièrement, pour le rendre plus digeste)
- Philippe Toffin, avec qui j'ai partagé également le bureau
- Marc Girault, avec qui j'ai eu des discussions très constructives
- Philippe Béguin, pour toutes les discussions que nous avons pu avoir au cours de ce stage
- tous les membres des départements de Mathématiques et d'Informatique de l'université de Caen qui m'ont très bien accueilli.

Introduction

La cryptographie a beau être une science très ancienne, la recherche en ce domaine est très récente (la cryptographie moderne ne date que de 1976). C'est le développement informatique qui a rendu son emploi indispensable pour protéger les accès, sans pour autant les interdire :

On a de plus en plus souvent besoin de prouver son identité, pour accéder à un réseau par exemple, ou de signer un message que l'on envoie par un tel réseau.

Depuis le premier protocole à clé publique (RSA en 1978), de nombreux autres protocoles ont été inventés, permettant ces preuves. Cependant, tous ceux-ci sont trop lourds pour être implémentés efficacement, tels quels, sur une carte à puce, interface de plus en plus courante.

Ce stage, effectué au sein du LAIAC (Laboratoire d'Algorithmique et d'Intelligence Artificielle de Caen), laboratoire commun à l'Université de Caen et à l'ISMRa, s'est déroulé dans le cadre d'un contrat avec le SEPT (Service d'Etudes communes de la Poste et de France Télécom).

Il avait pour but d'améliorer les schémas existants, quitte à perdre un peu de sécurité, pour les rendre utilisables sur processeurs à faibles caractéristiques.

Les orientations et les caractéristiques du matériel ont été définies par le SEPT.

Le plan de ce rapport est alors le suivant :

- Liste exhaustive des schémas d'authentification
  (description précise et comparaison des services rendus)
- Amélioration des schémas existants

Partie I

Liste exhaustive des schémas d'authentification

Description précise et comparaison des services rendus

Introduction

Dans cette première partie, nous tentons de répertorier les principaux schémas d'authentification à clé publique, afin de comparer les services qu'ils rendent, le temps qu'ils mettent à l'exécution, ainsi que l'espace mémoire qu'ils nécessitent. On désire en comparer les caractéristiques afin d'extraire, de cette liste, les schémas les plus adaptés à une implémentation sur une carte à puce.

Tout d'abord, nous donnons les définitions principales et rappelons les principes fondamentaux de la cryptographie à clé publique :

- Définition(s) d'un schéma d'authentification.
- Classification des problèmes "difficiles" utilisés.
- Description précise des principales opérations nécessaires.

Dans une seconde étape, nous passons en revue les principaux schémas en les regroupant selon le type de problème algorithmique qui constitue leur fondement, en détaillant leurs caractéristiques respectives.

Ensuite, dans un troisième temps, nous présentons des idées complémentaires, plus ou moins différentes des schémas classiques, mais qui, associées à ces derniers, peuvent donner de très bons résultats.

Enfin, nous terminons par un bilan sous forme de tableaux, regroupant les principaux schémas et leurs caractéristiques, dans des termes comparables, afin de permettre une sélection selon les moyens offerts.
Ces tableaux sont la conclusion de cette partie, et la base de la partie suivante.

1 Généralités

1.1 Qu'est-ce qu'un schéma d'authentification ?

Les schémas d'authentification –appelés aussi schémas d'identification– résolvent ou tentent de résoudre le problème général suivant :

Comment prouver qui on est ? Comment donner la preuve de son identité?

Ces schémas sont appelés aussi quelquefois schémas de signature car ils cherchent à remplacer de manière logicielle la signature qu'on appose au bas d'une lettre. Dans la description que nous ferons par la suite, il y aura toujours au moins deux individus : A pour Alice, et B pour Bob. Les rôles sont distribués une fois pour toutes; Alice est la personne active : c'est elle qui signe ou qui, plus généralement, veut faire la preuve de son identité. Bob, lui, joue un rôle plus passif; c'est le destinataire qu'Alice doit convaincre.

1.1.1 Les différents types de schémas.

On différencie plusieurs types de schémas

- selon qu'il existe un message accompagnant cette déclaration d'identité.
- selon qu'il existe un juge extérieur au couple (A, B) à qui Alice, Bob ou les deux doivent faire partager la conviction de la preuve d'identité. Ce juge sera désigné par C, diminutif de Charlie.

Ces schémas sont donc les suivants :

Identification (ou authentification d'entité).
Il n'y a ni message, ni juge, Alice doit faire la preuve à Bob que c'est bien elle qui communique avec lui.

Certification de messages (ou authentification de messages).
Il n'y a pas de juge, mais un message : Alice prouve à Bob qu'elle est l'auteur du message.

Signature de messages.
Il y a, à la fois, un message et un juge : La signature qui accompagne le message, dépendant du message et du signataire A, est une preuve qui doit entraîner la conviction du destinataire B, mais cette conviction doit également être partageable par toute entité extérieure au couple.

Ces trois situations sont donc à priori différentes, et les solutions envisagées seront distinctes. En particulier, lorsqu'il n'y a pas de juge, le schéma pourra être interactif et consister en un dialogue entre A et B; il est clair qu'une telle interaction ne pourra jamais constituer une preuve pour un juge externe C qui sera toujours tenté de croire qu'il y a une coalition entre A et B : un schéma de signature ne peut pas être interactif.

Le fait qu'il n'y ait pas de message ne simplifie pas forcément le problème. Comme l'on écarte le principe du mot de passe, A ne peut s'authentifier auprès de B toujours de

la même manière, sinon, un autre individu C, écoutant sur la ligne, pourrait par la suite se faire passer pour A. La preuve d'identité ne peut être seulement une fonction de A. Elle dépendra d'un aléa qui jouera, dans la vérification, le même rôle que le message.

1.1.2 Les différentes qualités cryptographiques d'un schéma.

Qui peut le plus peut le moins : si on sait résoudre le problème de la signature, on sait résoudre les deux autres problèmes. Une première mesure de la qualité d'un schéma est la *diversité des services* qu'il rend; en cela, un schéma de signature est souvent supérieur puisqu'il permet de résoudre le problème dans chacune des hypothèses. Mais comme souvent en cryptographie et en algorithmique en général, on cherche un compromis entre les qualités du service rendu et la quantité de moyens mis en oeuvre pour l'obtenir.

Essentiellement, les schémas d'authentification –au sens large du terme– sont fondés, dans la cryptographie à clef publique, sur le principe suivant :

> *Je suis A car je suis la seule à détenir un secret S_A, mais vous pouvez tous vérifier facilement que je possède ce secret. Cette vérification s'effectue à l'aide de ma clé publique que vous êtes tous à même de connaître.*

Mais il faut bien sûr que A prouve détenir ce secret sans rien en dévoiler. *La quantité d'information sur le secret que dévoile A* lors du déroulement du protocole est donc une seconde mesure de la qualité du schéma. En particulier, il existe des protocoles d'authentification où A peut faire la preuve de son identité sans donner aucune information sur son secret : on parle alors de protocole "zero-knowledge" ou encore de protocole *sans transfert de connaissance*.

On distingue plusieurs attaques contre un schéma d'authentification :

- l'attaque directe où l'on ne dispose que de la clé publique de A
- les attaques où l'ennemi dispose d'un certain nombre de messages accompagnés de la signature correspondante.

Si le protocole est sans transfert de connaissance, le cadre est alors le même pour chacune des attaques.

Dans ces attaques, on vise soit à retrouver le secret d'Alice pour pouvoir par la suite complètement signer à sa place, ou bien seulement s'identifier à elle, c'est-à-dire imiter sa signature ou sa preuve d'identité sans nécessairement connaître son secret.

On va donc classifier ces attaques selon l'attaquant et ce qu'il connaît, afin d'avoir de bons critères de comparaison :

(a) Charlie veut s'identifier à Alice, alors qu'il ignore le secret d'Alice.

(b) Charlie veut découvrir le secret d'Alice.

(c) L'autorité veut découvrir le secret d'Alice.

(d) L'autorité veut créer un nouveau couple (clé publique, clé secrète) associé à l'identité d'Alice.

(e) Charlie veut créer un nouveau couple associé à l'identité d'Alice, en d'autres termes, prendre la place de l'autorité.

Dans toutes ces attaques tentées par Charlie, Bob ne serait-il pas mieux placé, étant donné qu'il peut forcer le hasard lors d'une preuve interactive ?

Il faut remarquer que les résultats des attaques (d) et (e) sont très visibles et donc rapidement détectés : en effet, on constate la présence de 2 couples de clés associés à la même identité.

Bien évidemment, aucune protection contre l'attaque (d) ne peut être assurée, mais si, en revanche, l'attaque (e) est "quasi" impossible, l'autorité (ou un de ses membres) prendrait un énorme risque en falsifiant des clés, dans ce cas, l'attaque (d) peut être considérée comme inutile.

Par contre, si l'attaque (e) n'est pas protégée, l'attaque (d) devient tout à fait réaliste.

1.1.3 Les qualités algorithmiques d'un schéma.

Il y a aussi des qualités liées à la complexité des algorithmes utilisés dans le protocole : algorithme de production de la preuve d'identité, algorithme de vérification de la preuve. Il s'agit aussi bien de la complexité en temps que de la complexité en espace.

- *Complexité en espace.* On veut que la signature –ou plus généralement la preuve d'identité– soit courte en soi, ou courte par rapport au message qu'elle accompagne.
- *Complexité en temps.* On veut que cette preuve soit aisée à produire et aisée à vérifier.

Ces qualités sont particulièrement importantes dans le cadre qui nous intéresse ici, puisqu'on dispose de peu de ressources, à la fois en espace et en temps. Il faut alors s'attacher à rendre efficaces les opérations élémentaires utilisées :

- en utilisant des opérations câblées.
- en sous-traitant les opérations principales.
- en préparant à l'avance les calculs qui ne nécessitent pas la connaissance du message, ou de l'interaction de B (dans une phase de précalculs). D'ailleurs, dans la suite, on dénommera "précalcul" tout ce qui peut s'effectuer à l'avance (en particulier, la première étape d'un schéma interactif).

1.2 Les fondements de la cryptographie à clé publique.

Ici, nous nous intéressons à la cryptographie à clé publique : chaque individu possède une clé publique et une clé secrète.

Dans le problème de l'authentification, c'est la clé secrète d'Alice qui va lui servir pour s'identifier, tandis que les autres –Bob ou éventuellement un juge– utiliseront sa clé publique pour vérifier la preuve d'identité qu'elle a fournie.

1.2.1 Fonctions à sens unique.

L'algorithme de vérification de la preuve doit être très aisé pour toute personne possédant la clé publique d'Alice. Par contre l'algorithme de production de la preuve d'identité doit être pratiquement impossible à effectuer par toute autre personne qu'Alice : Il faut donc trouver des problèmes algorithmiquement faciles, qu'on tentera d'optimiser le plus possible, et des problèmes algorithmiquement difficiles. Par ailleurs, comme production de signature et vérification de signature sont des opérations inverses l'une de l'autre, les protocoles seront fondés sur l'existence de *fonctions à sens unique* : ce sont des fonctions faciles à calculer mais difficiles à inverser.

1.2.2 Gestion des clés.

Les clés publiques se trouvent en général dans un annuaire, et il faut, bien sûr, veiller à ce qu'elles ne soient pas modifiées. Si en effet Charlie modifie la clé publique d'Alice en mettant la sienne à la place, il pourra s'authentifier à la place d'Alice. Il y a essentiellement trois possibilités classiques pour résoudre ce problème, et ces trois techniques mettent en jeu ce qu'on appelle une autorité qui dispose, elle aussi, d'une clé publique et d'une clé secrète, et le plus souvent d'une fonction à sens unique.

- *certification des clés* - La clé publique de chaque individu peut être alors certifiée par l'autorité (qui sera alors désigné par KAC pour Key Authentication Center).
- *schéma fondé sur l'identité* - La clé publique de chaque utilisateur consiste en l'identité même de cet individu, la clé secrète de chacun est calculée par l'autorité en fonction de cette clé publique.
- *clés autocertifiées* - Le certificat de la clé publique n'est plus à vérifier par le vérifieur, sa validité est assurée par le bon fonctionnement du protocole d'identification. C'est-à-dire que si Bob accepte la preuve d'Alice, sans s'être préoccupé de la validité des clés, alors les clés sont correctes.

1.2.3 Protocoles interactifs

Un schéma interactif de preuve consiste en un dialogue entre Alice et Bob, Alice cherchant à prouver quelque chose à Bob, Bob posant des questions pour piéger un tricheur.

Par conséquent, un tel schéma doit avoir au moins deux propriétés fondamentales :

- *Complétude (completeness)* - La véritable Alice doit convaincre Bob avec une probabilité écrasante (aussi proche de 1 que l'on veut).

- *Consistance (soundness)* - Une fausse Alice doit être démasquée avec une probabilité comparable (aussi proche de 1 que l'on veut).

De façon plus formelle :
Soit n la longueur de l'entrée du schéma, alors

- Complétude : $\forall k \in \mathbf{N}^\star$

$$\Pr[\text{Bob convaincu par la véritable Alice}] \geq 1 - \frac{1}{n^k}$$

- Consistance : $\forall k \in \mathbf{N}^\star$

$$\Pr[\text{Bob convaincu par une fausse Alice}] \leq \frac{1}{n^k}$$

1.2.4 Protocoles "zero-knowledge"

On aimerait, en plus, que Bob n'apprenne rien du secret dont Alice prouve sa connaissance. C'est-à-dire que ce protocole soit sans transfert de connaissance *(zero-knowledge)*.

De façon intuitive, ceci est vérifié si Bob est capable de fabriquer, en temps raisonnable, un ruban contenant les informations échangées au cours du protocole. Bien évidemment, Bob doit parcourir les étapes en sens inverse pour fabriquer ce ruban.

Plus formellement, un système de preuve est *zero-knowledge* s'il existe une machine de Turing probabiliste polynomiale telle que la distribution de ses rubans de sortie soit polynomialement indistinguable de la distribution des rubans de sortie produits par ce système de preuve.

1.3 Les problèmes algorithmiquement difficiles utilisés.

Ces protocoles sont essentiellement fondés sur quatre problèmes qui utilisent la structure multiplicative du groupe $\mathbf{Z}^\star(n)$ formé par les éléments inversibles modulo l'entier n, premier ou composé –la factorisation entière, le problème de la racine carrée, le problème RSA, le problème du Logarithme Discret–. On a utilisé plus récemment d'autres problèmes qui n'utilisent plus la structure multiplicative modulaire, mais des problèmes "linéaires" qui reposent sur du calcul matriciel, toujours modulaire.

Nous rappelons l'énoncé des quatre problèmes multiplicatifs, puis l'énoncé des problèmes matriciels :

1.3.1 L'énoncé des problèmes.

Problème $FACT(n)$.

Soit un nombre n, produit de deux nombres premiers distincts p et q. On veut trouver ces facteurs p et q.

Problème $SQRT(n)$.

Soit un nombre n, produit de deux nombres premiers distincts inconnus p et q. On se donne alors un élément y de $\mathbf{Z}^\star(n)$ et on veut trouver –s'il existe– l'élément x qui vérifie $y \equiv x^2\ [n]$.

Problème $RSA(n, e)$.

Soient un nombre n, produit de deux nombres premiers distincts inconnus p et q, et un nombre e premier avec $\phi(n) = (p-1)(q-1)$. On se donne alors un élément y de $\mathbf{Z}^\star(n)$ et on veut trouver x qui vérifie $y \equiv x^e\ [n]$.

Variante "approchée" du Problème $RSA(n, e)$.

Soient deux nombres n et e. On se donne alors un élément y de $\mathbf{Z}^\star(n)$ et on veut trouver x qui vérifie $y \simeq x^e\ [n]$.

Problème du Logarithme discret.

Soient G un groupe fini, et b un élément du groupe dont l'ordre u est suffisamment grand. On note $< b >$, le groupe engendré par b. On se donne alors un élément y de $< b >$, et on veut trouver l'exposant x –défini modulo u– pour lequel on a $y = b^x$.

Là aussi, on utilise essentiellement les groupes $\mathbf{Z}^\star(n)$ et on note $LD(n, b)$ le problème du Logarithme Discret dans le groupe $\mathbf{Z}^\star(n)$ relatif à la "base" b.

Et en ce qui concerne les problèmes linéaires :

Problème du noyau permuté $PKP(p)$.

On travaille ici modulo un nombre premier p. Soit une matrice A à m lignes et n colonnes, et un vecteur V. Trouver une permutation π telle que le permuté V_π du vecteur V soit dans le noyau de A.

On considère également un autre problème qui n'est en fait qu'un cas particulier du précédent pour $p = 2$:

Problème des mots de poids faible dans les codes.

On travaille ici modulo 2. Soit une matrice A à m lignes et n colonnes. Trouver un vecteur V dans le noyau de A de poids donné d.

1.3.2 Les résultats algorithmiques concernant ces problèmes.

Tous ces problèmes sont réputés algorithmiquement difficiles : les problèmes de décision correspondants sont tous dans la classe NP, mais on ne connaît pas d'algorithmes polynomiaux pour les résoudre. L'hypothèse généralement admise est qu'il n'en existe pas. Ce postulat est fondamental pour construire, à partir de ces problèmes, des fonctions à sens unique. Pour ce qui est des deux problèmes linéaires, leur complètude a été prouvée (ils sont NP-complets), c'est-à-dire qu'ils sont plus difficiles que tout autre problème de la classe NP. Trois directions s'offrent à la recherche algorithmique sur ce domaine :

- On cherche des bornes inférieures de complexité.
- On cherche à construire de nouveaux algorithmes.
- On cherche seulement à comparer la difficulté relative de ces problèmes. De manière assez informelle, on dit qu'un problème R est plus difficile qu'un problème Q si l'existence d'un algorithme polynomial qui résout R entraîne l'existence d'un algorithme polynomial qui résout Q. On note en abrégé $R \geq Q$.

