(19) RÉPUBLIQUE FRANÇAISE

INSTITUT NATIONAL DE LA PROPRIÉTÉ INDUSTRIELLE

PARIS

(11) **N° de publication :** ***2 997 780***

(à n'utiliser que pour les commandes de reproduction)

(21) **N° d'enregistrement national :** **12 60553**

(51) Int Cl[8] : ***G 06 F 21/00*** *(2013.01), G 06 F 7/72*

(12) DEMANDE DE BREVET D'INVENTION A1

(22) **Date de dépôt :** 07.11.12.

(30) **Priorité :**

(43) **Date de mise à la disposition du public de la demande :** 09.05.14 Bulletin 14/19.

(56) **Liste des documents cités dans le rapport de recherche préliminaire :** *Se reporter à la fin du présent fascicule*

(60) **Références à d'autres documents nationaux apparentés :**

(71) **Demandeur(s) :** *INSIDE SECURE Société anonyme* — FR.

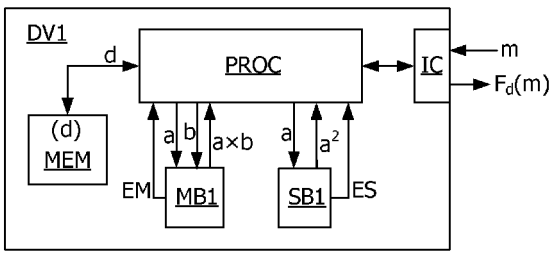
(72) **Inventeur(s) :** VERNEUIL VINCENT et CLAVIER CHRISTOPHE.

(73) **Titulaire(s) :** INSIDE SECURE Société anonyme.

(74) **Mandataire(s) :** OMNIPAT Société anonyme.

(54) PROCEDE DE CRYPTOGRAPHIE COMPRENANT UNE OPERATION D'EXPONENTIATION MODULAIRE.

(57) L'invention concerne un procédé de calcul itératif d'exponentiation d'une donnée de grande taille, le procédé étant mis en oeuvre dans un dispositif électronique (DV1) et comprenant des calculs d'élévation au carré et de multiplication de variables de grande taille effectués en parallèle, par des blocs d'élévation au carré (SB1) et de multiplication (SM1), le procédé comprenant, des étapes consistant à: tant qu'une mémoire tampon de stockage temporaire n'est pas pleine de carrés non utilisés, déclencher un calcul par le bloc d'élévation au carré pour un bit de l'exposant, lorsque le bloc d'élévation au carré est inactif, stocker chaque carré fourni par le bloc d'élévation au carré dans la mémoire tampon, si le bit de l'exposant correspondant est à 1, et tant que la mémoire tampon contient un carré non utilisé, déclencher un calcul par le bloc de multiplication portant sur le carré non utilisé, lorsque le bloc de multiplication est inactif.



FR 2 997 780 - A1

PROCEDE DE CRYPTOGRAPHIE COMPRENANT UNE OPERATION D'EXPONENTIATION MODULAIRE

La présente invention concerne un procédé de calcul itératif du résultat de l'exponentiation d'une donnée m par un exposant *d*, mis en œuvre dans un dispositif électronique.

Divers procédés de cryptographie connus reposent sur l'opération d'exponentiation modulaire, qui a pour expression mathématique :

m^d modulo(n),

m étant une donnée d'entrée, d un exposant et n un module. La fonction d'exponentiation modulaire consiste à calculer le reste de la division par n de m à la puissance d.

Une telle fonction est utilisée par divers algorithmes de cryptographie tels que l'algorithme RSA (Rivest, Shamir et Adleman), l'algorithme DSA ("Digital Signature Algorithm"), El Gamal, etc. La donnée m est généralement un message à chiffrer ou déchiffrer ou bien un message à signer ou une signature à vérifier, et l'exposant d est une clé privée ou publique.

Il est connu d'exécuter un calcul d'exponentiation modulaire au moyen de l'algorithme “Square & Multiply" ("élever au carré et multiplier") A1 ou A1' figurant en Annexe I.

L'algorithme A1 est dit "de gauche à droite" car les premières étapes de la boucle de calcul commencent par les bits de poids fort de l'exposant, pour aller vers les bits de poids faible. L'algorithme A1' est dit "de droite à gauche" car les premières étapes de la boucle de calcul commencent par les bits de poids faible de l'exposant, pour aller vers les bits de poids fort.

Ces algorithmes comprennent pour chaque itération, c'est-à-dire chaque bit de l'exposant, une multiplication de deux variables identiques de grande taille et si le bit de l'exposant traité par l'itération est égal à 1, une multiplication de deux variables différentes de grande taille. Il est généralement fait appel à des fonctions différentes pour exécuter chacune de ces opérations, la multiplication de deux variables identiques de grande taille étant exécutée au moyen d'une fonction d'élévation au carré ou fonction "CARRE", tandis que la multiplication de deux variables différentes de grande taille est exécutée au moyen d'une fonction de multiplication ou

fonction "MULT". Cette distinction est due au fait qu'il est possible de calculer plus rapidement x×y lorsque x=y que dans le cas contraire, au moyen de la fonction CARRE. Le ratio entre le temps d'exécution de la fonction CARRE et le temps d'exécution de la fonction MULT est généralement de l'ordre de 0,8 mais peut varier entre 0,5 et 1 suivant la taille des nombres considérés, la façon dont la multiplication est exécutée, etc.

Dans un dispositif électronique de type carte à puce, le calcul cryptographique est généralement exécuté par un processeur spécifique, tel un coprocesseur arithmétique ou un crypto-processeur. Le calcul de "m^d modulo n", et plus particulièrement l'exécution des multiplications de nombres de grande taille, occupe la majeure partie du temps de calcul du processeur relativement au temps total de calcul d'une signature, de vérification de signature, ou d'une opération de chiffrement ou de déchiffrement. Le fait d'utiliser alternativement la fonction CARRE ou la fonction MULT en fonction du type de calcul à effectuer permet donc d'optimiser le temps de calcul global de chiffrement, de déchiffrement, de signature ou de vérification de signature.

Toutefois, l'utilisation de deux fonctions différentes CARRE et MULT conduit à une fuite d'information détectable par analyse SPA ("Simple Power Analysis"), c'est-à-dire par analyse de la consommation électrique de la carte. La fonction CARRE ayant un temps d'exécution plus court que la fonction MULT, il est possible de différencier ces deux opérations en observant la courbe de consommation électrique du composant. On entend par "consommation électrique" toute grandeur physique observable trahissant le fonctionnement du composant électronique exécutant l'opération, notamment le courant électrique consommé ou le rayonnement électromagnétique du composant.

La figure 1 représente une courbe de consommation électrique d'un composant exécutant l'algorithme A1. On distingue clairement le profil de consommation de la fonction CARRE et celui de la fonction MULT. Une opération CARRE suivie d'une opération MULT (étape 2.1 suivie d'une étape 2.2) révèle que le bit de l'exposant d est égal à 1 puisque le branchement conditionnel vers l'étape 2.2 nécessite que la condition $d_s=1$ soit réalisée. Inversement, une opération CARRE suivie d'une autre opération CARRE (étape 2.1 suivie d'une autre étape 2.1) révèle que le bit de l'exposant est

égal à 0. Les bits de l'exposant d peuvent ainsi être découverts les uns après les autres par une simple observation de la courbe de consommation électrique.

Pour pallier cet inconvénient, les étapes 2.1 et 2.2 des algorithmes A1 et A1' pourraient être effectuées au moyen de la fonction MULT uniquement, sans utiliser la fonction CARRE. Toutefois, une analyse plus fine de la consommation électrique permettrait de distinguer l'étape 2.1 de l'étape 2.2 car l'algorithme A1 ou A1' n'est pas régulier. En effet, dans ce cas, le temps s'écoulant entre deux multiplications successives n'est pas le même lorsque les deux multiplications correspondent à l'exécution successive de deux étapes 2.1 (bit de l'exposant égal à 0) ou correspondent à l'exécution d'une étape 2.1 suivie d'une étape 2.2 (bit de l'exposant égal à 1). Un attaquant pourrait ainsi "zoomer" sur la partie de la courbe de consommation s'étendant entre les multiplications et observerait une dissymétrie temporelle révélatrice du branchement conditionnel et donc de la valeur du bit de l'exposant.

L'algorithme A2 figurant en Annexe I est une variante de l'algorithme A1 qui permet de pallier cet inconvénient. L'algorithme A2 est appelé "Square & Multiply Always" ("toujours élever au carré et multiplier") car une multiplication factice (étape 2.3) utilisant un paramètre factice b est insérée après une élévation au carré lorsque le bit de l'exposant d est égal à 0, grâce à un double branchement conditionnel "si" et "sinon".

La figure 2 montre la consommation électrique d'un composant exécutant l'algorithme A2. On constate une régularité des pics de consommation correspondant à la succession des étapes 2.1 et 2.2, qui protège l'algorithme contre une attaque SPA. On suppose donc que le double branchement conditionnel "si" et "sinon" ne produit pas de fuite détectable par analyse SPA, car on ne peut pas distinguer si la condition est vraie ou fausse, puisqu'une multiplication est toujours exécutée. L'algorithme A2 est dit "régulier" puisque l'attaquant voit une succession d'étapes identiques. Il ne répond toutefois pas au principe d'atomicité.

Le principe d'atomicité a été introduit par B. Chevallier-Mames, M. Ciet et M. Joye, dans un article intitulé "*Low-Cost Solutions for Preventing Simple Side-Channel Analysis: Side-Channel Atomicity*", paru dans IEEE Transactions on Computers, Volume 53, Issue 6 (June 2004), Pages: 760 -

768, 2004. Il est également décrit dans la demande internationale WO 03 083 645 ou le brevet US 7 742 595.

