Nouveaux records de factorisation et de calcul de logarithme discret

Fabrice Boudot, Pierrick Gaudry, Aurore Guillevic, Nadia Heninger, Emmanuel Thomé, Paul Zimmermann

Nouveaux records de factorisation et de calcul de logarithme discret

New factorization and discrete logarithm record computations

par **Fabrice Boudot**

Professeur de l'Éducation Nationale

Université de Limoges, XLIM, UMR 7252, F-87000 Limoges, France

par **Pierrick Gaudry**

Directeur de Recherche CNRS

Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France

par **Aurore Guillevic**

Chargée de Recherche Inria

Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France

par **Nadia Heninger**

Associate Professor

University of California, San Diego, USA

par **Emmanuel Thomé**

Directeur de Recherche Inria

Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France

par **Paul Zimmermann**

Directeur de Recherche Inria

Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France

Résumé

Cet article décrit deux nouveaux records établis fin 2019 : un record de factorisation d'entier avec la factorisation du nombre RSA-240, et un record de calcul de logarithme discret de même taille. Ces deux records correspondent à des nombres de 795 bits, soit 240 chiffres décimaux, et ont été établis avec le même logiciel libre (CADO-NFS), sur le même type de processeurs. Ces records servent de référence pour les recommandations en termes de taille de clé pour les protocoles cryptographiques.

Abstract

This article describes two new records established at the end of 2019 : an integer factorization record for the factorization of RSA-240, and a discrete logarithm record of the same size. These two records correspond to 795-bit numbers, or 240 decimal digits, and were established with the same open-source CADO-NFS software, on the same type of processors. These records serve as a reference for key size recommendations for cryptographic protocols.

Mots-clés

factorisation d'entier, logarithme discret, cryptographie à clé publique, crible algébrique, CADO-NFS

Keywords

integer factorization, discrete logarithm, public-key cryptography, Number Field Sieve, CADO-NFS

Table des matières

Introduction

La cryptographie à clé publique a connu un essor notable depuis son introduction en 1976–1977. Elle repose sur des fonctions mathématiques qui se calculent rapidement dans un sens mais dont l'inverse est extrêmement difficile à calculer. La multiplication de deux grands entiers premiers est simple sur un ordinateur, mais factoriser un tel produit est bien plus difficile et fait l'objet d'une compétition internationale. Cet article présente l'état de l'art pour le chiffrement RSA (Rivest–Shamir–Adleman) basé sur la difficulté de la factorisation de très grands entiers, et pour le chiffrement Diffie–Hellman basé sur la difficulté d'inverser une exponentiation dans certains groupes mathématiques. En 2019 le record de factorisation d'un produit de 240 chiffres décimaux a été obtenu en près de mille années-cœurs sur plusieurs grappes de calcul. L'intérêt de ces records est d'extrapoler les tailles cryptographiques de clé pour différents besoins de chiffrement et durées de protection.

1 État de l'art

1.1 Ron Rivest, Adi Shamir, Leonard Adleman, factorisation d'entier

L'algorithme de chiffrement RSA a été inventé par Ron Rivest, Adi Shamir et Leonard Adleman en 1977. Il fut l'un des premiers algorithmes de chiffrement dit à clé publique, c'est-à-dire que le chiffrement d'un message ne requiert pas que l'expéditeur partage un secret avec le destinataire. Il est devenu, depuis, l'un des algorithmes de chiffrement et d'authentification les plus utilisés au monde : on le retrouve ainsi dans le protocole `https` qui sécurise les sites internet ou dans le logiciel de chiffrement de messages PGP.
Lorsqu'Alice souhaite envoyer un message secret m à Bob, le fonctionnement de cet algorithme de chiffrement est le suivant :

— Bob choisit deux très grands nombres premiers p et q, et calcule leur produit $n = p \cdot q$. Il choisit également un nombre e premier avec $p-1$ et $q-1$, et calcule son inverse d modulo $(p-1)(q-1)$. Il envoie les deux nombres n et e à Alice.

— Pour chiffrer le message m, Alice calcule $c = m^e \bmod n$, et envoie le message chiffré c à Bob.

— Pour déchiffrer le message c, Bob calcule $c^d \bmod n$ et retrouve ainsi le message original m.

Taille de n	Date	Auteurs
100 chiffres	1er avril 1991	Lenstra
110 chiffres	14 avril 1992	Lenstra, Manasse
120 chiffres	9 juillet 1993	Denny, Dodson, Lenstra, Manasse
129 chiffres	26 avril 1994	Atkins, Graff, Lenstra, Leyland
130 chiffres	10 avril 1996	Cowie, Dodson, Elkenbracht-Huizing, Lenstra, Montgomery, Zayer
140 chiffres	2 février 1999	Cavallar, Dodson, Lenstra, Leyland, Lioen, Montgomery, Murphy, te Riele, Zimmermann
155 chiffres	22 août 1999	Cavallar, Dodson, Lenstra, Lioen, Montgomery, Murphy, te Riele, Aardal, Gilchrist, Guillerm, Leyland, Marchand, Morain, Muffett, Putnam, Putnam, Zimmermann
158 chiffres	19 janvier 2002	Bahr, Franke, Kleinjung
160 chiffres	1er avril 2003	Bahr, Franke, Kleinjung, Lochter, Böhm
174 chiffres	3 décembre 2003	Franke, Kleinjung, Montgomery, te Riele, Bahr, NFSNET
176 chiffres	2 mai 2005	Aoki, Kida, Shimoyama, Ueda
200 chiffres	9 mai 2005	Bahr, Böhm, Franke, Kleinjung
232 chiffres	12 décembre 2009	Kleinjung, Aoki, Franke, Lenstra, Thomé, Bos, Gaudry, Kruppa, Montgomery, Osvik, te Riele, Timofeev, Zimmermann
240 chiffres	2 décembre 2019	Boudot, Gaudry, Guillevic, Heninger, Thomé, Zimmermann
250 chiffres	28 février 2020	Boudot, Gaudry, Guillevic, Heninger, Thomé, Zimmermann

Tableau 1 – Records de factorisation d'entier depuis l'instauration du challenge RSA.

La sécurité de cet algorithme repose sur le fait qu'un attaquant qui observerait les communications entre Alice et Bob connaîtrait les valeurs de n, e et c mais ne pourrait retrouver la valeur du message m. En effet, ce qui permet à Bob de retrouver le message clair est de connaître la valeur de d, qui a elle-même été calculée en utilisant les deux nombres premiers p et q.
Cet attaquant devrait donc, à partir du nombre n, retrouver les deux facteurs premiers p et q qui le composent.