1.3.3 Construction de nouveaux algorithmes.

Il y a eu un progrès important au moins du point de vue de la complexité théorique [LLMP], pour les algorithmes résolvant $FACT(n)$ ou $LD(n)$ puisqu'on est passé d'une complexité en

$$\exp\,[\sqrt{\log(n)\log\log(n)}]$$

à une complexité en

$$\exp\,[\log(n)^{1/3}\log\log(n)^{2/3}].$$

Actuellement, on considère la factorisation d'un nombre de 512 bits infaisable en un temps raisonnable. Pour le Logarithme Discret, le module n de 512 bits s'il est premier, ou de 750 bits s'il est composé, et l'exposant x de 140 bits rendent le problème calculatoirement insoluble. La grande taille de l'exposant empêche la recherche exhaustive, ainsi que l'application de la méthode "des pas de bébé et pas de géant" [Sk] moins coûteuse. En effet, la recherche exhaustive est en $O(u)$, et cette dernière méthode n'est qu'en $O(\sqrt{u}\log u)$:

> Pour déterminer $\log_b y$, avec b d'ordre $u \leq s^2$, on sait qu'il existe $0 \leq i, j < s$
> tels que $y = b^{is+j}$.
> Par conséquent, on stocke la liste triée b^j pour $0 \leq j < s$,
> puis on calcule yb^{-is} jusqu'à ce que ce soit égal à un élément de la liste précédente.
> En effet, si $yb^{-is} = b^j$ alors $\log_b y = is + j$.

Toutefois, l'ordre u de b doit avoir au moins un grand facteur premier, car l'algorithme de Pohlig-Hellman [PH] permet de calculer le logarithme discret dans le cas contraire.

1.3.4 Comparaison de la Factorisation et du Logarithme discret.

On ne sait pas comparer exactement la difficulté de ces deux problèmes. On peut d'abord remarquer que ce sont les mêmes méthodes qui permettent actuellement de résoudre ces deux problèmes. C'est pourquoi on dispose actuellement d'algorithmes –voir 1.3.3– qui ont la même complexité.

De manière générale, dans le cas où l'entier n est le produit de deux nombres premiers p et q, on se donne un élément b d'ordre u dans $\mathbf{Z}^\star(n)$ qui a pour ordre v dans $\mathbf{Z}^\star(p)$ et w dans $\mathbf{Z}^\star(q)$. Rappelons que u est égal au ppcm de v et de w. Soit y un élément du groupe $\mathbf{Z}^\star(n)$ qui appartient au sous-groupe engendré par b. Supposons que l'on connaisse le logarithme x_p de y dans $\mathbf{Z}^\star(p)$ et le logarithme x_q de y dans $\mathbf{Z}^\star(q)$. Alors le logarithme x de y dans $\mathbf{Z}^\star(n)$ est connu modulo v et modulo w, et donc modulo le ppcm de v et w qui est justement u. Dans ce cas général, on a

$$[FACT(n) \text{ et } LD(b,p) \text{ et } LD(b,q)] \geq LD(b,n).$$

Plus récemment, Schrift et Shamir ont relié les deux problèmes $LD(n)$ et $FACT(n)$, toujours dans le cas ou l'entier n est le produit de deux nombres premiers p et q [SS]. Ils montrent que :

$$LD(b,n) \geq FACT(n).$$

1.3.5 Comparaison de la Factorisation et des racines e-èmes.

Lorsque le module est une puissance d'un nombre premier, le problème de la recherche des racines e-èmes est facile : on dispose d'algorithmes polynomiaux, déterministes ou probabilistes, pour résoudre ce problème. Il en est de même, via le théorème des restes chinois, lorsque le module n est composé de factorisation connue. On en déduit que $FACT(n)$ est plus difficile que chacun des trois problèmes $SQRT(n), RSA(n,e)$ et la version approchée de ce dernier.

Rabin a montré que les deux problèmes $FACT(n)$ et $SQRT(n)$ étaient en fait de même difficulté. On conjecture souvent qu'il en est de même pour les deux problèmes $RSA(n,e)$ et $FACT(n)$, mais ce n'est pas prouvé : c'est l'hypothèse RSA.

Le problème des racines e-èmes approchées est facile lorsque e est petit (2 ou 3), et ceci indépendamment de la factorisation du module n. Lorsque e est quelconque, le problème est également facile, même si le module n est de factorisation inconnue, pourvu que l'on connaisse une approximation x_0 du nombre x cherché. Par contre, on ne connaît pas d'algorithme polynomial pour résoudre le problème général : quand l'entier n est de factorisation inconnue, l'exposant $e \geq 4$, et qu'aucune approximation de la racine n'est donnée.

2 Description des principaux schémas

Nous décrivons maintenant les principaux schémas d'authentification, en les classant selon le type de problème difficile utilisé.

- les schémas classiques : RSA et Rabin fondés sur $SQRT(n)$ ou $RSA(n, e)$.
- le schéma ESIGN dû aux japonais Okamoto et Shirashi, puis ensuite à Fujioka, Okamoto et Miyaguchi, fondé sur la difficulté de la version approchée de RSA.
- le schéma de Fiat et Shamir, ainsi que ses trois variantes principales dues à Guillou-Quisquater, Ohta-Okamoto, et Ong-Schnorr : ce sont des schémas interactifs, sans transfert de connaissance fondés sur $SQRT(n)$ ou $RSA(n, e)$.
- le schéma de El Gamal revu par Schnorr : c'est un schéma interactif fondé sur le Logarithme discret.
- les schémas interactifs matriciels dus à Shamir, Stern ou Girault.

2.1 Utilisation des schémas à clé publique.

L'idée d'utiliser des systèmes à clé publique dans des schémas de signature semble due à Diffie et Hellman [DH].

2.1.1 L'idée générale.

Soient E_X et D_X les fonctions de chiffrement et de déchiffrement de l'utilisateur X. On suppose dans un premier temps que toutes ces fonctions ont le même domaine de définition et que E_X et D_X sont inverses l'une de l'autre.

Le schéma de signature se décrit de la manière suivante :

1. Alice signe le message m qu'elle veut envoyer avec $s = D_A(m)$ et elle envoie le couple (m, s).
2. Bob –ou tout autre utilisateur– vérifie que la signature est valide en calculant $E_A(s)$ et en vérifiant l'égalité de cette quantité avec m.

Il est facile de transformer un tel schéma en un schéma interactif d'authentification en remplaçant le message m par un aléa r envoyé par Bob.

1. Bob envoie un aléa r à Alice.
2. Alice répond avec $s = D_A(r)$.
3. Bob vérifie que la preuve d'identité est valide en calculant $E_A(s)$ et en vérifiant l'égalité de cette quantité avec r.

2.1.2 Le schéma de signature RSA - (Rivest, Shamir, Adleman 1978)

Ici, la sécurité du procédé réside dans la difficulté de l'inversion de la fonction élévation à la puissance e modulo n : $x \longmapsto x^e\ [n]$ lorsque la factorisation de n est cachée : c'est la difficulté du problème $RSA(n, e)$ [RSA].

Protocole

Chaque utilisateur dispose d'un couple formé d'un entier $n = pq$, où p et q sont deux nombres premiers distincts de même ordre de grandeur, et d'un entier e, premier avec $\phi(n) = (p-1)(q-1)$. Le couple (n, e) est rendu public par l'utilisateur. Celui-ci connaît de plus les entiers p et q qu'il garde secrets, et il peut donc calculer facilement l'inverse d de e modulo $\phi(n)$ et donc aussi l'inverse $x \longmapsto x^d\ [n]$ de la fonction $x \longmapsto x^e\ [n]$. Il peut donc calculer facilement –et lui seul semble pouvoir le faire– la fonction de déchiffrement $D : x \longmapsto x^d\ [n]$ tandis que chacun peut calculer la fonction de chiffrement $E : x \longmapsto x^e\ [n]$.

Valeurs des paramètres

Pour se protéger contre les algorithmes de factorisation connus, le module n doit être d'une longueur d'au moins 512 bits (et par conséquent, les entiers p et q seront de l'ordre de 256 bits).

L'exposant d (clé secrète) devant être difficile à trouver, il doit être quelconque. En revanche, l'entier e (clé publique) peut être aussi petit que l'on veut.

Complexité en temps

En général, on choisit e petit (dans le cadre des signatures, on prend couramment $e = 3$), par suite, d se trouve être presque aléatoire. Et donc, pour calculer la signature, Alice doit effectuer, en moyenne, 768 multiplications modulaires.

Complexité en espace

Données :	n, e, d	512×3 bits = 192 octets
	p, q	256×2 bits = 64 octets
Calculs :	Message	64 octets
	Valeurs intermédiaires	128 octets

Transferts

La signature et le message sont contenus dans $m^d\ [n]$, ce qui ne représente que 64 octets à transmettre par blocs.

Sécurité

(a) assuré par l'hypothèse RSA

(b)(c) sont assurés par la difficulté du problème $FACT$

(e) assuré si une autorité certifie les clés

Conclusion

Ce schéma présente un avantage rencontré nulle part ailleurs, il est fondé sur une permutation à trappe, et donc la quantité d'informations à transmettre est minimale (la longueur du message). Cependant, ces calculs de puissance sont coûteux en temps, et rendent ce schéma inutilisable sur des processeurs de faibles caractéristiques.

2.1.3 Le schéma de signature de Rabin (Rabin 1979)

La sécurité du procédé repose sur la difficulté d'extraire une racine carrée modulo un nombre composé : $SQRT(n)$, qui est prouvé être de même difficulté que $FACT$. Par conséquent, ce schéma est censé être plus sûr que le schéma précédent.

Protocole

Chaque utilisateur choisit un couple (p, q) de grands nombres premiers à partir desquels il calcule l'entier $n = pq$, ainsi qu'un entier $b < n$. Sa clé publique sera (b, n) et sa clé privée (p, q).
Tous les utilisateurs disposent d'une fonction de hachage commune : h
Soit m le message à signer par Alice.

1. Alice choisit un aléa u, et calcule $c = h(mu)$.

2. Elle vérifie si l'équation
$$c \equiv x(x + b)\ [n]$$
admet une solution, grâce au secret (p, q) qu'elle détient.

3. Si ce n'est pas le cas, elle choisit un autre aléa.
Remarque : Etant donnée la grande proportion de résidus quadratiques dans $\mathbf{Z}(n)$, elle trouvera un bon aléa assez rapidement.

4. Elle calcule donc une solution r de l'équation ci-dessus.

5. La signature qui accompagnera le message m sera alors (r, u).

6. La vérification s'effectue alors de la façon suivante :
A-t-on l'égalité $r(r + u) \equiv h(mu)\ [n]$?

Valeurs des paramètres

Les entiers n, p, q doivent être du même ordre de grandeur que dans le système RSA, car actuellement, ce sont les mêmes algorithmes qui permettent de résoudre l'un et l'autre des problèmes. Mais puisqu'Alice a des racines carrées à calculer, on a intérêt à prendre des valeurs, pour p et q, qui rendent plus aisé ce calcul. En effet, avec $p \equiv q \equiv 3\ [4]$, il existe un algorithme plus simple à mettre en oeuvre que dans les autres cas.

L'aléa u sert à obtenir un résidu quadratique à partir de m, par conséquent, un aléa de 8 bits suffit.

Complexité en temps

Pour trouver la racine x à l'équation posée ci-dessus, Alice doit extraire une racine carrée, ce qui revient, dans le cas particulier dans lequel on se trouve, à élever à une certaine puissance.

Par suite, Alice doit effectuer, en moyenne, 768 multiplications modulaires.

Complexité en espace

Données :	n, b	128 octets
	p, q	64 octets
Calcul :	c, r	128 octets

Transferts

Le message étant haché, il peut être de longueur quelconque.

Quant à la signature, elle est de 64 octets pour r et d'un seul pour u, soit un total de 65 octets à transférer.

Sécurité

(a) assuré par la difficulté de $SQRT$ (problème aussi difficile que $FACT$)

(b)(c) assurés par la difficulté de $FACT$

(e) assuré si une autorité certifie les clés

Conclusion

Ce schéma présente les mêmes inconvénients que le schéma RSA, tout en étant encore plus lent. En effet, il y a plus d'informations à transmettre, un certain nombre de multiplications à effectuer en plus. Cependant, ce schéma est fondé sur un problème à priori plus difficile que le précédent, il ne suppose pas "l'hypothèse RSA".

2.2 Le schéma ESIGN

(Okamoto, Shiraishi 1985) (Fujioka, Okamoto, Miyaguchi 1990)

2.2.1 Description du schéma

C'est un schéma qui est fondé sur le problème des racines e-èmes approchées modulo un nombre n composé. A l'origine, Okamoto et Shiraishi [OkS] avaient pris 2 pour valeur de l'exposant e. Ensuite, Fujioka, Okamoto et Miyaguchi [FOM] ont pris des valeurs supérieures. Ce problème est facile lorsque la factorisation du module n est connue, mais semble difficile quand la factorisation du module est inconnue.

Protocole

L'idée est d'accompagner le message m d'une signature s vérifiant

$$s^e \simeq h(m) \ \ [n]$$

où h est une fonction de hachage publique. Le module n est public de la forme $n = p^2q$ avec p et q deux nombres premiers distincts sensiblement de même taille. Les entiers p et q sont donc de l'ordre de $O(n^{1/3})$. Ces nombres p et q sont gardés secrets et forment la clé secrète de l'individu.

1. Précalculs : Alice choisit $x \in_R \mathbf{Z}^\star(pq)$.
   Elle calcule $F = x^e \ [n]$
   puis $G = ex^{e-1} \ [p]$
   et $T = G^{-1} \ [p]$.
2. Alice stocke x, F et T.
3. Signature : Alice calcule $W = \lceil \frac{h(m)-F}{pq} \rceil$
   $Y = WT \ [p]$
   et envoie $s = x + Ypq$.
4. Vérification :
   $$h(m) \leq s^e \ [n] < h(m) + n^{2/3}.$$

Valeurs conseillées

Les valeurs e=2 et 3 s'étant avérées peu sûres (*cf 2.2.2*), on a été amené à prendre des valeurs plus grandes pour e. En particulier, on prend usuellement e de la forme 2^i avec $i \geq 2$ pour permettre des calculs plus rapides. Mais e=4 semble parfaitement convenir.

Les entiers p et q sont choisis sur 256 bits, et donc n est codé sur 768 bits.

Complexité en temps

- Précalculs : 2 carrés modulo n
  1 multiplication et 2 décalages modulo p
  1 inversion modulo p

- Signature : 1 évaluation de hachage
  1 division
  1 multiplication
  1 multiplication modulo p

Complexité en espace

Données :	p, pq, n	192 octets
Précalculs :	x, F, G et T	224 octets
Signature :	W, Y et s	160 octets

Transferts

Le message est de longueur quelconque, et la signature est de 96 octets.

Sécurité

(a) assuré par la difficulté du problème $FACT$

(b)(c) sont assurés par l'hypothèse "RSA approché"

(e) assuré si une autorité certifie les clés

Conclusion

Ce schéma semble très prometteur. Malgré un mauvais départ (les valeurs $e = 2$ et $e = 3$ se sont très vite révélées inefficaces), ce schéma est parmi les plus rapides, et les moins coûteux en espace. De plus, les calculs peuvent s'effectuer en deux temps, pour accélérer la phase finale au moment de la signature.

2.2.2 Les attaques connues.

Cas $e = 2$

La première version d'Okamoto et Shiraishi avec $e = 2$ a été cassée par Brickell et Shamir en 1986, la deuxième par Vallée, Girault et Toffin en 1987 [VGT]. Leur méthode utilise la réduction des réseaux. En effet, à l'aide de l'algorithme LLL, on peut obtenir, pour x_0 et y_0 donnés, x proche de x_0 et y proche de y_0 tels que $x^2 \equiv y\ [n]$:

Soient x_0 et y_0 donnés, on veut u et v petits tels que

$$(x_0 + u)^2 \equiv y_0 + v\ [n]$$

c'est-à-dire

$$2x_0u - (v - u^2) \equiv y_0 - x_0^2\ [n].$$

Ceci revient à chercher un point du réseau de matrice suivante :

$$\begin{pmatrix} 1 & 0 \\ 2x_0 & n \end{pmatrix}$$

proche de $(0, y_0 - x_0^2)$ au sens d'une certaine norme. Cette norme particulière est ramenée à la norme sup à l'aide d'un système de multiplicateurs qui "dilate-contracte" ce réseau.

L'algorithme LLL nous donne alors un point de composantes (w_1, w_2) tel que

- w_1 proche de 0
- $2x_0w_1 + nw_2$ proche de $y_0 - x_0^2$

Il reste donc à résoudre le système $\begin{cases} w_1 \equiv u\ [n] \\ (2x_0w_1 + nw_2) - (y_0 - x_0^2) \equiv v - u^2\ [n] \end{cases}$ qui a clairement toujours une solution.

Cas $e = 3$

Quant à $e = 3$, pour $M = h(m)$, il s'agit de trouver s tel que

$$s^3 - M \equiv \delta\ [n] \text{ où } \mid \delta \mid \leq n^{2/3}.$$

Voici la méthode proposée par E.F. Brickell et J.M. DeLaurentis [BDL] :
Alors,

- on prend $r = [\frac{n}{3}]$
  (i.e. $r = \frac{n}{3} + \theta$ avec $\mid \theta \mid \leq \frac{1}{2}$)
- on calcule $z = M - r^3\ [n]$
- on prend x l'entier le plus proche de $z^{1/3}$, divisible par 3
  (i.e. $x = z^{1/3} + \epsilon$ avec $\mid \epsilon \mid \leq \frac{3}{2}$)
- puis on pose $s = r + x$.

On a donc

$$s^3 \equiv r^3 + 3r^2x + 3rx^2 + x^3\ [n]$$

$$s^3 \equiv r^3 + 3\left(\frac{n}{3} + \theta\right)^2 x + 3\left(\frac{n}{3} + \theta\right)x^2 + z + 3z^{2/3}\epsilon + 3z^{1/3}\epsilon^2 + \epsilon^3\ [n].$$

Comme x est divisible par 3, ceci se réduit en

$$s^3 \equiv M + (3\theta^2x + 3\theta x^2 + 3z^{1/3}\epsilon^2 + 3z^{2/3}\epsilon + \epsilon^3)\ [n].$$

Ce qui donne bien le résultat recherché :

$$s^3 \equiv M + \delta\ [n] \text{ avec } \mid \delta \mid \leq O(n^{2/3}).$$

2.3 Le schéma de Fiat-Shamir et ses variantes

Ce sont des protocoles interactifs sans transfert de connaissance utilisant les problèmes de factorisation

De fait, le schéma original de Fiat-Shamir [FS] est fondé sur la difficulté du problème $SQRT$. Par la suite, on a utilisé la difficulté du problème RSA. De plus, ce protocole a la propriété d'être fondé sur l'identité : la clé publique de chaque utilisateur est son identité, tandis que la clé secrète a été calculée par une autorité à partir de la clé publique. Comme tout schéma fondé sur l'identité, il peut se transformer en un schéma classique à clé publique avec des clés certifiées.

