

carte bancaire


PAR
Thomas Genet

NIVEAU DE LECTURE
Intermédiaire ● ● ●

PUBLIÉ LE
21/02/2008



Sans vraiment le savoir, nous utilisons tous, quotidiennement, la cryptographie asymétrique. Pire, nous réalisons quotidiennement des protocoles cryptographiques, sans même nous en douter. Il faut dire qu'ils sont indolores malgré leurs noms barbares. En revanche, si la sécurité de ces protocoles est mise en défaut, et pour peu qu'un fraudeur rôde, cela pourrait faire très mal à notre compte en banque.

Un protocole cryptographique est une succession d'échanges de messages chiffrés par des méthodes cryptographiques. De façon assez surprenante, pour saisir l'essence même d'un protocole cryptographique, il n'est généralement pas nécessaire de comprendre tous les détails des algorithmes de chiffrement. Nous présenterons la notation simplifiée communément utilisée pour décrire, de façon abstraite, les protocoles cryptographiques. Nous pourrions illustrer cette notation en nous référant au protocole [TLS](#) , *Transport Layer Security* (ou à son prédécesseur SSL, *Secure Socket Layer*) utilisé dans la sécurisation des pages Web, ou à un autre protocole en rapport direct avec l'informatique grand public. Mais nous avons choisi de prendre comme exemple un protocole

encore plus commun, le paiement par carte bancaire.

Notre but est aussi de faire comprendre les principales attaques menées par des fraudeurs sur ce protocole, pour en casser les sécurités et en retirer un avantage. Et nous montrerons les évolutions successives du protocole, mises en place par les sociétés de paiement bancaire afin de rendre inopérantes ces attaques.

Le protocole de paiement par carte, vu de l'extérieur

Le protocole utilisé pour le paiement par carte bancaire est un exemple idéal pour deux raisons. Premièrement, il est connu et utilisé couramment par tous. Deuxièmement, il illustre parfaitement les deux grands types de faiblesses d'un protocole : faiblesse cryptographique et faiblesse logique.

Nous ne nous intéresserons pas ici au protocole utilisé dans les guichets automatiques bancaires, qui est généralement plus complexe, mais au protocole que nous utilisons pour effectuer un paiement chez un commerçant. Ce protocole met en œuvre des agents qui sont les acteurs, humains ou matériels, de cette transaction.

Imaginons... A est la détentrice de la carte bancaire (nommons-la Alice), C la carte bancaire à puce détenue par Alice, T le terminal du commerçant sur lequel Alice compose son code secret et B , la banque d'Alice (ceci est une simplification, il s'agit en réalité d'un serveur central dédié à la vérification distante des cartes et de la solvabilité d'Alice).

Ces agents réalisent le protocole suivant :

- *Alice introduit sa carte C dans T*
- *Le commerçant saisit le montant m de la transaction sur T*


- *T authentifie C, le message « Authentication » est affiché sur T*
- *Le message « Code ? » est affiché sur T*
- *Alice tape son code, disons 3456, sur T*
- *T transmet le code à C*
- *Si le code est valide, C le signale à T*

Jusqu'à cette étape, tout se passe localement sur *T*. Vient ensuite une phase de vérification distante, à condition que le montant *m* dépasse 100 euros, et dans approximativement 20 % de ces cas.

- *T demande l'autorisation à B pour C et le montant m*
- *B donne l'autorisation*

Cette deuxième phase n'est pas systématiquement déclenchée, car elle est plus longue. Une des conséquences directes en est l'augmentation des temps d'attente et l'allongement des files aux caisses dans les grandes surfaces. Le seuil de 20 % de vérification a donc été choisi afin de pallier ce problème tout en gardant un quota de vérification dissuasif contre la fraude.

- *Dernière étape, le montant m est débité*

La version initiale de ce protocole comportait des faiblesses, largement exploitées en 1998 par Serge Humpich dans l'affaire dite des [Yescards](#) , pour attaquer le système de paiement par carte

bancaire. D'une part, cette attaque a démontré qu'il était possible de créer, de toutes pièces, une carte à puce ne pouvant être rattachée à aucun compte ou client réel à débiter. D'autre part, elle a prouvé que cette même carte traversait, sans encombre, la première phase de vérification locale du protocole. Elle pouvait donc être utilisée par une personne malveillante pour réaliser des paiements dont le montant était inférieur à 100 euros, sans fournir aucune indication sur le compte à débiter !