Problème de la factorisation d'entier. Étant donné un entier naturel $n = p \cdot q$, avec p et q premiers, retrouver p et q.

Résoudre ce problème est simple lorsque le nombre n est petit. Il s'agit simplement d'écrire, par exemple, le nombre 91 sous la forme $91 = 7 \cdot 13$ ou encore le nombre 2701 sous la forme $2701 = 37 \cdot 73$. Néanmoins, il n'existe pas, jusqu'à présent, d'algorithme efficace pour factoriser un nombre entier qui est le produit de deux nombres premiers de plusieurs centaines de chiffres.
La création de l'algorithme de chiffrement RSA a ainsi accéléré la recherche d'un algorithme efficace pour factoriser un nombre entier. Dans les années 1980, plusieurs nouveaux algorithmes sont apparus, comme par exemple le crible quadratique (C. Pomerance, 1981) ou la méthode des courbes elliptiques (H. W. Lenstra, Jr., 1987).
Afin de prouver la robustesse de leur algorithme de chiffrement face aux avancées mathématiques sur le problème de la factorisation d'entier, la société RSA, fondée par les trois inventeurs de l'algorithme de chiffrement, propose un défi en mars 1991. Ils fabriquent puis publient une liste de nombres n, dont la taille varie entre 100 et 617 chiffres, et proposent une prime, se montant à \$200 000 pour le plus grand de ces nombres, à la première personne qui trouvera sa factorisation.
Dès lors, et bien que la société RSA ait retiré les primes associées à chaque nombre, cette liste de nombres reste la référence pour réaliser les records de factorisation d'entier. Le tableau 1 donne la liste de ces records établis depuis 1991 et jusqu'à aujourd'hui.
Le record précédant nos travaux, la factorisation d'un entier de 232 chiffres, fut établi en 2009 et aura tenu pendant près de 10 ans. Celui-ci avait mobilisé cinq instituts académiques différents qui

unirent leurs forces pendant plus de deux ans, et avait demandé une puissance de calcul équivalente à 1700 années de fonctionnement d'un cœur processeur.

1.2 Whitfield Diffie, Martin Hellman, logarithme discret

En 1976, W. Diffie et M. Hellman publient un schéma d'échange de clé qui n'a pas besoin de secret commun préalable. Il repose sur la difficulté de calculer un *logarithme discret*. Ici aussi on manipule des entiers, modulo un nombre premier p. Cela veut dire que l'on considère les entiers de 0 à $p-1$. Lorsqu'une addition, soustraction ou multiplication de deux entiers est en dehors des bornes 0 et $p-1$, on considère le reste de sa division euclidienne par p pour se ramener dans l'intervalle, cela s'appelle calculer modulo p. On note cet ensemble $\mathbb{Z}/p\mathbb{Z}$. C'est un corps fini. Si l'on considère les éléments non nuls (entre 1 et $p-1$), on peut trouver un générateur g parmi eux, tel que $g, g^2, g^3, \ldots, g^{p-1} = 1$ (modulo p) prend toutes les valeurs possibles entre 1 et $p-1$. Ce sous-ensemble est appelé un groupe.

Problème du logarithme discret (DLP) dans $\mathbb{Z}/p\mathbb{Z}$. Étant donné un nombre premier p, un élément *générateur* g qui engendre tout le groupe, et un élément cible h, calculer un entier x, $0 \leqslant x < p$ tel que $g^x = h$.

On parle de *logarithme* car c'est une fonction inverse de l'exponentiation, *discret* car il prend des valeurs entières, entre 0 et $p-1$.
De façon simplifiée, deux entités A et B s'accordent sur un nombre premier p tel que $(p-1)/2$ est premier, et un générateur g. A choisit au hasard un entier a entre 1 et $p-1$, de même B choisit au hasard un entier b dans cet intervalle. A envoie g^a mod p à B et B envoie g^b mod p à A. Puis A connaissant g^b et a, calcule $(g^b)^a$ mod p, et de même B peut calculer $(g^a)^b$ mod p. Par la suite A et B connaissent le secret g^{ab}. Un attaquant qui peut lire les communications ne connaît que p, g, g^a, g^b et ne peut pas calculer g^{ab}, mais seulement g^{a+b}. Cette difficulté de casser ce schéma a été formalisée comme suit.

Problème de Diffie–Hellman. Étant donné un nombre premier p définissant $\mathbb{Z}/p\mathbb{Z}$, et un *générateur* g du groupe, pour (g, g^x, g^y) avec x, y inconnus, calculer g^{xy}.

Si l'on sait calculer efficacement un logarithme discret, alors cela permet de résoudre le problème de Diffie–Hellman.
Pour calculer un logarithme discret dans $\mathbb{Z}/p\mathbb{Z}$, les algorithmes génériques dits «pas-de-bébé–pas-de-géant» (coûteux en mémoire) ou «Pollard-ρ» ont une complexité en $O(\sqrt{p})$. La réduction de Pohlig–Hellman permet de paralléliser les algorithmes génériques dans chacun des sous-groupes d'ordre premier. Pour cela, il faut d'abord factoriser $p-1$. Pour éviter cette réduction, qui affaiblirait la construction cryptographique, on choisit des nombres premiers p tels que $(p-1)/2$ est premier, ou est divisible par un très grand nombre premier ℓ, et g est un générateur de ce sous-groupe de cardinal ℓ. Le logarithme discret x est alors un entier dans l'intervalle $0 \leqslant x < \ell$.
Il existe des algorithmes sous-exponentiels spécifiques aux corps finis, le cas des corps $\mathbb{Z}/p\mathbb{Z}$ (p premier) est détaillé ici. Ces algorithmes se sont améliorés depuis leur introduction dans les années 1970, la figure 1 retrace les records de calcul dans les corps premiers.

