Plateformes de TP interactives pour l'enseignement de la cryptographie

Charles Bouillaguet

▶ **To cite this version:**

Charles Bouillaguet. Plateformes de TP interactives pour l'enseignement de la cryptographie. Rendez-vous de la recherche et de l'enseignement de la sécurité des systèmes d'information (RESSI 2023), gdr securité, May 2023, Neuvy-sur-Barangeon, France. hal-04038343

HAL Id: hal-04038343
https://hal.science/hal-04038343

Submitted on 20 Mar 2023

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Plateformes de TP interactives pour l'enseignement de la cryptographie

Charles Bouillaguet (charles.bouillaguet@lip6.fr)

1 Introduction

Ce court article décrit mes tentatives de réalisation de plate-formes en ligne pour la réalisation de travaux pratiques de cryptologie par des étudiants de master.

Un cours de cryptologie standard est généralement composé de cours « magistraux », de séances d'exercices sur papier ou au tableau (travaux dirigés) et parfois de séances de travaux pratiques ou de la réalisation de projets. Autant des supports peuvent exister pour les deux premiers composants [9], autant pour la conception de TPs, les enseignants sont relativement livrés à eux-mêmes.

Une rapide recherche sur internet montre que beaucoup de cours de crypto (y compris celui que j'avais moi-même suivi) contiennent en fait *peu* de TPs, et que quelques tâches sont récurrentes : utilisation de PGP et/ou d'OpenSSL ; implantation du cryptosystème RSA et/ou de l'échange de clef Diffie-Hellman ; cryptanalyse d'un système très ancien (de César, de Vigenère, ...). Plus rarement, implantation de cryptosystèmes ou d'attaques reposant sur les LFSR, ou d'algorithmes issus de la théorie des nombres.

Je ne trouvais pas ceci très passionnant ni très en phase avec la pratique moderne de la cryptographie. J'avais en particulier envie de faire programmer des attaques cryptographiques simples aux étudiants, pour bien leur montrer « ce qu'il ne faut pas faire ».

2 TPs de crypto basés sur des *web-services*

À cette fin, j'ai commencé en 2013-2014 à mettre en place une collection de *web-services* hébergés dans le nuage, avec lesquels les étudiants pouvaient interagir via des requêtes HTTP. Pour essayer d'être dans l'air du temps, j'ai tenté d'adopter les standards du web 2.0 : les web-services exposent des API REST et reçoivent des arguments / renvoient des résultats sérialisés en JSON. Un client en python qui simplie l'envoi de requêtes et la réception des résultats est fourni aux étudiants.

L'une des première tâche consistait à récupérer la clef publique du web-service puis à réaliser un chiffrement avec et enfin à renvoyer le résultat au web-service (le tout en utilisant OpenSSL).

Ceci ne paye pas de mine au premier abord. Mais il y a déjà un avantage majeur par rapport à une séance de TP « traditionnelle ». En effet, les étudiants apprécient particulièrement l'aspect « en-ligne », c'est-à-dire que le système leur dit *tout de suite* s'ils ont réussi, et (essaye) de leur expliquer *pourquoi* ils n'ont pas réussi de manière instantanée. Plus précisément, j'ai essayé d'anticiper un certain nombre de manières dont les étudiants pourraient se tromper et j'ai prévu des messages d'erreurs circonstanciés. Mais j'ai appris à mes dépends que les étudiants sont très créatifs en matière d'entrées mal formées.

L'usage de web-services est un scénario réaliste d'utilisation de la cryptographie. De plus, cela ouvre tout un éventail de possibilités pédagogiques :

1) Les étudiants peuvent implanter des protocoles cryptographiques (avec le web-service comme « interlocuteur »).
2) Les web-services peuvent contenir des secrets cryptographiques. Ceci permet aux étudiants d'implanter des attaques pour tenter de les extraire.
3) Cela permet de rendre plus vivantes des définitions de sécurités (potentiellement abstraites) qui font intervenir des « jeux » dans lesquels un *challenger* interagit avec l'adversaire (le web-service jouant le rôle du *challenger*).

Ce système a de multiples autres avantages. Pour les enseignants, il supprime la corvée de la correction des rendus de TPs (le système a été utilisé avec 80 étudiants simultanément, et zéro correction !). La note de TP est fonction du nombre de tâches réalisées. Les étudiants, eux, apprécient de pouvoir y accéder n'importe quand et depuis n'importe où. En effet, pour leur permettre de travailler depuis chez eux et les rendre le plus autonomes possible, les web-services contenaient toute la documentation nécessaire, ainsi que de nombreuses indications sur la manière dont il faut atteindre la solution.