L'application du principe d'atomicité conduit à transformer une boucle non régulière, par exemple la boucle constituée par les étapes 2.1 et 2.2 de l'algorithme A1 ou A1', en une suite régulière de multiplications, sans utiliser de multiplication factice, pour un gain de temps dans l'exécution de l'algorithme.

A titre d'exemple, l'algorithme d'exponentiation A3 figurant en Annexe I, appelé “Multiply Always”, est la version atomique de l’algorithme A1. L’algorithme est parfaitement régulier en ce qu’il ne comporte que des multiplications et en ce que chaque itération de la boucle principale ne comprend qu’une seule multiplication.

La figure 3 représente la courbe de consommation électrique de l’algorithme A3 et montre la régularité des pics de consommation électrique.

Dans cet algorithme, certaines multiplications sont des multiplications de variables différentes et d’autres sont des multiplications de variables identiques. Or, dans l’article “*Distinguishing Multiplications from Squaring Operations*”, Selected Areas in Cryptography, volume 5381 of Lecture Notes in Computer Science, pages 346–360, Springer, 2008, les auteurs F. Amiel, B. Feix, M. Tunstall, C. Whelan, et W. Marnane divulguent un procédé d’analyse à canal caché qui exploite une différence intrinsèque entre la multiplication de deux variables différentes et la multiplication de deux variables identiques (équivalente à une élévation au carré), le résultat de la seconde ayant en moyenne un poids de Hamming inférieur au résultat de la première. L’algorithme A3 est donc exposé à ce type d’attaque, car il contient des multiplications de termes différents et des multiplications de termes égaux.

L’algorithme A2 “Square & Multiply Always" n’est pas sensible à ce type d’attaque car les multiplications exécutées à l’étape 2.2 sont toutes des multiplications de variables différentes, et l’étape 2.1 est exécutée avec la fonction CARRE. Il présente toutefois l’inconvénient d’un temps d’exécution non optimisé en raison de l’exécution de multiplications factices. De plus, il existe une classe d’attaques dite "safe errors" qui permettent de détecter les opérations factices qu’un algorithme comporte. Ces attaques consistent à injecter une faute dans un calcul cryptographique à un instant particulier, et à

observer si le résultat du calcul est juste ou faux. Ce type d'attaque appliqué à l'algorithme A2 permet de savoir si une multiplication est faite après un "si" ou après un "sinon". En effet, dans le second cas, le résultat de la multiplication factice n'est pas utilisé pour le calcul du résultat final. Une injection d'erreur dans une boucle dans laquelle le branchement conditionnel "sinon" est actif n'affecte donc pas le résultat et permet de savoir que le branchement conditionnel "sinon" a été retenu et non le branchement "si".

L'algorithme A4 figurant en Annexe I, appelé "Montgomery Ladder", permet également de réaliser un calcul d'exponentiation modulaire. Cet algorithme consiste à chaque itération à effectuer deux multiplications, l'une appliquée à deux nombres différents, et l'autre appliquée à deux nombres identiques et pouvant donc être effectuée par une multiplication ou une élévation au carré. A chaque itération, l'un des deux registres R_0, R_1 reçoit le produit du contenu des deux registres, et le contenu de l'autre des deux registres est élevé au carré. Si le bit d_s correspondant à l'itération en cours, de l'exposant d est à 0, le registre R_1 reçoit le résultat de la multiplication, et le registre R_0 est élevé au carré, et inversement si le bit d_s est à 1.

Cet algorithme présente l'avantage d'être parfaitement régulier en ce que chaque itération comporte systématiquement deux multiplications, ou une multiplication et une élévation au carré. Cet algorithme est également optimisé en terme de temps de calcul dans la mesure où il ne comporte pas d'opération factice. Cet algorithme présente également l'avantage d'être parallélisable moyennant l'utilisation d'un registre supplémentaire. L'algorithme A4' figurant en Annexe I est une variante parallélisée de l'algorithme A4. L'algorithme A4' utilise un registre supplémentaire R_2 (cf. étape 2.2 de l'algorithme A4' en Annexe I) pour ne pas écraser l'opérande R_0 ou R_1 de la multiplication, lors de l'opération d'élévation au carré.

Les figures 4A et 4B représentent des courbes de consommation électrique d'un composant exécutant l'algorithme A4'. Ces courbes de consommation montrent une régularité dans la succession des pics de consommation électrique, que la seconde multiplication soit réalisée par la fonction MULT (figure 4A) ou la fonction CARRE (figure 4B).

La demande EP 2 492 804 (US2012/0221618) déposée par le Demandeur propose de remplacer dans les algorithmes A1, A1' ou A2, les

opérations de multiplication par des opérations d'élévation au carré en utilisant l'une ou l'autre des formules suivantes :

(i) $x \times y = [(x+y) \times (x+y) - x \times x - y \times y]/2$

(ii) $x \times y = [(x+y)/2] \times [(x+y)/2] - [(x-y)/2] \times [(x-y)/2]$

L'algorithme A5 figurant en Annexe I présente un exemple de calcul d'exponentiation modulaire mettant en œuvre la formule (ii), de gauche à droite. Le fonctionnement de cet algorithme peut être compris en se référant à l'algorithme classique A1 décrit plus haut. Deux cas peuvent se produire :

1) Le bit d_i de l'exposant vaut 0 :

- l'algorithme calcule $R_0=R_0 \times R_0$ (étape 3.1), ce qui correspond à l'étape 2.1 de l'algorithme A1,

2) le bit d_i vaut 1, l'algorithme effectue trois itérations de la boucle "tant que" :

- $R_0 = R_0 \times R_0$ (étape 3.1), ce qui correspond à l'étape 2.1 de l'algorithme A1)

- $R_2 = ((R_0+m)/2) \times ((R_0+m)/2)$ (étapes 3.3, 3.4, puis 3.1)

- $R_0 = ((R_0-m)/2) \times ((R_0-m)/2)$ et $R_0 = R_2-R_0$ (étapes 3.2, 3.4, puis 3.1), ce qui correspond à l'étape 2.2 de l'algorithme A1 mise en œuvre avec la formule (ii)).

Le profil de consommation électrique d'un composant exécutant l'algorithme A5 se présente sous la forme d'une succession de pics correspondant à des appels à la fonction MULT (ou CARRE). Une telle courbe de consommation ne permet pas de déduire la valeur des bits de l'exposant secret et est donc protégée d'une attaque SPA. D'autre part, l'attaque consistant à distinguer une multiplication de deux variables différentes d'une multiplication de deux variables identiques n'est pas applicable puisque le procédé ne comporte que des multiplications de termes égaux.

Le temps d'exécution par itération (par bit de l'exposant d) de l'algorithme A5 peut être évalué à 2 fois le temps de calcul d'une élévation au carré, soit 1,6 fois le temps de calcul d'une multiplication,.

Une variante parallélisable peut être dérivée de l'algorithme A5. Ainsi, l'algorithme A5' figurant en Annexe I présente un autre exemple de calcul d'exponentiation modulaire mettant en œuvre la formule (ii), de droite à gauche, sous une forme atomique, parallélisée.

Le profil de consommation électrique d'un composant exécutant l'algorithme A5' se présente sous la forme d'une double succession de pics correspondant à des appels à la fonction MULT (ou CARRE). Une telle succession ne permet pas de déterminer les bits de l'exposant d.

Le temps d'exécution de cet algorithme dépend du nombre de registres mis en œuvre, et tend vers le temps de calcul d'un carré par bit de l'exposant d, au prix de l'utilisation d'un très grand nombre L de registres de la taille de la donnée m à élever à la puissance d.

Il pourrait donc être souhaité de réduire le temps de calcul et la taille de mémoire utilisée par un procédé d'exécution d'un calcul d'exponentiation appliqué à une donnée de grande taille. Il pourrait être également souhaité de protéger un tel procédé contre les attaques à canal caché mentionnées précédemment, en pénalisant le moins possible le temps de calcul.

Des modes de réalisation concernent un procédé de calcul itératif d'exponentiation d'une donnée de grande taille par un exposant formé d'un certain nombre de bits, le procédé étant mis en œuvre dans un dispositif électronique et comprenant un calcul d'élévation au carré d'une variable de grande taille pour chaque bit de l'exposant, et des calculs de multiplication de variables de grande taille pour traiter des bits à un de l'exposant. Selon un mode de réalisation, les calculs d'élévation au carré et de multiplication de variables de grande taille sont effectués en parallèle, par des blocs d'élévation au carré et de multiplication appartenant au dispositif électronique, le procédé comprenant, des étapes consistant à : tant qu'une mémoire tampon de stockage temporaire de résultats fournis par le bloc d'élévation au carré n'est pas pleine de résultats non utilisés par le bloc de multiplication, déclencher un calcul par le bloc d'élévation au carré pour un bit de l'exposant, lorsque le bloc d'élévation au carré est inactif, stocker chaque résultat fourni par le bloc d'élévation au carré dans la mémoire tampon, si le bit de l'exposant correspondant est à 1, et tant que la mémoire tampon contient un résultat d'élévation au carré non utilisé par le bloc de multiplication, déclencher un calcul par le bloc de multiplication portant sur le résultat d'élévation au carré non utilisé, lorsque le bloc de multiplication est inactif.

Selon un mode de réalisation, le procédé comprend un déclenchement de calcul factice par le bloc d'élévation au carré, si la

mémoire tampon est pleine, si le bit correspondant de l'exposant est à 1, et si le bloc d'élévation au carré est inactif.