1.3 Tailles de clé recommandées

Le choix de tailles de clé pour lesquelles le problème de la factorisation d'entier, ou le problème du logarithme discret, sont les plus difficiles possibles, est un domaine actif de recherche. La communauté scientifique s'accorde pour recommander l'emploi de systèmes qui s'appuient sur le problème du logarithme discret sur les courbes elliptiques, et à défaut, sur le problème RSA pour un entier n suffisamment grand, ou sur le problème de Diffie–Hellman dans des corps finis $\mathbb{Z}/p\mathbb{Z}$, avec p premier suffisamment grand. Ici, «suffisamment grand» est généralement compris comme au moins 2048 bits,

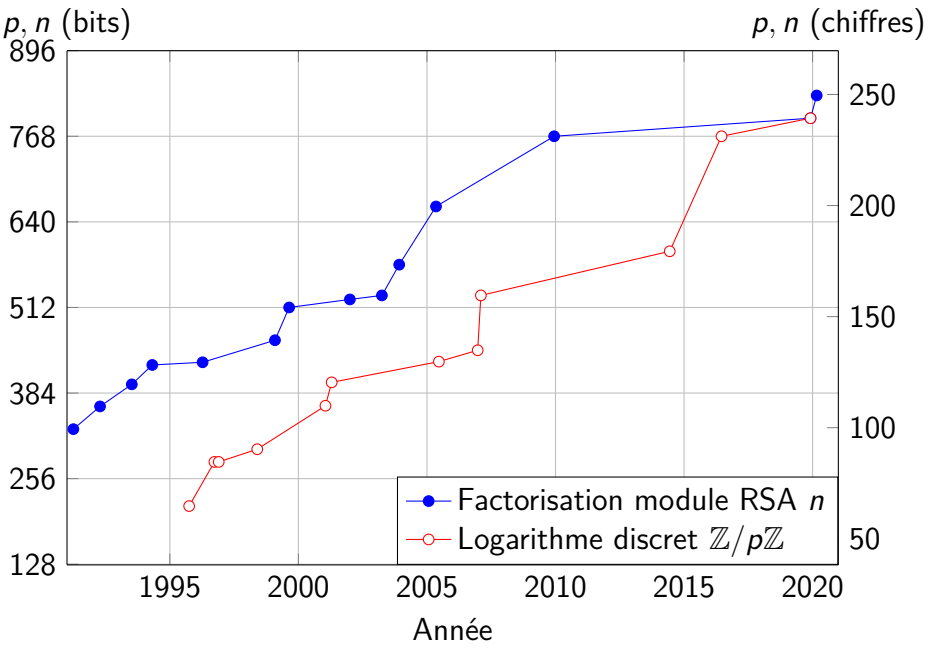

Figure 1 – Records de factorisation RSA et calculs de logarithme discret dans des corps premiers.

et dans le cas du logarithme discret, avec $(p-1)/2$ premier, ou ayant un facteur premier ℓ d'au moins 256 bits (cf par exemple [1, §2.2.1]).
Ces tailles de clé recommandées dépassent (heureusement !) les records de calculs publiés jusqu'ici. Le fossé entre ces tailles fait que dans bien des cas, des acteurs qui déploient des solutions cryptographiques passent outre les recommandations. Dans le but d'optimiser les ressources allouées aux calculs cryptographiques, ils choisissent des tailles de clé dans la «zone grise» des tailles pour lesquelles aucun calcul record n'a été réalisé, mais qui sont néanmoins bien en deçà des tailles recommandées. Un exemple parmi d'autres est celui des cartes bancaires en France, qui utilisent couramment des clés RSA de 1152 bits en 2020.
L'un des objectifs de notre travail est de fournir un point de donnée récent permettant l'appréciation du risque que représente la cryptanalyse pour des tailles de clés qui dépassent les records précédents.

1.4 Le logiciel libre CADO-NFS

L'algorithme utilisé pour les nouveaux records de factorisation et de logarithme discret est connu sous le nom de *crible algébrique* (*Number Field Sieve* ou NFS en anglais). Depuis l'invention de l'algorithme NFS par John Pollard en 1988, plusieurs implantations du crible algébrique ont existé. Certaines (les plus récentes) ont été développées dans des logiciels libres, d'autres non.
La première implantation était celle de Lenstra, Lenstra, Manasse et Pollard qui a permis de factoriser le neuvième nombre de Fermat $2^{2^9}+1$ en 1990. Une autre implantation a été développée au CWI (*Centrum voor Wiskunde en Informatica*, Pays-Bas) dans l'équipe de Herman te Riele, notamment par Peter Montgomery, et a été utilisée pour factoriser RSA-155 (512 bits) en 1999, record de factorisation à l'époque (voir tableau 1). La plupart des records de factorisation suivants ont été obtenus à l'aide de l'implantation développée à partir de 2002 par Jens Franke et Thorsten Kleinjung. Cette implantation a évolué pour permettre les records de factorisation RSA-768 et DL-768 [2, 3].
Une première implantation libre (GGNFS) a été réalisée par Chris Monico et diffusée sous licence GPL en 2003 : GGNFS a permis de factoriser des entiers jusque 148 chiffres décimaux, et des nombres SNFS jusque 195 chiffres. (Un nombre SNFS est un nombre ayant une forme particulière, comme $2^{1031}-1$, pour lequel il existe une variante plus efficace de l'algorithme NFS.) Une autre implantation libre (Msieve) a été initiée par Jason Papadopoulos ; focalisée au départ sur le crible quadratique, Msieve a été étendue au crible algébrique fin 2006. En 2010-2011, avec l'aide de Greg

Childers et Ilya Popovyan, la composante algèbre linéaire de Msieve a été adaptée à l'outil MPI de distribution des calculs. Ryan Propper a utilisé Msieve et GGNFS pour factoriser RSA-190 en 2013, mais la factorisation avec Msieve qui a fait le plus de bruit est sans doute celle de clés de signature des calculatrices Texas Instruments en 2009. Jason Papadopoulos a dû interrompre le développement de Msieve en 2014.

418 520 lignes de code
16 272 *commits*
716 tests unitaires et fonctionnels
41 contributeurs
15 langages (dont 57% de C et 23% de C++)
13 années de développement

Tableau 2 – CADO-NFS en chiffres.