2.3.1 Le schéma de base - (Fiat, Shamir 1986)

L'autorité dispose d'un entier n, qui est le produit de deux nombres premiers p et q distincts. L'entier n est rendu public par l'autorité, qui garde secrets les entiers p et q. Chaque individu s'est mis précédemment en rapport avec l'autorité en lui transmettant son identité I. En retour, l'autorité lui a calculé sa clé secrète S qui est une racine carrée modulo n de l'identité I :

$$S^2 \equiv I \ [n]$$

Protocole

1. Alice choisit un aléa r
   calcule $t = r^2 \ [n]$
   envoie t et son identité I à Bob.
2. Bob choisit au hasard un bit c
   envoie c à Alice.
3. Alice calcule $y = rS^c$
   envoie y à Bob.
4. Vérification : $y^2 = tI^c \ [n]$.

Sécurité

Ce protocole assure qu'un faux prouveur sera détecté au moins une fois sur deux. En réitérant ce protocole ℓ fois, la probabilité de détecter le faux prouveur dépasse $1 - \frac{1}{2^\ell}$.

Complexité

L'inconvénient principal d'un tel schéma est la nécessité de le répéter une vingtaine de fois afin d'obtenir la sécurité souhaitée. D'où l'apparition de nombreuses variantes.

2.3.2 Quelques premières variantes.

Il y a plusieurs variantes dans ce schéma d'authentification :

- Alice peut disposer d'un k-uplet de secrets $S = (S_1, S_2, \ldots, S_k)$ correspondant à une identité I découpée en "morceaux" $I = (I_1, I_2, \ldots, I_k)$. [FFS]
  Si x et y sont deux k-uplets d'entiers positifs , on note

$$x^y = \prod_{i=1}^{k} x_i^{y_i}.$$

  Alors la seule modification du protocole est la suivante : dans l'étape (2), Bob choisit un k-uplet de bits $c = (c_1, c_2, \ldots, c_k)$.
  En le répétant ℓ fois, un faux prouveur n'arrive à tromper Bob qu'avec une probabilité inférieure à $\frac{1}{2^{k\ell}}$. Par conséquent, en prenant k suffisamment grand, on peut limiter le nombre de répétitions à effectuer pour atteindre la sécurité voulue.

- Ce schéma peut être transformé aisément en schéma de signature en supprimant l'interaction et en utilisant une fonction à sens unique f.
  L'étape (2) est donc supprimée et les étapes (1) et (3) comprimées en une seule comme suit :

  * Alice choisit un aléa r
    calcule $t = r^2$ $[n]$
    puis $c = f(t, m)$
    et $y = rS^c$
    envoie le couple (c, y) à Bob, comme signature du message m.
  * Vérification : $c = f(y^2 I^c, m)$.

2.3.3 Le passage de l'exposant 2 à l'exposant e

(Guillou-Quisquater 1988) [QG]

Ici, la sécurité du procédé réside dans la difficulté de l'inversion de la fonction élévation à la puissance e modulo n : $x \longmapsto x^e$ $[n]$ lorsque la factorisation de n est cachée : c'est la difficulté du problème $RSA(n, e)$.
L'autorité dispose d'un couple RSA formé d'un entier n qui est le produit de deux nombres premiers p et q distincts et d'un entier e premier avec $\phi(n)$.
Le couple (n, e) est rendu public par l'autorité, qui garde secrets les entiers p et q, et qui peut donc calculer facilement l'inverse d de e modulo $\phi(n)$.
Chaque individu se met préalablement en rapport avec l'autorité en lui transmettant son identité I.
En retour, l'autorité lui calcule sa clé secrète S qui est, par définition, égale à $S = I^{-d}$.

Protocole

1. Alice choisit un nombre aléatoire x de $\mathbf{Z}^\star(n)$
   envoie $t = x^e$ $[n]$ à Bob.

2. Bob choisit au hasard un nombre $c \in [0, e-1]$
   envoie c à Alice.
3. Alice envoie $y = xS^c \ [n]$ à Bob.
4. Vérification : $t \equiv I^c y^e \ [n]$.

On peut bien sûr répéter ce processus plusieurs fois.

Valeurs des paramètres

Pour minimiser le nombre de multiplications, on a intérêt à prendre $e = 2^k + 1$ premier avec $\phi(n)$, avec k entre 10 et 20. Et dans ce cas mémoriser $S, S^2, \ldots$

Pour ce qui est de la taille des entiers n, p, q, c'est, une fois de plus, comparable à celle des paramètres du système RSA, puisque c'est le problème RSA qui est derrière ce schéma.

Complexité en temps

Dans le précalcul, l'évaluation de t nécessite k carrés modulo n.

Quant au calcul de S^c, il nécessite autant de multiplications modulo n que le développement binaire de c comporte de 1 , donc en moyenne $\frac{k}{2}$ si on a mémorisé les S^{2^i}, plus k carrés modulo n dans le cas contraire.

Enfin, le calcul de y nécessite 1 multiplication modulo n.

Complexité en espace

Données :	S	64 octets
	éventuellement : S^{2^i} $1 \leq i \leq k$	$k \times 64$ octets
Précalcul :	t, x	128 octets
Calcul :	c, y	512+k bits

Transferts

Seuls t et y sont à transmettre, ce qui représente 128 octets.

Sécurité

(a) La probabilité pour que Charlie parvienne à prendre l'identité d'Alice est inférieure à $\frac{1}{2^k}$

(b) Pour découvrir le secret d'Alice, Charlie doit casser RSA.

(c) En revanche, il n'y a pas de secret pour l'autorité, puisque c'est elle qui calcule la clé secrète.

Conclusion

La principale motivation de ce schéma a été de diminuer le nombre d'interactions; bien entendu, ceci engendre une croissance du nombre de multiplications, mais dans un ordre tout à fait raisonnable. Il faut remarquer que ce schéma est également basé sur l'identité.

2.3.4 Les propositions de Ong-Schnorr (1989)

Leur idée reprend, en fait, la première variante énoncée, mettant en jeu plusieurs secrets. Ceci, toujours dans le même but : réduire le nombre d'interactions. [OS]

Initialisation

- L'autorité choisit deux nombres premiers p et q
  un schéma quelconque de signature
  afin de certifier les clés des utilisateurs
  calcule le produit $n = pq$
  publie n, ainsi que les éléments publics de son schéma de signature.
- L'utilisateur choisit
  - sa clé secrète : un k-uplet d'éléments de $\mathbf{Z}^\star(n)$, $S = (S_1, S_2, \ldots, S_k)$
  - sa clé publique : $v_i = {S_i}^{-2^t}\ [n]$ pour $i = 1, \ldots, k$.
- L'autorité vérifie l'identité de l'utilisateur et génère I (identité avec redondances)
  certifie (I, v) en $\mathcal{S}\rangle\}$ à l'aide de son schéma de signature.

Utilisation

1. Alice choisit $r \in_R [1, n]$
   calcule $x \equiv r^{2^t}\ [n]$
   envoie x à Bob.
2. Bob choisit $e = (e_{11}, \ldots, e_{tk}) \in_R \{0,1\}^{tk}$
   envoie e à Alice.
3. Alice calcule $y = r \prod_{j=1}^{k} S_j^{\sum_{i=1}^{t} e_{ij}.2^{i-1}}\ [n]$
   envoie y à Bob.
4. Vérification : $\mathcal{S}\rangle\}$ est bien la signature de (I, v)
   $x = y^{2^t} \prod_{j=1}^{k} v_j^{\sum_{i=1}^{t} e_{ij}.2^{i-1}}\ [n]$.

Valeur des paramètres

Les ordres de grandeurs de n, p et q sont les mêmes que dans RSA.

Les entiers t et k sont les garants de la sécurité du protocole. Par conséquent, pour obtenir la sécurité habituellement souhaitée de 10^{-6}, il faut prendre $t \geq 4$ et $k \leq 20$ tels que $kt = 20$.

Complexité en temps

Lors du précalcul (première étape), le calcul de x nécessite t carrés modulo n.

L'étape (3), quant à elle, demande $\frac{t(k+2)}{2} - 1$ multiplications modulo n, en moyenne.

Complexité en espace

Données :	Clé secrète	$k \times 64$ octets
Précalcul :	x et t	128 octets
Calcul :	y	64 octets

Transferts

Première étape :	x	512 bits
Deuxième étape :	e	tk bits = 20 bits
Troisième étape :	y	512 bits

Sécurité

(a) La probabilité pour Charlie, de se faire passer pour Alice, est inférieure à $\frac{1}{2^{kt}}$

(b)(c) Pour trouver le secret d'Alice, Charlie et l'autorité doivent casser RSA.

Conclusion

Ce schéma donne une sécurité supplémentaire, étant donné que l'autorité ne connaît pas les clés secrètes, mais en contre-partie, il n'est plus basé sur l'identité. Il faut donc recourir à la certification des clés.

2.3.5 Variante d'Ohta et Okamoto (1988)

Cette variante est identique à celle de Guillou-Quisquater, au détail près qu'elle n'est pas fondée sur l'identité : l'utilisateur calcule sa clé publique à partir de sa clé secrète, par exponentiation. [OO]

Cependant, il faut remarquer que ce protocole est un système interactif de preuve parfaitement zero-knowledge.

2.4 Schémas fondés sur le logarithme discret

Dans la première partie, nous avons remarqué que le Logarithme Discret est également un problème difficile. ElGamal a été le premier à penser à l'utiliser dans un schéma de signature. Sa méthode a été revue par Schnorr, puis le NIST (National Institute of Standards and Technology) a fait un mélange des deux pour élaborer le DSS qu'il veut imposer comme standard international.

2.4.1 Système de signature de El Gamal (1985)

Ce schéma est donc le premier a être fondé sur ce problème algorithmique : le Logarithme Discret. Et il souffre de certains défauts. [EG]

Protocole

Initialisation

- L'autorité choisit un nombre premier p tel que $p-1$ ait un grand facteur premier q
  ainsi qu'un élément primitif α de $\mathbf{Z}^\star(p)$
  publie p et α.
- L'utilisateur choisit sa clé secrète x.
- Et calcule sa clé publique $y = \alpha^x \ [p]$.

Signature de m

1. Alice choisit $k \in_R \{0, \ldots, p-1\}$, premier avec $p-1$.
2. Elle calcule $r \equiv \alpha^k \ [p]$.
3. Et cherche s tel que $m \equiv xr + ks \ [p-1]$, c'est-à-dire
$$s \equiv k^{-1}(m - xr) \ [p-1].$$
4. Alice envoie (r, s) à Bob, signature de m.
5. Vérification : $\alpha^m \equiv y^r r^s \ [p]$.

Valeurs conseillées

- pour assurer la difficulté du Logarithme Discret, on prend $\mid p \mid \geq 512$ avec $q \mid p-1$ assez grand, $\mid q \mid \geq 160$.
- pour éviter les recherches exhaustives, on doit choisir $\mid k \mid \simeq 140$ pour une signature.
- Bien évidemment, ce protocole peut être transformé en preuve interactive d'identité, dans cas cas on prend $\mid k \mid \simeq 80$.

Complexité en temps

Dans le précalcul, pour r, il faut en moyenne 210 multiplications modulo p. On peut également effectuer à ce moment l'inversion modulo $p-1$.

Ensuite, lors de la signature, le calcul de s nécessite 2 multiplications modulo $p-1$.

Complexité en espace

Données :	p, x, y	1164 bits = 146 octets
Précalcul :	k, r	652 bits =82 octets
Calcul :	m, s	1024 bits = 128 octets

Remarque : On peut hacher m, afin de ne pas avoir à signer plusieurs blocs, si ce dernier fait plus de 512 bits.

Transferts

Il faut transmettre la clé publique y, le message m, ainsi que la signature (r, s), soit 2048 bits = 256 octets.

Ce schéma n'étant pas fondé sur l'identité, Il faut certifier les clés, donc transmettre en plus l'identité et la certification, puis le vérifieur doit contrôler cette identité.

Sécurité

(a) Trouver r et s tels que $\alpha^m = y^r r^s \ [p]$

- Pour r fixé, trouver s revient à résoudre le problème $LD(p)$
- Pour s fixé, le problème à résoudre est estimé aussi difficile que $LD(p)$

(b)(c) Trouver x à partir de y revient à casser $LD(p)$

(e) Certification des clés nécessaire.

Conclusion

Cette idée semble avoir un bel avenir, mais pour le moment, elle reste encore trop coûteuse en temps. De plus, elle contient une inversion qui ne semble pas indispensable.

2.4.2 Schéma interactif d'identification de Schnorr (1989)

Justement, Schnorr a voulu accélérer le schéma d'ElGamal, en s'inspirant du schéma de Fiat-Shamir. Tout d'abord, il a supprimé l'inversion, très coûteuse, puis a cherché une méthode pour produire des couples aléatoires (r, α^r) pour rendre ce schéma praticable sur un processeur de faibles caractéristiques. [Sc1, Sc2]

Initialisation

- Le KAC choisit deux nombres premiers p et q tels que $q \mid p-1$
  un élément α de $\mathbf{Z}^\star(p)$ d'ordre q
  un protocole quelconque de signature pour la certification des clés
  publie p, q, α et les éléments publics de son schéma de signature

- L'utilisateur choisit un secret $s \in \{1, \ldots, q\}$
  calcule sa clé publique $v = \alpha^{-s}$ $[p]$
  envoie son identité Id au KAC, ainsi que sa clé publique v.
- Le KAC vérifie $v^q \equiv 1$ $[p]$, puis l'identité Id de l'utilisateur
  génère I, version redondante de Id
  signe $(I,\ v)$ en $\mathcal{S}\rangle\}$.

Protocole d'identification

1. Alice choisit $r \in_R \{1, \ldots, q\}$
   calcule $x = \alpha^r$ $[p]$
   envoie $I,\ v,\ \mathcal{S}\rangle\}$, x à Bob.
2. Bob vérifie que $\mathcal{S}\rangle\}$ est bien la signature de $(I,\ v)$
   choisit $e \in_R \{1, \ldots, 2^t\}$
   envoie e à Alice.
3. Alice calcule $y = r + se$ $[q]$
   envoie y à Bob.
4. Vérification : $x \equiv \alpha^y v^e$ $[p]$.

Protocole de signature

Soit h une fonction de hachage publique, et m le message à signer.

1. choisir $r \in_R \{1, \ldots, q\}$
   calculer $x = \alpha^r$ $[p]$
2. calculer $e = h(x, m) \in_R \{1, \ldots, 2^t\}$
3. calculer $y = r + se$ $[q]$
   envoyer $I,\ v,\ \mathcal{S}\rangle\}$, m et la signature $(e,\ y)$
4. Vérification : $\mathcal{S}\rangle\}$ correcte
   $z = \alpha^y v^e$ $[p]$
   $e = h(z, m)$

Valeurs conseillées

- Pour une signature, il semble nécessaire de prendre :
  - $\mid q \mid \geq 140$ et $\mid p \mid \geq 512$
  - $t = 72$
- Par contre, pour un protocole d'identification, il suffit de prendre :
  - $\mid q \mid \geq 80$ et $t = 20$
  - $\mid p \mid \geq 512$ pour conserver la complexité du Logarithme Discret

Complexité en temps

- Précalcul : x nécessite en moyenne 210 multiplications pour la signature
  et seulement 120 pour l'identification
  Mais on peut accélérer ce précalcul
  (cf. méthode des précalculs de Schnorr)
- Calcul : une seule multiplication modulo q
  une addition modulo q

Méthode des précalculs de Schnorr

Soient $k, d \geq 6$

Initialisation : Tirer au hasard k couples (r_i, x_i), tels que $x_i = \alpha^{r_i}$
poser $\nu = 1$

1. Tirer $a(0), \ldots, a(d-3) \in_R \{1, \ldots, k\}$
   Poser $a(d-2) = a(d) = \nu - 1 \ [k]$ et $a(d-1) = \nu$
2. Pour le protocole, on garde le couple (r, x)
   $r = r_\nu + 2r_{\nu-1}$ et $x = x_\nu x_{\nu-1}^2$
3. Le renouvellement s'effectue en calculant
   $r_\nu = \sum_{i=0}^{d} r_{a(i)} 2^i \ [q]$
   $x_\nu = \prod_{i=0}^{d} x_{a(i)}^{2^i} \ [p]$
4. Puis $\nu = \nu + 1 \ [k]$

Le choix des derniers éléments $a(d-2), a(d-1), a(d)$ a pour but de préserver la distribution uniforme des k-uplets $(r_1, \ldots, r_k)$, tandis que le choix des premiers termes $a(0), \ldots, a(d-3)$ permet un mélange suffisant des paires précalculées.

D'après Schnorr, il semble que la seule attaque possible utilise une suite d'au moins k termes consécutifs (r, x). Alors, il suffit de deviner les $d-2$ éléments $a(0), \ldots, a(d-3)$ pour déterminer les nombres initiaux $(r_1, \ldots, r_k)$. Donc, pour éviter qu'on puisse deviner par recherche systématique, Schnorr conseille de choisir les deux paramètres égaux successivement à $k = 8, d = 6$.

L'attaque de De Rooij, présentée à Eurocrypt'91 [R], utilise les événements suivants :

$$A(m) = \{a(i,m) = \nu - 1 \text{ ou } a(i,m) = \nu \ \text{ pour tout } \ i\}$$

où $a(i, m)$ indique la valeur de $a(i)$ au moment m.

Ces événements sont à priori peu probables, mais, il est démontré que trois occurrences de cet événement, à des instants donnés bien choisis, permettent de retrouver le secret S avec très grande probabilité. En particulier, avec la valeur des paramètres proposées par Schnorr, on peut espérer retrouver la clé secrète au bout de 2^{37} étapes. Il faut donc ou bien augmenter la valeur des paramètres -c'est ce que propose de Rooij-, ou bien empêcher les événements $A(m)$ de se réaliser. Alors une modification simple consiste à choisir les $a(i)$ pour $i = 0, \ldots, d-3$ tels qu'ils représentent une permutation d'une partie de $\{1, \ldots, k\}$. Mais De Rooij a proposé une nouvelle attaque utilisant la forme des relations de la deuxième étape, sans aléa. Il conviendrait donc d'insérer de l'aléa dans cette étape.

Complexité en espace

Données :	I, v, $\mathcal{S}\rangle\}$, s	1224 bits = 153 octets
Précalcul :	r_i	$k \times 80$ bits = 80 octets
	x_i	$k \times 512$ bits = 512 octets
	$a(i)$	$(d+1) \times 3$ bits = 21 bits
	r, x	652 bits
Calcul :	y	140 bits

Transferts

Pour la signature : x, e, y 724 bits
Pour l'identification : x, e, y 612 bits
En plus de l'identité, de la clé publique et de la certification.