Ceci a pour conséquence de faire basculer les travaux pratiques dans un fonctionnement de *classe inversée* : les séances de TP prévues dans l'emploi du temps de la formation ne servent alors pas tant à venir travailler —puisqu'on peut le faire à un autre moment— qu'à venir soumettre des problèmes ou des questions aux enseignants. J'ai fini par déclarer que la présence y était facultative. Sans surprise, cette formule semble très bien convenir aux étudiants. Bien que ça n'ait pas été sa vocation première, ce système s'est avéré particulièrement utile lors du passage brutal à l'enseignement à distance pendant les périodes de confinement.

Le revers de la médaille, évidemment, c'est que les étudiants peuvent tricher en partageant entre eux le code qui leur sert à accomplir les tâches demandées. Et, de fait,

certains trichent. Dans le fond, j'ai décidé de tolérer ce phénomène. Pour le limiter un peu, j'annonce dès le début du cours qu'ils devront rendre leurs programmes, ou les traces de leur travail, une fois celui-ci terminé. En réalité, c'est une manoeuvre psychologique car je n'ai jamais eu le courage d'examiner ce qui m'a été rendu.

3 TÂCHES CRYPTOGRAPHIQUES

Voici une petite liste non-exhaustive de tâches cryptographiques qu'un tel système peut permettre de réaliser [1].

Utilisation de nombreuses fonctionnalités de la bibliothèque openssl ; implantation d'un protocole d'identification *challenge/response* ; du procole Kerberos ; de l'échange de clef Diffie-Hellman ; de l'échange de clef authentifié par mot de passe SPEKE ; du protocole d'identification de Schnorr ; de la signature de Schnorr ; des chiffrement/signatures RSA, Elgamal, des *ring-signatures*, du protocole de dépense d'argent électronique de Chaum ; mise en place d'un « web-of-trust »de clefs publiques signées ; détection de certificats corrompus cachés au milieu d'autres valides.

Cryptanalyse de générateurs pseudo-aléatoires répandus (Mersenne Twitser, fonction `drand48()`, ...), de mauvais MACs, ...

Théorie des nombres : générer des nombres premiers de taille contrainte, et des racines primitives modulo ces nombres premiers ; produire des certificats de primalité de Pratt ; implémenter des algorithmes de factorisation (méthode ρ, $p-1$, ...) et factoriser des défis de plus en plus durs ; calculer de petits logs discrets avec l'algorithme de Shanks ou la méthode ρ ; implanter les réductions qui montrent que le bit de poids fort du chiffrement RSA ou de l'exponentiation mod p sont *hardcore*, reconstruction du partage de secret de Shamir, ...

Implantation d'attaques : recherche de mot de passe par dictionnaire ; exploitation d'un oracle de padding sur le bourrage CBC ou sur RSA PKCS#1 v1.5 (célèbre attaque de Bleichenbacher) ; multi-collisions de Joux (un programme qui trouve UNE collision sur MD5 est fourni) ; *length-extension attack* sur SHA-256 ; forge de signatures DSA, ECDSA ou de Schnorr en présence de *nonce reuse* ; sur 3 ou 4 tours d'un réseau de Feistel ; exploitation de la malléabilité de RSA ; forge de signature RSA PKCS#1 v1.5 avec $e = 3$ en présence d'un bug dans la procédure de vérification (autre attaque de Bleichenbacher), exploitation d'une fuite d'information dans le chiffrement Elgamal via la résiduosité quadratique, *cold boot attacks* sur RSA ou l'AES,...

Réseaux euclidiens : attaque de Wiener sur RSA a petit exposant secret, cryptanalyse de générateurs linéaires congruentiels tronqués, d'un générateur de Blum-Blum-Schub qui sort trop de bits, de la PRF de Boneh, Halevi et Howgrave-Graham, du chiffrement doublement homomorphe de Smart-Vercauteren, ...

4 VERS UN JEU DE RÔLE EN MODE TEXTE

J'ai conçu cette collection de web-services au fur-et-à-mesure d'un semestre de cours. Cela a pour conséquence qu'ils ont tous des interfaces légèrement différentes, ce qui est une source de confusion pour les étudiants. Pour améliorer cela, j'ai décidé à la fin du semestre une refonte générale de l'ensemble des web-services. L'objectif principal était de leur donner des interfaces prévisibles et uniformes. Ils ont tous été rassemblés pour former un seul « TP continu »qui s'étend sur toute la durée du cours, c'est-à-dire généralement un semestre.