Mais avant d'expliquer cette attaque plus en détail, précisons comment fonctionne ce protocole.

Le protocole de paiement par carte, vu de l'intérieur

La notion centrale des protocoles cryptographiques est celle du chiffrement, une opération visant à transformer un texte intelligible en une version incompréhensible, à l'aide d'une clé. À l'inverse, le déchiffrement permet de produire la version intelligible à partir du texte chiffré. Pour connaître les principes mathématiques sous-jacents, voir [Cryptographie : les mathématiques au service de la protection de l'information](#).

Afin de s'abstraire, d'une part, des principes mathématiques des algorithmes cryptographiques et, d'autre part, des détails de programmation de ces protocoles, ceux-ci sont généralement présentés en utilisant une notation simplifiée comme celle présentée ci-dessous :

- $\{m\}_K$, le message m chiffré par la clé K ,
- $A \rightarrow B : m$, l'envoi par l'agent A (Alice) d'un message m à l'agent B .

Après avoir vu comment noter un message chiffré par une clé, voyons maintenant le type de chiffrement utilisé : le chiffrement asymétrique ([l'algorithme RSA](#) ou le logiciel PGP, *Pretty Good Privacy*, en sont des exemples). Dans ce type de chiffrement, chaque agent dispose d'un couple de clés. Par exemple, Alice dispose d'une clé publique K_A distribuée à tous les agents et d'une clé privée K_A^{-1} connue d'elle seule. Un message chiffré avec la clé K_A ne pourra être déchiffré qu'avec

la clé inverse K_A^{-1} . Réciproquement, un message chiffré avec la clé K_A^{-1} ne pourra être déchiffré qu'avec la clé inverse K_A . Dans tous les cas, on suppose qu'une tentative de déchiffrement réalisée avec une mauvaise clé échoue.

Voyons maintenant les propriétés associées aux messages chiffrés que l'on peut construire avec ces clés : confidentialité et authentification. Le message $\{m\}_{K_B}$, qui correspond au chiffrement du message m par la clé publique de B , ne pourra être déchiffré que par B , car lui seul détient la clé inverse K_B^{-1} . En conséquence, c'est un moyen simple pour n'importe quel agent d'envoyer de façon confidentielle un message m à B .

Le message $\{m\}_{K_B^{-1}}$ qui correspond au chiffrement du message m par la clé privée de B , pourra être déchiffré par tous les agents et leur permettra ainsi d'authentifier l'émetteur de ce message. On appelle aussi ce type de procédé signature numérique. La raison pour laquelle il est possible d'associer ce message à B est dû au fait que seul B détient K_B^{-1} et qu'il est donc le seul agent en mesure de construire ce message.

Nous pouvons maintenant donner une description plus précise du protocole cryptographique relatif au paiement par carte bancaire (voir l'article de Jacques Patarin, « La cryptographie des cartes bancaires », *Dossier spécial Pour la science*, Numéro 36, Juillet 2002). Nous nous focalisons ici sur la première phase du protocole, dite d'authentification hors-ligne, n'utilisant que la puce et non les serveurs distants. La version du protocole que nous présentons ici est volontairement simplifiée. En particulier, nous avons fait abstraction de l'utilisation de fonctions de hachage, nécessaires pour des raisons de performances mais inutiles à connaître pour comprendre le protocole.

Les clés et les données utilisées dans ce protocole sont les suivantes :

- La banque B possède un couple de clés asymétriques K_B et K_B^{-1} .
- La carte C possède la donnée $Data = \text{nom, prénom, numéro carte, date de validité, ainsi qu'une valeur de signature}$: la donnée $\{Data\}_{K_B^{-1}}$. Il faut comprendre que ce message chiffré est calculé, une fois pour toutes par B , et enregistré sur la carte au moment de son initialisation. En conséquence, la carte ne dispose pas de K_B^{-1} . Par ailleurs, la carte connaît son code à 4 chiffres. Ce code a pour valeur 3456 dans notre exemple.
- Le terminal T possède la clé publique de la banque, K_B .