Selon un mode de réalisation, le procédé comprend un déclenchement de calcul factice par le bloc de multiplication si la mémoire tampon est vide, et si le bloc de multiplication est inactif.

Selon un mode de réalisation, tous les résultats fournis par les blocs d'élévation au carré et de multiplication sont utilisés pour l'obtention du résultat de l'exponentiation.

Selon un mode de réalisation, plusieurs résultats fournis par le bloc d'élévation au carré, sont stockés dans la mémoire tampon.

Selon un mode de réalisation, la mémoire tampon est gérée de manière cyclique, avec un indice d'écriture, un indice de lecture et un compteur de résultats non utilisés.

Selon un mode de réalisation, la mémoire tampon est configurée pour stocker trois à cinq résultats fournis par le bloc d'élévation au carré.

Selon un mode de réalisation, les calculs réalisés par les blocs d'élévation au carré et de multiplication sont des opérations modulaires.

Selon un mode de réalisation, plusieurs bits de l'exposant sont traités à chaque itération.

Des modes de réalisation concernent également un dispositif électronique comprenant un processeur configuré pour calculer une exponentiation d'une donnée de grande taille par un exposant, un bloc de calcul d'élévation au carré d'une variable de grande taille, et un bloc de calcul de multiplication de variables de grande taille. Selon un mode de réalisation, le dispositif comprend une mémoire tampon pouvant stocker plusieurs résultats fournis par le bloc d'élévation au carré, le dispositif étant configuré pour mettre en œuvre le procédé tel que précédemment défini.

Selon un mode de réalisation, la mémoire tampon est configurée pour stocker trois à cinq résultats fournis par le bloc d'élévation au carré.

Selon un mode de réalisation, les blocs d'élévation au carré et de multiplication comprennent chacun un coprocesseur de type à unité centrale programmable, ou un coprocesseur entièrement hardware de type machine d'état, ou bien sont réalisés chacun par une tache exécutée de manière indépendante d'un programme principal, les taches et le programme principal étant exécutés par le processeur de type multi-cœur.

Des exemples de réalisation de l'invention seront décrits dans ce qui suit, à titre non limitatif en relation avec les figures jointes parmi lesquelles :

la figure 1 précédemment décrite représente la courbe de consommation électrique d'un composant exécutant un premier algorithme d'exponentiation classique,

la figure 2 précédemment décrite représente la courbe de consommation électrique d'un composant exécutant un second algorithme d'exponentiation classique,

la figure 3 précédemment décrite représente la courbe de consommation électrique d'un composant exécutant un troisième algorithme d'exponentiation classique,

les figures 4A, 4B représentent des courbes de consommation électrique d'un composant exécutant deux variantes d'un algorithme d'exponentiation classique,

la figure 5 représente un dispositif électronique mettant en œuvre un algorithme d'exponentiation selon un mode de réalisation,

les figures 6 et 7 représentent des courbes de consommation électrique du dispositif électronique exécutant deux modes de réalisation d'un algorithme d'exponentiation,

la figure 8 représente un dispositif électronique mettant en œuvre un algorithme d'exponentiation selon un autre mode de réalisation.

L'invention concerne un procédé de calcul cryptographique comportant des multiplications de variables identiques de grande taille et des multiplications de variables différentes de grande taille. Il est mis en œuvre par un dispositif électronique configuré pour exécuter des calculs cryptographiques, incluant des exponentiations. Le procédé peut être un calcul d'exponentiation RSA, DSA, El Gamal, etc.

Selon un mode de réalisation, ce procédé se base sur une dérivation de l'algorithme A1', dans lequel les calculs d'élévation au carré sont effectués en utilisant une multiplication optimisée tenant compte de l'égalité des opérandes, et dans lequel les calculs de multiplication et d'élévation au carré effectués à chaque itération sont réalisés en parallèle d'une manière asynchrone. Ainsi, un nouveau calcul d'élévation au carré peut être déclenché dès qu'un précédent calcul d'élévation au carré s'est terminé,

sans attendre qu'une multiplication éventuellement en cours de calcul soit terminée. De même, un nouveau calcul de multiplication peut être déclenché dès qu'un précédent calcul de multiplication s'est terminé, sans attendre qu'une élévation au carré éventuellement en cours de calcul soit terminée. De cette manière, le résultat d'une élévation au carré ou d'une multiplication peut être exploité dès qu'il est disponible.

L'algorithme A6 figurant en Annexe II est un exemple d'algorithme d'exponentiation, de droite à gauche, selon un mode de réalisation.

Dans l'algorithme A6 et dans les algorithmes décrits plus loin :

M(a,b) désigne une fonction de calcul asynchrone du produit de nombres a et b de grande taille,

S(a) désigne une fonction de calcul asynchrone du carré du nombre a de grande taille,

FinM() désigne une fonction fournissant un indicateur à FAUX tant qu'un calcul de produit est en cours (fonction M active) et qui passe à VRAI dès qu'un produit est disponible (fonction M inactive), et

FinS() désigne une fonction fournissant un indicateur à FAUX tant qu'un calcul d'élévation au carré est en cours (fonction S active) et qui passe à VRAI dès qu'un carré est disponible.

L'algorithme A6 utilise un ensemble de L registres R de la taille de la donnée m à élever à la puissance d. Cet ensemble peut avantageusement être géré comme une mémoire tampon de type FIFO (First In – First Out) cyclique (étapes 3.1.1.1.2 et 3.2.1.2). A cet effet, un compteur k mémorise le nombre de carrés chargés dans les registres R, non utilisés par les multiplications. Un indice d'écriture j définit le registre R dans lequel le résultat de l'élévation au carré suivante doit être chargé, et un indice de lecture i définit le registre R où se trouve l'opérande de la multiplication suivante. Dans cet algorithme, l'étape 3 est un contrôle de boucle sur le nombre v de bits de l'exposant d. Les étapes 3.1.x sont exécutées lorsqu'un calcul d'élévation au carré s'est terminé (FinS = VRAI – étape 3.1), et lorsque la mémoire tampon formée par les registres n'est pas pleine (k≠L) et qu'un carré doit être chargé dans un registre R (d_s = 1 - étape 3.1.1.1). Un calcul d'élévation au carré (étape 3.1.1.2) est effectué pour chaque bit d_s de l'exposant d, mais seuls les résultats des élévations au carré sont chargés dans l'un des registres R (étape 3.1.1.1.1) lorsque le bit d_s de l'exposant en

cours de traitement est égal à 1. Les étapes 3.2.1.x sont exécutées lorsqu'un calcul de produit s'est terminé (FinM = VRAI – étape 3.2) et lorsque la mémoire tampon n'est pas vide (k≠0 – étape 3.2.1). Un calcul de produit (étapes 3.2.1.1 et 4.1.1) est effectué pour chaque calcul d'élévation au carré chargé dans un registre R. Les étapes 4.x permettent d'effectuer les multiplications restantes une fois que toutes les élévations au carré nécessaires ont été effectuées. Les étapes 5.x forment une boucle d'attente de la fin du dernier calcul de multiplication ou d'élévation au carré, le dernier calcul de multiplication fournissant le résultat du calcul d'exponentiation.

Le temps d'exécution par itération de cet algorithme dépend du nombre L de registres R utilisés. Plus ce nombre est grand, plus la probabilité que tous les registres R soient pleins durant l'exécution de l'algorithme est faible, et donc plus la probabilité qu'un calcul d'élévation au carré soit retardé est faible. A partir de L=3, le temps d'exécution par itération de cet algorithme atteint sensiblement une valeur minimum égale au temps de calcul d'une élévation au carré, soit environ 0,8 fois le temps de calcul d'une multiplication. L'algorithme A6 permet donc d'atteindre les performances de l'algorithme A5' avec beaucoup moins de registres de la taille de la donnée m à élever à la puissance d.

La figure 5 représente sous forme de schéma bloc un dispositif électronique DV1 configuré pour exécuter un calcul cryptographique incluant l'algorithme A6. Le dispositif DV1 peut être un circuit intégré sur microplaquette de semi-conducteur agencé sur un support portatif comme une carte en matière plastique, l'ensemble formant une carte à puce. Le dispositif DV1 peut également être n'importe quel appareil équipé d'un processeur multi-tâche, tel qu'un téléphone mobile intelligent ("smartphone"), un lecteur multimédia, une tablette tactile, ou un ordinateur personnel. Le dispositif DV1 peut également être un composant d'un tel appareil, par exemple relié à un processeur principal de l'appareil.

Le dispositif DV1 comprend un processeur PROC, un bloc de calcul MB1 configuré pour exécuter la fonction M(a,b) de multiplication de variables a, b, de grande taille un bloc de calcul SB1 configuré pour exécuter la fonction S(a) d'élévation au carré d'une variable a de grande taille, une mémoire MEM et un circuit d'interface de communication IC. Le circuit d'interface IC peut être du type à contact ou sans contact, par exemple un

circuit d'interface RF ou UHF fonctionnant par couplage inductif ou par couplage électrique. Les blocs de calcul MB1, SB1 peuvent comprendre chacun un coprocesseur équipé d'une unité centrale programmable, un coprocesseur entièrement hardware de type machine d'état, ou une tache ou fil d'exécution (thread) exécuté de manière indépendante d'un programme principal, notamment par un processeur multi-cœur. Les deux blocs de calcul MB1, SB1 peuvent être intégrés dans un même composant (coprocesseur ou machine d'état) s'ils peuvent fonctionner indépendamment l'un de l'autre.