Le développement de CADO-NFS a commencé en 2007, et le logiciel a été continûment amélioré depuis (cf. Tableau 2). L'objectif initial de CADO-NFS était double : d'une part développer un logiciel libre de référence pour le crible algébrique (CADO est l'acronyme de Crible Algébrique : Distribution, Optimisation), d'autre part pouvoir factoriser de manière routinière des entiers jusque 512 bits (soit 155 chiffres décimaux).
La structure globale de CADO-NFS a peu changé depuis le départ : un script principal (au début en Perl et aujourd'hui en Python) qui appelle différents programmes correspondant aux différentes étapes de l'algorithme NFS (programmes écrits en C ou C++). L'architecture client–serveur (due à Alexander Kruppa) permet de tirer au maximum parti des processeurs parallèles et/ou des grappes de calcul. Le développement de CADO-NFS a été au départ principalement guidé par l'objectif initial de factoriser de manière routinière des entiers de 512 bits. Cet objectif a été rempli dès 2012, quand Zachary Harris a démontré avec CADO-NFS que Google utilisait des clés RSA trop petites pour l'authentification dans le logiciel DKIM. Mentionnons aussi le *Factoring as a Service* de Nadia Heninger, qui lui permet dès 2013 de factoriser un entier de 512 bits en 44 heures et 241 dollars sur le *cloud* Amazon EC2 (chiffres ramenés à 4 heures et 75 dollars en 2015).
En 2010, l'algorithme NFS-DL, qui permet de résoudre le problème du logarithme discret, a été ajouté à CADO-NFS. Cet algorithme comporte de nombreuses phases identiques ou très similaires à NFS pour la factorisation d'entier, aussi il était logique d'implanter les deux variantes dans le même logiciel.
Depuis 2012, le développement de CADO-NFS a surtout été motivé par l'établissement de nouveaux records, ou la cryptanalyse de systèmes cryptographiques. Citons par exemple l'attaque FREAK menée en 2015 par Karthikeyan Bhargavan et ses coauteurs, où CADO-NFS a permis de factoriser une clé RSA de 512 bits en quelques heures (via Amazon EC2). Toujours en 2015, CADO-NFS est au cœur de l'attaque LogJam, en permettant de calculer un logarithme discret de 512 bits en 10 minutes seulement, après un pré-calcul d'une semaine environ.
CADO-NFS est lui-même basé sur d'autres logiciels libres, afin de ne pas réinventer la roue. Citons le logiciel GF2X qui est crucial dans l'étape intermédiaire d'algèbre linéaire (*Lingen*), la bibliothèque GMP d'arithmétique en précision arbitraire (utile notamment dans l'étape de racine carrée), le logiciel HWLOC permettant de répartir de manière optimale les différents *threads* d'un programme sur une machine parallèle.
CADO-NFS a aussi servi de laboratoire pour tester et valider de nouveaux algorithmes : l'étape de *rootsieve* lors de la sélection polynomiale (Bai, Brent, Thomé, 2015), une nouvelle famille de polynômes pour NFS (Bai, Bouvier, Kruppa, Zimmermann, 2016), un nouvel algorithme parallèle pour l'étape d'élimination gaussienne structurée (Bouillaguet, Zimmermann, 2019), un nouvel indicateur de qualité pour la sélection polynomiale (David, Zimmermann, 2020), de nouveaux algorithmes pour

la racine carrée (Thomé, 2012), une amélioration majeure pour l'étape d'algèbre linéaire (Dumas, Kaltofen, Thomé, Villard, 2016), ... Il est intéressant de noter que CADO-NFS a été aussi employé comme outil de recherche par d'autres chercheurs que les développeurs du logiciel. Ainsi Yang, Meng, Wang, Wang et Zhang se sont servis de CADO-NFS en 2013 pour expérimenter de nouvelles idées pour la sélection polynomiale. En mathématiques, Cosgrave et Dilcher ont utilisé CADO-NFS en 2014 dans le contexte du théorème de Gauss-Wilson. Certaines applications de CADO-NFS furent inattendues. Ainsi, en 2014, Perigaud et Pernet ont cassé un rançongiciel nommé BitCrypt basé sur une clé RSA de 128 chiffres décimaux, soit environ 426 bits (on soupçonne que les auteurs de BitCrypt avaient à l'esprit une clé de 128 *octets*, soit 1024 bits, mais se sont trompés et ont implanté à la place une clé de 128 *chiffres*, bien plus facile à casser).
Une fois la barre des 512 bits atteinte, il était très tentant d'aller plus loin. RSA-704 (704 bits) a été factorisé en 2012 (Bai, Thomé, Zimmermann) RSA-220 (729 bits) en 2016 (Bai, Gaudry, Kruppa, Thomé, Zimmermann), puis RSA-240 (795 bits) en 2019. Pour chaque record, il a fallu réécrire une partie du code. Ainsi la factorisation de RSA-220 a été interrompue pendant de longs mois car le code de la phase intermédiaire d'algèbre linéaire (*Lingen*) ne « passait » pas. De la même manière, la factorisation de RSA-240 a nécessité une refonte majeure du code de collecte de relations. Mentionnons enfin la factorisation de RSA-232 (768 bits) en février 2020 par trois chercheurs russes, Zamarashkin, Zheltkov, et Matveev : à part pour l'étape d'algèbre linéaire, ils ont utilisé CADO-NFS, sans aucune aide de l'équipe de développement. C'est un véritable succès du caractère libre de CADO-NFS.

À retenir

- RSA en 1977 et Diffie–Hellman en 1976 sont deux systèmes cryptographiques largement déployés, et reposent sur la difficulté de la factorisation d'entier pour le premier, et la difficulté du calcul de logarithme discret pour le second.
- Le crible algébrique est un algorithme qui permet de factoriser des entiers de plus de cent chiffres et de calculer des logarithmes discrets. Une connaissance fine permet de choisir les tailles de clé suffisantes pour assurer une marge de sécurité pour RSA et Diffie–Hellman. L'ANSSI (Agence Nationale de la Sécurité des Systèmes d'Information) en France émet de telles recommandations.
- Le logiciel libre CADO-NFS développé principalement à Nancy depuis 2007 a permis les derniers records de calcul : la factorisation de RSA-240 et un calcul de logarithme discret de 240 chiffres en décembre 2019, et la factorisation de RSA-250 en février 2020.

2 Factorisation de RSA-240

Les étapes principales du crible algébrique peuvent être divisées comme suit. La proximité entre le crible algébrique pour factoriser un entier n, et le crible algébrique pour calculer des logarithmes discrets dans $\mathbb{Z}/p\mathbb{Z}$ est telle que cette liste est pertinente dans les deux situations.

- L'étape de **sélection polynomiale**. Il est question de choisir une paire de polynômes irréductibles à coefficients entiers, notés $f_0 \in \mathbb{Z}[x]$ et $f_1 \in \mathbb{Z}[x]$, qui sont bien adaptés au problème à résoudre. À partir de f_0 et f_1, un contexte mathématique propice peut être installé. Toutes les paires de polynômes ne se valent pas, certaines permettent un meilleur rendement que d'autres dans la suite des calculs.
- L'étape de **collecte des relations**. Il s'agit ici d'explorer un espace gigantesque de paramètres, indexés par deux entiers a et b, pour lesquels on espère que deux événements plutôt rares interviennent simultanément. On souhaite que les deux entiers $\mathrm{Res}(a-bx, f_0)$ et $\mathrm{Res}(a-bx, f_1)$ soient simultanément *friables*, c'est-à-dire que leur plus grand facteur premier soit plus petit

qu'une borne fixée à l'avance. Lorsque c'est le cas, on obtient une *relation*. (Ici comme ailleurs dans l'article, Res( ) désigne le résultant de deux polynômes ; c'est un entier.)

