Groupe de travail Réseau
Request for Comments : 3766
BCP : 86
Catégorie : Bonnes pratiques actuelles

H. Orman,Purple Streak Dev.
P. Hoffman, VPN Consortium
avril 2004
Traduction Claude Brière de L'Isle

Détermination de la force des clés publiques utilisées pour l'échange de clés symétriques

Statut de ce mémoire
Le présent document spécifie les bonnes pratiques actuelles de l'Internet pour la communauté de l'Internet, et appelle à des discussions et suggestions pour son amélioration. La distribution du présent mémoire n'est soumise à aucune restriction.

Résumé
Les mises en œuvre de systèmes qui utilisent la cryptographie à clé publique pour échanger des clés symétriques ont besoin de rendre les clés publiques résistantes à certains niveaux d'attaque prédéterminés. Ce niveau de résistance à l'attaque est la force du système, et les clés symétriques qui sont échangées doivent être au moins aussi fortes que les exigences de force du système. Les trois quantités, force du système, force de la clé symétrique, et force de la clé publique, doivent être en cohérence pour tout usage d'un protocole réseau.

Bien qu'il soit très facile d'exprimer les exigences de force d'un système en termes de longueur de clé symétrique et de choisir un chiffrement qui ait une longueur de clé égale ou supérieure à ces exigences, il est plus difficile de choisir une clé publique qui ait une force cryptographique qui satisfasse à l'exigence de force d'une clé symétrique. Le présent document explique comment déterminer la longueur d'une clé asymétrique comme fonction d'une exigence de force d'une clé symétrique. Il donne des règles approximatives d'estimation de la résistance équivalente aux attaques à grande échelle contre divers algorithmes. Le document traite aussi de la façon dont les changements des tailles des grands entiers sous-jacents (modulo, tailles de groupes, exposants, et ainsi de suite) changent les temps d'utilisation des algorithmes d'échange de clé.

Table des matières

1. Modèle de protection des clés symétriques avec des clés publiques

De nombreux livres sur la cryptographie et la sécurité expliquent la nécessité d'échanger des clés symétriques en public ainsi que les nombreux algorithmes qui sont utilisés à cette fin. Cependant, peu de ces exposés expliquent comment sont en rapport la force des clés publiques et celle des clés symétriques.

Pour comprendre cela, imaginons une maison avec une forte serrure sur la porte d'entrée. À côté de la porte d'entrée se trouve une petite boîte verrouillée qui contient la clé de la porte d'entrée. Un candidat cambrioleur qui veut pénétrer dans la maison par la porte d'entrée a deux options : attaquer la serrure de la porte d'entrée, ou attaquer le verrou de la boîte afin de récupérer la clé. Il est clair que le cambrioleur ferait mieux de s'attaquer à la plus faible des deux serrures. Dans cette situation, le propriétaire doit s'assurer qu'ajouter l'option de la seconde entrée (la boîte verrouillée qui contient la clé de la porte d'entrée) est au moins aussi fort que la serrure de la porte d'entrée, afin de ne pas rendre trop facile le travail du voleur.

Celui qui conçoit un système pour échanger des clés symétriques en utilisant la cryptographie à clé publique doit prendre une décision similaire. Supposons qu'un attaquant veuille apprendre le contenu d'un message qui est chiffré avec une clé symétrique, et que la clé symétrique soit échangée entre l'envoyeur et le receveur en utilisant la cryptographie à clé publique. L'attaquant a deux options pour récupérer le message : une tentative en force brute pour déterminer la clé symétrique par des essais répétés, ou la détermination mathématique de la clé privée utilisée comme clé d'échange de clé. Un attaquant intelligent va travailler sur le plus facile de ces deux problèmes.

Une réponse naïve au problème de la mise en œuvre est de s'assurer que le système d'échange de clé est toujours significativement plus fort que la clé symétrique ; cela peut être fait en choisissant une très longue clé publique. Un tel concept n'est généralement pas une bonne idée parce que les échanges de clé deviennent beaucoup plus coûteux en termes de temps de traitement lorsque la longueur des clés publiques augmente. Donc, la mise en œuvre se trouve confrontée à la tâche d'essayer de faire correspondre la difficulté de l'attaque contre la clé symétrique avec la difficulté d'une attaque contre le chiffrement de clé publique. Cette analyse n'est pas nécessaire si l'échange de clé peut être effectué avec une extrême sécurité pour un coût presque nul en termes de temps passé ou en travail du CPU ; malheureusement, ce n'est pas le cas pour les méthodes actuelles de clé publique.

Une troisième considération est l'exigence minimum de sécurité de l'utilisateur. Supposons que celui-ci chiffre avec CAST-128 et exige une clé symétrique avec un temps de résistance contre une attaque en force brute de 20 ans. Il pourrait commencer par choisir une clé avec 86 bits d'aléa et ensuite utiliser une fonction unidirectionnelle telle que SHA-1 pour "gonfler" cela en un bloc de 160 bits, et ensuite prendre 128 de ces bits comme clé pour CAST-128. Dans un tel cas, l'algorithme d'échange de clé a seulement besoin de correspondre à une difficulté de 86 bits, et non de 128 bits.

La procédure de sélection est :

1. Déterminer la résistance aux attaques nécessaire pour satisfaire aux exigences de sécurité de l'application. Le faire en estimant le nombre minimum d'opérations informatiques que l'attaquant sera forcé de faire afin de compromettre la sécurité du système et ensuite prendre le logarithme base deux de ce nombre. Appelons cette valeur de logarithme "n".
   Un rapport de 1996 recommandait 90 bits comme un bon choix tout compris pour la sécurité des systèmes. Le nombre de 90 bits devrait être augmenté d'environ 2/3 bits/an, ou environ 96 bits en 2005.
2. Choisir un chiffrement symétrique qui ait une clé d'au moins n bits et au moins cette force cryptanalytique.
3. Choisir un algorithme d'échange de clé avec une résistance à l'attaque d'au moins n bits.
   Une quatrième considération pourrait être la méthode d'authentification de clé publique utilisée pour établir l'identité de l'usager. Cela pourrait être une signature numérique RSA ou une signature numérique DSA. Si le module pour la méthode d'authentification n'est pas assez grand, toute la base de confiance de la communication pourrait alors s'écrouler. L'étape suivante doit donc être ajoutée :
4. Choisir un algorithme d'authentification avec une résistance à l'attaque d'au moins n bits. Cela assure qu'une clé échangée similaire ne peut pas être falsifiée entre les deux parties durant la durée de vie secrète du matériel chiffré. Cela peut n'être pas strictement nécessaire si les clé d'authentification sont changées fréquemment et si elles ont une durée de vie d'utilisation bien comprise, mais au lieu de cela, le conseil des n bits est toujours utile.