Conclusion

Ces nouveaux schémas contiennent un nouvel inconvénient pour une implémentation sur un système de peu d'espace mémoire. En effet, ils nécessitent des calculs dans plusieurs anneaux. En particulier, ce schéma de Schnorr effectue des calculs modulo p et modulo q. A part cela, ce schéma tient la route.

2.4.3 Variante de Girault-Paillès (1990)

Cette variante a l'intérêt d'être basée sur l'identité. Sinon, elle est semblable au schéma de Schnorr. [GP]

Initialisation

- L'autorité choisit deux nombres premiers $p = 2fp' + 1$ et $q = 2fq' + 1$ tels que
  les entiers f, p', q' sont premiers
  l'entier f est supérieur à tous les secrets
  les facteurs p', q' sont très grands devant f
  un élément g de $\mathbf{Z}^\star(n)$ d'ordre f
- et publie le produit $n = pq$, et l'entier f
  l'exposant e tel que $ed = 1\ [\phi(n)]$
  l'élément g.
- L'utilisateur choisit son secret $s < f$
  calcule $t = g^s\ [n]$
  envoie t à l'autorité, ainsi que son identité I.
- L'autorité vérifie I
  calcule $V = I^{-d}\ [n]$ $P = Vt^{-1}\ [n]$.

On a donc comme éléments publics et communs à tous :
n, e, g et $h = g^e$
L'utilisateur a comme clé publique I, P
et comme clé secrète s.

Protocole

1. Alice choisit $r \in_R \{1, \ldots, f\}$
   calcule $x = h^r \ [n]$
   envoie I, P, x à Bob.
2. Bob choisit $c \in_R \{1, \ldots, e\}$
   envoie c à Alice.
3. Alice calcule $y = r + sc \ [f]$
   envoie y à Bob.
4. Vérification : $h^y (P^e I)^c \equiv x \ [n]$.

Valeurs des paramètres

- $\mid f \mid \geq 150$
- $\mid p' \mid$ et $\mid q' \mid \geq 225$
- alors $\mid n \mid \simeq 750$
- La taille de e définit la sécurité : $\mid e \mid \geq 20$

Complexité en temps

Ce protocole est comparable à celui de Schnorr, ses précalculs peuvent s'appliquer, sinon, en moyenne, 210 multiplications modulo n (sur 750 bits)

Complexité en espace

Données :	I (375 bits), P (750 bits)	1125 bits
	h	750 bits
	s	150 bits
Précalcul :	r et x	900 bits
Calcul :	y	150 bits

Transferts

Dans la première étape	I, P, x	1275 bits
Dans la deuxième	c	20 bits
Dans la troisième	y	150 bits

Sécurité

(a) Charlie a 1 chance sur e de tromper Bob (dans ce cas $\frac{1}{2^{20}}$)

(b) Charlie connaît P et I, et donc $h^{-s} = P^e I \ [n] : LD(n)$

(c) L'autorité connaît $t = g^s \ [n] : LD(n)$

(e) Ce protocole étant basé sur l'identité, pour trouver un nouveau P relatif à I et à une nouvelle clé secrète, il faut casser RSA.

Conclusion

Ce schéma semble beaucoup plus coûteux en temps que le précédent, mais cela provient simplement du fait que les auteurs ont pris des tailles de paramètres plus élevées pour tenter de présenter une plus grande résistance aux algorithmes de calcul de logarithmes discrets existants et à venir.

2.4.4 Variante de Brickell-McCurley (1992)

Modifications [BMcC]

Principalement, la factorisation de $p-1$ reste secrète :

- L'autorité choisit les nombres premiers w, p et q tels que
$$\begin{gathered} qw \mid p-1 \\ q^2 \not| p-1 \\ \mid q \mid \text{et} \mid w \mid \geq \text{k} \\ \mid qw \mid \geq 512 \\ \text{un élément } \alpha \text{ de } \mathbf{Z}^\star(p) \text{ d'ordre } q \end{gathered}$$
- et publie p, α, ainsi que sa clé publique de certification.

Le protocole reste le même en remplaçant les calculs modulo q par des calculs modulo $p-1$

Valeurs des paramètres

- $t = 40$
- $k = 140$

2.5 Digital Signature Standard (DSS)

(National Institute of Standards and Technology 1991)

Ce protocole de signature qui voudrait s'imposer comme standard mondial, est inspiré des schémas de ElGamal et de Schnorr. [NIST]

Protocole

Initialisation

- L'autorité choisit deux nombres premiers p et q tels que $q \mid p-1$
  un élément h de $\{1,\ldots,p-1\}$, tel que $h^{\frac{p-1}{q}}\ [p] > 1$
  une fonction de hachage à sens unique H.
- Elle publie p, q ,H et $g = h^{\frac{p-1}{q}}\ [p]$.
- L'utilisateur choisit sa clé secrète $x \in \{1,\ldots,q-1\}$.
- Et publie sa clé publique $y = g^x\ [p]$.

Utilisation

Pour envoyer un message signé m, Alice doit faire comme suit :

- Précalcul : Choisir $k \in_R \{1,\ldots,q-1\}$
  Calculer $r = (g^k\ [p])\ [q]$
  Puis $a = xr\ [q]$ et $b = k^{-1}\ [q]$
- Signature : $s = b(H(m)+a)\ [q]$

  La signature est alors le couple (r,s)
- Vérification : Bob reçoit m, et la signature (r,s), il a trois conditions à vérifier :
  - $0 < r < q$
  - $0 < s < q$
  - $r = (g^{u_1} y^{u_2}\ [p])\ [q]$ où $u_1 = H(m)w\ [q]$
    $u_2 = rw\ [q]$
    avec $w = s^{-1}\ [q]$

Valeurs des paramètres

- $511 <\mid p \mid < 512$
- $159 <\mid q \mid < 160$

Complexité en temps

- Précalcul : Le calcul de r demande une exponentiation modulo p,
  (240 multiplications modulaires sur 512 bits en moyenne).
  Pour a, il suffit d'une multiplication modulo q (sur 160 bits).
  Pour b, une inversion modulo q (sur 160 bits)
- Le calcul de s ne nécessite qu'une multiplication modulo q (sur 160 bits)
  et une évaluation de hachage.

Complexité en espace

Données :	p, q, g, x	168 octets
Précalcul :	r, a, b	60 octets
Calcul :	s	20 octets

Transferts

Le message est haché, donc peut être de longueur quelconque. A ce texte, s'ajoutent la signature (r, s) de 320 bits, ainsi que la clé publique y de 512 bits.

Sécurité

La sécurité de ce protocole repose, comme le schéma de Schnorr, sur le Logarithme Discret.

(a) Charlie est face au problème $LD(p)$

(b)(c) Charlie et l'autorité connaissent $y = g^s \ [p] : LD(p)$

(e) Il faut faire certifier les clés

Conclusion

Ce schéma refait apparaître le calcul de l'inverse, est-ce nécessaire ? Il contient les mêmes inconvénients cités dans les paragraphes précédents.

2.6 Problèmes matriciels

Tous les protocoles décrits jusque là présentent l'inconvénient de nécessiter de longs calculs (puissances, exponentiations), et surtout font appel à des problèmes dont la difficulté n'est nullement prouvée. L'idée des protocoles suivants est de n'utiliser que des problèmes linéaires dont la difficulté est prouvée, en l'occurrence, les problèmes suivants sont NP-complets.

2.6.1 PKP - (Shamir 1989)

Voici l'un des tous premiers schémas fondés sur un problème linéaire. Celui-ci repose sur PKP. [Sh2]

Problème

Données

- un nombre premier p
- une matrice $m \times n$, A
- un vecteur de dimension n, V

Question Existe-t-il une permutation π telle que $V_\pi \in KerA$?

Protocole

Initialisation

- L'autorité publie les données p et A, ainsi qu'une fonction de hachage h
- L'utilisateur choisit : sa clé secrète : une permutation π
  sa clé publique : un vecteur V tel que $V_\pi \in KerA$.

Utilisation

1. Alice choisit : $R \in_R \mathbf{Z}_p^n$
   σ bijection aléatoire de $\{1, \ldots, n\}$
   calcule : $x_1 = h(\sigma, AR)$
   $x_2 = h(\pi\sigma, R_\sigma)$
   envoie x_1 et x_2 à Bob.
2. Bob choisit $c \in_R \mathbf{Z}_p$
   envoie c à Alice.
3. Alice calcule $W = R_\sigma + cV_{\pi\sigma}$
   envoie W à Bob.
4. Bob choisit $d \in_R \{0, 1\}$
   envoie d à Alice.

5. Alice envoie σ si $d = 0$
   $\pi\sigma$ si $d = 1$.

6. Vérification

   - si $d = 0$ alors $x_1 = h(\sigma, A_\sigma W)$
   - si $d = 1$ alors $x_2 = h(\pi\sigma, W - cV_{\pi\sigma})$

   On répéte ce processus k fois.

Valeurs conseillées des paramètres

n	p	m
32	251	16
64	251	37

D'après de récentes recherches [Ge], le premier cas semble insuffisant, traitons donc le deuxième cas.

Complexité en temps

- Précalcul : Choix d'une permutation de $\{1, \ldots, n\}$
  produit de 2 permutations
  produit matrice vecteur modulo p
  2 évaluations de hachage sur 64 bits
- Calcul : produit d'un entier par un vecteur modulo p

Complexité en espace

Données :	$A = A' \mid I$	$m\times$(n-m)=1728 éléments de $\mathbf{Z}_p$	= 1728 octets
	Secret : π	$n \log n$=384 bits	=48 octets
	Public : V	n=64 éléments de $\mathbf{Z}_p$	= 64 octets

Remarque : On peut définir A', sous forme de fonction pseudo-aléatoire : $A'_{i,j} = f(i, j)$

Précalcul :	R	64 éléments de $\mathbf{Z}_p$	64 octets
	σ, $\pi\sigma$	$2n \log n$=768 bits	96 octets
	x_1, x_2	128 bits	16 octets
Calcul :	c, W, d		65 octets + 1 bit

Soit un total de 353 octets et 1 bit, plus le stockage de A'

Transferts

x_1, x_2	128 bits	16 octets
c, W, d		65 octets et 1 bit
σ ou $\pi\sigma$	384 bits	48 octets

Soit un total de 1033 bits en plus de (I, V)

Sécurité

(a) C'est une preuve interactive zero-knowledge d'identité, donc la probabilité pour que Charlie y arrive peut être rendue aussi faible que l'on veut.
En effet, à chaque boucle, la probabilité de réussite est de $\frac{p+1}{2p} = 0{,}502$. Donc au bout de k boucles cette probabilité chute à environ $\frac{1}{2^k}$

(b)(c) Pour découvrir la clé secrète d'Alice, Charlie ainsi que l'autorité doivent casser PKP, soit tenter toutes les permutations possibles, c'est-à-dire environ 2^{142}. En effet, ce protocole étant zero-knowledge, même en choisissant les questions, Bob ne pourra tirer d'informations lui permettant d'accélérer la recherche.

(e) La certification des clés l'en empêche, il lui faut casser la signature du KAC (par exemple le schéma RSA)

Conclusion

Ce schéma semble très performant : il n'effectue que des calculs linéaires et ne requiert que très peu de place mémoire, contrairement à ce que l'on aurait pu croire en raison de l'utilisation d'une matrice. Mais le gros problème se situe aux échanges de données. En effet, pour atteindre la sécurité voulue, ce schéma nécessite une vingtaine d'interactions, et, alors, énormément de temps se perd dans le protocole de transmission. Cependant, on descend largement en dessous des 5 secondes (limite infranchissable par les protocoles précédents qui nécessitaient une centaine de multiplications modulaires).

2.6.2 Codes correcteurs d'erreurs - (Stern 1993)

Cet autre type de problème linéaire, cas particulier du précédent, PKP, a connu de nombreuses applications sous forme de schémas d'authentification. Harari [H] a lancé l'idée, mais son schéma était très lourd, et de plus, les instances du problème, sur lequel le schéma était basé, n'étaient pas dures. D'autres ont donc tenté de rendre son schéma plus performant, tels Stern [St1] qui a publié une variante en cinq passes de ce schéma, de complexité en espace comparable, Girault [Gi2] qui en a publié une en trois passes, zero-knowledge. Cependant, cette dernière nécessitait des transferts encore plus considérables.

Enfin, Stern vient de publier un nouveau schéma, sur la même idée, zero-knowledge et praticable [St2].

Protocole

Initialisation

- L'autorité communique une matrice H de taille $n \times k$ sur $\mathbf{Z}(2)$, aléatoire. Elle peut donc être considérée comme matrice de parité d'un code correcteur d'erreurs C.
- L'utilisateur reçoit sa clé secrète, s, un vecteur de taille n et de poids p.
  Sa clé publique est alors
  $$i = H(s)$$
- La fonction $Hash$ de hachage est connue de tous.

Utilisation

1. Alice choisit un vecteur y de taille n
   une permutation σ de $\{1, \ldots, n\}$
   calcule $c_1 = Hash(\sigma, H(y))$
   $c_2 = Hash(\sigma(y))$
   $c_3 = Hash(\sigma(y \oplus s))$
   envoie c_1, c_2, c_3 à Bob.

2. Bob choisit $b \in_R \{0, 1, 2\}$
   envoie b à Alice.

3. Alice envoie y et σ, si $b = 0$
   $y \oplus s$ et σ, si $b = 1$
   $\sigma(y)$ et $\sigma(s)$, si $b = 2$
   à Bob.

4. Vérification
   Si $b = 0$, Bob vérifie c_1 et c_2
   Si $b = 1$, Bob vérifie c_1 et c_3 car $H(y) = H(y \oplus s) \oplus i$
   Si $b = 2$, Bob vérifie c_2 et c_3 puis que le poids de $\sigma(s)$ vaut bien p.

Valeurs conseillées des paramètres

- Il faut choisir p près de la borne supérieure de Warshamov-Gilbert.
- n de l'odre de $2k$

En pratique, il semble convenable de prendre :

- $n = 512$, $k = 256$ et $p \sim 56$
- $n = 1024$, $k = 512$ et $p \sim 110$

Complexité en temps

Tout étant linéaire, les calculs s'effectuent en un temps négligeable.

Complexité en espace

Commun :	H matrice 512×256	16kO
Clés :	s	512 bits = 64 octets
	i	256 bits = 32 octets
Calcul :	σ	près de 600 octets

Remarque : La matrice H peut être stockée sous forme de fonction pseudo-aléatoire comme dans le protocole précédent. Par conséquent, il n'y a plus de problème de place mémoire.

De même, la permutation peut être stockée et communiquée comme origine d'un générateur pseudo-aléatoire de 120 bits.

Transferts

Les transferts sont très faibles car la matrice H est commune à tous les utilisateurs. Puis à l'aide des générateurs pseudo-aléatoires, les transferts se réduisent à 128 octets.

Sécurité

(a) Charlie a 2 chances sur 3, à chaque tour, de tromper Bob.

(b)(c) L'autorité et Charlie sont confrontés au problème de base : trouver un mot de poids p d'image i par H.

(e) Certification des clés.

Ce protocole est zero-knowledge.

Conclusion

Ce protocole admet de nombreuses variantes, notamment pour alléger le calcul de la première étape et pour diminuer le nombre de répétitions. En effet, la probabilité de fraude en un tour est de 2/3, il faut donc plus de 30 tours pour passer sous les 10^{-6}. Cette variante ramène la probabilité de fraude d'un tour aux environs de 1/2, ce qui nécessite encore 20 itérations. De plus, elle devient en 5 passes.

Les mêmes problèmes qu'avec le schéma précédent (PKP) sont à prévoir. En effet, le coût des transferts est comparable, ainsi que la complexité des calculs.

Toutefois, Stern propose également une dernière variante basée sur l'identité.

Mais comme pour PKP, ce schéma est peu coûteux en calculs, et les transferts deviennent prédominants. Le coût de ces derniers étant peu quantifiable, il semble que seule l'implémentation pratique permette de trancher.

3 Idées nouvelles

3.1 $3DM$ - Three Dimensional Matching Problem

(Henri Gilbert)

Problème

Données

- un entier $p \in \mathbf{N}^\star$
- un entier $n \in \mathbf{N}^\star$
- n triplets $(x_1, y_1, z_1), \ldots, (x_n, y_n, z_n)$ d'éléments de $\{1, \ldots, p\}$

Question

Existe-t-il p indices $i_1, \ldots, i_p \in \{1, \ldots, n\}$ distincts tels que

$$\{x_{i_1}, \ldots, x_{i_p}\} = \{y_{i_1}, \ldots, y_{i_p}\} = \{z_{i_1}, \ldots, z_{i_p}\} = \{1, \ldots, p\} \ ?$$

Non seulement ce problème est dans la classe de complexité NP, mais en plus, il est NP-complet.

Protocole

Génération des clés

Alice choisit aléatoirement

- p éléments distincts $i_1, \ldots, i_n$ de $\{1, \ldots, n\}$
- 3 permutations $(x_{i_1}, \ldots, x_{i_p}), (y_{i_1}, \ldots, y_{i_p}), (z_{i_1}, \ldots, z_{i_p})$ de $\{1, \ldots, p\}$
- $n - p$ triplets (x_j, y_j, z_j) pour $j \neq i_1, \ldots, i_p$

Remarque : Toutes les instances de $3DM$ peuvent être ainsi générées.

La clé publique est cette instance de $3DM$
La clé secrète en est la solution $(i_1, \ldots, i_p)$.

Utilisation

1. Alice choisit une permutation σ de $\{1, \ldots, n\}$.
   On définit $x'_i = x_{\sigma(i)}$
   $y'_i = y_{\sigma(i)}$
   $z'_i = z_{\sigma(i)}$
   et $\{j_1 < \ldots < j_p\} = \{\sigma^{-1}(i_1), \ldots, \sigma^{-1}(i_p)\}$
   alors, $j_1, \ldots, j_p$ est une solution du problème (x'_i, y'_i, z'_i).

Alice calcule $h_0 = h(j_1, \ldots, j_p)$
$h_1 = h(x'_1, \ldots, x'_n)$
$h_2 = h(y'_1, \ldots, y'_n)$
$h_3 = h(z'_1, \ldots, z'_n)$.
Alice envoie h_0, h_1, h_2 et h_3 à Bob.

2. Bob choisit $q \in \{0, 1, 2, 3\}$
   envoie q à Alice.

3. selon la valeur de q :
   si $q = 0$

   - Alice envoie σ.
   - Vérification : $h_1 = h(x_{\sigma(1)}, \ldots, x_{\sigma(n)})$
     $h_2 = h(y_{\sigma(1)}, \ldots, y_{\sigma(n)})$
     $h_3 = h(z_{\sigma(1)}, \ldots, z_{\sigma(n)})$.

   si $q = 1$

   - Alice envoie $(x'_1, \ldots, x'_n)$ et $(j_1, \ldots, j_p)$.
   - Vérification : $h_0 = h(j_1, \ldots, j_p)$
     $h_1 = h(x'_1, \ldots, x'_n)$
     $\{x'_{j_1}, \ldots, x'_{j_p}\} = \{1, \ldots, p\}$.

   si $q = 2$, on opère de même avec y.
   si $q = 3$, on opère avec z.