— L'étape d'**algèbre linéaire**. Un système linéaire, construit à partir des relations collectées, doit être résolu. Pour la factorisation d'entier, ce système est défini sur $\mathbb{Z}/2\mathbb{Z}$. Pour le logarithme discret dans $\mathbb{Z}/p\mathbb{Z}$, on doit résoudre un système défini sur $\mathbb{Z}/\ell\mathbb{Z}$, où ℓ est le plus grand facteur premier de $p-1$.

— Les **calculs finaux** : retrouver les facteurs premiers p et q de n, ou calculer effectivement des logarithmes discrets dans $\mathbb{Z}/p\mathbb{Z}$.

Sur ces quatre étapes, la collecte de relations et l'algèbre linéaire sont de loin les plus coûteuses. Nous donnons ici quelques éléments relatifs à la factorisation de RSA-240. Une présentation plus complète peut être obtenue dans l'article [4].

2.1 Sélection polynomiale

Notre cible est la factorisation de l'entier RSA-240, défini ainsi :

$$\begin{aligned}\text{RSA-240} = {} & 124620366781718784065835044608106590434820374651678805754818\\ & 788883289666801188210855036039570272508747509864768438458621\\ & 054865537970253930571891217684318286362846948405301614416430\\ & 468066875699415246993185704183030512549594371372159029236099.\end{aligned}$$

L'algorithme utilisé pour rechercher la paire de polynômes (f_0, f_1) est dû à Kleinjung (2006, 2008), avec les améliorations de Bai, Bouvier, Kruppa et Zimmermann (2016). Nous avons recherché des polynômes de degré $\deg f_0 = 1$ et $\deg f_1 = 6$, ce qui est apparu meilleur que les alternatives. Pour optimiser le rendement des étapes ultérieures de l'algorithme, nous avons également contraint le terme dominant de f_1 à être divisible par l'entier friable $110880 = 2^5 \cdot 3^2 \cdot 5 \cdot 7 \cdot 11$. Ces contraintes sont minces, et l'espace des paires de polynômes potentielles est immense. Pour guider notre choix, nous avons tiré parti de l'algorithme, qui est un pourvoyeur de paires de bonne qualité, et nous avons classé ces dernières au moyen d'indicateurs communément notés α, MurphyE, ou encore l'indicateur de qualité E' proposé par David et Zimmermann (2020). Après 76 années-cœurs de calcul (distribué sur de nombreuses machines, de sorte que le temps réel n'a pas dépassé deux semaines), et des tests de performance poussés pour départager les paires les plus prometteuses, nous avons arrêté notre choix sur la paire suivante, qui vérifie $|\text{Res}(f_0, f_1)| = 120n$:

$$\begin{aligned} f_0 &= 17780390513045005995253\,x - 10548775373296986022379504129586051738 0\\ f_1 &= 10853204947200\,x^6 - 22117558884229911759056454260997701656719186 0\\ &- 4763683724115259920\,x^5 + 15957125533693354304961257950831466885 23\,x\\ &- 63817444612798679419616 70\,x^4 + 17920057353366572131021064073806117 0\,x^2\\ &+ 97444893485386480769067506703 7\,x^3 \end{aligned}$$

2.2 Collecte de relations

La collecte de relations est l'étape la plus coûteuse, particulièrement dans le cas de la factorisation d'entier. Les détails de cette phase sont ardus, et détaillés dans [4]. De nombreux paramètres (si l'on descend au grain le plus fin, on en compte aisément plusieurs dizaines) contrôlent cette phase. Nous avons choisi ces paramètres avec les considérations suivantes en ligne de mire.

— Puisque l'étape d'algèbre linéaire est comparativement moins coûteuse que la collecte de relations dans le cas de la factorisation d'entier, il est raisonnable de faire pencher (légèrement) la balance vers le fait de produire des relations vite, même si leur prise en compte amène un système linéaire un peu encombrant. Ainsi, nous avons fixé comme objectif que les entiers $\mathrm{Res}(a - bx, f_0)$ et $\mathrm{Res}(a - bx, f_1)$ aient tous leurs facteurs premiers inférieurs à 2^{36} et 2^{37}, respectivement.

— La distribution du calcul étant une nécessité, au vu du vaste nombre de paires (a, b) à tester, il est important que le calcul soit efficace sur la plupart des machines auxquelles nous pouvons avoir accès. Le souci d'efficacité, ici, est en vérité multiforme : il est préférable d'éviter qu'une trop grande quantité de mémoire soit nécessaire, il est bon de pouvoir tirer parti au mieux des machines possédant de nombreux cœurs de calcul, et enfin il est également important de veiller à ce que la performance des accès mémoire, notamment au travers des différents niveaux de mémoire cache, soit au rendez-vous. Outre l'important travail sur l'implémentation elle-même, ces considérations ont un impact dans le choix des paramètres.

— L'orchestration du calcul sur un parc de plusieurs centaines de machines travaillant en parallèle soulève également quelques défis, et le choix de bons paramètres de calcul peut aider à alléger certaines des difficultés dans ce domaine (par exemple pour la fréquence des connexions client–serveur, ou le nombre de fichiers de données à traiter).

Pour RSA-240, la collecte de relations a coûté environ 794 années-cœurs, en prenant comme machine de référence un cœur d'un processeur Intel Xeon 6130 (2.1 GHz).

2.3 Algèbre linéaire

Après l'étape de collecte de relations, et à l'issue d'un traitement intermédiaire, l'étape d'algèbre linéaire pour la factorisation de RSA-240 nécessite de trouver un élément du noyau (à gauche) d'une matrice définie sur $\mathbb{Z}/2\mathbb{Z}$, avec 282 millions de lignes et de colonnes. Cette matrice est particulièrement *creuse*, puisque seulement 200 éléments non nuls sont présents dans chaque ligne. Au regard de la dimension de certains systèmes linéaires manipulés dans des contextes numériques, cette dimension peut sembler modeste. Toutefois, une telle comparaison est invalide, à cause d'une différence fondamentale. Dans le cas qui nous intéresse, les calculs sont exacts, et il n'y a pas de notion de convergence qui vaille. La quête d'une solution approchée n'a pas de sens. Si l'algorithme que nous avons utilisé, appelé algorithme de Wiedemann par blocs, peut être rangé dans la catégorie des algorithmes dits itératifs, celui-ci ne produit la solution recherchée qu'à l'issue du nombre maximal d'itérations (soit 282 millions d'itérations, au lieu de quelques-unes seulement en calcul numérique). Cette étape d'algèbre linéaire est donc un vrai défi, pour lequel l'implantation de CADO-NFS met à contribution les différents atouts technologiques des machines dédiées au calcul à haute performance : placement des processus sur des cœurs spécifiques, et réseaux d'interconnexion à débit très élevé (100 Gbps).