De façon en soi classique, une variable est dite "de grande taille" lorsque sa taille (en nombre de bits) est supérieure à celle des registres de calcul du processeur PROC. Ce dernier effectue lui-même, sans faire appel aux blocs de calcul MB1, SB1, des multiplications de variables de petite taille, c'est-à-dire de taille inférieure ou égale à celle de ses registres de calcul, et fait appel aux blocs de calcul MB1, SB1 pour les multiplications et les élévations au carré de variables de grande taille, c'est-à-dire supérieure à la taille de ses registres de calcul. Par exemple, si la taille des registres de calcul du processeur PROC est de 32 bits, une variable de grande taille est une variable de plus de 32 bits. En cryptographie, les variables manipulées peuvent atteindre plusieurs centaines ou plusieurs milliers de bits (512, 1024, 2048, 4096 bits).

La mémoire MEM est couplée au processeur PROC et permet au dispositif DV1 de mémoriser une clé secrète d. Le processeur PROC reçoit, par l'intermédiaire du circuit d'interface IC, un message m à chiffrer ou à signer, et renvoie un message chiffré ou une signature du type $F_d(m)$, F étant une fonction de cryptographie basée sur la clé d comprenant un calcul d'exponentiation de type m^d modulo(n) exécuté au moyen de l'algorithme A6. Pendant le calcul d'exponentiation, le processeur PROC fait appel aux blocs de calcul MB1, SB1, en fournissant des variables a, b au bloc de calcul MB1 qui retourne a×b, et en fournissant une variable a au bloc de calcul SB1 qui retourne a^2. Le bloc MB1 fournit un signal EM dans un état par exemple actif lorsqu'un résultat de multiplication est disponible, et dans un état inactif lorsqu'il est en train de calculer un produit. De même, le bloc SB1 fournit un signal ES dans un état par exemple actif lorsqu'un carré est disponible, et dans un état inactif lorsqu'il est en train d'effectuer une élévation au carré. Une partie de la mémoire MEM peut également être utilisée comme mémoire

tampon pour mémoriser le contenu des registres R mentionnés dans l'algorithme A6.

La figure 6 représente un profil de consommation électrique d'un composant tel que le dispositif DV1, exécutant l'algorithme A6. Ce profil de consommation se présente sous la forme d'une double succession de pics correspondant à l'activité respective des blocs SB1 et MB1, c'est-à-dire aux appels des fonctions M() et S(). Au début de l'exécution de l'algorithme A6, aucun appel de la fonction M() ne peut être effectué tant que la fonction S() n'a pas fourni de résultat mémorisé dans un registre R, c'est-à-dire pour un bit d_s égal à 1 de l'exposant d. L'activité du bloc SB1 comprend des périodes T1 durant lesquelles le bloc est inactif. Ces périodes correspondent aux périodes durant lesquelles la mémoire tampon R[L] est pleine (de résultats d'élévation au carré non utilisées par une multiplication précédente) et qu'un résultat d'élévation au carré doit être chargé dans cette mémoire tampon ($d_s = 1$ - condition de l'étape 3.1.1. non satisfaite). L'activité du bloc MB1 comprend également des périodes T2 durant lesquelles le bloc est inactif. Ces périodes correspondent aux périodes durant lesquelles la mémoire tampon R[L] est vide (ne contient aucun résultat d'élévation au carré non utilisé par une multiplication précédente). Une analyse de l'activité des blocs SB1, MB1 peut donc permettre de déterminer la valeur de l'exposant d. Par conséquent, l'algorithme A6 est vulnérable aux attaques SPA. Cependant, l'analyse de cette activité pour en déduire l'exposant d peut être rendue plus difficile en augmentant le nombre L de registres R.

L'algorithme A7 figurant en Annexe II est un autre exemple d'algorithme d'exponentiation, de droite à gauche, selon un mode de réalisation. Cet algorithme qui peut être mis en œuvre dans le dispositif DV1, diffère de l'algorithme A6 en ce qu'il comporte des opérations factices permettant de neutraliser les attaques SPA. Ces opérations factices figurent aux étapes 3.1.2.1 et 3.2.2.1 où le résultat S(b) d'une élévation au carré factice et le résultat M(a,b) d'une multiplication factice sont chargés dans un registre x quelconque. Il en résulte que les profils de consommation électrique des blocs SB1, MB1 ne comportent plus de périodes d'inactivité T1, T2. Ces profils ne permettent donc pas de déterminer les bits de l'exposant d. Cependant, ce résultat est obtenu au prix de l'ajout d'opérations factices et donc d'une augmentation du temps d'exécution de

l'exponentiation. Ici encore, une augmentation du nombre L de registres R peut permettre de diminuer le nombre d'opérations factices à exécuter, en particulier le nombre d'élévations au carré factices.

L'algorithme A8 figurant en Annexe II est une version atomisée, selon un mode de réalisation, de l'algorithme A7. Cet algorithme qui peut également être mis en œuvre dans le dispositif DV1, diffère de l'algorithme A7 en ce que tous les branchements conditionnels ont été remplacés par une matrice binaire Q_L de 4(L+1) lignes et 3 colonnes (L étant le nombre de registres R), qui définit les opérations à réaliser à chaque itération. Par ailleurs, les étapes 4.x ont été intégrées dans la boucle formée par les étapes 3.x, le test de fin de boucle (étape 3) étant modifié de manière à inclure le cas traité par les étapes 4.x de l'algorithme A6 ou A7, c'est-à-dire lorsqu'une élévation au carré a été effectuée pour chacun des bits de l'exposant d et qu'il reste des carrés non utilisés par les multiplications dans les registres R. L'algorithme A8 calcule à l'étape 3.3 un indice de ligne g de lecture de la matrice Q_L en utilisant une fonction pos(x) qui retourne 1 si x>0 et sinon 0. La valeur de l'indice de ligne g dépend du numéro s du bit d_s en cours de traitement de l'exposant d, de la valeur de ce bit d_s, du nombre k de carrés non utilisés dans les registres R, et du nombre L de registres R. Cet algorithme utilise également deux paires de registres a[0], a[1], b[0], b[1] en remplacement des registres a et b, les registres a[0] et b[0] étant prévus pour recevoir le résultat des calculs factices d'élévation au carré et de multiplication, et les registres a[1] et b[1] les résultats des calculs non factices.

Il peut être noté que l'algorithme A8 continue à effectuer des élévations au carré une fois que tous les bits d_s de l'exposant d ont été traités, tant qu'il reste des carrés non utilisés dans les registres R. Toutefois, ces élévations au carré supplémentaires ne pénalisent pas le temps global d'exécution du calcul d'exponentiation étant donné qu'elles sont réalisées en même temps que des multiplications nécessaires au calcul d'exponentiation.

Le contenu de la matrice Q_L dépend du mode de calcul de l'indice de ligne g calculé à l'étape 3.3. Dans l'exemple du calcul de l'indice de ligne g effectué à l'étape 3.3 de l'algorithme A8, le contenu et la taille de la matrice Q_L dépend de la valeur du nombre L de registres R. L'Annexe III fournit les valeurs de la matrice Q_L pour L égal à 1, 2 et 3. La matrice Q_2 (L = 2) est

construite à partir de la matrice Q_1 (L = 1) en ajoutant à cette dernière à la troisième ligne en partant des première et dernière lignes de la matrice, les deux blocs de deux lignes (0,1,1 / 1,1,1) et (0,0,1 / 0,0,1) indiqués en gras dans la matrice Q_2 fournie en Annexe III. La matrice Q_3 est construite à partir de la matrice Q_2 en dupliquant les blocs de deux lignes ajoutés dans la matrice Q_2, à la troisième ligne en partant de la première ligne et de la dernière ligne. Les lignes ainsi ajoutées sont indiquées en gras dans la matrice Q_3 fournie en Annexe III. De même, la matrice Q_L (L>1) est construite en ajoutant à la troisième ligne à partir des première et dernière lignes de la matrice Q_1, respectivement (L-1) fois les deux blocs (0,1,1 / 1,1,1) et (0,0,1 / 0,0,1).

D'autres formules de calcul de l'indice de ligne pour lire la matrice Q_L peuvent être aisément imaginées. Ainsi, si l'on initialise l'indice g (à l'étape 1) tel que $g= 4*(1-d_s)+d_s+2$ $(= 6-3d_s)$, et que l'on choisit pour le calcul des indices g suivants, la formule suivante :

$$g = pos(v-s) * [4*(1-d_s) + d_s + 2*pos(L-k)] + pos(k) \quad (1)$$

on obtient la matrice référencée Q, fournie en Annexe IV. Il peut être noté que la matrice Q est indépendante du nombre L de registres R et présente une taille fixe de 8 lignes et 3 colonnes.

L'algorithme A9 figurant en Annexe II est une version dérivée de l'algorithme A6 avec application d'une fenêtre glissante de 2 bits sur l'exposant d, selon un mode de réalisation. En d'autres termes, l'algorithme A9 est obtenu en modifiant l'algorithme A6 de manière à traiter à chaque itération non pas un, mais deux bits consécutifs de l'exposant d. Plus précisément, l'algorithme A9 réalise une seule multiplication par groupe de deux bits de l'exposant d comportant au moins un bit à 1, et réalise finalement une élévation au carré et deux multiplications. L'algorithme A9 peut être aisément modifié pour traiter les bits de l'exposant d par groupes de plus de deux bits. Dans ce cas, une seule multiplication sera effectuée par groupe de bits comportant au moins un bit à un. Plusieurs élévations au carré et plusieurs multiplications seront nécessaires à la fin de l'algorithme pour déterminer le résultat. Il peut donc en résulter une réduction du nombre de multiplications de nombres de grande taille, au détriment de l'utilisation de p-1 registres supplémentaires de grande taille, p étant le nombre de bits par groupe.