Valeurs des paramètres

- L'entier n doit être de plusieurs centaines de bits
- L'entier p doit être de l'ordre de $n/3$

Sécurité

(a) A chaque tour, la probabilité pour qu'un imposteur soit accepté est inférieure à $\frac{3}{4}$, ce qui est beaucoup. Par conséquent, pour obtenir une sécurité acceptable, il faut répéter ce protocole une cinquantaine de fois.
Cependant, ce protocole est parfaitement "zero-knowledge"

(b)(c) Pour trouver le secret d'Alice, il faut résoudre le problème $3DM$.

3.2 Accélération des modulos

Dans leur article sur ESIGN, Fujioka, Okamoto et Miyaguchi [FOM] ont fourni un algorithme qui permet d'effectuer les calculs de modulos, et par suite de divisions euclidiennes, relativement rapidement. Cet algorithme consomme bien évidemment un peu d'espace mémoire, mais raisonnablement, comparé au gain temporel.

Supposons qu'on ait un grand nombre de réductions modulo n à effectuer. Alors on définit les entiers $n_i = 2^{9-i}n$ pour $i = 1, \ldots, 9$

MOD(b,n)

1. Si $\mid b \mid_2 < \mid n \mid_2$ alors retourner b
2. Si $\mid b \mid_2 = \mid n \mid_2$ alors
   (a) Si $b < n$ alors retourner b
   (b) Sinon retourner $b - n$
3. Si $\mid b \mid_2 > \mid n \mid_2$ alors
   (a) Chercher i tel que le premier octet de n_i soit plus petit que le premier de b
   (b) Calculer $b = b - 2^{|b|_2 - |n_i|_2} n_i$
   (c) Retourner à l'étape 1

3.3 Exponentiation rapide

(E.F. Brickell, D.M. Gordon, K.S. McCurley)

Brickell, Gordon et McCurley ont étudié le problème de l'exponentiation et ont permis un gain de temps considérable, en contre-partie, un stockage important, mais raisonnable, est nécessaire [BGMcC].

3.3.1 Méthode

On veut calculer $x = \alpha^r \ [n]$ où l'exposant r est de taille t et n, le module, de 512 bits. La méthode habituelle consiste à développer r en base 2, puis à multiplier les carrés successifs de α. Une première généralisation est donc de prendre une base b quelconque, que l'on déterminera en fonction de l'exposant.
Il convient, alors, de stocker

$$\alpha_i = \alpha^{b^i} \ [n] \ \text{ pour } i = 0, \ldots, m-1$$

où m est la taille de r dans la base b : $m = \left\lceil \dfrac{t}{\log_2 b} \right\rceil$

Alors

$$r = \sum_{i=0}^{m-1} r_i b^i \text{ avec } r_i \in \{0, \ldots, b-1\}$$

Et donc,

$$x = \prod_{i=0}^{m-1} \alpha_i^{r_i} = \prod_{d=0}^{b-1} \Big(\prod_{r_i = d} \alpha_i \Big)^d \ [n]$$

On calcule

$$c_d = \prod_{r_i = d} \alpha_i \ [n] \ \text{ pour } d = 1, \ldots, b-1$$

Enfin

$$x = \prod_{d=1}^{b-1} c_d^d \ [n]$$

3.3.2 Complexité

Calcul des c_d
Supposons que pour k valeurs de d, parmi $\{1, \ldots, b-1\}$, on a $c_d \neq 1$.
Alors le calcul de tous les c_d nécessite $m - k$ multiplications au maximum et $\frac{b-1}{b} m - k$ en moyenne.

Calcul de x
Quant à x, son évaluation s'effectue de la manière suivante :

$$x = \prod_{d=1}^{b-1} c_d^d = c_{b-1}(c_{b-1}c_{b-2}) \ldots (c_{b-1} \ldots c_1) \ [n]$$

Alors, ceci se fait en $(k-1) + (b-2)$ multiplications.

Complexité globale

Au total, le nombre de multiplications à effectuer se réduit,

au maximum, à $m + b - 3$

en moyenne, à $\frac{b-1}{b}m + b - 3$.

Il ne faut pas oublier les m valeurs à stocker.

3.3.3 Efficacité

A titre de comparaison, étudions la complexité d'une exponentiation $x = \alpha^r \ [n]$, où l'exposant r et le module n sont sur 512 bits.

Avec la méthode habituelle (calcul des carrés successifs, puis multiplication de ces derniers si besoin) on obtient une complexité, dans le cas le pire, de 511 carrés et 511 multiplications, et, en moyenne, de 511 carrés et 256 multiplications. Ceci, en náyant effectué aucun précalcul préalable.

Avec cette méthode, on obtient les résultats suivants :

t	b	m	Stockage (octets)	mult maximum	mult moyenne
512	2	512	32.768	511	255
	4	256	16.384	257	193
	8	171	10.944	176	154.625
	16	128	8.192	141	133
	32	103	6.592	132	128,781
	64	86	5.504	147	145,656
	128	74	4.736	199	198,422

3.4 On-line/Off-line

C'est une notion introduite par Even, Goldreich et Micali en 1989 [EGM]. Mais en fait, avec l'attention portée aux précalculs par Schnorr, il n'y avait plus qu'un pas à faire.

En effet, il s'agit de minimiser la quantité de calculs à effectuer au moment même de la signature ou de l'identification. Par conséquent, essayer de reporter le gros des calculs dans la première étape d'un schéma interactif (avant que Bob n'envoie son aléa) ou dans les étapes ne nécessitant pas la connaissance du message. Ceci réduit donc les protocoles à deux phases :

- Phase de précalculs (off-line) : Ne fait intervenir que l'aléa d'Alice, peut être effectuée à n'importe quel moment, puis stockée.
- Phase d'authentification (on-line) : C'est la phase finale, Alice a un message particulier ou a reçu l'aléa de Bob, il faut faire la preuve de son identité très rapidement.

Une fois un tel protocole trouvé, ce qui est le cas (ElGamal, Schnorr et d'autres), on fait faire ces "précalculs" à notre puce pendant ses moments d'inactivité.

Alors là, trois cas sont à différencier :

1. La puce a une totale autonomie (sa propre alimentation), elle peut donc faire ses précalculs à tout moment. Ces précalculs peuvent éventuellement mettre un certain temps. La puce peut faire plusieurs précalculs à l'avance et les conserver en mémoire.
2. La puce ne peut fonctionner que lorsqu'elle est reliée à une alimentation extérieure. C'est-à-dire quelques secondes avant, et quelques secondes après son utilisation pour une signature ou une authentification. Les précalculs doivent rester de complexité limitée.
3. Une situation intermédiaire serait de ranger notre carte à puce en contact avec une alimentation, soit directement dans le porte-carte, soit chez soi.

3.5 Signatures retardées

(David Chaum et Niels Ferguson) [CF]

Cette idée pourrait être considérée comme un cas particulier du paragraphe précédent, cependant, elle a un aspect très original.

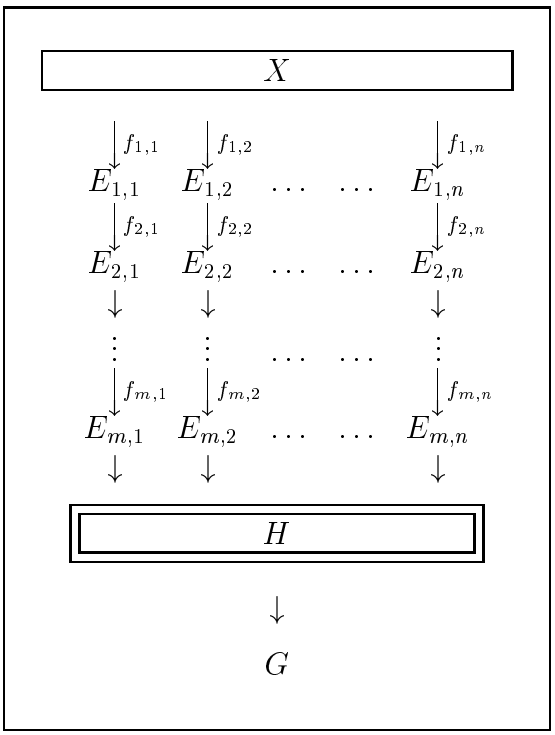
En effet, on dispose :

- d'une famille de fonctions à sens unique, ou d'une fonction à plusieurs variables : $f_{i,j}$ pour $i = 1, \ldots, m$ et $j = 1, \ldots, n$
- d'une fonction de hachage H
- d'un protocole quelconque de signature à clé publique

Précalcul

- Alice choisit un aléa X
- calcule $E_{1,j} = f_{1,j}(X)$ pour $j = 1, \ldots, n$
- ainsi que $E_{i,j} = f_{i,j}(E_{i-1,j})$ pour $i = 2, \ldots, m$ et $j = 1, \ldots, n$
- et $G = H(E_{m,1}, \ldots, E_{m,n})$
- puis signe G.

Et alors, Alice stocke X et la signature de G.



Signature

Supposons qu'Alice veuille signer un message M, représenté sous la forme d' un vecteur de $\mathbf{Z}(m)^n$: $M = (M_1, \ldots, M_n)$.

Elle envoie donc ce message M à Bob accompagné de la signature suivante :

- La signature de G
- Les valeurs $E_{M_i+1,i}$ pour $i = 1, \ldots, n$ qu'elle recalcule rapidement à partir de X

Vérification

- Bob calcule G à partir de M et des $E_{M_i+1,i}$, à l'aide des fonctions à sens unique publiques et de la fonction de hachage.
- Puis il vérifie que la signature de G est bien correcte.

Limitations

Il est bien clair que, désormais, Bob pourra signer n'importe quel message M' tel que $M'_i \geq M_i$ pour tout i, sous le nom d'Alice. Il faut donc imposer que $M_i > M'_i$ pour au moins une valeur de i.

Pour cela, on ajoute k composantes au début du message qui devront vérifier la relation suivante :

$$\sum_{l=1}^{k} m^{l-1} M_l = \sum_{l=k+1}^{n} (m - 1 - M_l)$$

En effet, si $M' \neq M$ avec $M'_i \geq M_i$ pour $i = k+1, \ldots, n$, alors il existe $1 \leq j \leq k$ tel que $M'_i < M_i$.

La valeur de k souhaitable est en $O(\log n)$, ce qui ne provoque donc pas trop de pertes de place pour le message.

Complexité

- La complexité en espace est très faible :
  - Pour les précalculs, seuls X, le $E_{i,j}$ qu'on calcule et la valeur intermédiaire de hachage (si cette dernière a de bonnes propriétés) sont à stocker.
  - Pour ce qui est des fonctions à sens unique, il est bien sûr conseillé de n'en avoir qu'une seule prenant également i et j comme variables. Par exemple le DES.
- La complexité en temps reste également assez faible :
  - quelques centaines d'évaluation DES lors des précalculs, si on envisage des signatures sur 10 octets (ce qui est suffisant pour signer une somme). Et la moitié, en moyenne, au moment de la signature –soit de l'ordre de 5 secondes–
  - quant aux transferts, ils restent limités. Dans le cas ci-dessus, guère plus d'un kbit à transmettre.

Sécurité

L'exemple donné ci-dessus a pris pour paramètres $m = 2^4 = 16$, $n = 22$ et $k = 2$. Ces valeurs permettent, en effet, de signer un message de 80 bits. Si on peut considérer ces messages aléatoires, à l'aide d'un précalcul, on pourra signer, en moyenne, une douzaine de messages.

La sécurité repose sur les fonctions à sens unique.

4 Bilan général

4.1 Tableaux récapitulatifs

Dès la lecture de tous ces schémas, certains peuvent déjà être exclus des applications qui nous intéressent actuellement (cartes à puce de faible puissance). En particulier, les schémas nécessitant une exponentiation ou un calcul de puissance.

En revanche, d'autres schémas semblent, à première vue, convenir à nos exigences. Nous les regroupons donc dans quatre tableaux afin de comparer rigoureusement leurs propriétés respectives.

Le premier tableau met en évidence les services rendus par chacun. Le second, la mémoire nécessaire à la gestion des diverses données des schémas. Les deux derniers tentent de comparer les temps de calculs dans les deux phases des protocoles. Mais pour cela, il faut prendre des données comparables, nous ramènerons donc tout à une unité commune, le temps de calcul d'une multiplication modulaire sur 512 bits. Nous respecterons alors les notations, ainsi que les relations suivantes :

- $M(b)$: temps de calcul d'une multiplication modulaire sur b bits
- $Sq(b)$: temps de calcul d'un carré modulaire sur b bits
- $Inv(b)$: temps de calcul d'un inverse modulaire sur b bits
- $M^\star(b)$: temps de calcul d'une multiplication (non modulaire) sur b bits
- $Alea(b)$: temps nécessaire à la génération de b bits aléatoires

Leurs liens seront les suivants :

- $\frac{M(b)}{b^2} = \frac{M(b')}{b'^2}$
- selon les cas :
  - $Sq(b) = \frac{3}{4}M(b)$
  - $Sq(b) = M(b)$
- $Inv(b) = 10M(b)$
- On majorera
  - $M^\star(b)$ par $M(b)$
  - $Alea(b)$ par $M(b)$
- Décalage (Déc), division (Div) et hachage (Hach) $= \epsilon$

4.1.1 Services rendus

Schéma	Services rendus		Problème de base	Zero know-ledge	Basé sur l'identité	Nombre de passes	Nombre de répétitions
	Id	Sign					
RSA		x	RSA			1	1
Rabin		x	$SQRT$			1	1
ESIGN		x	RSA approché			1	1
Fiat-Shamir	x		$SQRT$	x	x	3	20
variante 20 secrets	x		$SQRT$		x	3	1
Guillou Quisquater	x		RSA		x	3	1
Schnorr	x		$LD(p)$			3	1
Girault Paillès	x		$LD(n)$		x	3	1
DSS		x	$LD(p)$			1	1
PKP	x		PKP	x		5	20

4.1.2 Espace mémoire nécessaire

Schéma	Longueur du message	Problème			Résultats du précalcul	Transferts *(longueur de la signature)*
		Données	Clé publique	Clé secrète		
RSA	512	0	1024	512	0	512
Rabin	haché	0	1024	512	0	520+m
ESIGN	haché	0	768	512	1536	768+m
Fiat-Shamir		512	512	512	1024	512+1+512 =1025
variante 20 secrets		512	10240	10240	1024	512+20+512 =1044
Guillou Quisquater		512	512	512	1024	512+20+512 =1044
Schnorr		1104	512	80	592	512+20+80 =612
Girault Paillès		2250	1500	150	900	750+20+150 =920
DSS	haché	1184	512	160	640	320+m
PKP		8 +matrice	512	384	512	128+8+512+1+384 =1033

4.1.3 Comparaison des temps de calculs

Schéma	Précalcul						
	Aléa (bits)	Mult (mod)	Sq (mod)	Inv (mod)	Déc (mod)	Total* ($M = \frac{3}{4}Sq$)	Total* ($M = Sq$)
RSA						0	0
Rabin						0	0
ESIGN	1 (512)	1 (256)	2 (768)	1 (256)	2 (256)	7,125+2ϵ	8,25+2ϵ
Fiat-Shamir	1 (512)		1 (512)			1,75	2
variante 20 secrets	1 (512)		1 (512)			1.75	2
Guillou Quisquater	1 (512)		20 (512)			16	21
Schnorr	1 (80)	40 (512) [+40]	80 (512)			100,02 [+40]	120,02 [+40]
Girault Paillès	1 (150)	75 (750) [+75]	150 (750)			402,42 [+160,93]	482,88 [+160,93]
DSS	1 (160)	1 (160) 80 (512) [+80]	160 (512)	1 (160)		201,17 [+80]	241,17 [+80]

Schéma	Calcul						
	Aléa (bits)	Mult (mod)	Sq (mod)	Div (mod)	Hach	Total* ($M = \frac{3}{4}Sq$)	Total* ($M = Sq$)
RSA		256 (512) [+256]	512 (512)			640 [+256]	768 [+256]
Rabin	1 (8)	256 (512) [+256]	512 (512)		1	640+ϵ [+256]	768+ϵ [+256]
ESIGN		1 (512) 1*(384)		1	1	1,56+2ϵ	1,56+2ϵ
Fiat-Shamir		1 (512)				1	1
variante 20 secrets		10 (512) [+10]				10 [+10]	10 [+10]
Guillou Quisquater		11 (512) [+10]	20 (512)			26 [+10]	31 [+10]
Schnorr		1 (80)				0,0244	0,0144
Girault Paillès		1 (150)				0,086	0,086
DSS		1 (160)			1	0,098+ϵ	0,098+ϵ

* ramené au temps d'une multiplication modulo 512 bits

4.2 Conclusion

Sur les cartes à puce bon marché, le temps d'une multiplication modulaire sur 512 bits est de l'ordre de 200ms; donc, pour rester dans les limites raisonnables de temps pour une authentification, on ne doit pas dépasser la dizaine d'unités.

Les schémas RSA et de Rabin sont alors rejetés. Il en est de même pour le schéma de Schnorr et ses variantes, à moins d'optimiser les calculs du point de vue temporel. Celui de Fiat-Shamir l'est également en raison du nombre d'itérations à effectuer trop élevé. Cependant, ses variantes qui tentent de diminuer ce nombre d'itérations sont à garder en vue.

Pour ce qui est de PKP, ce dernier n'apparaît pas dans le tableau des temps de calcul, car tout est linéaire, les calculs sont donc négligeables. Mais l'expérience a montré que les échanges de données étaient très coûteux en temps. Les conclusions sont les mêmes pour le schéma basé sur les codes correcteurs d'erreur de Stern : seule l'implémentation permettra de juger.

Les survivants sont donc ESIGN, ainsi que le schéma de Schnorr et ses dérivés, à condition de revoir la technique de calcul de l'exponentielle, puis quelques variantes de Fiat-Shamir. Il ne faudra pas pour autant perdre de vue les algorithmes de calcul des modulos et de l'exponentielle, car tout gain de temps est bon à prendre, ainsi que l'idée des signatures retardées qui peut se révéler très intéressante.