En utilisant les notations précédentes, la première phase du protocole, sans authentification distante, correspond aux échanges de messages suivants :

- $T \rightarrow A : \text{« Authentification »}$
- $C \rightarrow T : Data, \{Data\}_{K_B^{-1}}$
- $T \rightarrow A : \text{« Code ? »}$
- $A \rightarrow T : 3456$
- $T \rightarrow C : 3456$
- $C \rightarrow T : ok$



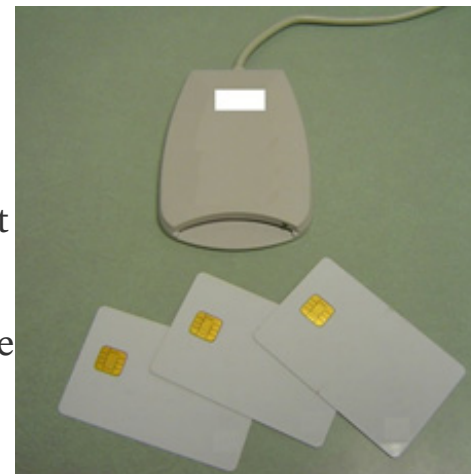
Voici maintenant le détail des événements correspondants à ces messages. Le message 1 est affiché sur l'écran du terminal à destination d'Alice. Dans le message 2, la carte envoie son champ *Data* et la valeur de signature au terminal. Ensuite, le terminal utilise la clé publique de la banque K_B pour déchiffrer $\{Data\}_{K_B}^{-1}$ et vérifier que ce qu'il obtient concorde bien avec le champ *Data* envoyé en clair dans la première partie du message 2 de la carte. Le message 3 est affiché sur l'écran du terminal pour Alice. Dans le message 4, Alice tape son code sur le terminal et celui-ci est transmis à la carte dans le message 5. Enfin, le message 6 est un message à destination du terminal confirmant que la carte a accepté le code qui lui a été transmis.

L'attaque sur le protocole de paiement

Le protocole de paiement par carte bancaire comporte plusieurs faiblesses qui ont été exploitées par Serge Humpich en 1998. Connues du groupement des cartes bancaires, ces faiblesses, et l'attaque possible en résultant, étaient même évoquées dès 1988 par Louis Guillou dans son article [Techniques à clé publique](#)  (paru dans les Annales des Télécommunications). Mais la facilité avec

laquelle cette attaque pouvait être mise en œuvre et reproduite à grande échelle a été sous-estimée. Elle concernait uniquement les transactions avec authentification hors-ligne, donc *a priori*, il était impossible d'attaquer les distributeurs qui pratiquent systématiquement l'authentification en ligne avec un central.

Initialement, la sécurité du paiement par carte à puce reposait beaucoup sur le secret autour du protocole employé et sur l'impossibilité de créer une réplique de la carte. Or, la première sécurité s'est effondrée par rétro-ingénierie d'un terminal bancaire et la seconde, par la mise sur le marché d'outils grand public de programmation de cartes à puce. De plus, ce protocole souffrait d'une faiblesse cryptographique, car les clés asymétriques RSA de 320 bits n'étaient déjà plus sûres depuis 1988, en raison des progrès faits par les algorithmes de factorisation. En clair, il était possible en 1998, à partir d'une clé publique K_B extraite de la mémoire d'un terminal, de retrouver par calcul K_B^{-1} la clé privée de la banque. Enfin, ce protocole souffrait d'une faiblesse logique que nous détaillons maintenant.



Exemple d'outil de programmation de cartes à puce.

Supposons que ma belle-mère Alice, A , dispose d'une carte bancaire C . Lorsqu'elle fait des achats chez un commerçant, elle réalise — sans le savoir je pense — des sessions de protocole de la forme :

- $T \rightarrow A : \text{« Authentication »}$
- $C \rightarrow T : Data, \{Data\}_{K_B}^{-1}$
- $T \rightarrow A : \text{« Code ? »}$
- $A \rightarrow T : 3456$

- $T \rightarrow C : 3456$
- $C \rightarrow T : ok$



Par ailleurs, moi l'intrus I , avec ma carte C' dont les informations bancaires sont $Data'$ et le code 6789, je réalise des transactions, sans vraiment y penser non plus d'ailleurs, de la forme :

- $T \rightarrow I : \text{« Authentication »}$
- $C' \rightarrow T : Data', \{Data'\}_{K_B}^{-1}$
- $T \rightarrow I : \text{« Code ? »}$
- $I \rightarrow T : 6789$
- $T \rightarrow C' : 6789$
- $C' \rightarrow T : ok$