1.1 Algorithmes d'échange de clés

La méthode Diffie-Hellman utilise un groupe, un générateur, et des exposants. Dans les normes de l'Internet d'aujourd'hui, l'opération de groupe se fonde sur la multiplication modulaire. Ici, le groupe est défini par le groupe multiplicateur d'un entier, normalement un nombre premier $p = 2q + 1$, où q est un premier, et l'arithmétique est faite modulo p ; le générateur (qui est souvent simplement 2) est noté par g.

Dans Diffie-Hellman, Alice et Bob se mettent d'abord d'accord (en public ou en privé) sur les valeurs de g et p. Alice choisit un grand entier secret au hasard (a), et Bob choisit un grand entier secret au hasard (b). Alice envoie à Bob A, qui

est g^a mod p ; Bob envoie à Alice B, qui est g^b mod p. Ensuite, Alice calcule B^a mod p, et Bob calcule A^b mod p. Ces deux nombres sont égaux, et les participants utilisent une simple fonction de ce nombre comme clé symétrique k.

Noter que l'échange de clé Diffie-Hellman peut être fait sur différentes sortes de représentations de groupe. Par exemple, les courbes elliptiques définies sur des champs finis sont un moyen particulièrement efficace pour calculer l'échange de clé [SCH95].

Pour l'échange de clé RSA, on suppose que Bob a une clé publique (m) qui est égale à p*q, où p et q sont deux nombres premiers secrets, et un exposant de chiffrement e, et un exposant de déchiffrement d. Pour l'échange de clé, Alice envoie à Bob E = k^e mod m, où k est la clé symétrique secrète échangée. Bob récupère k en calculant E^d mod m, et les deux parties utilisent k comme clé symétrique. Alors que l'exposant de chiffrement de Bob, e, peut être assez petit (par exemple, 17 bits) son exposant de déchiffrement, d, aura autant de bits qu'en a m.

2. Détermination de l'effort de factorisation

La méthode RSA de chiffrement à clé publique est immunisée contre les attaques de découverte en force brute parce que le module (et donc, l'exposant secret d) aura au moins 512 bits, et que cela fait trop de possibilités à deviner. L'échange Diffie-Hellman est lui aussi sûr contre la conjecture parce que les exposants vont avoir au moins deux fois plus de bits que la clé symétrique qui sera déduite d'eux. Cependant, les deux méthodes sont susceptibles d'attaques mathématiques pour déterminer la structure des clés publiques.

Factoriser un module RSA va résulter en la compromission complète de la sécurité de la clé privée. Résoudre le problème du logarithme discret pour un système d'exponentiation modulaire Diffie-Hellman va de même détruire la sécurité de tous les échanges de clé qui utilisent ce module particulier. Le présent document suppose que la difficulté de résoudre le problème du logarithme discret est équivalente à la difficulté de factoriser les nombres qui ont la même taille que le module. En fait, il est légèrement plus difficile parce qu'il requiert plus d'opérations ; sur la base d'une évidence empirique jusqu'à présent, le ratio des difficultés est d'au moins 20, et pouvant éventuellement monter à 64. Résoudre l'un ou l'autre problème exige une grande quantité de mémoire pour la dernière étape de l'algorithme, celle de la réduction de matrice. Il reste à voir si cette exigence de mémoire continuera ou non d'être le facteur limitant de la résolution des problèmes de grands entiers. Pour le moment, ce ne l'est pas, et il y a d'actives recherches sur les algorithmes de matrices parallèles qui pourraient atténuer les exigences de mémoire pour ce problème.

Le filtre de champ numérique (NFS, *number field sieve*) [GOR93], [LEN93] est aujourd'hui la meilleure méthode pour résoudre le problème du logarithme discret. La formule pour estimer le nombre d'opérations arithmétiques simples nécessaires pour factoriser un entier, n, en utilisant la méthode NFS est :

L(n) = k * e^((1,92 + o(1)) * cubrt(ln(n) * (ln(ln(n)))^2))

(Note du traducteur : Malgré des recherches poussées, il ne m'a pas été possible d'identifier le terme cubrt dans cette formule, et les auteurs n'ont pas daigné répondre à ma demande. Je sollicite des lecteurs qui connaîtraient la réponse qu'ils veuillent bien me la communiquer à claude.brieredelisle@wanadoo.fr *Merci)*

Nombreux sont ceux qui préfèrent parler du nombre de millions d'instructions par seconde par an (*MIPS years*, ou MY) qui sont nécessaires pour de grosses opérations telles que le filtre de champ numérique. Pour une telle estimation, une opération dans la formule L(n) est une instruction d'ordinateur. L'évidence empirique indique que 4 ou 5 instructions peuvent être une correspondance plus proche, mais c'est un facteur mineur et le présent document s'en tient à une opération/une instruction pour les besoins de cet exposé.

2.1 Choix des paramètres pour l'équation

L'expression ci-dessus a deux paramètres qui peuvent être estimés par des moyens empiriques : k et o(1). Pour la gamme de nombres qui nous intéresse, il y a peu de distinctions à faire entre eux.

On pourrait supposer que k est 1 et o(1) est 0. C'est raisonnablement valide si l'expression n'est utilisée que pour estimer l'effort relatif (au lieu de l'effort réel) et on suppose que le terme o(1) est très petit sur la gamme des nombres qui sont à factoriser.

Ou, on pourrait supposer que o(1) est petit et en gros constant et donc que sa valeur peut être centrée sur k ; puis, estimer k à partir des quantités rapportées d'efforts dépensés à factoriser de grands entiers dans des essais.

Le présent document utilise la seconde approche afin d'obtenir une estimation de la signification du facteur. Il apparaît qu'il est mineur, sur la base des calculs suivants.