Partie II

Amélioration des protocoles existants

Introduction

Tout d'abord, rappelons les schémas conservés, suite à la première étude :

- Les variantes de Fiat-Shamir :
  - La variante à plusieurs secrets
  - La version de Guillou-Quisquater
- Le schéma de Schnorr, à condition d'accélérer le calcul de l'exponentielle
- ESIGN
- Les signatures retardées

Tous ces schémas, dans leur version initiale, sont trop coûteux en temps, il est donc nécessaire soit de diminuer légèrement la sécurité, soit de stocker des calculs préliminaires afin d'accélérer les calculs lors de la signature.

Nous allons donc étudier tous ces points de plus près, et tentant d'optimiser les calculs a l'aide de l'algorithme de Brickell, Gordon et McCurley, ou d'autres techniques similaires, ainsi qu'en utilisant toute la mémoire disponible pour stocker des précalculs.

Le cadre technique de la carte à puce proposée est le suivant :

- 128 octets de RAM
- 3kO de ROM
- 3kO d'EEPROM
- l'unité de temps (multiplication modulaire sur 512 bits) est estimée à 200ms.

1 Variante de Fiat-Shamir à plusieurs secrets

1.1 Eléments publics

Le module n, produit de deux nombres premiers p et q gardés secrets par l'autorité.
Une suite $f_i : \mathbf{Z}(512) \to \mathbf{Z}(512)$ de fonctions à sens unique indexée par $i \in \mathbf{N}$.
Le nombre de secrets k, paramètre direct de la sécurité du protocole.
Sur une idée du SEPT, nous avons tenté de regrouper les secrets par blocs, et d'effectuer quelques calculs à l'avance, afin d'optimiser les calculs de l'identification :
Le produit $k = \alpha\beta$ où α est le nombre de blocs, et β le nombre de secrets par bloc.

1.2 Protocole

Initialisation

- Alice choisit une identité Id
  envoie Id à l'autorité.
- L'autorité vérifie cette identité
  calcule les k premiers résidus quadratiques de la forme $f_i(Id)$:
  I_j pour $j = i_1, \ldots, i_k$
  (pour simplifier les notations, on suppose $i_m = m$ pour $m = 1, \ldots, k$)
  puis S_j une racine carrée de I_j modulo n.

  Puisque $k = \alpha\beta$, les secrets peuvent être regroupés comme suit :

$$\{S_1, \ldots, S_\beta\}, \{S_{\beta+1}, \ldots, S_{2\beta}\}, \ldots, \{S_{(\alpha-1)\beta+1}, \ldots, S_{\alpha\beta}\}.$$

  L'autorité calcule
  $T_1 = [S_\beta, S_{\beta-1}, S_{\beta-1}S_\beta, S_{\beta-2}, \ldots, S_1S_2 \ldots S_{\beta-1}S_\beta]$
  $\ldots$
  $T_\alpha = [S_{\alpha\beta}, S_{\alpha\beta-1}, S_{\alpha\beta-1}S_{\alpha\beta}, S_{\alpha\beta-2}, \ldots, S_{(\alpha-1)\beta+1}S_{(\alpha-1)\beta+2} \ldots S_{\alpha\beta-1}S_{\alpha\beta}]$

  En fait, si $j = j_1 \ldots j_\beta$ est la décomposition binaire de j
  l'élément général $T_i[j]$ est de la forme :

$$T_i[j] = S_{(i-1)\beta+1}^{j_1} \ldots S_{(i-1)\beta+\beta}^{j_\beta} = \prod_{l=1}^{\beta} S_{(i-1)\beta+l}^{j_l}.$$

  L'autorité envoie T_i pour $i = 1, \ldots \alpha$ ainsi que les indices i_m à Alice.

Identification

1. Alice choisit $r \in \mathbf{Z}(n)$
   calcule $t = r^2$ $[n]$
   envoie t, Id puis les indices des résidus quadratiques $i_1, \ldots, i_k$ à Bob.

2. Bob choisit $c_1, \ldots, c_k \in \mathbf{Z}(2)$
   regroupe ces bits par blocs : $d_1 = c_1 \ldots c_\beta, \ldots, d_\alpha = c_{(\alpha-1)\beta+1} \ldots c_{\alpha\beta} \in \mathbf{Z}(2^\beta)$
   envoie $d_1, \ldots, d_\alpha$ à Alice.

3. Alice calcule
$$y = r \prod_{\substack{i=1 \\ d_i \neq 0}}^{\alpha} T_i[d_i] \ [n]$$
   puis envoie y à Bob.

4. Vérification :
$$y^2 = t \prod_{j=1}^{k} I_j^{c_j} \ [n].$$
   En fait, ce produit s'effectue sur les indices des résidus quadratiques.

1.3 Complexités

Complexité en espace

Ce protocole met en jeu k secrets, répartis en α blocs de β valeurs. Pour chaque bloc, toutes les combinaisons possibles de ces secrets sont précalculées puis stockées dans le tableau T. Par conséquent, pour chaque bloc, il y a $2^\beta - 1$ valeurs à stocker.

Au total, les précalculs des secrets représentent $\alpha(2^\beta - 1)$ nombres de 512 bits. A ceci, s'ajoutent l'identité Id ainsi que les k indices des résidus quadratiques (que l'on peut stocker sur 6 bits pour $k \leq 30$).

Le tout prend une place de $(\alpha(2^\beta - 1) + 1) \times 64 + 0,75 \times k$ octets.

Complexité en temps

Le calcul du carré de la première étape est une nécessité que l'on ne peut supprimer. Mais ici, c'est le calcul de l'étape 3 qui est le plus coûteux en temps.

En raison des précalculs effectués et stockés, ce calcul ne demande, au maximum, qu'α multiplications. Puisque la probabilité pour que d_i soit nul est de $\frac{1}{2^\beta}$, alors, en moyenne, Bob doit effectuer $\alpha\frac{2^\beta-1}{2^\beta}$ multiplications modulaires pour calculer y.

1.4 Transferts

Les transferts sont limités :

1. t, Id et les i_m soit 1024+6k bits à envoyer
2. $d_1, \ldots, d_\alpha$ soit k bits à recevoir
3. y soit 512 bits à envoyer

1.5 Sécurité

La sécurité de ce protocole dépend du nombre de secrets, ainsi que du problème algorithmique $SQRT$. En effet, sous réserve du fait que $SQRT$ est difficile, Bob a une chance sur 2^k de tromper Alice, de même que pour toute personne espionnant la ligne.

1.6 Tableau récapitulatif

Nous allons donc énumérer les différentes valeurs possibles et acceptables des paramètres en question, afin de déterminer les valeurs optimales.

Rappelons les équations pour encadrer ces paramètres :

- la sécurité :
$$k = \alpha\beta \in \{16, \ldots, 24\}$$
- l'espace mémoire :
$$(\alpha(2^\beta - 1) + 1) \times 64 + 0,75\alpha\beta < 4 \cdot 2^{10}$$
- le temps :
$$\alpha \text{ faible}$$

La mémoire impose $\beta \leq 4$:

Moins de 2kO

k	α	β	mémoire (octets)	transfert (bits)	mult max	mult moyen
16	8	2	1.612	1648	8	6,00
18	9	2	1.805,5	1662	9	6,75
20	10	2	1.999	1676	10	7,50

Moins de 3kO

k	α	β	mémoire (octets)	transfert (bits)	mult max	mult moyen
18	6	3	2.765,5	1662	6	5,25

Moins de 4kO

k	α	β	mémoire (octets)	transfert (bits)	mult max	mult moyen
16	4	4	3.916	1648	4	3,75
21	7	3	3.215,75	1683	7	6,13
24	8	3	3.666	1704	8	7,00

Avec ces précalculs stockés dans l'EEPROM, les calculs du protocole de Fiat-Shamir ne prennent plus que de 0,8 seconde à 2 secondes selon la sécurité demandée.

Dans la situation des 3kO, les valeurs optimales des paramètres sont :

$$k = 18 \text{ avec } \alpha = 6 \text{ et } \beta = 3.$$

En effet, on peut effectuer une identification en guère plus d'une seconde, avec une sécurité de 1 pour 250.000. Ce qui est tout à fait acceptable pour une carte à puce, pour un usage totalement consacré à l'authentification.

1.7 Conclusion

Une bonne gamme de valeurs est balayée par les paramètres de temps et de mémoire. Il ne faut cependant pas oublier l'aléa et le carré de la première étape qui s'ajoutent au coût de ces multiplications (environ 2 unités)

2 Modification de Guillou-Quisquater

2.1 Protocole

Initialisation

- L'autorité a choisi un module RSA $n = pq$, ainsi qu'un couple (e, d).
- Alice choisit une identité Id
  envoie Id à l'autorité.
- L'autorité vérifie cette identité
  calcule la clé secrète $S = Id^{-d}\ [n]$
  envoie S à Alice.

Identification

1. Alice choisit un nombre aléatoire x de $\mathbf{Z}^\star(n)$
   envoie $t = x^e\ [n]$ à Bob.
2. Bob choisit au hasard un nombre $c \in [0, e-1]$
   envoie c à Alice.
3. Alice envoie $y = xS^c\ [n]$ à Bob.
4. Vérification : $t \equiv I^c y^e\ [n]$.

2.2 Détails pratiques

Pour rendre la première étape la moins coûteuse possible, il semble convenable de prendre $e = 2^{20}+1$ (il est très probable que cette valeur soit inversible modulo n). En effet, le calcul de $t = x^e\ [n]$ s'effectue par 20 mises au carré successives, suivies d'une multiplication. Soit un coût irréductible de 16 unités (ou 21 selon l'implémentation).

Et en fait, c'est cette étape qui se trouve être prédominante en coût. En effet, à moins d'effectuer une technique de précalculs semblables à ceux de Schnorr, on ne peut pas diminuer le coût.

Quant à la troisième étape, plusieurs accélérations sont envisageables. On peut d'ailleurs espérer réduire à moins de 5 multiplications en moyenne :

2.2.1 Exponentiation rapide

En effet, cette troisème étape revient à mettre S à une certaine puissance, l'exposant étant compris entre 0 et $e-1$, soit sur 20 bits, puis à le multiplier par x.
L'algorithme de Brickell, Gordon, McCurley (*cf* I 3.3) semble bien adapté. Voyons ce qu'il donne comme résultats :

t	b	m	Stockage (octets)	mult maximum	mult moyenne
18	2	18	1.152	18	9
	3	12	768	13	9
	4	9	576	11	8,75
	5	8	512	11	9,4
	8	6	384	12	11,25
20	2	20	1.280	20	10
	3	13	832	14	9,667
	4	10	640	12	9,5
	5	9	576	12	10,2
	8	7	448	13	12,125

2.2.2 Stockage plus intensif

En s'inspirant de la technique utilisée dans la variante de Fiat-Shamir et de la méthode de Brickell, Gordon et McCurley, on peut réduire encore plus ce nombre de multiplications, mais au prix d'un stockage plus important.

En effet, on prend une base b, et on précalcule

$$S_{i,j} = S^{ib^j} \text{ pour } i = 1, \ldots, b-1 \text{ et } j = 0, \ldots, \left\lceil \frac{t}{m} \right\rceil - 1$$

où t est la taille de l'exposant, donc, dans ce cas, environ 20, et m la taille de b.

Par conséquent, $\lceil \frac{t}{m} \rceil (b-1)$ valeurs sont à stocker.

Et le calcul nécessite $\lceil \frac{t}{m} \rceil$ multiplications dans le cas le pire et $\frac{b-1}{b} \lceil \frac{t}{m} \rceil$ en moyenne.

t	b	m	Stockage (octets)	mult maximum	mult moyenne
18	2	1	1.152	18	9
	4	2	1.728	9	6,75
	8	3	2.688	6	5,25
	16	4	4.800	5	4,69
20	2	1	1.280	20	10
	4	2	1.920	10	7,5
	8	3	3.136	7	6,125
	16	4	4.800	5	4,59

2.3 Conclusion

Pour un stockage et une sécurité comparable, la variante de Fiat-Shamir est beaucoup moins gourmande en mémoire que n'importe quelle modification de Guillou-Quisquater.

3 Version interactive d'ESIGN

3.1 Inconvénients du schéma d'origine

A l'origine, ce schéma de Fujioka, Okamoto et Miyaguchi m'a semblé remplir toutes les exigences :

- Signature avec une sécurité absolue (sous réserve que le problème RSA approché est difficile).
- Par conséquent, ce protocole peut s'effectuer en une seule passe, soit une transaction très brève : pas besoin d'interactivité.
- Peu de mémoire nécessaire.
- Et surtout, très peu de calculs : 4 multiplications et une inversion.

Malheureusement, le coût de l'inversion s'est révélé plus important que prévu initialement, et ramène alors ce protocole au rang des autres au niveau vitesse.

En effet, le coût d'une inversion modulo un nombre de 512 bits avait été estimé comme équivalent à celui d'une dizaine de multiplications modulaires sur 512 bits (d'après un rapport de source japonaise). Mais les expérimentations semblent montrer un coût équivalent à celui de plus d'une centaine de multiplications !

3.2 Nouveauté

L'idée consiste à faire calculer cet inverse coûteux par le vérifieur, ce dernier a des moyens de calcul plus importants qu'une puce, qui doit être à certaines normes au niveau de la taille. Mais il faut prendre quelques précautions :

1. Le vérifieur ne doit rien pouvoir découvrir de plus par l'intermédiaire de ces données supplémentaires.
2. Il faut vérifier que le résultat renvoyé est correct. En effet, le vérifieur risquerait de découvrir plus facilement le secret en renvoyant volontairement un résultat erroné.
3. Toutes ces précautions ne doivent pas rendre le coût plus important qu'avant.

Comment s'y prendre ?
Dans le cas présent, nous avons à faire calculer $y = x^{-1} \bmod p$ lorsque p et x sont secrets. La seule chose que le vérifieur connaisse est l'entier $n = p^2 q$.
La démarche est donc la suivante :

1. Choisir un nombre aléatoire $t \in \mathbf{Z}(n)$.
2. Effectuer le produit $\pi = xt\ [n]$.
   Ainsi, x est complètement masqué par l'aléa t.

3. Demander le calcul de $z = \pi^{-1}$ $[n]$.
   La révélation de π ne dévoile aucun secret, sauf si π n'est pas inversible modulo n, ce qui est très peu probable (la probabilité pour que π ne soit pas inversible est inférieure à 2^{-255}).

4. Enfin, on calcule $r = zt$ $[p]$, qui est le résultat souhaité.

5. Reste à vérifier la bonne foi du vérifieur : $rx = 1$ $[p]$?

Il est clair que par cette technique, les 2 précautions de sécurité sont prises. Calculons maintenant le coût d'un tel schéma pour le prouveur :

1. 1 $M(768) = 9/4$ $M(512)$ (En fait un nombre aléatoire sur 768 bits)

2. 1 $M(768) = 9/4$ $M(512)$

3. transfert de données

4. 1 $M(768) = 9/4$ $M(512)$

5. 1 $M(256) = 1/4$ $M(512)$

Soit un total de 7 $M(512)$. C'est-à-dire moins d'une seconde et demie de calcul.

3.3 Schéma d'identification

Protocole

Initialisation

Clé secrète : un couple (p, q) de nombres premiers de 256 bits.
Clé publique : le produit $n = p^2q$.
Alice communique son identité I ainsi que sa clé publique n au centre de certification des clés qui lui renvoie, après vérification de ces 2 données, une signature Sig de (I, n).

Identification

1. Alice choisit $x \in_R \mathbf{Z}(pq)$
   $y \in_R \mathbf{Z}(n)$
   calcule $F = x^4$ $[n]$
   $G = 4x^3$ $[p]$
   $h = Gy$ $[n]$
   envoie I, n, Sig et h à Bob.

2. Bob vérifie Sig=signature(I, n)
   calcule $t = h^{-1}$ $[n]$
   choisit $m \in_R \mathbf{Z}(n)$
   envoie m et t à Alice.

3. Alice calcule $T = ty\ [p]$
   vérifie $TG = 1\ [p]$
   calcule $W = \left\lceil \frac{m-F}{pq} \right\rceil$
   $Y = WT\ [p]$
   $s = x + Y(pq)$
   envoie s à Bob.
4. Vérification :
$$m \leq s^4\ [n] < m + n^{2/3}.$$

Complexités

Complexité en temps

Ce protocole se réduit donc à des calculs de multiplications modulaires. Leur coût global s'élève à :

- Précalcul :

Aléa (bits)	Mult (mod)	Sq (mod)	Dec (mod)	Total ($M = \frac{3}{4}Sq$)	Total ($M = Sq$)
1 (512)	1 (256)		2 (256)	9,125+2ϵ	10,25+2ϵ
1 (768)	1 (768)	2 (768)			

  Ce qui nécessite un peu plus de 2 secondes.

- Calcul :

Mult (mod)	Mult (taille)	Div (taille)	Total
3 (256)	384	512	1,75+ϵ

  Ce qui s'effectue en moins d'une demi-seconde.

Complexité en espace

Aucune place supplémentaire n'est nécessaire, si ce n'est l'implémentation d'un générateur pseudo-aléatoire sur 768 bits, à la place de l'inversion modulaire.

Transferts

Le plus grand inconvénient de ce nouveau schéma consiste en la nécessité d'une interaction, ce qui supprime un des avantages du schéma initial.

Les volumes des transferts sont les suivants :

1. 320 octets
2. 192 octets
3. 96 octets

3.4 Schéma de signature

Protocole
La fonction de hachage, $Hash$, est publiée à tous les utilisateurs.
Soit m le message à signer.

1. Alice choisit $x \in_R \mathbf{Z}(pq)$
   $y \in_R \mathbf{Z}(n)$
   calcule $F = x^4\ [n]$
   $G = 4x^3\ [p]$
   $h = Gy\ [n]$
   envoie I, n, Sig et h à Bob.

2. Bob vérifie Sig=signature(I, n)
   calcule $t = h^{-1}\ [n]$
   envoie t à Alice.

3. Alice calcule $T = ty\ [p]$
   vérifie $TG = 1\ [p]$
   calcule $W = \left\lceil \frac{Hash(m)-F}{pq} \right\rceil$
   $Y = WT\ [p]$
   $s = x + Y(pq)$
   envoie m et s à Bob.

4. Vérification :
$$Hash(m) \leq s^4\ [n] < Hash(m) + n^{2/3}.$$

Complexités
Les diverses complexités sont les mêmes que ci-dessus, aussi bien en temps, qu'en espace et qu'en transferts de données, avec le message en plus.

3.5 Conclusion

Cette modification assez simple décharge complètement le prouveur du calcul le plus coûteux qu'il avait à faire, le temps d'une signature s'en trouve considérablement diminué. Même si la différence théorique ne semble pas importante, en pratique, la durée totale d'une signature est divisée par 4 (plus de 10 secondes avec le protocole initial, moins de 2 secondes et demie pour cette nouvelle version).