Trois étapes principales composent l'algorithme de Wiedemann par blocs : *Krylov*, *Lingen* et *Mksol*. L'étape centrale, appelée *Lingen*, est différente des deux autres. En effet, l'aspect «par blocs» de l'algorithme recouvre une flexibilité que nous avons largement employée. Il est possible de diviser l'étape *Krylov* en sous-tâches indépendantes, avec un *scaling* de 100% : plutôt que de travailler pendant un temps T sur une ressource, nous travaillons pendant un temps T/k sur k ressources en parallèle, pour effectuer la même tâche. (Par ressource, on entend par exemple une grappe de calcul, ou éventuellement un sous-ensemble d'une telle grappe.) C'est un gain très appréciable, car il permet d'éviter des communications et synchronisations coûteuses entre les ressources qui participent au calcul. Si cette division du travail est parfaite pour l'étape *Krylov*, il n'en est pas de même pour l'étape *Lingen*, dont le coût croît avec k. Un des défis importants de ce calcul d'algèbre linéaire a donc été de faire en sorte que les calculs de l'étape *Lingen* puissent être menés.

Pour la factorisation de RSA-240, nous avons mené l'étape d'algèbre linéaire sur la grille de calcul Grid'5000/SILECS (https://www.grid5000.fr/). Les étapes *Krylov* et *Mksol* ont employé le mode *best-effort* : nos programmes ont utilisé exclusivement les ressources de calcul laissées libres par les autres utilisateurs de la plate-forme. Les trois étapes *Krylov*, *Lingen*, et *Mksol* ont respectivement coûté 69, 0.8, et 13 années-cœurs. Un tel calcul rencontre souvent quelques erreurs, et ce fut le cas ici : une panne temporaire d'un nœud de stockage de données a invalidé une mince partie des données calculées, qui ont échappé à nos tests de cohérence. Le diagnostic de cette incohérence n'a pas été simple, mais sa réparation n'a pas nécessité de calculs significatifs.

2.4 Calculs finaux

Pour retrouver les facteurs secrets p et q de l'entier composé RSA-240, les calculs finaux de l'algorithme nécessitent d'employer des techniques d'arithmétique multi-précision sur des entiers de très grande taille (plusieurs gigaoctets). Bien que de tels calculs soient comparativement peu chers au regard du reste de l'entreprise, l'imminence de la solution nous a encouragés à améliorer l'implémentation pour obtenir le résultat au plus vite (pour y passer quelques heures plutôt que quelques jours). Nous avons ainsi obtenu la factorisation suivante : l'entier RSA-240 se décompose en $n = p \cdot q$, où

$$\begin{aligned} p &= 509435952285839914555051023580843714132648382024111473186660 \\ &\quad 296521821206469746700620316443478873837606252372049619334517, \\ q &= 244624208838318150567813139024002896653802092578931401452041 \\ &\quad 221336558477095178155258218897735030590669041302045908071447. \end{aligned}$$

À retenir

— La factorisation d'entier avec l'algorithme NFS comprend les étapes suivantes : sélection polynomiale, collecte de relations, algèbre linéaire sur une matrice très creuse modulo 2, et calcul de racine carrée d'un entier gigantesque. Les étapes les plus coûteuses sont la collecte de relations qui se parallélise facilement sur des milliers de cœurs, et l'algèbre linéaire qui est plus délicate à paralléliser et a besoin d'une grappe de calcul avec des débits de données très rapides entre les différents nœuds.

— La factorisation de RSA-240 en 2019 a pris 953 années-cœurs réparties sur des milliers d'ordinateurs.

3 Calcul d'un logarithme discret sur 240 chiffres

Au premier abord, la variante NFS-DL de l'algorithme NFS est assez similaire à ce qui est utilisé pour la factorisation d'entier. Et en effet, les grandes étapes sont les mêmes (cf. §2). D'ailleurs certaines briques logicielles sont entièrement partagées entre les deux variantes. C'est le cas du programme qui permet de récolter les milliards de relations qui, une fois filtrées forment la matrice sur laquelle on applique l'étape d'algèbre linéaire. Malgré cela, il y a des différences importantes à chaque étape, soit que l'algorithme est différent, soit que l'on règle les paramètres de manière différente. Sans rentrer dans les détails mathématiques ou algorithmiques, nous allons redonner pour chaque étape quelques points clefs, ainsi que des données numériques liées au présent record, qui concerne le nombre premier $p = \text{RSA-240} + 49204$, qui est le plus petit nombre premier sûr plus grand que RSA-240 (i.e., $\ell = (p-1)/2$ est aussi un nombre premier).

3.1 Sélection polynomiale

L'algorithme utilisé pour cette étape est dû à Antoine Joux et Reynald Lercier (2003). Il fournit des polynômes de bien meilleure qualité que l'algorithme de Kleinjung utilisé pour la factorisation,

en tirant parti du fait que le nombre p modulo lequel on travaille est premier. L'objectif de cette étape est de trouver deux polynômes f_0 et f_1 à coefficients entiers, avec $\deg f_0 = 3$, $\deg f_1 = 4$, tels que le résultant de f_0 et f_1 égale p ou un petit multiple de p, tout en essayant d'avoir des coefficients de f_0 les plus petits possibles (sachant que ceux de f_1 sont petits par construction). D'autres critères sont à prendre en compte pour évaluer la qualité d'une paire de polynômes, et l'on utilise le même indicateur *MurphyE* que pour la factorisation afin de présélectionner rapidement les paires prometteuses. Comme pour la factorisation, ce critère est imparfait, et la paire choisie est *in fine* décidée à partir de quelques expériences de collecte de relations.
Pour le calcul record, le temps passé pour cette étape a été d'environ 152 années-cœurs. La parallélisation de l'algorithme de Joux-Lercier se fait de manière très simple, et comme cela ne demande pas de ressources particulières (notamment en mémoire), il est facile d'utiliser des nœuds de calcul de provenance diverse. Les polynômes retenus pour la suite sont

$$\begin{aligned} f_0 &= 2865121727006754119869668463943599248745765364087863680 56x^3 \\ &+ 24908820300715766136475115982439735516581888603817255539890x^2 \\ &- 18763697560013016564403953928327121035580409459944854652737x \\ &- 23661040882700025625019083822082412299787899459578543220 2599 \\ f_1 &= 39x^4 + 126x^3 + x^2 + 62x + 120 \end{aligned}$$

et l'on a $|\mathrm{Res}(f_0, f_1)| = 540p$.