Bien entendu, une version robuste aux attaques SPA de l'algorithme A9 peut être dérivée d'une manière analogue de l'un ou l'autre des algorithmes A7 et A8.

Le résultat du calcul d'exponentiation fourni par le processeur PROC peut être modulaire ou non modulaire. Le dispositif DV1 représenté sur la figure 5 réalise des opérations d'exponentiation non modulaires. Les blocs MB1 et SB1 du dispositif DV1 réalisent des opérations d'élévation au carré et de multiplication non modulaires. Par ailleurs, il peut être noté que les algorithmes A6, A7, A8 ne nécessitent pas d'être modifiés pour obtenir un résultat d'exponentiation modulaire ou non modulaire, seuls les blocs de calcul MB1 et SB1 devant être adaptés à cet effet.

La figure 8 représente un dispositif DV2 réalisant des exponentiations modulaires. A cet effet, le dispositif DV2 diffère du dispositif DV1 en ce qu'il comprend des blocs de multiplication MB2 et d'élévation au carré SB2 réalisant des calculs modulaires. Les blocs MB2, SB2 comprennent une entrée supplémentaire pour recevoir le module n des calculs modulaires. Le processeur PROC est alors configuré pour fournir une valeur du module n aux blocs MB2, SB2. Le message m à élever à la puissance d est alors inférieur au module n.

Les temps d'exécution par bit de l'exposant d et le nombre de registres utilisés pour chacun des algorithmes précédemment présentés sont rassemblés dans le tableau 1 suivant :

Tableau 1

Algorithme	Sécurisé SPA	Temps d'exécution par bit de l'exposant d	Nombre de registres
A1, A1'	NON	≈ 1S + 0.5M = 1.30M	2
A2	OUI	= 1S + 1M = 1.80M	2
A3	OUI	≈ 1.5M	2
A4	OUI	≈ 1M + 1S = 1,80M	2
A4'	OUI	≈ 1M	3
A5	OUI	≈ 2S = 1.60M	3
A5'	OUI	≈ 7/6S ≈ 0,93M	5 (L = 1)
A5'	OUI	≈ 15/14S ≈ 0,86M	7 (L = 3)
A6	NON	≈ 0.80M	3 (L = 1)
A7	OUI	≈ 0,84M	4 (L = 1)

A7	OUI	≈ 0,80M	6 (L = 3)
A8	OUI	≈ 0,84M	5 (L = 1)
A8	OUI	≈ 0,80M	7 (L = 3)

Il ressort du tableau 1 que seuls les algorithmes A5' à A8 atteignent un temps de calcul par bit de l'exposant, inférieur au temps de calcul M d'une multiplication de nombres de grande taille. Par ailleurs, les algorithmes A6, A7 et A8 permettent d'atteindre un temps de calcul sensiblement égal à celui d'une élévation au carré d'un nombre de grande taille (S ≈ 0,8M) par bit de l'exposant d, avec un nombre de registres notablement plus faible que l'algorithme A5'.

Il apparaîtra clairement à l'homme de l'art que la présente invention est susceptible de diverses variantes de réalisation et diverses applications. En particulier, des multiplications factices, notamment au démarrage d'un calcul d'exponentiation, peuvent être réalisées sans effectuer d'élévations au carré factices (suppression des étapes 3.1.2 et 3.1.2.1 dans l'algorithme A7). Inversement, des élévations au carré factices peuvent être réalisées sans effectuer de multiplications factices (suppression des étapes 3.2.2, et 3.2.2.1 dans l'algorithme A7). D'autres modes de gestion des registres R(L) peuvent être mis en œuvre dans les algorithmes A6, A7 et A8. Par exemple ces registres peuvent être utilisés en prévoyant des étapes telles que 2.1.1.4, 2.1.2.3 et 2.2.2.1 de l'algorithme A5', pour décaler le contenu de ces registres.

ANNEXE I (Faisant partie intégrante de la description)

Algorithme A1 - Exponentiation "Square & Multiply", de gauche à droite"

Entrée : *un message m et un module n entiers tels que m < n*
un exposant d entier de v bits tel que d = $(d_{v-1}\ d_{v-2}....\ d_0)_2$
Requiert : *un registre a de la taille du message m*
Sortie : *m^d modulo n*

Étape 1 : a = 1
Étape 2 : pour s allant de v-1 à 0 faire :
Étape 2.1 : $a = (a^2) \bmod n$ (CARRE)
Étape 2.2 : si $d_s = 1$ alors $a = (a \times m) \bmod n$ (MULT)
Étape 3 : Retourner a

Algorithme A1' - Exponentiation "Square & Multiply, de droite à gauche"

Entrée : *un message m et un module n entiers tels que m < n*
un exposant d entier de v bits tel que d = $(d_{v-1}\ d_{v-2}....\ d_0)_2$
Requiert : *deux registres a et b de la taille du message m*
Sortie : *m^d modulo n*

Étape 1 : a = 1 ; b = m
Étape 2 : pour s allant de 0 à v-1 faire :
Étape 2.1 : si $d_s = 1$ alors $a = (a \times b) \bmod n$ (MULT)
Étape 2.2 : $b = (b^2) \bmod n$ (CARRE)
Étape 3 : Retourner a

Algorithme A2 - Exponentiation "Square & Multiply Always", de gauche à droite

Entrée : *un message m et un module n entiers tels que m < n*
un exposant d entier de v bits tel que $d = (d_{v-1}\ d_{v-2} \ldots\ d_0)_2$
Requiert : *deux registres a et b de la taille du message m*
Sortie : m^d *modulo n*

Étape 1 : a = 1, b = 1
Étape 2 : pour s allant de v-1 à 0 faire :
Étape 2.1 : $a = (a^2) \bmod n$ (CARRE)
Étape 2.2 : si $d_{v-s} = 1$ alors $a = (a \times m) \bmod n$ (MULT)
Étape 3 : sinon $b = (a \times m) \bmod n$ (MULT)
Étape 3 : Retourner a

Algorithme A3 - "Multiply Always", de gauche à droite, version atomique

Entrée : *un message m et un module n entiers tels que m < n*
un exposant d entier de v bits tel que $d = (d_{v-1}\ d_{v-2} \ldots\ d_0)_2$
Requiert : *deux registres* R_0 *et* R_1 *de la taille du message m*
Sortie : m^d *modulo n*

Étape 1 : $R_0 = 1$, $R_1 = m$, s = v-1, k = 0
Étape 2 : tant que $s \geq 0$ **faire** :
Étape 2.1 : $R_0 = R_0 \times R_k \bmod n$ (MULT)
Étape 2.2 : $k = k \oplus d_s$; $s = s - 1 + k$
Étape 3 : Retourner R_0

L'opération "a ⊕ b" désigne le OU Exclusif bit à bit des variables a et b.

Algorithme A4 - Montgomery Ladder (de gauche à droite)

Entrée : *un message m et un module n entiers tels que m < n*
un exposant d entier de v bits tel que d = $(d_{v-1}\ d_{v-2}....\ d_0)_2$
Requiert : *deux registres R_0 et R_1 de la taille du message m*
Sortie : *m^d modulo n*

Étape 1 : $R_0 = 1$, $R_1 = m$
Étape 2 : pour s allant de v-1 à 0 faire:
Étape 2.1 : $b = d_s$
Étape 2.2 : $R_{1-b} = R_0 \times R_1 \bmod n$ (MULT)
Étape 2.3 : $R_b = (R_b)^2 \bmod n$ (CARRE)
Étape 3 : Retourner R_0

Algorithme A4' - Montgomery Ladder, version parallélisée

Entrée : *un message m et un module n entiers tels que m < n*
un exposant d entier de v bits tel que d = $(d_{v-1}\ d_{v-2}....\ d_0)_2$
Requiert : *trois registres R_0, R_1 et R_2 de la taille du message m*
Sortie : *m^d modulo n*

Étape 1 : $R_0 = 1$, $R_1 = m$
Étape 2 : pour s allant de v-1 à 0 faire:
Étape 2.1 : $b = d_s$
Étape 2.2 : $R_{1-b} = R_0 \times R_1 \bmod n \ ||\ R_2 = (R_b)^2 \bmod n$ (MULT||CARRE)
Etape 2.3 : $R_b = R_2$
Étape 3 : Retourner R_0

(le symbole || sépare des opérations effectuées en parallèle)

Algorithme A5 – "Square always" avec (ii), de gauche à droite,

Entrée : *un message m et un module n entiers tels que m < n*
un exposant d entier de v bits tel que d = $(d_{v-1}\ d_{v-2}\ldots\ d_0)_2$
Requiert : *trois registres R_0, R_1, R_2 de la taille du message m*
Sortie : *m^d modulo n*

Étape 1 : $R_0 = 1$; $R_1 = m \bmod n$; $R_2 = 1$
Étape 2 : u = 0 ; s = 2 ; w = 2 ; t = 0 ; i = v–1 ; j = 1
Étape 3 : tant que $i \geq 0$ faire :
Étape 3.1 : $R_u = (R_u)^2 \bmod n$ (CARRE)
Étape 3.2 : si w = 0 alors $R_w = (R_t - R_j) \bmod n$
Étape 3.3 : sinon $R_w = (R_t + R_j) \bmod n$
Étape 3.4 : $R_s = R_s/2 \bmod n$
Étape 3.5 : t = u ; $j = (t+1) \bmod 3$; $u = w * d_i$
Étape 3.6 : $s = (u+2) \bmod 4$; $w = t \oplus s$
Étape 3.7 : $i = i - (w >> 1)$
Étape 4 : Retourner R_0

L'opération a>>b désigne le décalage à droite de la variable a de b bits
L'opération a*b désigne la multiplication de variables de petite taille, qui est exécutée sans faire appel faire appel à un bloc de multiplication ou d'élévation au carré.