Cependant, un gros inconvénient demeure : un certain nombre de calculs modulaires sont effectués avec un module n de 768 bits. Ces calculs étant néanmoins possibles, sont très coûteux sur une puce ne possédant que 128 octets de RAM. En effet, on ne peut pas mettre deux nombres dans cette RAM. Les calculs doivent donc être effectués par petits bouts et faire appel à l'EEPROM qui est extrêmement lente. Le coût de tels calculs est donc difficilement quantifiable.

4 Signatures retardées

Cette idée de Chaum et Ferguson m'a semblé intéressante, mais il fallait déterminer les paramètres adaptés à la situation.

4.1 Détermination des paramètres

Rappelons les notations de ce schéma :
Un message à signer est coupé en n blocs de longueur q (chaque bloc a donc une valeur comprise entre 0 et $m-1 = 2^q - 1$), le message ayant une longueur de nq bits. Lors de la phase de précalcul, il y a donc $n \cdot 2^q = nm$ valeurs à calculer, et lors de la signature, en moyenne, la moitié.

Pour avoir une certaine généralité, on doit être capable de signer un message de 128 bits (résultat du hachage du message réel de longueur quelconque). Par conséquent, lors d'une signature, le signeur a au moins une centaine d'évaluations à effectuer, soit, environ, cinq secondes.

Alors, une variante consiste à stocker quelques valeurs intermédiaires : 1 ligne toutes les p lignes.

Nous pouvons déterminer plus précisément les valeurs de ces différents paramètres n, q et p.
Tout d'abord, on veut pouvoir signer 128 bits, donc

$$nq \geq 128 \quad (1).$$

Ensuite, nous sommes limités par la mémoire de la puce, or $\frac{n}{p}2^q$ valeurs sont à stocker, ce qui représente $\frac{n}{p}2^{q+3}$ octets. Et seuls 3kO sont à notre disposition, d'où l'inéquation suivante :

$$\frac{n}{p}2^{q+3} \leq 3 \cdot 2^{10} \quad (2).$$

Enfin, la dernière restriction, essentielle, est au niveau du temps de calcul : pas plus d'une trentaine d'évaluations pour ne pas excéder les 2 secondes. Or, en moyenne, il y a $n(p-1)/2$ évaluations, d'où

$$n\frac{p-1}{2} \leq 2^5 \quad (3).$$

(1) et (2) entraînent

$$\frac{2^q}{q} \leq 3.p$$

puis (1) et (3) conduisent à

$$q \geq 2(p-1).$$

Finalement, on obtient

$$\frac{2^q}{3q} \leq p \leq \frac{q}{2} + 1$$

et donc $q \leq 6$.

q	n	qn	p	Stockage (octets)	évaluations max	évaluations moy
1	128	128	1	2048	0	0
2	64	128	1	2048	0	0
			2	1024	64	32
3	43	129	1	2752	0	0
			2	1376	43	21,5
4	32	128	2	2048	32	16
			3	1360	64	32
5	26	130	3	2219	52	26
6	22	132	4	2816	66	33

Il se trouve que la solution élémentaire est parmi les meilleures : n=128, q=1. Elle permet la signature de messages de 128 bits de façon quasi-instantanée.

4.2 Nombres de messages

Soit (M_i) la suite des messages à signer où $M_i = (M_{i,j})_{j=1,\ldots,n}$ et $M_{i,j} \in \{0, \ldots, m-1\}$. Un message i pourra être signé si

$$(\exists j \in \{1, \ldots, n\})(\forall k < i)(M_{i,j} < M_{k,j})$$

L' expérience est encore une fois décevante :
Si on suppose cette suite de messages aléatoire, le nombre maximal de messages que l'on peut signer, en moyenne, est de l'ordre d'une dizaine. Bien sûr, Chaum et Ferguson conseillaient l'ajoût de colonnes de sécurité pour augmenter ce nombre de messages, mais ça ne l'augmente pas de façon suffisante.

De plus, ce petit nombre de messages n'assurerait aucune sécurité. En effet, pour avoir confiance en ce protocole, il faudrait que la signature de k messages ne permettent pas à autrui de signer un $(k+1)$ème message. Pour cela, il conviendrait de limiter la carte à la signature de k_{max} messages, avec k_{max} défini de façon à ce que la probabilité pour qu'un $(k_{max}+1)$ème message puisse être signé par un intrus soit inférieure à une valeur ϵ définissant la sécurité.
Vouloir imposer $\epsilon = 10^{-6}$ (valeur habituelle de la sécurité souhaitée) entraîne une valeur de k_{max} très faible, ce qui n'a pas de sens. Mais même pour des valeurs supérieures pour ϵ, k_{max} reste ridiculement faible comparativement aux moyens employés.

4.3 Identification

Pour une signature, il est nécessaire de stocker des valeurs intermédiaires, en revanche, pour une identification, avec $m = 2$, ce schéma a un avenir plus certain : aucun stockage de valeurs intermédiaires n'est nécessaire, on peut donc préparer un grand nombre d'identifications à l'avance.

Protocole

Initialisation

- Une autorité précalcule un grand nombre de couples (X, G)
  où $G = H(f_{2,1}f_{1,1}(X), \ldots, f_{2,k}f_{1,k}(X))$
  et signe les couples $(G, I) : \mathcal{S}\rangle\}$.
  C'est une sorte de certification de la clé publique G.

- L'utilisateur stocke X et la signature $\mathcal{S}\rangle\}$.

Identification

1. Bob choisit k bits $b_1, \ldots, b_k$ tels que $b_1 + \ldots + b_k = \frac{k}{2}$
   envoie ces bits à Alice.
2. Alice calcule $v_i = E_{b_i+1,i}$ pour $i = 1, \ldots, k$ en fonction de X
   envoie son identité I, la signature de l'autorité $\mathcal{S}\rangle\}$ et les v_i à Bob.
3. Vérification : Bob calcule $E_{2,i}$ pour tout i, en fonction des v_i et des b_i
   puis $G' = H(E_{2,1}, \ldots, E_{2,k})$
   vérifie que $\mathcal{S}\rangle\}$ est bien la signature de (G', I).

Sécurité

Bien évidemment, chaque clé n'est utilisée qu'une seule fois. Pour avoir la sécurité souhaitée, on fait varier k :

Après une authentification, la probabilité de recevoir la même distribution des b_i que la fois précédente est d'une chance sur $C_k^{k/2}$. En prenant k=24, on obtient une sécurité de 2^{-21}.

Complexité en temps

Maintenant, quel est le coût temporel d'un tel schéma ?
Puisque pour exactement $k/2$ valeurs de i, b_i vaut 1, le nombre d'évaluations est égal à $3k/2$, soit 36, dans ce cas. Une évaluation DES nécessitant 40ms, ceci représente un peu moins d'une seconde et demie.

Complexité en espace

Pour chaque identification, il faut stocker un aléa X et la signature $\mathcal{S}\rangle\}$ associée. Ce qui occupe une place de 128 bits. Ainsi par kO, 64 préparations peuvent être stockées, soit 192 dans les 3kO de l'EEPROM.

Une solution moins gourmande en mémoire est de ne stocker que l'environnement* du générateur pseudo-aléatoire afin de recalculer à chaque étape, le futur aléa X. Ainsi, seul un environnement est à stocker (il évoluera à chaque identification), et la signature $\mathcal{S}\rangle\}$ de chaque identification : 64 bits. D'où, 128 signatures peuvent tenir dans 1kO, donc 384 dans les 3kO de l'EEPROM.

*l'ensemble des paramètres qui permettent de générer la valeur "pseudo-aléatoire" suivante.

4.4 Conclusion

Ce schéma de signature n'apporte pas les résultats escomptés. De plus, aucune sécurité ne peut être prouvée, seules des valeurs moyennes peuvent être étudiées.
En revanche, il permet de préparer un grand nombre d'identifications qui s'effectueront en un temps relativement faible : moins de 2 secondes. La sécurité dépend de la qualité des fonctions à sens unique utilisées.

5 Schéma de Schnorr

Comme l'a montré l'étude de la première partie, le schéma de Schnorr est très intéressant pour une implémentation sur une carte à puce en raison du faible coût des calculs dans la phase finale. Malheureusement, la phase de précalcul est très lourde (une exponentiation). Nous allons donc étudier divers moyens pour l'alléger.

5.1 Exponentiation rapide

Appliquons cette méthode à notre cas particulier : Exposant de l'ordre de 120 bits.

Tableau des résultats

t	b	m	Stockage (octets)	mult maximum	mult moyenne
140	8	47	3.008	52	46,125
	12	40	2.560	49	45,667
	13	38	2.432	48	45,077
	14	37	3.368	48	45,357
	16	36	2.304	49	46,75
	20	33	2.112	50	48,35
120	4	61	3.904	62	46,75
	8	40	2.560	45	40
	9	38	2.432	44	39,778
	10	37	2.368	44	40,3
	12	34	2.176	43	40,167
	16	31	1.984	44	42,0625
	20	28	1.792	45	43,6
100	4	51	3.264	52	39,25
	8	34	2.176	39	34,75
	12	28	1.792	37	34,667
	16	26	1.664	39	37,375
	20	24	1.536	41	39,80
80	4	41	2.624	42	31,75
	8	27	1.728	32	28,625
	12	23	1.472	32	30,083
	16	21	1.344	34	32,6875

Optimisation
Il semble tout de même plus adapté de prendre une base b de la forme 2^n, par conséquent, les paramètres optimaux sont les suivants :

t	b	m	Stockage (octets)	mult maximum	mult moyenne
140	8	47	3.008	52	46,125
	16	36	2.304	49	46,75
120	8	40	2.560	45	40
	16	31	1.984	44	42,0625
100	8	34	2.176	39	34,75
	16	26	1.664	39	37,375
80	8	27	1.728	32	28,625

Conclusion
Il faut tout de même remarquer que le rendement de cette méthode est assez exceptionnel, et pour un stockage très faible (pour moins de 2kO de stockage, on divise par plus de 5 la complexité dans le cas le pire). Cependant, ceci reste toujours trop coûteux, on descend tout juste sous la barre des 10 secondes.

5.2 Stockage massif des couples

En consultant le tableau de l'espace mémoire nécessaire qui conclut la première partie, on constate que les précalculs du schéma de Schnorr sont les moins volumineux. Pourquoi ne pas en stocker un grand nombre à l'avance, afin d'effectuer des identifications ultra-rapides ?
De plus, Schnorr, pour minimiser le nombre de bits à transférer, suggérait d'utiliser une fonction de hachage f :
Au lieu d'envoyer la valeur complète de x, Alice envoie $f(x)$, dans la première étape. Cette fonction f n'a pas besoin d'être à sens unique car le calcul $x = \alpha^r \ [p]$ l'est déjà. Cependant, pour atteindre le niveau de sécurité 2^t, $f(x)$ doit être au moins sur t bits. Pour la fonction $f(x)$ qui extrait les t bits les moins significatifs de x, aucune attaque particulière n'est connue.
On peut donc légèrement modifier le schéma de base de la façon suivante :

On considère une fonction de troncature f qui, par exemple, à x, associe les t bits les moins significatifs de x.

1. Alice choisit $r \in_R \{1, \ldots, q\}$
   calcule $x = \alpha^r \ [p]$
   envoie $f(x)$, $(I,\ v)$ et Sig à Bob.

2. Bob vérifie la validité de $(I,\ v)$
   choisit $e \in \{1, \ldots, 2^t\}$
   envoie e à Alice.

3. Alice calcule $y = r + se \ [q]$
   envoie y à Bob.

4. Vérification : Bob calcule $z = \alpha^y v^e \ [p]$
   compare $f(z)$ et $f(x)$.

Dans ce cas, le couple $(r,\ f(\alpha^r))$ occupe un espace réduit.

Taille de ce couple

- Pour assurer une sécurité à toute épreuve, même pour une identification, il semble nécessaire de prendre r sur 140 bits.
  Par conséquent, un tel couple $(r,\ f(\alpha^r))$ occupe $140+t$ bits.

- Cependant, la méthode de Shanks ("Pas de bébé, pas de géant"), nécessite le stockage et le tri de $\sqrt{q}$ valeurs et $2\sqrt{q}$ multiplications sur 512 bits qui chacune entraîne 4096 multiplications sur des octets.
  Bref, en prenant un exposant sur 80 bits, comme le suggère Schnorr, pour découvrir le secret, il faut stocker et trier 2^{40} nombres, et effectuer 2^{53} multiplications sur des octets, sans compter les réductions modulaires.
  Ainsi, un couple $(r,\ f(\alpha^r))$ n'occupe plus que $80+t$ bits.

- Une troisième possibilité est de prendre q sur 140 bits, de fixer 80 bits de l'exposant r (répartis uniformément sur l'ensemble des 140 bits) et donc de n'avoir que les autres d'aléatoires.
  Ainsi, seulement 60 bits de r sont à stocker et la recherche exhaustive est d'une complexité inférieure à celle de la méthode "pas de bébé, pas de géant" : plus de 2^{60} multiplications.
  Dans ce cas, le stockage du couple $(r,\ f(\alpha^r))$ nécessite $60+t$ bits.

- Enfin, la solution la plus économique, et aussi sûre que la première, est de ne stocker, à chaque authentification, que l'environnement du générateur pseudo-aléatoire.
  Ainsi, on recalcule r à chaque fois, mais la valeur $f(x)$ est déjà précalculée :

  Notons g la fonction qui fait évoluer l'environnement* du générateur, et k la fonction qui calcule l'aléa : $r_i{=}k(g^i(env_{init}))$
  On stocke env_{init}
  On calcule $r_i{=}k(g^i(env_{init}))$ pour toutes les valeurs possibles de i
  Puis $x_i{=}\alpha^{r_i}\ [p]$
  Enfin, on stocke $f(x_i)$ pour tout i.
  Pour chaque authentification, on calcule l'aléa $r{=}k(env)$
  on utilisera r et $f(x_i)$
  on incrémente i
  on renouvelle l'environnement : $env := g(env)$
  Ainsi, seuls t bits sont à stocker.

*l'ensemble des paramètres qui permettent de générer la valeur "pseudo-aléatoire" suivante.

	1 kO		2 kO		3 kO	
Solution	Ident t=20	Sign t=72	Ident t=20	Sign t=72	Ident t=20	Sign t=72
1ère r sur 140 bits	41	31	92	70	144	108
2ème r sur 80 bits	66	43	148	97	230	151
3ème r formaté	83	50	185	112	288	174
4ème env sur 512 bits	307	85	716	199	1126	312

Comment charger ces couples ?

Il paraît irréaliste de vouloir stocker ces couples après un précalcul intensif. En effet, le calcul de 100 couples prendrait aux environs d'une heure !

Les calculs doivent donc être faits par le serveur, en qui l'on doit avoir entière confiance.

Avantages - Inconvénients

Les avantages de ce système sont indéniables :

- après le chargement complet, le propriétaire de cette carte peut effectuer plusieurs centaines d'identifications en un temps record (certainement moins d'une demi-seconde) : 140 (ou 290 si l'on formate r) avec q sur 140 bits, et 230 avec q sur 80 bits, et plus de 1000 si l'on regénère à chaque fois l'aléa.

- après chaque identification, l'espace mémoire antérieurement utilisé pour stocker le couple de l'authentification est libéré, ce qui permet d'y mettre des données relatives à la transaction.

- les valeurs précalculées et stockées ne dépendent pas du secret, donc n'importe qui peut les calculer.

En contre-partie,

- la sécurité est plus difficile à assurer : il faut parvenir à effectuer le chargement sans risque d'espionnage.

- ce type de stockage interdit toute manipulation des couples, par exemple, des combinaisons semblables aux précalculs de Schnorr, afin d'augmenter le nombre d'idenfications à peu de frais (faible complexité temporelle et affaiblissement raisonnable de la sécurité.)

  Pour cela, il faut stocker la valeur complète de x.

5.3 Stockage en vue de précalculs affaiblis

Les *Précalculs de Schnorr*, non seulement ont été prouvés peu sûrs (quoique suffisants si l'on se contente de quelques centaines d'utilisations), mais encore, ils nécessitent beaucoup trop de multiplications.

Cependant, la notion de précalculs affaiblis (dans le sens où la distribution des r n'est pas *très* aléatoire) peut être approfondie.

Il semble préférable d'avoir un maximum de couples stockés pour uniformiser la distribution. Or, un couple (r, α^r) est codé sur 652 bits, par suite, les 3kO de l'EEPROM, peuvent contenir 36 couples (r_i, x_i) ainsi que l'identité I, la clé publique v puis le cerficat Sig. Notons n ce nombre de couples précalculés (pour 3kO, n=36; pour 2kO, n=23, et pour 1kO, n=11). Il convient de regénérer un couple à chaque utilisation, puis, éventuellement d'en préparer un pour l'identification à venir. Ces deux nouveaux couples peuvent être calculés de la façon suivante :

Un couple (r, x) est créé à partir de k couples précalculés $(r_{i_1}, x_{i_1}), \ldots, (r_{i_k}, x_{i_k})$ suivant la combinaison

$$r = \sum_{j=1}^{k} a_j r_{i_j} \ [q]$$

et donc

$$x = \prod_{j=1}^{k} x_{i_j}^{a_j} \ [p]$$

où $a_j \in \{1, \ldots, m\}$

Précalculs affaiblis

Initialisation : on précalcule les n couples
on pose $\nu = 1$

1. préparation de l'identification suivante :
   - Choisir $k-1$ indices $i_1, \ldots, i_{k-1}$ distincts et différents de ν
   - Poser $i_k = \nu$
   - Choisir k coefficients $a_1, \ldots, a_k$ compris entre 1 et m
   - Générer $r^* = \sum_{j=1}^{k} a_j r_{i_j} \ [q]$
     $x^* = \prod_{j=1}^{k} x_{i_j}^{a_j} \ [p]$
2. regénération :
   - Choisir $l-1$ indices $i_1, \ldots, i_{l-1}$ distincts et différents de ν
   - Poser $i_l = \nu$
   - Choisir l coefficients $b_1, \ldots, b_l$ compris entre 1 et t
   - Regénérer $r_\nu := \sum_{j=1}^{l} b_j r_{i_j} \ [q]$
     $x_\nu := \prod_{j=1}^{l} x_{i_j}^{b_j} \ [p]$
3. Incrémenter ν : $\nu := \nu + 1$ modulo n

Les paramètres k, l et m, t peuvent être variables :

		possibilités*			Multiplications			
k	m	3kO	2kO	1kO	$Sq = M$		$Sq = \frac{3}{4}M$	
l	t	n=36	n=23	n=11	max	moy	max	moy
2	1	35	22	10	1	1	1	1
	2	140	88	40	3	2	2,5	1,75
	3	315	198	90	5	3	4,5	2,667
	4	560	352	160	5	3,5	4,5	3
3	1	595	231	45	2	2	2	2
	2	4.760	1.848	360	5	3,5	4,25	3,125
	3	16.065	6.237	1.215	8	5	7,25	4,5
	4	38.080	14.784	2.880	8	5,75	7,25	5
	5	74.375	28.875	5.625	11	6,8	8,75	5,75
4	2	104.720	24.640	1.920	7	5	6	4,5
	3	530.145	124.740	9.720	11	7	10	6,333
	4	1.675.520	394.240	30.720	11	8	10	7
	5	4.090.625	962.500	75.000	15	9,4	12	8

* Nombre de choix possibles des entiers a_j, b_j et des indices i_j

Le meilleur candidat, rapport possibilités/coût, pour la regénération est $l = 4$ et $t = 2$. En effet, le nombre de possibilités représente le coût de la recherche exhaustive, à chaque étape.