□

Ma belle-mère me prête sa carte bancaire C . Ne me faisant pas particulièrement confiance — et elle a raison —, belle-maman ne m'a pas communiqué le code de sa carte. Supposons maintenant que

pendant la transaction, il soit possible d'intervertir les cartes bancaires sans éveiller les soupçons du commerçant ou faire réagir le terminal. Dans ce cas, je pourrais commencer une transaction avec la carte C de ma belle-mère et, après l'authentification, la remplacer par ma propre carte C' . Je pourrais réaliser des transactions de la forme :

- $T \rightarrow I : \ll \text{Authentication} \gg$
- $C \rightarrow T : Data, \{Data\}_{K_B}^{-1}$
- $T \rightarrow I : \ll \text{Code ?} \gg$
- $I \rightarrow T : 6789$
- $T \rightarrow C' : 6789$
- $C' \rightarrow T : ok$

□

Lors de cette transaction, quelle carte serait débitée ? Celle de la personne dont les coordonnées bancaires $Data$ sont acceptées en début de transaction, c'est-à-dire ma belle-mère en l'occurrence. Évidemment, me direz-vous, il n'est pas possible d'intervertir les cartes en cours de transaction. Vous avez raison, mais là où le bât blesse, c'est qu'en programmant une carte à puce, il est possible de réaliser de telles transactions sans interversion de carte ! En recopiant les données publiques $Data$ et $\{Data\}_{K_B}^{-1}$ de la carte de ma belle-mère sur une carte à puce vierge et en ajoutant un programme qui renvoie toujours la valeur *ok* quel que soit le code confidentiel saisi, j'obtiens une carte qui réalise la même transaction sans bloquer le terminal.

À ce stade, nous n'avons utilisé que la faiblesse logique du protocole qui permet, en entrelaçant


astucieusement deux sessions différentes, de le prendre en défaut. En revanche, si on la combine avec la faiblesse cryptographique des clés asymétriques utilisées jusqu'en 1998, on peut retrouver la clé privée K_B^{-1} de la banque et forger de toutes pièces de fausses coordonnées bancaires reconnues par le terminal. Soit par exemple XXX un ensemble de coordonnées : nom, prénom, numéro de carte, date de validité imaginaires. Puisque nous disposons, maintenant, de la clé K_B^{-1} , il nous est possible de construire $\{XXX\}_{K_B}^{-1}$. Nous pouvons ainsi construire une Yescard en ajoutant XXX et $\{XXX\}_{K_B}^{-1}$ dans la zone publique de la carte et en ajoutant le programme répondant toujours *ok* quel que soit le code tapé. Les transactions effectuées à l'aide de cette Yescard deviennent :



- $T \rightarrow I : \text{« Authentication »}$
- $C \rightarrow T : XXX, \{XXX\}_{K_B}^{-1}$
- $T \rightarrow I : \text{« Code ? »}$
- $I \rightarrow T : 0000$
- $T \rightarrow C : 0000$
- $C \rightarrow T : ok$

□

Corrections du protocole

Par la modification du protocole cryptographique, les faiblesses précédentes sont en passe d'être corrigées par les sociétés de carte bancaire. À la suite de l'affaire Humpich de 1998, la première réaction du groupement des cartes bancaires fut d'allonger en 1999 la taille des clés RSA utilisées,


afin de pallier la faiblesse cryptographique du protocole. Cet allongement de la taille des clés rendait impossible non seulement la déduction des nouvelles clés K_B^{-1} à partir des nouvelles clés publiques K_B , mais aussi la création d'identités bancaires factices, impossibles à débiter. En revanche, subsistait toujours la possibilité de recopier les données publiques d'une carte valide vers une Yescard et de débiter le compte d'un tiers. C'est d'ailleurs un principe très exploité qui donne aujourd'hui encore régulièrement lieu à des escroqueries. Une des dernières en date, [révélée le 9 février 2007](#) , a ainsi atteint un montant estimé à 640 000 euros.

Le fait que le « secret » entourant les rouages du protocole de paiement n'ait pas suffi à le prémunir contre les attaques est sans doute à la base du changement de comportement des grandes sociétés de cartes bancaires. Contrairement aux habitudes en la matière, le consortium [EMVCo](#)  (pour Europay, MasterCard et Visa) a publié sur le Web les spécifications détaillées de son successeur [EMV](#) . Celui-ci propose, en particulier, non pas un mais trois protocoles de transaction pouvant être activés en fonction de leur disponibilité sur la carte et/ou le terminal. Nous allons présenter les deux premiers : SDA et DDA. Comme précédemment, nous nous attacherons essentiellement à la phase d'authentification hors-ligne, celle-ci étant la plus fragile. Et pour faciliter la compréhension, les deux protocoles sont eux aussi présentés en faisant abstraction des détails techniques.