Un échantillon de valeurs tirées de travaux récents sur le filtrage du champ numérique inclut :

Nom de l'essai	Nbre de chiffres décimaux	Nombre de bits	MY d'effort
RSA130	130	430	500
RSA140	140	460	2000
RSA155	155	512	8000
RSA160	160	528	3000

Il y a peu de mesures précises de la quantité de temps utilisée pour ces factorisations. Dans la plupart des essais de factorisation, des centaines ou des milliers d'ordinateurs sont utilisés sur une période de plusieurs mois, mais le nombre de leurs cycles qui ont été utilisés pour le projet de factorisation, la distribution précise des types de processeur, les vitesses, et ainsi de suite, ne sont habituellement pas rapportés. Cependant, dans tous les cas ci-dessus, la quantité d'efforts utilisés a été très inférieure à ce que la formule L(n) aurait prédit si k était 1 et o(1) était 0.

Une estimation similaire des efforts, faite en 1995, se trouve dans [ODL95].

On trouvera dans [DL] des résultats qui indiquent que pour la méthode de factorisation de filtrage de champ numérique, le nombre réel d'opérations est inférieur à ce qui était attendu.

2.2 Choix de k à partir de rapports empiriques

En résolvant k à partir de rapports empiriques, il apparaît que k est approximativement 0,02. Cela signifie que la "force de clé effective" de l'algorithme RSA est d'environ 5 ou 6 bits de moins que ce qui est impliqué par la naïve application de l'équation L(n) (qui est de régler k à 1 et o(1) à 0). Ces estimations de k sont très stables sur les nombres rapportés dans le tableau. L'estimation est limitée à un seul chiffre significatif de k parce qu'il exprime des incertitudes réelles ; cependant, l'effet de chiffres supplémentaires n'aurait produit que de minuscules changements de la taille de clé recommandée.

La factorisation de RSA130 a utilisé environ 1700 MY, mais il a été estimé que cela était bien trop élevé pour des besoins de prédiction ; en mettant plus de mémoire sur les machines, il aurait été facilement possible de réduire le temps à 500 MY. Donc, la valeur utilisée pour préparer le tableau ci-dessus était 500. Cette histoire sous évalue cependant la difficulté d'obtenir une mesure précise de l'effort. Le présent document prend l'effort rapporté pour factoriser RSA155 comme étant la mesure la plus précise.

Du résultat de l'examen des données empiriques, il apparaît que la formule L(n) peut être utilisée avec le terme o(1) réglé à 0 et avec k réglé à 0,02 lorsque on parle de factorisation de nombres dans la gamme de 100 à 200 chiffres décimaux. L'équation devient :

L(n) = 0,02 * e^(1,92 * cubrt(ln(n) * (ln(ln(n)))^2))

Pour convertir à partir de simples instructions mathématiques L(n) en MY, on divise par 3*10^13. L'équation pour le nombre de MY nécessaires pour factoriser un entier n se réduit alors à :

MY = 6 * 10^(-16) * e^(1,92 * cubrt(ln(n) * (ln(ln(n)))^2))

Avec quelle confiance cette formule peut-elle être utilisée pour prédire la difficulté de factoriser des nombres légèrement plus grands ? La réponse est que ce devrait être une limite supérieure proche, mais chaque effort de factorisation est normalement marqué par des améliorations des algorithmes ou de leur mise en œuvre qui raccourcit un peu plus le temps de fonctionnement que ce qu'indiquerait la formule.

2.3 Méthode rho de Pollard

Dans les échanges Diffie-Hellman, il y a une seconde approche, la méthode rho de Pollard [POL78]. L'algorithme s'appuie sur la recherche de collisions entre les valeurs calculées dans un espace de grands nombres ; son taux de succès est proportionnel à la racine carrée de la taille de l'espace. À cause de la méthode rho de Pollard, l'espace de recherche de la clé dans un échange de clé DH (l'exposant dans un terme g^a) doit être deux fois celui de la clé symétrique. Donc, pour déduire en toute sécurité une clé de K bits, une mise en œuvre doit utiliser un exposant avec au moins 2*K bits. Voir les détails dans [ODL99].

Lorsque l'échange de clé Diffie-Hellman est fait en utilisant une méthode à courbe elliptique, les méthodes NFS ne sont

d'aucun secours. Cependant, la méthode de collision est encore efficace, et le besoin d'un exposant (appelé un multiplicateur dans les courbes elliptiques) avec 2*K bits subsiste. Le module utilisé pour le calcul peut aussi être de 2*K bits, et cela sera substantiellement plus petit que le module nécessaire pour les méthodes d'exponentiation modulaire lorsque le niveau de sécurité désiré augmente au delà d'une résistance de 64 bits à une attaque en force brute.

On peut se demander comment on peut comparer le nombre d'instructions d'ordinateur réellement nécessaires pour une attaque de logarithme discret sur le nombre nécessaire pour chercher l'espace de clés d'un chiffrement ? En comparant les efforts, on devrait considérer ce qu'est une "opération de base". Pour une recherche en force brute sur l'espace de clés d'un algorithme de chiffrement symétrique comme DES, l'opération de base est le temps de faire un établissement de clé et le temps de faire un chiffrement. Pour les logarithmes discrets, l'opération de base est une élévation au carré modulaire. Le logarithme du ratio de ces deux opérations peut être utilisé comme "facteur de normalisation" entre les deux sortes de calculs. Cependant, même pour de très grands modules (16 kbits) ce facteur se résume à quelques bits d'effort supplémentaire.

2.4 Limites des grandes mémoires et de nombreuses machines

Robert Silverman a examiné la question de quand il est pratique de factoriser des modules RSA plus grands que 512 bits. Son analyse se fonde non seulement sur le nombre théorique d'opérations, mais il inclut aussi des attentes sur la disponibilité des machines actuelles pour effectuer le travail (le présent document se fonde seulement sur le nombre théorique d'opérations). Il examine la question de savoir si on peut ou non s'attendre à ce qu'il y ait assez de machines, de mémoire, et de communication pour factoriser un très grand nombre.

La meilleure méthode de factorisation a besoin d'une mémoire à accès aléatoire pour collecter les relations de données (filtrage) et une étape finale critique qui fait une réduction de rangées sur une grande matrice. Les exigences de mémoire se rapportent à la taille du nombre à mettre en facteurs (ou soumis à la résolution du logarithme discret). Silverman [SILIEEE99] [SIL00] a avancé qu'il y a une limite pratique au nombre de machines et à la quantité de RAM qui peuvent être rassemblées pour s'attaquer à un seul problème dans un avenir prévisible. Il voit deux problèmes à attaquer un module RSA à 1024 bits : les machines qui font le filtrage vont avoir besoin d'espaces d'adresses de 64 bits et la machine de réduction de rangée de la matrice va avoir besoin de plusieurs téraoctets de mémoire. Silverman note que très peu de machines à 64 bits ayant les 170 giga octets de mémoire nécessaire pour le filtrage ont été commercialisées. Presque un milliard de ces machines sont nécessaires pour le filtrage en une quantité de temps raisonnable (un an ou deux).