3.2 Collecte de relations

Cette étape est très similaire à ce qui se passe en factorisation. Toutefois, si l'on règle les paramètres de manière identique, on va collecter suffisamment de relations en un temps relativement court, mais la matrice produite sera extrêmement difficile à traiter à l'étape suivante d'algèbre linéaire. Il s'agit donc de trouver des paramètres différents afin de rééquilibrer les choses : passer un peu plus de temps sur la collecte de relations, afin d'obtenir une matrice plus petite et donc réduire le temps de calcul global.
L'approche naïve consistant à calculer bien plus de relations que nécessaire, puis à en extraire celles qui sont de meilleure qualité n'est pas optimale, loin de là. Mieux vaut cibler directement les bonnes relations, ce qui ouvre la voie à de nouvelles optimisations. C'est dans cet esprit que Kleinjung avait utilisé pour la première fois une méthode appelée *batch smoothness detection* lors du précédent record [3]. Pour le présent calcul, nous avons repris cette idée, mais l'avons conjuguée à une nouvelle technique appelée *composite special-*$\mathfrak{q}$. Tout ceci se combine très bien, et nous avons ainsi pu accélérer grandement cette phase de collecte de relations, tout en maintenant une matrice de taille contrôlée. Expliquer le fonctionnement de ces méthodes nous emmènerait trop loin, et nous renvoyons le lecteur intéressé à l'article scientifique [4] pour les détails.
Les calculs de cette étape se distribuent facilement sur de nombreuses machines indépendantes. Toutefois, notre implémentation de l'algorithme de collecte de relations est paramétrable en fonction de la quantité de mémoire disponible par cœur, et bien sûr, l'efficacité croît avec la mémoire disponible (jusqu'à un certain point).
Des machines très variées ont été utilisées, mais si l'on re-calibre pour une machine de calcul typique, l'ensemble de cette phase a pris l'équivalent de 2400 années-cœurs et a produit environ 3.8 milliards de relations, dont presque 40% de doublons, ce qui est inévitable avec l'algorithme utilisé.

3.3 Algèbre linéaire

Les relations, une fois pré-traitées, produisent une matrice ayant 36 millions de lignes et colonnes, pour une densité moyenne de 253 coefficients non nuls par ligne. Il s'agit alors de calculer un vecteur non nul du noyau de cette matrice, mais à la grande différence de la factorisation, cette fois-ci les

calculs doivent s'effectuer modulo $\ell = (p-1)/2$ et non pas modulo 2. Chaque opération élémentaire est donc très coûteuse puisqu'elle implique des entiers de presque 800 bits. Et malheureusement, il n'est pas possible dans ce contexte de manipuler des approximations flottantes de ces entiers. On doit donc traiter une matrice gigantesque, avec en plus de l'arithmétique multi-précision. C'est d'ailleurs ce point là qui fait que cette étape d'algèbre linéaire est bien plus coûteuse qu'en factorisation et qu'on a dû repenser la collecte de relations pour limiter la taille de la matrice.
Mise à part cette complication de taille, l'algorithme utilisé est de nouveau Wiedemann par blocs, avec ses trois sous-étapes *Krylov*, *Lingen* et *Mksol*. La phase la plus coûteuse est *Krylov*, et l'on peut la paralléliser, mais avec certaines contraintes. Sans rentrer dans les détails, cette phase d'algèbre linéaire nécessite des ressources dédiées, car on a besoin d'avoir des paquets de nœuds de calcul qui soient reliés par du réseau à très haut débit (la latence importe assez peu). Au total, cette étape a pris l'équivalent de 625 années-cœurs (Tableau 3).

3.4 Calculs finaux

Le vecteur calculé à l'étape précédente contient les logarithmes discrets d'une multitude d'éléments du corps fini. Il s'agit d'éléments « petits » en un sens algébrique assez compliqué. Ceci forme une base de données qui peut ensuite être utilisée pour calculer le logarithme discret de n'importe quel autre élément, grâce à une méthode appelée *special-*$\mathfrak{q}$ *descent*. Peu d'effort a été fait pour optimiser cette étape dont le coût est négligeable par rapport aux autres étapes (quelques milliers d'heures-cœurs, assez bien parallélisable). Notons toutefois qu'un attaquant réel pourrait vouloir calculer de très nombreux logarithmes discrets d'éléments appartenant tous à un même corps fini, chacun représentant une clef cryptographique. Dans ce cas, il deviendrait intéressant de réduire au maximum le coût de cette étape.

L'élément dont nous avons calculé le logarithme discret est l'entier h dont l'écriture hexadécimale correspond à l'encodage ASCII de la phrase "The magic words are still Squeamish Ossifrage", qui est une référence à une phrase célèbre dans le monde de la factorisation d'entier (i.e., $h = \texttt{0x54...65}$). Dans $\mathbb{Z}/p\mathbb{Z}$, le logarithme discret de h en base $g = 5$ est

$$\begin{aligned} x = \log_5 h \;=\; & 92603135928144195363094955331732855502961099191437611616729 4 \\ & 20475898744562365366788100548099072093487548258752802923326 4 \\ & 47367244150096121629264809207598195062213366889859186681126 9 \\ & 28982506005127728321426751244111412371767375547225045851716\,. \end{aligned}$$

	sélection polynomiale	collecte de relations	algèbre linéaire	total
RSA-240	76	794	83	953
DLP-240	152	2400	625	3177

Tableau 3 – Comparaison des records RSA-240 et DLP-240 en nombre d'années-cœurs utilisées.

À retenir
— Un calcul de logarithme discret avec CADO-NFS réutilise de nombreuses parties du logiciel dédiées à la factorisation. Une des différences majeures est l'algèbre linéaire qui est faite modulo un grand nombre premier ℓ au lieu de modulo 2. Pour rééquilibrer les temps de calcul entre collecte de relations et algèbre linéaire, des choix d'optimisation différents sont faits. — Le record de calcul d'un logarithme discret de 240 chiffres a pris environ trois fois plus de temps que la factorisation de RSA-240. — À la différence de la factorisation, les étapes coûteuses jusqu'à l'algèbre linéaire produisent une gigantesque base de données de logarithmes qui peut être réutilisée pour casser en moins de quelques minutes tout échange de clés utilisant l'algorithme de Diffie–Hellman avec le même nombre premier p.