Dans le protocole le plus simple, SDA (*Static Data Authentication*) pour l'authentification locale, on retrouve sur la carte les données publiques suivantes : les données bancaires *Data* et la valeur de signature $\{Data\}_{K_B}^{-1}$. Dans les données publiques fournies par la carte, on trouvera également un certificat, ici $\{K_B\}_{K_S}^{-1}$. Ce certificat est le résultat de la signature de la clé publique de banque K_B par une autorité de certification (S). Seule la clé publique de l'autorité de certification K_S sera connue du terminal. Ce dernier pourra retrouver K_B en déchiffrant $\{K_B\}_{K_S}^{-1}$ avec K_S . Une transaction avec le

protocole SDA se déroule de la façon suivante :

- $T \rightarrow A : \ll \textit{Authentication} \gg$
- $C \rightarrow T : \{KB\}_{K_S}^{-1}, Data, \{Data\}_{K_B}^{-1}$
- $T \rightarrow A : \ll \textit{Code ?} \gg$
- $A \rightarrow T : 3456$
- $T \rightarrow C : 3456$
- $C \rightarrow T : ok$

Ce protocole est sensible aux mêmes faiblesses logiques que le protocole original. En particulier, il est toujours possible de recopier les données publiques $\{K_B\}_{K_S}^{-1}, Data, \{Data\}_{K_B}^{-1}$ et ainsi, de produire une Yescard débitant le compte correspondant à *Data*. Fort heureusement, depuis la fin mai 2007, il semble que la quasi totalité des terminaux bancaires et une grande quantité des cartes en circulation soient en mesure d'utiliser le protocole [DDA](#)  (*Dynamic Data Authentication*) qui est, pour l'instant, invulnérable à ce type d'attaques. Une des raisons justifiant le délai d'exploitation de ce nouveau protocole est qu'il nécessite des cartes à puce capables de réaliser un chiffrement asymétrique RSA dans des temps raisonnables. En effet, les cartes à puce utilisées dans les cartes bancaires ne disposent que de peu de ressources et RSA est un algorithme de chiffrement très gourmand.

Dans ce protocole, la carte *C* dispose également d'un couple de clés asymétriques, K_C et K_C^{-1} et de la faculté de réaliser un chiffrement avec l'une ou l'autre. Dans sa zone publique, la carte dispose, en plus de la valeur de signature, des certificats $\{K_B\}_{K_S}^{-1}$ et $\{K_C\}_{K_B}^{-1}$. Ainsi, par des déchiffrements en cascades, à partir de la seule clé publique K_S , le terminal est capable de retrouver successivement

K_B à partir du certificat $\{K_B\}_{K_S}^{-1}$, puis K_C à partir de K_B et de $\{K_C\}_{K_B}^{-1}$. Une transaction avec DDA prend la forme suivante :

- $T \rightarrow A : \text{« Authentication »}$
- $C \rightarrow T : \{K_B\}_{K_S}^{-1}, \{K_C\}_{K_B}^{-1}, Data, \{Data\}_{K_B}^{-1}$
- $T \rightarrow C : N_T$
- $C \rightarrow T : \{N_T\}_{K_C}^{-1}$
- $T \rightarrow A : \text{« Code ? »}$
- $A \rightarrow T : 3456$
- $T \rightarrow C : \{3456\}_{K_C}$
- $C \rightarrow T : ok$

Afin d'avoir une authentification plus forte que dans le cas des protocoles précédents, ce protocole utilise la notion classique de « challenge ». À l'étape 3, le terminal génère un nombre aléatoire N_T et l'envoie à la carte. À l'étape 4, la carte calcule le résultat du chiffrement de N_T par K_C^{-1} et l'envoie au terminal. Ce dernier peut ensuite déchiffrer $\{N_T\}_{K_C}^{-1}$ avec la clé publique K_C et vérifier qu'il obtient bien le nombre N_T qu'il avait envoyé à l'étape 3. Si tel est le cas, il a la garantie qu'il communique bien avec la carte (C), car seule cette carte dispose de K_C^{-1} et est capable de répondre à ce « challenge ». De plus, le code de la carte, transmis en clair par les autres protocoles, est chiffré par K_C et ainsi transmis de façon confidentielle à la carte.