La conclusion de Silverman, sur la base de l'histoire des efforts de factorisation et de la Loi de Moore, est que les modules RSA de 1024 bits ne seront pas mis en facteurs avant 2037. Cela implique une durée de vie pour les clés RSA beaucoup plus longue que ce qu'indique l'analyse théorique. Il avance que les prévisions sur le nombre de machines et sur les modules de mémoire qui seront disponibles peuvent être faites avec une grande confiance, sur la base des extrapolations de la Loi de Moore et de l'histoire récente des efforts de factorisation.

On devrait donner un grand poids aux considérations pratiques, mais dans une analyse de risques, le monde physique est moins prévisible que ne l'indiquent les courbes de tendance. Quand on considère quelle confiance on peut mettre en l'incapacité de l'industrie informatique à satisfaire la voracité des factoriseurs, on doit avoir quelques idées des considérations économiques qui sont plus compliquées que les mathématiques de factorisation. La demande en mémoire d'ordinateurs est difficile à prédire parce qu'elle se fonde sur les applications : une "application tueuse" peut surgir n'importe quand et plonger l'industrie de la mémoire dans une frénésie de ventes. Le nombre de processeurs disponibles sur les ordinateurs de bureau peut être limité par le nombre de bureaux, mais les systèmes incorporés à grande capacité comptent plus dans les ventes de processeur que les ordinateurs de bureau. Comme les systèmes incorporés absorbent des fonctions de réseautage, il n'est pas inimaginable que des millions de processeurs à 64 bits avec au moins plusieurs giga octets de mémoire envahissent notre environnement.

La limite de cela est que les recommandations de longueur de clé prédites par la théorie pourraient être trop prudentes, mais ce sont elles que nous avons utilisé pour le présent document. Cette question de disponibilité de machine est une de celles qui devraient être reconsidérées à la lumière de la technologie actuelle de façon régulière.

2.5 Machines dédiées à un objet particulier

En août 2003, un concept de "machine de triage" dédiée (TWIRL) est apparu [Shamir2003], et il a substantiellement changé les estimations de coût de mise en facteurs de nombres jusqu'à des tailles de 1024 bits. En appliquant de nombreux composants VLSI à grande vitesse en parallèle, une telle machine pourrait être capable de mener à bien le filtrage de nombres de 512 bits en 10 minutes pour un coût du matériel de 10 000 $. Une plus grosse version pourrait filtrer un nombre de 1024 bits en un an pour un coût de 10 M $. L'article cite certaines avancées dans les approches de l'étape de réduction

des rangées en concluant que la sécurité des modules RSA de 1024 bits est mise en question.

Les estimations sur le temps et le coût de la factorisation de nombres de 512 bits et 1024 bits correspondent à un facteur d'accélération d'environ 2 million par rapport à ce qui pouvait être réalisé avec les processeurs courants d'il y a quelques années.

3. Temps de calcul pour les algorithmes

Cette section décrit le temps que prend l'utilisation des algorithmes pour effectuer les échanges de clés. Là encore, il est important de considérer l'accroissement du temps que prend un échange de clés symétriques lorsque on augmente la longueur des clés publiques. Il est important d'éviter de choisir des longues clés publiques infaisables.

3.1 Échange de clé Diffie-Hellman

Un échange de clés Diffie-Hellman est fait avec un groupe cyclique fini G avec un générateur g et un exposant x. Comme noté dans la méthode rho de Pollard, l'exposant a deux fois le nombre de bits nécessaires pour la clé finale. Soit p la taille du groupe G, soit j le nombre de bits de la représentation en base 2 de p, et soit K le nombre de bits dans l'exposant.

En faisant les opérations qui résultent en une clé partagée, un générateur est élevé à une puissance. La façon la plus efficace de le faire implique d'élever K fois un nombre au carré et de le multiplier plusieurs fois le long du chemin. Chacun des nombres a j/w mots binaires en lui, où w est le nombre de bits dans un mot binaire (aujourd'hui ce sera 32 ou 64 bits). Une supposition naïve serait qu'on a besoin de faire j élévations au carré et j/2 multiplications ; heureusement, une mise en œuvre efficace aura besoin de moins (NB : pour la suite de ce paragraphe, n représente j/w).

Une opération d'élévation au carré n'a pas besoin d'utiliser autant d'opérations qu'une multiplication ; une estimation raisonnable est que l'élévation au carré prend 60 % du nombre des instructions machine d'une multiplication. Si on prépare un tableau des temps avec plusieurs valeurs de petites puissances entières du générateur g, est nécessaire seulement environ un cinquième des multiples que ce que suggère la formule naïve. Donc, on a besoin de faire le travail d'approximativement 0,8*K multiples de n par n nombres de mots. De plus, chaque multiplication et élévation au carré doit être suivie par une réduction modulaire, et une bonne hypothèse est qu'il est aussi difficile de faire une réduction modulaire qu'il l'est de faire une multiplication de mot n par n. Donc, cela prend K réductions pour les élévations au carré et 0,2*K réductions pour les multiplications. En additionnant tout cela, l'effort total pour un échange de clé Diffie-Hellman avec des exposants de K bits et un module de n mots est approximativement de 2*K multiplications de mots n par n.

Pour les processeurs à 32 bits, les entiers qui utilisent moins d'environ 30 mots d'ordinateur dans leur représentation exigent au moins n^2 instructions pour une multiplication de mot n par n. Les grands nombres vont utiliser moins de temps, en se servant des multiplications de Karatsuba, et ils vont s'adapter à environ n^(1,58) pour les plus grands n, mais on laissera cela de côté pour le présent exposé. Noter que les processeurs à 64 bits poussent le nombre de la "transition de Karatsuba" à encore plus de bits.

Le résultat de base est que si on double la taille du groupe d'exponentiation modulaire Diffie-Hellman, on quadruple le nombre d'opérations nécessaires pour le calcul.