Conclusion

Les algorithmes RSA et Diffie–Hellman sont au cœur des protocoles modernes de communication. D'après le *ICSI Certificate Notary* (https://notary.icsi.berkeley.edu/), 90% des certificats mis en œuvre pour des échanges chiffrés sur Internet en juillet 2020 ont utilisé des clés RSA, et l'échange de clé Diffie–Hellman est requis pour SSH (*secure shell*), pour TLS 1.3 (la dernière version du protocole SSL/TLS permettant de chiffrer les échanges par mail ou sur Internet), et pour IPsec (pour les réseaux virtuels privés).
Établir de nouveaux records de factorisation et de calcul de logarithme discret est le meilleur moyen d'encourager les utilisateurs de ces protocoles à mettre à jour les tailles de clé en faveur de valeurs plus sûres. Alors que les recommandations officielles sont depuis 2010 d'au moins 2048 bits pour les clés RSA et Diffie–Hellman, de nombreux sites utilisent encore des clés plus petites : le *SSL Pulse* de Qualys indique qu'en juillet 2020, 9% des sites web couramment visités prennent en charge des clés Diffie–Hellman de 1024 bits, et environ 1% des clés de 512 ou 768 bits.
Les records que nous avons établis constituent un pas décisif vers la factorisation d'une clé de 1024 bits dans un futur proche. Grâce aux améliorations logicielles que nous avons effectuées, et à notre procédé algorithmique d'estimation de temps de calcul, nous pouvons affirmer qu'aucune barrière technologique ne s'oppose plus à un record de 1024 bits. La principale difficulté réside dans l'accès à suffisamment de ressources de calcul. Un rapide calcul montre que la factorisation de RSA-896 (896 bits) nécessitera entre 6 et 7 fois plus de temps de calcul que celle de RSA-250 (829 bits). Ensuite, la factorisation de RSA-1024 (1024 bits) demandera environ 30 fois plus de temps de calcul que RSA-896, soit de l'ordre de 500 000 années-cœurs.
Cette quantité de ressources est actuellement inaccessible pour nous. Cependant, la combinaison de plusieurs facteurs (des améliorations algorithmiques, de plus grandes capacités de calcul via la loi de Moore, et des partenariats avec des organismes ayant accès à de gros moyens de calcul) devrait permettre la factorisation de RSA-1024 d'ici quelques années.
En 1994, Peter Shor a inventé un algorithme permettant de factoriser un entier plus rapidement sur un ordinateur quantique. Pour factoriser un entier de n bits, il faut disposer d'un ordinateur quantique parfait avec environ $2n$ qubits (bits quantiques). Cela semble extrêmement difficile à réaliser, si bien que le record de factorisation quantique actuel est ridiculement petit : il s'agit de $4\,088\,459 = 2017 \times 2027$ en 2018, et encore il est loin d'être clair que le procédé utilisé permette de factoriser n'importe quel nombre de 7 chiffres. Bref, il est fort probable que RSA-1024 sera factorisé bien avant que l'ordinateur quantique entre dans la course.

Glossaire

modulo n ; $\bmod n$

Un entier x modulo un autre entier positif n est le reste de la division euclidienne de x par n : $x = qn + r$ avec q entier et le reste $0 \leqslant r < n$, et on note $x = r \bmod n$, ou encore $x \equiv r \pmod n$.

RSA (*Rivest, Shamir, Adleman*)
Initiales des auteurs, par extension, cryptosystème à clé publique publié en 1977 par ces auteurs.
DH (*Diffie, Hellman*)
Initiales des auteurs, par extension, cryptosystème asymétrique publié en 1976 par ces auteurs.
CADO-NFS (*Crible Algébrique : Distribution, Optimisation*)
Logiciel *open-source* de factorisation et de calcul de logarithme discret utilisant l'algorithme NFS.
NFS (*Number Field Sieve ; crible algébrique*)
Algorithme de factorisation et de calcul de logarithme discret.

Nouveaux records de factorisation et de calcul de logarithme discret

par **Fabrice Boudot**

Professeur de l'Éducation Nationale

Université de Limoges, XLIM, UMR 7252, F-87000 Limoges, France

par **Pierrick Gaudry**

Directeur de Recherche CNRS

Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France

par **Aurore Guillevic**

Chargée de Recherche Inria

Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France

par **Nadia Heninger**

Associate Professor

University of California, San Diego, USA

par **Emmanuel Thomé**

Directeur de Recherche Inria

Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France

par **Paul Zimmermann**

Directeur de Recherche Inria

Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France

Sources bibliographiques

[1] Agence nationale de la sécurité des systèmes d'information. *Référentiel général de sécurité, v2.03, Annexe B1*. Téléchargeable via https://www.ssi.gouv.fr/uploads/2014/11/RGS_v-2-0_B1.pdf. 2014.

[2] Thorsten Kleinjung et al. "Factorization of a 768-Bit RSA Modulus". In : *CRYPTO 2010*. Sous la dir. de Tal Rabin. LNCS 6223. Springer, Heidelberg, 2010, p. 333-350. doi : 10.1007/978-3-642-14623-7_18.

[3] Thorsten Kleinjung et al. "Computation of a 768-Bit Prime Field Discrete Logarithm". In : *EUROCRYPT 2017, Part I*. Sous la dir. de Jean-Sébastien Coron et Jesper Buus Nielsen. LNCS 10210. Springer, Heidelberg, 2017, p. 185-201. doi : 10.1007/978-3-319-56620-7_7.

[4] Fabrice Boudot et al. “Comparing the difficulty of factorization and discrete logarithm : a 240-digit experiment”. In : *Proceedings of Advances in Cryptology (CRYPTO)*. Sous la dir. de D. Micciancio et T. Ristenpart. LNCS 12171. 2020, p. 62-91.

[5] Pierrick Gaudry, Emmanuel Thomé et Paul Zimmermann. “RSA : la fin des clés de 768 bits”. In : *Techniques de l'Ingenieur* IN131 (2011). url : https://hal.inria.fr/hal-00641592.

Sites Internet

Computations of discrete logarithms, Laurent Grémy, https://dldb.loria.fr
Wikipedia Integer factorization records https://en.wikipedia.org/wiki/Integer_factorization_records
Wikipedia RSA numbers https://en.wikipedia.org/wiki/RSA_numbers
(pages consultées le 4 août 2020)
Glossaire de l'ANSSI https://www.ssi.gouv.fr/particulier/glossaire/
(page consultée le 16 septembre 2020)