Vers des protocoles infaillibles ?

Le mécanisme de « challenge » précédent semble à même de garantir la bonne authentification de

la carte par le terminal. Malgré tout, il reste très difficile de prouver qu'un protocole est exempt de failles. Une large communauté de chercheurs s'applique à proposer des outils automatiques permettant de donner des garanties de sûreté sur les protocoles. On sait déjà, grâce à des résultats théoriques récents, qu'il n'existe pas de programme pouvant démontrer la sécurité d'un protocole cryptographique en général. Voir par exemple l'article « Vérifier les protocoles cryptographiques » de Véronique Cortier (à [télécharger en PostScript](#)) pour un panorama de la recherche ainsi que des résultats d'indécidabilité obtenus dans ce domaine.

Cependant, si l'on se restreint aux failles logiques, il existe des outils automatiques capables de détecter des attaques. De plus, pour certaines catégories de protocoles, si aucune attaque logique n'est détectée, ces outils sont parfois capables de démontrer leur absence, fournissant ainsi de bonnes garanties quant à leur sûreté. C'est ce genre de tâche que réalisent, par exemple, l'outil [ProVerif](#), l'outil en ligne [AVISPA](#) et sa version autonome [SPAN](#).

Les détails liés à la programmation et au déploiement d'un protocole sont parfois à la source de ses faiblesses. C'est notamment le cas pour EMV qui est sensible à des attaques requérant un matériel spécifique, comme en témoigne par exemple une présentation de Steven J. Murdoch, [téléchargeable en PDF](#) (en anglais, 1,3 Mo). Ces attaques sont difficiles à parer mais, à la différence des failles logiques qui remettent en question la sécurité globale du protocole, celles-ci sont généralement plus difficiles à mettre en œuvre et à reproduire à grande échelle. Par conséquent, même s'ils ne travaillent qu'au niveau logique, les outils de vérification automatique de protocoles cryptographiques donnent des garanties importantes sur la sûreté des protocoles. Ceci explique le début de transfert des outils de vérification du domaine de la recherche vers le monde industriel.

Matériel utilisé par Steven J. Murdoch pour son attaque.

Photo : Steven J. Murdoch.



Niveau de lecture

Aidez-nous à évaluer le niveau de lecture de ce document.

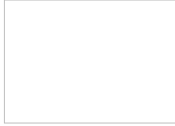
Il vous semble :

- ☐ facile à lire (affiché : facile)
- ☐ totalement accessible avec un léger bagage scientifique (facile)
- ☐ accessible en grande partie avec un léger bagage scientifique (intermédiaire)
- ☐ accessible avec des connaissances scientifiques (avancé)
- ☐ difficile d'accès (avancé)

Si vous souhaitez expliquer votre choix, vous pouvez ajouter un commentaire (qui ne sera pas publié).

Enregistrer

SUR LE MÊME SUJET

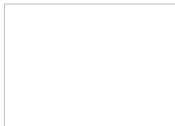


Nombres premiers et cryptologie : l'algorithme RSA

RECOMMANDATIONS



Le protocole cryptographique de paiement par carte bancaire



Nombres premiers et cryptologie : l'algorithme RSA

Les réseaux de pair à pair

Rubriques

De la recherche
Découvrir
Approfondir
Itinéraires

Médias

Podcast
Vidéos
Animations
Tous les documents

Spécial

Milieux : interactions,
interfaces, homogénéité,
ruptures (TIPE 2017-2018)
Optimalité : choix,

À propos

Qui sommes-nous ?
Équipe éditoriale
Tous les auteurs
Conseils aux auteurs

[C'était hier](#)
[Débattre](#)
[Ludique](#)
[Lire et voir](#)
[Archives](#)

[contraintes, hasard \(TIPE
2016-2017\)](#)
[L'informatique - ou
presque - dans les films](#)
[Idées reçues](#)
[Ressources pour les
lycéens](#)
[Données - Vie privée](#)

[Sites complémentaires](#)
[Témoignages](#)
[Contactez-nous](#)
[Twitter](#)
[Facebook](#)

[Interstices.info](#) Revue de culture scientifique en ligne publiée par Inria