3.1.1 Diffie-Hellman avec groupes de courbes elliptiques

Noter que les ratios pour l'effort de calcul comme une fonction de la taille du module tiennent même si on utilise un groupe de courbe elliptique (EC) pour Diffie-Hellman. Cependant, pour une sécurité équivalente, on peut utiliser de plus petits nombres dans le cas de courbes elliptiques. Supposons que quelqu'un ait choisi un groupe d'exponentiation modulaire avec un module de 2048 bits comme étant une mesure de sécurité appropriée pour une application Diffie-Hellman et qu'il veuille déterminer quel avantage il y aurait à utiliser à la place un groupe EC. Le calcul est relativement direct si on suppose qu'en moyenne, il faut un effort 20 fois supérieur pour faire une élévation au carré ou une multiplication dans un groupe EC que dans un groupe d'exponentiation modulaire. Une estimation grossière est qu'un groupe EC avec une sécurité équivalente a environ 200 bits dans sa représentation. Puis, en supposant que le temps est dominé par les opérations de mot n par n, le temps relatif est calculé par : ((2048/200)^2)/20 ≈ 5 : ce qui montre qu'une mise en œuvre de courbe elliptique devrait être cinq fois plus rapide qu'une mise en œuvre d'exponentiation modulaire.

3.2 Chiffrement et déchiffrement RSA

Supposons qu'une clé publique RSA utilise un module avec j bits ; ses facteurs sont deux nombres d'environ j/2 bits chacun. Les temps de calcul espérés pour le chiffrement et le déchiffrement sont différents. Comme précédemment, on note le nombre de mots dans la représentation machine du module par le symbole n.

Le plupart des mises en œuvre de RSA utilisent un petit exposant pour le chiffrement. Un chiffrement peut impliquer seulement 16 élévations au carré et une multiplication, utilisant des opérations de mots n par n. Chaque opération doit être suivie par une réduction modulaire, et donc la complexité du temps est d'environ 16*(0,6 + 1) + 1 + 1 ≈ 28 multiplications de mots n par n.

Le déchiffrement RSA doit utiliser un exposant qui a autant de bits que le module, j. Cependant, le théorème du reste chinois s'applique, et tous les calculs peuvent être faits avec un module de seulement n/2 mots et un exposant de seulement j/2 bits. Le calcul doit être fait deux fois, une pour chaque facteur. L'effort est équivalent à 2*(j/2) (n/2 par n/2) multiplications de mots. Comme multiplier des nombres avec n/2 mots est seulement 1/4 de la difficulté de multiplier des nombres avec n mots, l'effort équivalent pour le déchiffrement RSA est j/4 multiplications de mots n par n.

Si on double la taille du module pour RSA, les multiplications n par n vont prendre quatre fois plus de temps. De plus, le temps de déchiffrement double parce que l'exposant est plus grand. Le coût d'adaptation global est un facteur de 4 pour le chiffrement, et un facteur de 8 pour le déchiffrement.

3.3 Exemples tirés de la réalité

Pour donner un peu plus de réalité à ces nombres, voici quelques exemples de mises en œuvre logicielles qui fonctionnent sur un matériel qui était courant quelques années avant la publication de ce document. Les exemples sont inclus pour montrer des estimations grossières de mises en œuvre raisonnables ; ce ne sont pas des étalonnages. Comme avec tout logiciel, les performances vont dépendre des détails exacts de la spécialisation du code au problème et du matériel spécifique.

Le meilleur temps informellement rapporté pour une exponentiation modulaire de 1024 bits (le côté déchiffrement de RSA à 2048 bits) est 0,9 ms (environ 450 000 cycles de CPU) sur un processeur Itanium à 500 MHz. Cela montre que les processeurs les plus récents ne perdent pas pied sur les opérations de grands nombres ; le nombre d'instructions est moins que ce qu'un processeur à 32 bits utilise pour une exponentiation modulaire de 256 bits.

Pour des temps de processeurs moins avancés, les deux tableaux suivants (calculés par Tero Monenen à SSH Communications) pour une exponentiation modulaire, telle qu'il en serait fait dans un échange de clé Diffie-Hellman.

Celeron 400 MHz ; compilé avec le compilateur C GNU, avec des optimisations de codages spécifiques de la plateforme :

Type de groupe	Taille de module	Taille d'exposant	Temps
mod	768	~150	18 ms
mod	1024	~160	32 ms
mod	1536	~180	82 ms
ecn	155	~150	35 ms
ecn	185	~200	56 ms

Le type de groupe vient de la [RFC2409] et est soit exponentiation modulaire ("mod") soit courbe elliptique ("ecn"). Toutes les tailles, ici et dans les tableaux suivants, sont en bits.

Alpha 500 MHz compilé avec le compilateur C de Digital, optimisé, pas de code spécifique de la plateforme:

Type de groupe	Taille de module	Taille d'exposant	Temps
mod	768	~150	12 ms
mod	1024	~160	24 ms
mod	1536	~180	59 ms
ecn	155	~150	20 ms
ecn	185	~200	27 ms

Les deux tableaux suivants (calculés par Eric Young) étaient à l'origine pour des opérations de signature RSA, en utilisant la représentation du reste chinois. Pour faciliter la compréhension, les paramètres sont présentés ici pour montrer les calculs internes, c'est-à-dire, la taille du module et l'exposant utilisé par le logiciel.

Dual Pentium II-350 :

Taille de module équivalent	Taille d'exposant équivalent	Temps équivalent
256	256	1,5 ms
512	512	8,6 ms
1024	1024	55,4 ms
2048	2048	387 ms

Alpha 264 600 Mhz :

Taille de module équivalent	Taille d'exposant équivalent	Temps équivalent
512	512	1,4 ms

Les puces récentes qui accélèrent l'exponentiation peuvent effectuer des exponentiations de 1024 bits (module de 1024 bits, exposant de 1024 bits) en environ 3 millisecondes ou moins.

4. Équivalences des tailles de clé

Afin de déterminer quelle force doit avoir une clé publique pour protéger une clé symétrique particulière, on a d'abord besoin de déterminer quel effort est nécessaire pour casser la clé symétrique. De nombreux protocoles de sécurité Internet exigent l'utilisation de TripleDES pour un chiffrement symétrique fort, et on s'attend à ce que la norme de chiffrement évolué (AES, *Advanced Encryption Standard*) soit adoptée sur l'Internet dans les années à venir. Donc, ces deux algorithmes sont exposés ici. Dans cette section, pour illustrer notre propos, on va implicitement supposer que l'exigence de sécurité du système est de 112 bits ; cela ne signifie pas que 112 bits soient recommandés. En fait, 112 bits est indiscutablement trop fort pour toute application pratique. C'est utilisé pour illustration, simplement parce que c'est la limite supérieure de la force de TripleDES.