Algorithme A5' – "Square always" avec (ii), de droite à gauche, parallélisé

Entrée : *un message m et un module n entiers tels que m < n*

un exposant d entier de v bits tel que $d = (d_{v-1}\ d_{v-2} \ldots\ d_0)_2$

Requiert : *L+4 registres de la taille du message m : a,* $R_0, R_1, \ldots R_{L+2}$

Sortie : m^d *modulo n*

Étape 1 : $a = 1$; $R_1 = m$; $k = 0$

Étape 2 : pour s allant de 0 à v-1 faire :

- **Étape 2.1 :** si $d_s = 1$ alors
  - **Étape 2.1.1 :** si $k < L$ alors
    - **Étape 2.1.1.1 :** $R_0 = (a - R_1)^2 \bmod n$ || $R_{k+2} = (R_{k+1})^2 \bmod n$
    - **Étape 2.1.1.2 :** $a = (a + R_1)^2 \bmod n$ || $R_{k+3} = (R_{k+2})^2 \bmod n$
    - **Étape 2.1.1.3 :** $a = (a - R_0)/4 \bmod n$
    - **Étape 2.1.1.4 :** $(R_1, R_2, \ldots R_{L+1}) = (R_2, R_3, \ldots R_{L+2})$
    - **Étape 2.1.1.5 :** $k = k+1$
  - **Étape 2.1.2 :** sinon
    - **Étape 2.1.2.1 :** $R_0 = (a - R1)^2 \bmod n$ || $a = (a + R_1)^2 \bmod n$
    - **Étape 2.1.2.2 :** $a = (a - R_0)/4 \bmod n$
    - **Étape 2.1.2.3 :** $(R_1, R_2, \ldots R_{L+1}) = (R_2, R_3, \ldots R_{L+2})$
    - **Étape 2.1.2.4 :** $k = k-1$
- **Étape 2.2 :** sinon
  - **Étape 2.2.1 :** si $k = 0$ alors
    - **Étape 2.2.1.1 :** $R_1 = (R_1)^2 \bmod n$
  - **Étape 2.2.2 :** sinon
    - **Étape 2.2.2.1 :** $(R_1, R_2, \ldots R_{L+1}) = (R_2, R_3, \ldots R_{L+2})$
    - **Étape 2.2.2.2 :** $k = k-1$

Étape 3 : Retourner a

ANNEXE II (Faisant partie intégrante de la description)

Algorithme A6 – "Square & Multiply", parallélisé, de droite à gauche

Entrée : *un message m entier*
un exposant d entier de v bits tel que $d = (d_{v-1}\ d_{v-2} \ldots\ d_0)_2$
Requiert : deux registres a et b *de la taille du message m*, et
L registres R de la taille du message m : R[0] à R[L-1], avec L ≥ 1
Sortie : m^d

Étape 1 : a = 1 ; b = m ; s = 0
Étape 2 : i,j,k = 0,0,0
Étape 3 : Tant que s < v faire :
Étape 3.1 : Si FinS() alors :
Étape 3.1.1 : Si d_s = 0 OU k ≠ L alors :
Étape 3.1.1.1 : Si d_s = 1
Étape 3.1.1.1.1 : R[j] = b
Étape 3.1.1.1.2 : j = j+1 mod L
Étape 3.1.1.1.3 : k = k+1
Étape 3.1.1.2 : b = S(b) (CARRE)
Étape 3.1.1.3 : s = s+1
Étape 3.2 : Si FinM() alors :
Étape 3.2.1 : Si k ≠ 0 alors :
Étape 3.2.1.1 : a = M(a,R[i]) (MULT)
Étape 3.2.1.2 : i = i+1 mod L
Étape 3.2.1.3 : k = k–1
Étape 4 : Tant que k ≠ 0 faire :
Étape 4.1 : Si FinM() alors :
Étape 4.1.1 : a = M(a,R[i]) (MULT)
Étape 4.1.2 : i = i+1 mod L
Étape 4.1.3 : k = k–1
Étape 5 : Tant que FinS() = FAUX OU FinM() = FAUX faire :
Étape 5.1 : RIEN
Étape 6 : Retourner a

Algorithme A7 – "Square & Multiply", parallélisé, de droite à gauche

Entrée : *un message m entiers*
un exposant d entier de v bits tel que $d = (d_{v-1}\ d_{v-2} \ldots\ d_0)_2$
Requiert : trois registres a, b et x de *la taille du message m,* et
L registres R de la taille du message m : R[0] à R[L-1], avec $L \geq 1$
Sortie : m^d

Étape 1 : a = 1 ; b = m ; s = 0
Étape 2 : i,j,k = 0,0,0
Étape 3 : Tant que s < v faire :
  Étape 3.1 : Si FinS() alors :
    Étape 3.1.1 : Si d_s = 0 OU k ≠ L alors :
      Étape 3.1.1.1 : Si d_s = 1
        Étape 3.1.1.1.1 : R[j] = b
        Étape 3.1.1.1.2 : j = j+1 mod L
        Étape 3.1.1.1.3 : k = k+1
      Étape 3.1.1.2 : b = S(b) (CARRE)
      Étape 3.1.1.3 : s = s+1
    Étape 3.1.2 : Sinon faire :
      Étape 3.1.2.1 : x = S(b)
  Étape 3.2 : Si FinM() alors :
    Étape 3.2.1 : Si k ≠ 0 alors :
      Étape 3.2.1.1 : a = M(a,R[i])
      Étape 3.2.1.2 : i = i+1 mod L
      Étape 3.2.1.3 : k = k–1
    Étape 3.2.2 : Sinon faire :
      Étape 3.2.2.1 : x = M(a,R[i]) (MULT)
Étape 4 : Tant que k ≠ 0 faire :
  Étape 4.1 : Si FinM() alors :
    Étape 4.1.1 : a = M(a,R[i]) (MULT)
    Étape 4.1.2 : i = i+1 mod L
    Étape 4.1.3 : k = k–1
Étape 5 : Tant que FinS() = FAUX OU FinM() = FAUX faire :
  Étape 5.1 : RIEN
Étape 6 : Retourner a

Algorithme A8 – "Square & Multiply", parallélisé, de droite à gauche, version atomique

Entrée : *un message m entier*
un exposant d entier de v bits tel que $d = (d_{v-1}\ d_{v-2} \ldots\ d_0)_2$
Requiert : *L registres de la taille du message m : R[0] à R[L-1], avec* $L \geq 1$
2 registres *de la taille du message m* : a[0] et a[1]
2 registres *de la taille du message m* : b[0] et b[1]
une matrice Q_L de 4(L+1) lignes et 3 colonnes de bits
Sortie : m^d

Étape 1 : a[1] = 1 ; b[1] = m ; s = 0 ; g = d_s
Étape 2 : i,j,k = 0,0,0
Étape 3 : Tant que s < v OU k ≠ 0 faire :
Étape 3.1 : Si FinS() alors :
Étape 3.1.1 : R[j] = Q_L[g,0]*b[1] + (1–Q_L[g,0])*R[j]
Étape 3.1.2 : j = j+Q_L[g,0] mod L
Étape 3.1.3 : k = k+Q_L[g,0]
Étape 3.1.4 : b[Q_L[g,1]] = S(b[1]) (CARRE)
Étape 3.1.5 : s = s+Q_L[g,1]
Étape 3.2 : Si FinM() alors :
Étape 3.2.1 : a[Q_L[g,2]] = M(a[1],R[i]) (MULT)
Étape 3.2.2 : i = i+Q_L[g,2] mod L
Étape 3.2.3 : k = k–Q_L[g,2]
Étape 3.3 : g = (1–pos(v–s))*2(L+1) + 2*k + pos(v–s)*d_s
Étape 4 : Tant que FinS() = FAUX OU FinM() = FAUX faire :
Étape 4.1 : RIEN
Étape 5 : Retourner a[1]

Algorithme A9 – "Square & Multiply ", parallélisé, de droite à gauche, avec fenêtre glissante de 2 bits

Entrée : *un message m entier*

un exposant d entier de v+1 bits tel que d = (0 d_{v-1} d_{v-2}.... $d_0)_2$ (avec d_v=0)

Requiert : trois registres a[0], a[1] et b *de la taille du message m*, et

L registres R de la taille du message m : R[0] à R[L-1], avec L ≥ 1

H un tableau de L bits

Sortie : m^d

Étape 1 : a[0] = 1 ; a[1] = 1 ; b = m ; s = 0

Étape 2 : i,j,k = 0,0,0

Étape 3 : Tant que s < v faire :

Étape 3.1 : Si FinS() alors :

Étape 3.1.1 : Si d_s = 0 OU k ≠ L alors :

Étape 3.1.1.1 : Si (d_{s+1},$d_{s)}$ = (0,1)

Étape 3.1.1.1.1 : R[j] = b

Étape 3.1.1.1.2 : H[j] = 0

Étape 3.1.1.1.3 : j = j+1 mod L

Étape 3.1.1.1.4 : k = k+1

Étape 3.1.1.2 : Si (d_{s+1},$d_{s)}$ = (1,1)

Étape 3.1.1.2.1 : R[j] = b

Étape 3.1.1.2.2 : H[j] = 1

Étape 3.1.1.2.3 : j = j+1 mod L

Étape 3.1.1.2.4 : d_{s+1} = 0

Étape 3.1.1.2.5 : k = k+1

Étape 3.1.1.3 : b = S(b) (CARRE)