De plus, autant un brassage important des r_i est nécessaire, que les r^* peuvent être construits de façon plus rapide. Cependant, le *hasard* est indispensable pour rendre la recherche des r_i, à partir de l'observation, plus difficile.

Par conséquent, pour la préparation du couple qui servira à la prochaine identification, on peut prendre les mêmes paramètres, ou éventuellement des valeurs plus faibles ($k = 2$ et $m = 2$, ou $k = 3$ et $m = 1, 2$), pour ne pas trop ralentir le processus.

Sécurité

De la même manière que l'a fait Schnorr, on peut prouver que les couples utilisés lors de n authentifications consécutives sont indépendants (*local randomization*). En revanche, la preuve de l'aléa interne (*internal randomization*), qui traduit l'aléa apporté par les b_i, n'est pas conservée.

En effet :

Notations : Notons T_ν, la matrice $n \times n$ de transformation des r_i au tour ν,*i.e.* au tour ν, $R^t := T_\nu R^t$ où R=$(r_1, \ldots, r_n)$. Puis r_j^* l'exposant utilisé à la j-ème identification.

Local randomization **Lemme :** *Si le vecteur initial $(r_1, \ldots, r_n)$ est uniformément distribué sur $\{1, \ldots, q\}^n$, alors cette distribution uniforme est conservée (si $t < q$).*

Preuve : La matrice T_ν est égale à l'identité, exceptée la ligne ν, qui est déterminée par la transformation : $r_\nu := b_l r_\nu + \sum_{j=1}^{l-1} b_j r_{i_j}\ [q]$
D'où $\det T_\nu = b_l \in \{1, \ldots, t\}$
Par suite, $\det T_\nu \not\equiv 0\ [q]$, donc T_ν est inversible modulo q et conserve bien la distribution uniforme sur $\{1, \ldots, q\}^n$.

Comme les r^* sont construits de manière semblable à la regénération, une preuve similaire montre le théorème suivant.

Théorème : *Si le vecteur $(r_1, \ldots, r_n)$ est uniformément distribué sur $\{1, \ldots, q\}^n$, alors, pour tout i_j et pour tous les a_j et b_j, quelquesoit $v \geq 0$, le vecteur $(r^*_{v+1}, \ldots, r^*_{v+n})$ est uniformément distribué sur $\{1, \ldots, q\}^n$.*

Ainsi, une attaque doit utiliser au moins $n+1$ authentifications consécutives. Mais une telle attaque à partir de $n+1$ authentifications consécutives, en recherche exhaustive, demanderait un temps de calcul supérieur à celui du calcul du logarithme discret :
Résoudre le système linéaire $n+1 \times n+1$
$y_\nu = r^*_\nu + se_\nu = a_k r_\nu + \sum_{j=1}^{k-1} a_j r_{i_j} + se_\nu$ pour $\nu = v, \ldots, v+n$,
après avoir deviné

- les coefficients a_j. Pour cela, on doit énumérer $\left(m^k C_{n-1}^{k-1}\right)^{n+1}$ cas (*cf. tableau Complexité 1*).
- les coefficients b_j de regénération. Cette recherche a une complexité comparable : $\left(t^l C_{n-1}^{l-1}\right)^n$ (*cf. tableau Complexité 2*).

		Complexité 1			Complexité 2		
k	m	3kO	2kO	1kO	3kO	2kO	1kO
l	t	n=36	n=23	n=11	n=36	n=23	n=11
2	1	190	107	40	184	103	37
	2	264	155	64	257	149	59
	3	307	183	78	299	176	71
	4	338	203	88	329	195	81
3	1	341	188	66	332	181	60
	2	452	260	102	440	250	93
	3	517	303	123	503	290	113
	4	563	332	138	548	319	126
	5	599	356	150	583	341	137
4	2	617	350	131	600	336	120
	3	704	406	159	685	389	146
	4	765	446	179	744	428	164
	5	813	477	194	791	457	178

Logarithme en base 2 de la complexité

Internal randomization En revanche, en raison de la présence d'aléa dans les coefficients de regénération, on ne peut pas prouver l'inversibilité de la fonction

$$f : b_{1,1}, \ldots, b_{l,1}, \ \cdots, \ b_{1,\nu}, \ldots, b_{l,\nu} \longmapsto T_1 \ldots T_\nu$$

où $b_{i,j}$ est le coefficient b_i au tour j.

Cependant, intuitivement, cet aléa supplémentaire semble préférable ...

5.4 Conclusion

Si l'on veut garder la sécurité initiale du schéma de Schnorr et si l'on ne veut pas être limité en quantité d'identifications faisables, le coût temporel demeure prohibitif :

Même à l'aide de méthodes de calcul de l'exponentielle optimisées, on reste aux environs des 10 secondes.

En revanche, si on accepte la restriction à quelques centaines d'identifications, on obtient un protocole sans concurrence :

Temps d'identification purement négligeable, sécurité intacte.

Finalement, effectuer des *précalculs affaiblis* augmente considérablement le nombre d'identifications faisables, tout en assurant, sans doute, une sécurité suffisante.

Ces deux dernières variantes représentent de bons états de transition dans l'attente de nouvelles puces plus performantes. Le vérifieur étant inchangé, quelle que soit la technique utilisée, par le prouveur, pour calculer l'exponentielle.

6 DSS

L'idée du stockage intensif peut-être reprise pour DSS, et ca devient également un bon moyen de transition en attendant la venue de puces performantes bon marché. Surtout que DSS risque de devenir le standard de demain (tout du moins, c'est la volonté des autorités américaines). Autant s'y préparer progressivement.

Pour cela, on peut procéder comme suit :

- Précalcul : Choisir $k \in_R \{1, \ldots, q-1\}$
  Calculer $r = (g^k \ [p]) \ [q]$
  $b = k^{-1} \ [q]$
- Signature : Calculer $a = xr \ [q]$
  puis $s = b(H(m) + a) \ [q]$
  Envoyer y (clé publique) et le certificat
  puis m et (r, s)

Ces entiers r et b sont sur 160 bits, par conséquent 320 bits préparent une signature, soit 40 octets. Ainsi, une EEPROM de 3kO peut contenir 75 précalculs de signature.

Bien évidemment, on a les mêmes avantages et inconvénients que précédemment. Ces précalculs ne dépendent pas, non plus, du secret x.

Partie III

Conclusion

1 Problème

En fonction des caractéristiques réclamées au protocole, ainsi que des capacités en mémoire disponibles à son implémentation, on peut obtenir des performances temporelles très variables.

En effet, plusieurs paramètres dépendent de l'application souhaitée :

- taille des paramètres de sécurité et des secrets, selon ce qui risque d'être découvert
  - secret de l'autorité (qui mettrait en péril toute l'application)
  - secret individuel

  et selon le type d'authentification
  - signature
  - identification
- nombre de transactions souhaitées
- mémoire utilisée par l'application
  - fichier de données
  - stockage des transactions effectuées
- programme en ROM ou en EEPROM

Rappelons les caractéristiques initiales de la carte à proposée :

- 128 octets de RAM
- 3kO de ROM
- 3kO d'EEPROM
- l'unité de temps (multiplication modulaire sur 512 bits) est estimée à 200ms

2 Cas pratique

On permet l'implémentation du programme en ROM, et aucune mémoire (ou très peu) n'est nécessaire à l'application (tout du moins, initialement).
Puis le nombre d'utilisations est limité à quelques centaines pendant la durée de vie de la carte, ou entre deux rechargements.
Ex : identification pour un contrôle d'accès, porte-monnaie électronique où seul le solde est à stocker.

Dans ce cas, le schéma de Schnorr, avec un stockage massif de couples, est parfaitement adapté. Avec une sécurité satisfaisante, on peut préparer plusieurs centaines authentifications à l'avance (300 signatures ou 1100 identifications)

Cette méthode a des avantages incomparables :

- une authentification sera instantanée (quelques dixièmes de seconde)
- au fur et à mesure, de la place se libère sur la carte, on peut ainsi stocker des données relatives à chaque transaction (quelques octets)
- ce schéma de Schnorr permet aussi bien les identifications que les signatures

Bien évidemment, le stockage massif associé à DSS convient également. Cependant, il permet moins d'authentifications.

Il ne faut pas oublier le schéma de signatures retardées qui permet, dans ces conditions, de préparer près de 400 identifications. Mais ces dernières prendront une temps non négligeable : entre une seconde et demie et deux secondes.

Cette situation est optimale, mais d'autres sont envisageables.

3 Autres cas

Nombre limité d'authentifications

Pour un nombre limité d'authentifications, mais plus important tout de même (de l'ordre du millier, sans doute), les *précalculs affaiblis* doivent convenir. Malheureusement, aucune sécurité ne peut être réellement prouvée. Cependant, même avec un espace mémoire réduit (1 ou 2 kO), ils permettent un nombre important d'authentifications en un temps raisonnable (moins de 2 secondes).

Sécurité absolue et nombre d'authentifications illimité

Là, il est clair que le schéma de Schnorr ne convient plus, quelle que soit la variante appliquée. En revanche, la variante de Fiat-Shamir (plusieurs secrets) et celles de Guillou-Quisquater peuvent convenir, avec une préférence pour la première qui permet de descendre aux environs de la seconde.

La place de la mémoire disponible n'influera alors que sur la fonction qui, à la sécurité, associe le temps de calcul. Mais pour une identification, la variante à plusieurs secrets doit satisfaire toutes les exigences.

Signature

Bien sûr, pour une signature, on doit parer les attaques off-line, par conséquent, les coûts deviennent trop importants. Seul le schéma de Schnorr (ainsi que DSS) permet ces signatures, sans augmenter le coût, mais dans les mêmes conditions que précédemment (peu d'authentifications, assez de mémoire disponible).

Il y a également la version interactive d'ESIGN, mais elle doit être utilisée avec prudence, car l'usage d'une carte à puce qui ne contient que 128 octets de RAM risque de ralentir le protocole par rapport aux résultats attendus.

Schémas linéaires

Aucun résultat n'est donné sur ces schémas, car seule l'implémentation permettrait de déterminer le coût. Mais, pour des identifications, ils sont probablement intéressants.

4 Résumé

Nombre d'authentifications	Mémoire	Schéma	Service Ident	Service Sign	Sécurité	mult max	mult moy
illimité	moins de	Fiat-Shamir (*plusieurs secrets*)	x		2^{-18}	8	7,25
	3kO	Guillou-Quisquater (*stockage intensif*)	x		2^{-18}	27	26,25
	moins de	Fiat-Shamir (*plusieurs secrets*)	x		2^{-20} 2^{-18}	12 11	9,50 8,75
	2kO	Guillou-Quisquater (*stockage intensif*)	x		2^{-20} 2^{-18}	31 30	28,5 27,75
	aucune	ESIGN (*interactif*)		x	totale	~ 12*	
important		Schnorr (*précalculs*)	x	x	à volonté	~ 10	
1100 300	3kO	Schnorr (*stockage massif*)	x	 x	2^{-20} 2^{-72}	ϵ ϵ	
380	3kO	Signatures retardées	x		2^{-21}	8,2	
75	3kO	DSS (*stockage massif*)	x	x		$1+\epsilon$	

* à utiliser avec précautions sur puce de 128 octets de RAM

Bibliographie

[A] L. Adleman, Factoring using singular integers, Proc. of STOC'91, pp 64-71.

[B] E. F. Brickell, A survey of Hardware Implementation of RSA, Advances in Cryptology, Proc. of Crypto 89, Lecture Notes in Computer Science 435 (1990), pp 368-370.

[BDL] E. F. Brickell, J. M. DeLaurentis, An attack on a signature scheme proposed by Okamoto and Shiraishi, Lecture Notes in Computer Sciences 218.

[BGMcC] E. F. Brickell, D. Gordon, K.S. McCurley, Fast exponentiation with precomputation, Advances in Cryptology, Proc. of Crypto 92, Lecture Notes in Computer Science.

[BMcC] E. Brickell, K.S. McCurley, An Interactive Identification Scheme based on Discrete Logarithms and Factoring, Journal of Cryptology 1992.

[CF] D. Chaum, N. Ferguson, Delayed Signatures and Applications.

[COS] D. Coppersmith, A. Odlyzko, R. Schroeppel, Discrete logarithms, Algorithmica, Vol 1 Number 1, pp 1-16.

[DH] W. Diffie, M.E. Hellman, New directions in cryptography, IEEE Transactions on Information Theory (nov 76), Vol IT-22, No6, pp 644-654.

[EG] T. ElGamal, A public key cryptosystem and a signature scheme based on discrete logarithms, IEEE Transactions on Information Theory (juillet 85), Vol IT-31, No4, pp 469-472.

[EGM] S. Even, O. Goldreich, S. Micali, On-line/Off-line digital signatures, Proc. of Crypto 89.

[FFS] U. Feige, A. Fiat, A. Shamir, Zero-knowledge Proofs of Identity, Journal of Cryptology 1988.

[FOM] A. Fujioka, T. Okamoto, S. Miyaguchi, ESIGN : An Efficient Digital Signature Implementation for Smart Cards

[FS] A. Fiat, A. Shamir, How to prove yourself: practical solutions of identification and signature problems, Advances in Cryptology, Proc. of Crypto 86, Lecture Notes in Computer Science, 263, (1987) pp 186-194.

[Ge] J. Georgiades, Some Remarks on the security of the identification Scheme Based on Permuted Kernels, Journal of Cryptology 1992.

[Gi1] M. Girault, An identity-based identification scheme based on discrete logarithms modulo a composite number, Proc. of Eurocrypt'90, pp 481-486.

[Gi2] M. Girault, A (non-practical) three-pass identification protocol using coding theory.

[GP] M. Girault, J.C. Paillès, An identity-based identification scheme providing zero-knowledge authentication and authenticated key exchange, Proc. of Esorics '90, pp 173-184.

[H] S. Harari, A new authentication algorithm.

[Kn] H.J. Knobloch, A Smart Card Implementation of the Fiat-Shamir Identification Scheme.

[LLMP] A.K. Lenstra, H.W. Lenstra, M.S. Manasse, J.M. Pollard, The number field sieve, Proc. of AMS-STOC'90, pp 564-572.

[MS] S. Micali, A. Shamir, An improvement of the Fiat-Shamir Scheme, Advances in Cryptology, Proc. of Crypto 88, Lecture Notes in Computer Science 403 (1990).

[NIST] Proposed FIPS for Digital Signature Standard (DSS).

[Od] A. Odlyzko, Discrete logarithms and their cryptographic significance, Proc. of Eurocrypt'84.

[Ok] T. Okamoto, A fast signature scheme based on congruential polynomial operations, IEEE Transactions on Information Theory, Vol IT-36, No1, pp 47-53.

[OkS] T. Okamoto, A. Shiraishi, A digital signature scheme based on quadratic inequalities, IEEE (Avril 1985) pp 123-132.

[OO] K. Ohta, T. Okamoto, A modification of the Fiat-Shamir Scheme, Advances in Cryptology, Proc. of Crypto 88, Lecture Notes in Computer Science 403 (1990) pp 232-243.

[OS] H. Ong, C.P. Schnorr, Fast signature generation with a Fiat Shamir-like scheme, Advances in Cryptology, Proc. of Eurocrypt'90, Lecture Notes in Computer Science 473 (1991) pp 432-440.

[PH] S. Pohlig, M. Hellman, An improved algorithm for computing logarithms over $GF(p)$ and its cryptographic significance, IEEE Transactions on Information Theory (1978), Vol IT-24, No 1, pp 106-110.

[QG] J.J. Quisquater, L. Guillou, A practical zero-knowledge protocol fitted to security microprocessor minimizing both transmission and memory, Advances in Cryptology, Proc. of Eurocrypt'88, Lecture Notes in Computer Science, 330, (1988), pp 123-128.

[R] P.J.N. de Rooij, On the security of the Schnorr Scheme using preprocessing, Proc. of Eurocrypt'91, pp 29-34.

[RSA] R.L. Rivest, A. Shamir, L. Adleman, A method for obtaining digital signatures and public-key cryptosystems, Communications of the ACM, Vol 21, No 2, 1978, pp 120-126.

[Sc1] C.P. Schnorr, Efficient identification and signatures for smart cards, Advances in Cryptology, Proc. of Crypto 89, Lecture Notes in Computer Science, 435, pp 235-251.

[Sc2] C.P. Schnorr, Efficient Signature Generation by Smart Cards, Journal of Cryptology 1991.

[Sh1] A. Shamir, Identity-based cryptosystems and signature schemes, Advances in Cryptology, Proc. of Crypto 84, Lecture Notes in Computer Science 196, (1985), pp 47-53.

[Sh2] A. Shamir, An efficient identification scheme based on permuted kernels, Advances in Cryptology, Proc. of Crypto 89, Lecture Notes in Computer Science.

[Sk] D. Shanks, Class number, a theory of factorization, and genera, Proc. symp. Pure Math. vol 20 (1971), AMS pp 415-440.

[SS] A. Schrift, A. Shamir, The discrete logarithm is very discreete, Proc. of STOC'90, pp 405-415.

[St1] J. Stern, An alternative to the Fiat-Shamir protocol, Advances in Cryptology , Proc. of Eurocrypt'89, Lecture Notes in Computer Science.

[St2] J. Stern, A new identification scheme based on syndrome decoding (à paraître).

[VGT] B. Vallée, M. Girault, Ph. Toffin, "How to guess ℓ-th roots modulo n by reducing lattice bases", Proc. of Issac-Aaecc'88, LNCS 357, pp 427-442.

Table des matières