Si on pouvait simplement déterminer le nombre de MY qu'il faut pour casser TripleDES, la tâche de calculer la taille de la clé publique de force équivalente serait aisée. Malheureusement, ce n'est pas le cas ici parce qu'il y a beaucoup d'exemples de matériels spécifiques de DES qui chiffrent plus vite que DES dans un logiciel sur un CPU standard. On doit plutôt déterminer le coût équivalent pour qu'un système casse TripleDES et qu'un système casse la clé publique qui protège une clé TripleDES.

En 1998, la Fondation Frontière électronique (EFF, *Electronic Frontier Foundation*) a construit une machine à casser DES [GIL98] pour 130 000 US $ qui pouvait tester environ 1e11 clés DES par seconde (de l'argent a été dépensé en plus pour la conception de la machine). Les constructeurs de la machine admettent tout à fait que celle-ci n'est pas entièrement optimisée, et il est estimé que dix fois cette somme pourrait probablement créer une machine environ 50 fois plus rapide. En supposant une meilleure optimisation en imaginant qu'un système pour tester les clés TripleDES fonctionne à peu près aussi vite qu'un système pour tester les clés DES, donc approximativement 1 million de US $ pourrait tester 5e12 clés TripleDES par seconde.

Au cas où vos adversaires seraient beaucoup plus riches que l'EFF, on pourrait supposer qu'ils auront 1 milliard de US $, assez pour tester 5e18 clés par seconde. Une recherche exhaustive de l'espace effectif de TripleDES de 2^112 clés avec ce système assez coûteux prendrait environ 1e15 secondes soit environ 33 millions d'années. (Noter qu'un tel système aurait aussi besoin de 2^60 octets de RAM [MH81], ce qui est considéré comme gratuit dans ce calcul). Cela semble une valeur trop prudente. Cependant, si la vitesse de la logique informatique continue d'augmenter conformément à la Loi de Moore (doublement de la vitesse tous les un an et demi) on peut alors s'attendre à ce que dans environ 50 ans, le calcul pourrait être réalisé en seulement un an. Pour les besoins de notre illustration, cette résistance de 50 ans contre un milliardaire est supposée être l'exigence minimale de sécurité pour un ensemble d'applications.

Si 112 bits de résistance à une attaque est l'exigence de sécurité du système, alors le système d'échange de clés pour TripleDES devrait avoir une difficulté équivalente ; c'est à dire que si l'attaquant a 1 milliard de US $, vous voulez qu'il dépense tout cet argent pour acheter aujourd'hui du matériel en sachant qu'il va "casser" l'échange de clés en pas moins de 33 millions d'années. (Évidemment, un attaquant rationnel va attendre environ 45 ans avant de dépenser réellement l'argent, parce qu'il pourra alors obtenir un bien meilleur matériel, mais tous les attaquants bénéficient de cette sorte d'égalité dans l'attente.)

On estime qu'un CPU de PC normal d'il y a juste quelques années peut générer plus de 500 MIP et pourrait être acheté pour environ 100 US $ en quantités ; donc on obtient plus de 5 MIP/US $. Là encore, ce nombre double environ tous les 18 mois. Pou un milliard de dollars US, un attaquant peut obtenir 5e12 MIP an d'instructions informatiques sur ce matériel récent. Ce chiffre est utilisé dans les estimations suivantes des coûts équivalente pour casser les systèmes d'échange de clés.

4.1 Équivalence de clé contre matériel dédié au passage en force

Si l'attaquant milliardaire utilise des CPU conventionnels pour "casser" un échange de clés pour une clé de 112 bits dans le même temps que la machine dédiée passe à une recherche en force brute pour la clé symétrique, le système d'échange de clés doit utiliser un module de grandeur appropriée. En supposant que le milliardaire effectue 5e12 MIP d'instructions par an. Utiliser l'équation suivante pour estime la taille de module à utiliser avec le chiffrement RSA ou l'échange de clé DH :

5*10^33 = (6*10^-16)*e^(1,92*cubrt(ln(n)*(ln(ln(n)))^2))

Ce qui, résout approximativement pour n, donne : n = 10^(625) = 2^(2077)

Donc, en supposant des vitesses logiques similaires et l'efficacité actuelle du filtre de champ numérique, des modules d'environ 2100 bits auront environ la même résistance contre l'attaque qu'une clé TripleDES de 112 bits. Cela indique que le chiffrement de clé publique RSA devrait utiliser un module avec environ 2100 bits ; pour un échange de clés Diffie-Hellman, on pourrait utiliser un module légèrement plus petit, mais la différence n'est pas significative.

4.2 Équivalence de clé contre attaque en force de CPU conventionnel

Un autre façon d'estimer cela est de supposer que l'attaquant a moins d'exigences : il doit seulement "casser" l'échange de clés en moins de temps qu'une recherche en force brute contre la clé symétrique prendrait avec des ordinateurs non dédiés. C'est une comparaison "de pommes à pommes", parce qu'elle suppose que l'attaquant a seulement besoin de faire un calcul proportionnel à son effort, et non construit sur sa fortune personnelle ou celle du pays. Le module de clé publique sera plus long que celui du paragraphe 4.1, parce que la clé symétrique va devoir être viable pour une plus longue durée.

On suppose que le nombre d'instructions de CPU pour chiffrer un bloc de matériel utilisant TripleDES est 300. Le nombre estimé d'instructions d'ordinateur pour casser une clé TripleDES de 112 bits est :

300 * 2^112 = 1,6 * 10^(36)
= 0,02*e^(1,92*cubrt(ln(n)*(ln(ln(n)))^2))

Résoudre cela donne approximativement pour n : n = 10^(734) = 2^(2439)

Donc, pour des attaques de CPU tout venant, on peut supposer que les modules avec environ 2400 bits auront à peu près la même force contre l'attaque qu'une clé TripleDES de 112 bits. Cela indique que le chiffrement de clé publique RSA devrait utiliser un module avec environ 2400 bits ; pour un échange de clés Diffie-Hellman, on pourrait utiliser un module légèrement plus petit, mais cela ne fait pas une différence significative.