Étape 3.1.1.4 : s = s+1

Étape 3.2 : Si FinM() alors :

Étape 3.2.1 : Si k ≠ 0 alors :

Étape 3.2.1.1 : a[H[i]] = M(a[H[i]],R[i]) (MULT)

Étape 3.2.1.2 : i = i+1 mod L

Étape 3.2.1.3 : k = k–1

Étape 4 : Tant que k ≠ 0 faire :

Étape 4.1 : Si FinM() alors :

Étape 4.1.1 : a[H[i]] = M(a[H[i]],R[i]) (MULT)

Étape 4.1.2 : i = i+1 mod L

Étape 4.1.3 : k = k–1

Étape 5 : Tant que FinS() = FAUX OU FinM() = FAUX faire :

Étape 5.1 : RIEN

Étape 6 : b = S(a[1]) (CARRE) // calcul de $a[1]^2$

Étape 7 : b = M(a[1],b) (MULT) // calcul de $a[1]^3$

Étape 8 : Retourner M(a[0],b) (MULT) // résultat = $a[0] \times a[1]^3$

Annexe III (faisant partie intégrante de la description)

$$Q_1 = \begin{vmatrix} 0 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{vmatrix}$$

$$Q_2 = \begin{vmatrix} 0 & 1 & 0 \\ 1 & 1 & 0 \\ \mathbf{0} & \mathbf{1} & \mathbf{1} \\ \mathbf{1} & \mathbf{1} & \mathbf{1} \\ 0 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ \mathbf{0} & \mathbf{0} & \mathbf{1} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{vmatrix}$$

$$Q_3 = \begin{vmatrix} 0 & 1 & 0 \\ 1 & 1 & 0 \\ \mathbf{0} & \mathbf{1} & \mathbf{1} \\ \mathbf{1} & \mathbf{1} & \mathbf{1} \\ \mathbf{0} & \mathbf{1} & \mathbf{1} \\ \mathbf{1} & \mathbf{1} & \mathbf{1} \\ 0 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ \mathbf{0} & \mathbf{0} & \mathbf{1} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{vmatrix}$$

Annexe IV (faisant partie intégrante de la description)

$$Q = \begin{vmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{vmatrix}$$

REVENDICATIONS

1. Procédé de calcul itératif d'exponentiation d'une donnée (m) de grande taille par un exposant (d) formé d'un certain nombre de bits (d_s), le procédé étant mis en œuvre dans un dispositif électronique (DV1, DV2) et comprenant un calcul d'élévation au carré d'une variable de grande taille pour chaque bit de l'exposant, et des calculs de multiplication de variables de grande taille pour traiter des bits à un de l'exposant,

caractérisé en ce que les calculs d'élévation au carré et de multiplication de variables de grande taille sont effectués en parallèle, par des blocs d'élévation au carré (SB1, SB2) et de multiplication (SM1, SM2) appartenant au dispositif électronique, le procédé comprenant, des étapes consistant à :

tant qu'une mémoire tampon (R) de stockage temporaire de résultats fournis par le bloc d'élévation au carré n'est pas pleine de résultats non utilisés par le bloc de multiplication, déclencher un calcul par le bloc d'élévation au carré pour un bit de l'exposant, lorsque le bloc d'élévation au carré est inactif,

stocker chaque résultat fourni par le bloc d'élévation au carré dans la mémoire tampon, si le bit de l'exposant correspondant est à 1, et

tant que la mémoire tampon contient un résultat d'élévation au carré non utilisé par le bloc de multiplication, déclencher un calcul par le bloc de multiplication portant sur le résultat d'élévation au carré non utilisé, lorsque le bloc de multiplication est inactif.

2. Procédé selon la revendication 1, comprenant un déclenchement de calcul factice par le bloc d'élévation au carré (SB1, SB2), si la mémoire tampon est pleine (R), si le bit (d_s) correspondant de l'exposant (d) est à 1, et si le bloc d'élévation au carré est inactif.

3. Procédé selon la revendication 1 ou 2, comprenant un déclenchement de calcul factice par le bloc de multiplication (SM1, SM2) si la mémoire tampon (R) est vide, et si le bloc de multiplication est inactif.

4. Procédé selon la revendication 1, dans lequel tous les résultats fournis par les blocs d'élévation au carré (SB1, SB2) et de multiplication (SM1, SM2) sont utilisés pour l'obtention du résultat de l'exponentiation.

5. Procédé selon l'une des revendications 1 à 4, dans lequel plusieurs résultats fournis par le bloc d'élévation au carré (SB1, SB2), sont stockés dans la mémoire tampon (R).

6. Procédé selon l'une des revendications 1 à 5, dans lequel la mémoire tampon (R) est gérée de manière cyclique, avec un indice d'écriture (j), un indice de lecture (i) et un compteur (k) de résultats non utilisés.

7. Procédé selon l'une des revendications 1 à 6, dans lequel la mémoire tampon (R) est configurée pour stocker trois à cinq résultats fournis par le bloc d'élévation au carré (SB1, SB2).

8. Procédé selon l'une des revendications 1 à 7, dans lequel les calculs réalisés par les blocs d'élévation au carré (SB2) et de multiplication (SM2) sont des opérations modulaires.

9. Procédé selon l'une des revendications 1 à 8, dans lequel plusieurs bits de l'exposant (d) sont traités à chaque itération.

10. Dispositif électronique (DV1, DV2) comprenant un processeur (PROC) configuré pour calculer une exponentiation d'une donnée de grande taille (m) par un exposant (d), un bloc de calcul d'élévation au carré (SB1, SB2) d'une variable de grande taille, et un bloc de calcul de multiplication (SM1, SM2) de variables de grande taille,

caractérisé en ce qu'il comprend une mémoire tampon (R) pouvant stocker plusieurs résultats fournis par le bloc d'élévation au carré, le dispositif étant configuré pour mettre en œuvre le procédé selon l'une des revendications 1 à 9.

11. Dispositif selon la revendication 10, dans lequel la mémoire tampon (R) est configurée pour stocker trois à cinq résultats fournis par le bloc d'élévation au carré (SB1, SB2).

12. Dispositif selon la revendication 10 ou 11, dans lequel les blocs d'élévation au carré (SB1, SB2) et de multiplication (MB1, MB2) comprennent chacun un coprocesseur de type à unité centrale programmable, ou un coprocesseur entièrement hardware de type machine d'état, ou bien sont réalisés chacun par une tache exécutée de manière indépendante d'un programme principal, les taches et le programme principal étant exécutés par le processeur (PROC) de type multi-cœur.

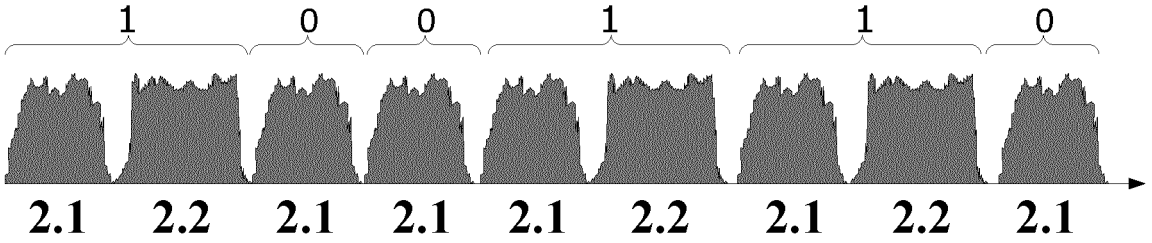


Fig. 1
(Art Antérieur)

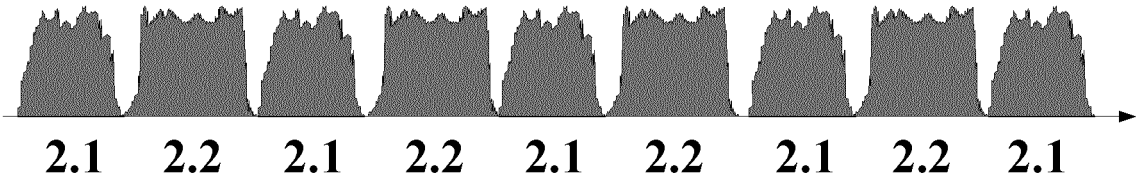


Fig. 2
(Art Antérieur)

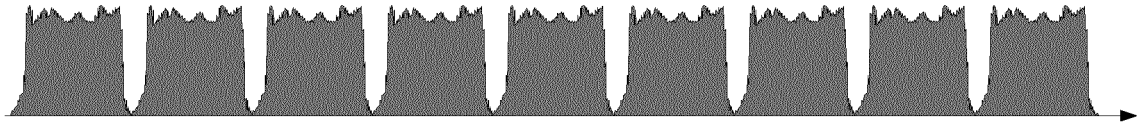


Fig. 3
(Art Antérieur)

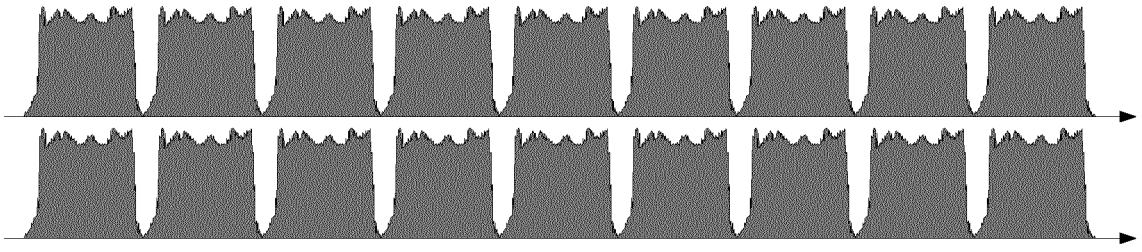


Fig. 4A
(Art Antérieur)