Noter que certains auteurs supposent que les algorithmes sous-jacents au crible de champs numérique vont continuer de s'améliorer au fil du temps. Ces auteurs recommandent un module encore plus grand, de plus de 4000 bits, pour protéger une clé symétrique de 112 bits sur 50 ans. Cela souligne la difficulté de la sécurité cryptographique à long terme : il est tout à fait impossible de prédire les progrès des mathématiques et de la physique sur d'aussi longues périodes.

4.3 Attaque d'un an : 80 bits de force

En supposant qu'un milliardaire dépense son argent pour acheter aujourd'hui du matériel, quelle taille d'échange de clés pourrait il "casser" en un an ? Il peut effectuer 5*e12 MY d'instructions, ou

3*10^13 * 5*10^12 = 0,02*e^(1,92*cubrt(ln(n)*(ln(ln(n)))^2))

Résoudre cela pour une approximation de n donne : n = 10^(360) = 2^(1195)

C'est à peu près autant d'opérations qu'il faudrait pour casser une clé symétrique de 80 bits par force brute.

Donc, pour protéger des données qui ont une exigence de secret d'un an contre un attaquant incroyablement riche, un module d'échange de clés d'environ 1200 bits protégeant une clé symétrique de 80 bits est sûr, même contre les ressources d'une nation.

4.4 Équivalence de clé pour autres chiffrements

L'extension de cette logique à AES est très facile. Pour les besoins d'une estimation de la recherche de clé, on peut penser que les 128 bits d'AES sont au moins plus forts de 16 bits que TripleDES mais trois fois plus rapide. Le temps et le coût d'une attaque en force brute est approximativement 2^(16) plus que pour TripleDES, et donc, dans l'hypothèse où 128 bits

de force est l'objectif de sécurité désiré, la taille recommandée du module d'échange de clés est plus long d'environ 700 bits.

Si il est possible de concevoir un matériel pour casser AES qui soit considérablement plus efficace que le matériel pour casser DES, alors (là encore, dans l'hypothèse que la force de l'échange de clé doit correspondre à l'effort de force brute) les modules pour protéger l'échange de clé peuvent être plus petits. Cependant, l'existence de tels projets est une pure spéculation à ce moment précoce de la vie d'AES.

Les chiffrements AES ont des tailles de clés de 128 bits jusqu'à 256 bits. Est ce qu'une exigence prudente de sécurité minimum, et donc les modules d'échange de clés, ont des forces similaires ? La réponse à cette question dépend de si on pense que la Loi de Moore va continuer de s'appliquer sans ralentir. Si elle continue, on devrait s'attendre à ce que les clés de 128 bits soient sûres pour environ 60 ans, et les clés de 256 bits seraient sûres pour encore 400 ans au delà, bien plus loin que n'importe quelle exigence de sécurité imaginable. Mais un tel progrès est difficile à prédire, car il excède les capacités physiques des appareils d'aujourd'hui et impliquerait l'existence de technologies logiques qui sont inconnues ou infaisables aujourd'hui. Le calcul quantique est un candidat, mais on en sait trop peu aujourd'hui pour faire des prédictions fiables sur son applicabilité à la cryptographie (qui peut elle-même changer sur les 100 prochaines années !).

Si la Loi de Moore ne continue pas à tenir, si aucun nouveau paradigme de calcul n'émerge, alors les clés de plus de 100 bits de long pourraient être sûres "pour toujours". Noter cependant que d'autres ont fait des estimations fondées sur des hypothèses d'émergence de nouveaux paradigmes de calcul. Par exemple, l'article de Lenstra et Verheul fondé sur la Toile "Choisir des tailles de clés de chiffrement" choisit une analyse plus prudente que celle du présent document.

4.5 Fonctions de hachage pour déduire des clés symétriques d'algorithmes de clé publique

L'algorithme Diffie-Hellman résulte en une clé qui est longue de centaines ou de milliers de bits, mais le chiffrement demande beaucoup moins de bits que cela. Comment peut on distiller une longue clé en une courte sans perdre de la force ?

Les fonctions de hachage cryptographiques unidirectionnelles sont les pierres angulaires de cette construction, et tant qu'elles utilisent toutes la clé Diffie-Hellman pour déduire chaque bloc de la clé symétrique, elles produisent des clés de force suffisante.

La recommandation habituelle est d'utiliser une bonne fonction unidirectionnelle appliquée au matériel de base (le résultat de l'échange de clés) et d'utiliser un sous-ensemble du résultat de la fonction de hachage pour la clé. Cependant, si la longueur de clé désirée est supérieure au résultat de la fonction de hachage, on peut se demander comment réconcilier les deux.

Les étapes de la déduction de bits de clé supplémentaires doivent satisfaire les exigences suivantes :
- Les bits doivent ne révéler aucune information sur l'échange de clés secrètes
- Les bits ne doivent pas être corrélés entre eux
- Les bits doivent dépendre de tous les bits de l'échange de clé secrète.

Toute bonne fonction de hachage cryptographique satisfait à ces trois exigences. Noter que le nombre de bits de résultat de la fonction de hachage n'est pas spécifié. C'est parce que même une fonction de hachage avec un résultat très court peut être itérée pour produire plus de bits non corrélés en faisant un peu attention.

Par exemple, SHA-1 a 160 bits de résultat. Pour déduire une clé de résistance aux attaques de 160 bits ou moins, SHA(DHkey) produit une bonne clé symétrique.

Supposons qu'on veuille une clé avec une résistance aux attaques de 160 bits, mais qu'elle soit utilisée avec un chiffrement qui utilise des clés de 192 bits. On peut itérer SHA-1 comme suit :

Bits 1-160 de la clé symétrique = K1 = SHA(DHkey | 0x00) (c'est-à-dire, enchaîner un seul octet de valeur 0x00 au côté droit de la clé DH, et ensuite hacher)
Bits 161-192 de la clé symétrique = K2 = choisir_32_bits(SHA(K1 | 0x01))

Mais qu'en est-il si on veut 192 bits de force pour le chiffrement ? Le calcul approprié est :

Bits 1-160 de la clé symétrique = SHA(0x00 | DHkey)
Bits 161-192 de la clé symétrique = choisir_32_bits(SHA(0x01 | DHkey))

(Noter que dans la description ci-dessus, au lieu d'enchaîner un octet complet, enchaîner un seul bit serait aussi suffisant.)