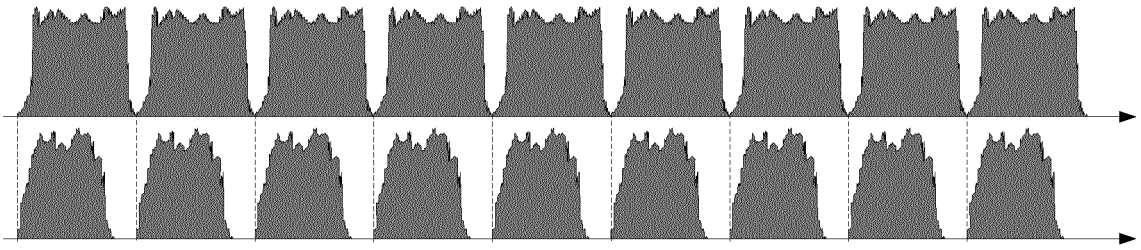


Fig. 4B
(Art Antérieur)

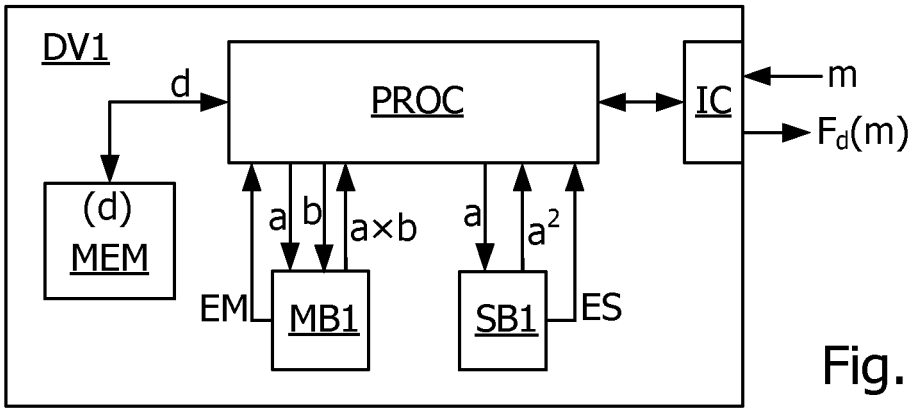


Fig. 5

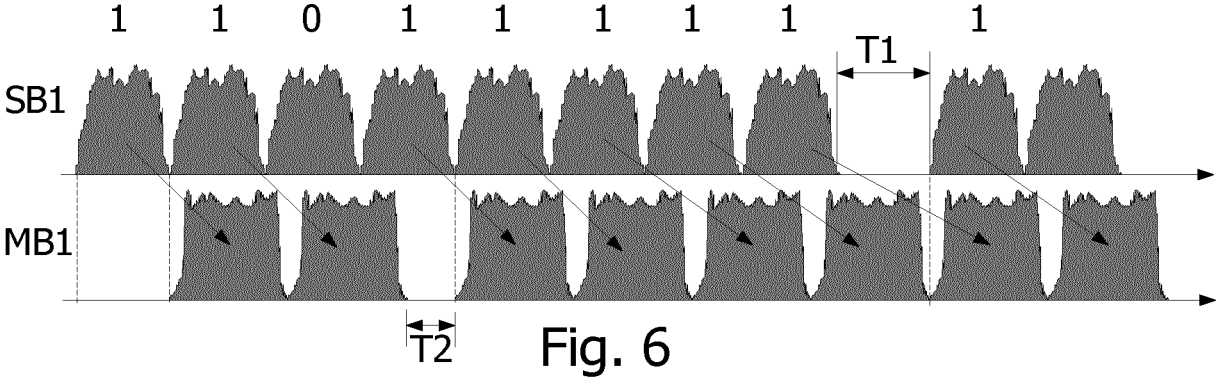


Fig. 6

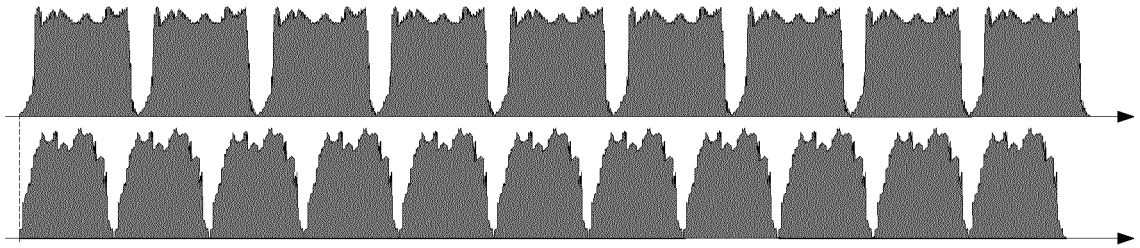


Fig. 7

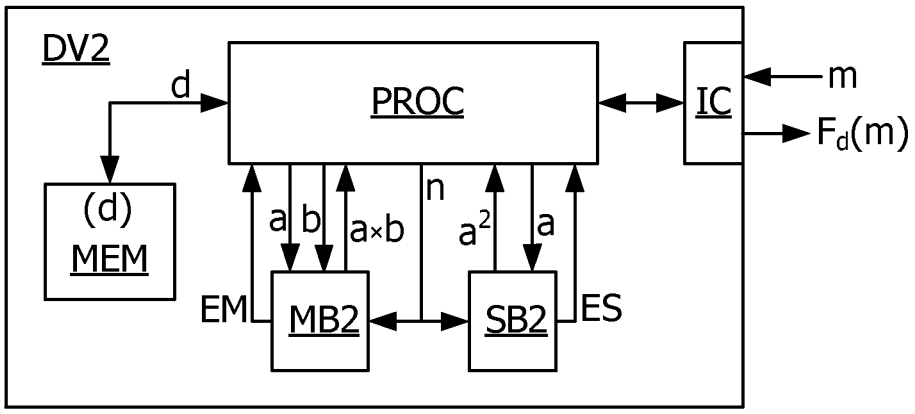


Fig. 8

RÉPUBLIQUE FRANÇAISE

INSTITUT NATIONAL DE LA PROPRIÉTÉ INDUSTRIELLE

RAPPORT DE RECHERCHE PRÉLIMINAIRE

établi sur la base des dernières revendications déposées avant le commencement de la recherche

N° d'enregistrement national

FA 773412
FR 1260553

DOCUMENTS CONSIDÉRÉS COMME PERTINENTS		Revendication(s) concernée(s)	Classement attribué à l'invention par l'INPI
Catégorie	Citation du document avec indication, en cas de besoin, des parties pertinentes		
Y,D	US 2012/221618 A1 (FEIX BENOIT [FR] ET AL) 30 août 2012 (2012-08-30) * le document en entier * -----	1-12	G06F21/00 G06F7/72
Y	ANSARI B ET AL: "Parallel scalar multiplication for elliptic curve cryptosystems", COMMUNICATIONS, CIRCUITS AND SYSTEMS, 2005. PROCEEDINGS. 2005 INTERNAT IONAL CONFERENCE ON HONG KONG, CHINA MAY 27-30, 2005, PISCATAWAY, NJ, USA,IEEE, vol. 1, 27 mai 2005 (2005-05-27), pages 71-73, XP010827154, DOI: 10.1109/ICCCAS.2005.1493364 ISBN: 978-0-7803-9015-7 * section III * -----	1-12	
A	Manfred Aigner ET AL: "Power Analysis Tutorial", , 1 janvier 2000 (2000-01-01), pages 1-15, XP055078152, Extrait de l'Internet: URL:http://www.iaik.tugraz.at/content/research/implementation_attacks/introduction_to_impa/dpa_tutorial.pdf [extrait le 2013-09-09] * section 4, paragraphe 2 * -----	1-12	DOMAINES TECHNIQUES RECHERCHÉS (IPC) G06F

Date d'achèvement de la recherche	Examinateur
9 septembre 2013	Prins, Leendert

CATÉGORIE DES DOCUMENTS CITÉS

X : particulièrement pertinent à lui seul
Y : particulièrement pertinent en combinaison avec un autre document de la même catégorie
A : arrière-plan technologique
O : divulgation non-écrite
P : document intercalaire

T : théorie ou principe à la base de l'invention
E : document de brevet bénéficiant d'une date antérieure à la date de dépôt et qui n'a été publié qu'à cette date de dépôt ou qu'à une date postérieure.
D : cité dans la demande
L : cité pour d'autres raisons
.....
& : membre de la même famille, document correspondant

EPO FORM 1503 12.99 (P04C14)

ANNEXE AU RAPPORT DE RECHERCHE PRÉLIMINAIRE RELATIF A LA DEMANDE DE BREVET FRANÇAIS NO. FR 1260553 FA 773412

La présente annexe indique les membres de la famille de brevets relatifs aux documents brevets cités dans le rapport de recherche préliminaire visé ci-dessus.
Les dits membres sont contenus au fichier informatique de l'Office européen des brevets à la date du 09-09-2013
Les renseignements fournis sont donnés à titre indicatif et n'engagent pas la responsabilité de l'Office européen des brevets, ni de l'Administration française

Document brevet cité au rapport de recherche	Date de publication	Membre(s) de la famille de brevet(s)	Date de publication
US 2012221618 A1	30-08-2012	CN 102684876 A	19-09-2012
		EP 2492804 A1	29-08-2012
		FR 2972064 A1	31-08-2012
		US 2012221618 A1	30-08-2012

EPO FORM P0465