La distinction importante est que dans le second cas, la clé DH est utilisée pour chaque partie de la clé symétrique. Cela assure que l'entropie de la clé DH n'est pas perdue par l'itération de la fonction de hachage sur les mêmes bits.

Du point de vue de l'efficacité, si la clé symétrique doit avoir une grosse quantité d'entropie, il est probablement meilleur d'utiliser une fonction de hachage cryptographique avec un grand bloc de résultat (192 bits ou plus) plutôt que d'itérer un plus petit bloc.

De nouveaux algorithmes de hachage avec de plus longs résultats (comme SHA-256, SHA-384, et SHA-512) peuvent être utilisés avec le même niveau de sécurité que l'algorithme d'extension décrit ci-dessus.

4.6 Importance de l'aléa

Certains des calculs décrits dans ce document exigent des entrées aléatoires; par exemple, les exposants secrets Diffie-Hellman doivent être choisis sur la base de n bits vraiment aléatoires (où n est l'exigence de sécurité du système). Le nombre de bits vraiment aléatoires est extrêmement important pour déterminer la force du résultat des calculs. On ferme souvent les yeux sur l'utilisation de nombres vraiment aléatoires, et de nombreuses applications de sécurité ont été significativement affaiblies par l'utilisation d'entrées insuffisamment aléatoires. Une description beaucoup plus complète de l'importance des nombres aléatoires se trouve dans la [RFC1750].

5. Conclusion

Dans le tableau qui suit on suppose que les attaquants utilisent des ordinateurs tout venant, que le matériel est acheté en 2000, et que les connaissances mathématiques pertinentes pour le problème restent les mêmes qu'aujourd'hui. C'est une pure comparaison de "pommes avec des pommes" qui montre comment le délai pour n échanges de clés s'adapte par rapport à l'exigence de force. La taille de sous-groupe pour DSA est incluse, si il est utilisé pour prendre en charge l'authentification au titre du protocole ; le module DSA doit être aussi long que le module DH, mais la taille du sous-groupe "q" est aussi pertinente.

Exigences système pour la résistance aux attaques (bits)	**Taille de clé symétrique** (bits)	**Taille de module RSA ou DH** (bits)	**Taille de sous-groupe DSA** (bits)
70	70	947	129
80	80	1228	148
90	90	1553	167
100	100	1926	186
150	150	4575	284
200	200	8719	383
250	250	14596	482

5.1 Correction de TWIRL

Si la machine TWIRL devient une réalité, et si il y a des avancées dans le parallélisme pour la réduction des rangées dans la factorisation, les estimations prudentes vont soustraire environ 11 bits de la colonne Sécurité du système dans le tableau. Donc, afin d'obtenir 89 bits de sécurité, on aura besoin d'un module RSA d'environ 1900 bits.

6. Considérations pour la sécurité

Les équations et valeurs données dans le présent document sont aussi précises que possible, sur la base de l'état de l'art dans les ordinateurs tout venant au moment de la rédaction de ce document. Aucune prévisions ne peut être complètement précise, et les formules données ici ne sont pas conçues comme des déclarations définitives de fait sur les forces cryptographiques. Par exemple, certains des résultats empiriques utilisés dans le calibrage des formules de ce document ne sont probablement pas entièrement précises, et cette imprécision affecte les estimations. L'espoir des auteurs est que les chiffres présentés ici diffèrent aussi peu que possible de l'expérience du monde réel.

7. Références

[DL] Dodson, B. et A. K. Lenstra, "NFS with four large primes: an explosive experiment", Proceedings Crypto 95, Lecture Notes in Comput. Sci. 963, (1995) 372-385.

[GIL98] John Gilmore (Ed.), "Cracking DES: Secrets of Encryption Research, Wiretap Politics & Chip Design", Electronic Frontier Foundation, 272 pages, mai 1998, O'Reilly & Associates ; ISBN : 1565925203

[GOR93] Gordon, D., "Discrete logarithms in GF(p) using the number field sieve", SIAM Journal on Discrete Mathematics, 6 (1993), 124-138.

[LEN93] Lenstra, A. K. et H. W. Lenstra, Jr. (eds), The development of the number field sieve, Lecture Notes in Math, 1554, Springer Verlag, Berlin, 1993.

[MH81] Merkle, R.C., et Hellman, M., "On the Security of Multiple Encryption", Communications of the ACM, v. 24 n. 7, 1981, pp. 465-467.

[ODL95] A. M. Odlyzko, "The Future of Integer Factorization", RSA Labs Cryptobytes, Volume 1, n° 2 - été 1995;

[ODL99] A. M. Odlyzko, "Discrete logarithms: The past et the future, Designs, Codes, et Cryptography" (1999).

[POL78] J. Pollard, "Monte Carlo methods for index computation mod p", Mathematics of Computation, 32 (1978), 918-924.

[RFC1750] D. Eastlake, 3rd et autres, "Recommandations d'aléa pour la sécurité", décembre 1994. *(Info., remplacée par la* RFC4086)

[RFC2409] D. Harkins et D. Carrel, "L'échange de clés Internet (IKE)", novembre 1998. *(Obsolète, voir la RFC4306)*

[SCH95] R. Schroeppel, et al., "Fast Key Exchange With Elliptic Curve Systems", In Don Coppersmith, editor, Advances in Cryptology – CRYPTO, 31 août 1995. Springer-Verlag

[SHAMIR03] Shamir, Adi et Eran Tromer, "Factoring Large Numbers with the TWIRL Device", Advances in Cryptology - CRYPTO 2003, Springer, Lecture Notes in Computer Science 2729.

[SIL00] R. D. Silverman, "A Cost-Based Security Analysis of Symmetric et Asymmetric Key Lengths". RSA Laboratories Bulletin, n° 13 - avril 2000,

[SILIEEE99] R. D. Silverman, "The Mythical MIPS Year", IEEE Computer, août 1999.

8. Adresse des auteurs

Hilarie Orman
Purple Streak Development
500 S. Maple Dr.
Salem, UT 84653
mél : hilarie@purplestreak.com
ho@alum.mit.edu

Paul Hoffman
VPN Consortium
127 Segre Place
Santa Cruz, CA 95060 USA
mél : paul.hoffman@vpnc.org

9. Déclaration complète de droits de reproduction

Remerciement

Le financement de la fonction d'édition des RFC est actuellement fourni par l'Internet Society.