Sur le plan technique, j'avais utilisé le *micro web-framework* Bottle [1] pour construire tous mes web-services. Ils étaient *stateless* (rien n'était sauvegardé côté serveur). Cela avait l'avantage de la simplicité, mais s'est finalement avéré trop restrictif (impossible de mettre en place une PKI par exemple). Finalement, j'ai décidé de passer à un véritable *web-framework* avec un support natif pour l'utilisation d'une base de données SQL. J'ai choisi django [3] qui était l'option principale, et je ne l'ai jamais regretté.

Comme j'en avais alors la possibilité technique, et aussi par goût, j'ai choisi de plaquer un « scénario » (largement improvisé) sur le tout. Le TP se présente alors comme une sorte de jeu de rôle dans univers virtuel. Les étudiants reçoivent pour mission d'infiltrer un système d'exploitation inconnu nommé `UGLIX` (que simule l'application web). Au fur-et-à-mesure qu'ils réalisent des tâches, ils parviennent à usurper les comptes de différents utilisateurs et à progresser dans l' « histoire ». Le principe de ce scénario a été éhontément plagié dans l'excellent concours de programmation organisé en 2006 par l'*International Conference on Functional Programming* [7], [10], avec des éléments tirés des jeux vidéos Uplink (Introversion Software, 2001) ou encore Chrono Trigger (Square, 1995).

Les étudiants peuvent explorer le système de fichier de l'OS inconnu en effectuant des requêtes HTTP : des pointeurs leur sont donnés vers `/bin` et `/usr/share/doc`. Une requête sur un répertoire renvoie la liste des fichiers qu'il contient. Une requête sur un fichier renvoie son contenu. Une requête sur un programme déclenche son exécution.

Bien sûrs certains répertoires leur sont inaccessibles (erreur HTTP 403), tels que `/root`, `/home/machintruc`, etc. Le programme `/bin/login`, invoqué avec les bons arguments (`username` et `password`), place un *Cookie* qui donne accès à son propre repertoire personnel. Lire un fichier se fait avec une requête HTTP GET (c'est idempotent), invoquer un programme nécessite une requête HTTP POST (car cela peut modifier l'état du système).

Initialement, les étudiants n'ont accès qu'au compte invité. Là, ils trouvent un moyen d'accéder à un autre compte, celui d'un employé d'un *helpdesk* cryptographique foutraque aux clients mécontents... Une fois que suffisamment de tickets ont été fermés, ils reçoit une invitation pour des travailler sur des missions plus... sensibles, au service des autorités, etc.

Certaines tâches forcent les étudiants à collaborer entre eux, par exemple pour reconstruire un secret partagé, ou bien pour construire entre eux un « *web-of-trust* » en signant mutuellement leurs clefs publiques.

Le système `UGLIX` est similaire aux *UNIX* (ça a l'avantage que les étudiants le trouvent familier). Il est multi-utilisateurs et offre un support natif du courrier électronique. Vu que les étudiants n'ont grosso-modo accès qu'au système de fichiers, il copie sur Plan9 [6] le concept de « tout est un fichier » poussé au maximum. Par exemple, la boite

1. Si vous avez d'autres idées, partagez-les avec moi !

de réception d'un utilisateur est en fait un répertoire dans lequel les messages individuels apparaissent comme des fichiers. L'annuaire de la PKI est un répertoire où chaque utilisateur correspond à un fichier, etc.

Les étudiants peuvent explorer presque tout le « système de fichiers » dès le début. C'est un choix délibéré : ils ont très rapidement accès à (presque) toutes les tâches, et ils peuvent donc choisir de les faire dans l'ordre qui leur convient et de ne pas faire celles qui ne leurs conviennent pas. Ils apprécient beaucoup la possibilité de pouvoir basculer d'une tâche à une autre lorsqu'ils sont coincés.

Un client python qui simplifie l'envoi des requêtes, la gestion des *cookies* de session, etc. est fourni aux étudiants. En plus, il me permet d'exécuter du code arbitraire sur leur machine via un mécanisme légèrement obfusqué ! Outre l'intéressante discussion sur la sécurité de leurs *autres* applications que cela permettait, je me suis servi de cette possibilité pour ajouter une bande son et des effets sonores.

Du point de vue des étudiants, l'accès à l'application web (donc au « système d'exploitation » UGLIX) se fait dans un interpréteur python ou un *notebook* Jupyter. Ils soumettent des requêtes et affichent ou traitent les résultats. Une session peut donc ressembler à ça :

```
>>> import client
>>> c = client.Connection()
>>> print(c.get('/'))
[...]

>>> print(c.get('/home/guest'))
# ServerError('ERREUR 403, Session not opened.')

>>> print(c.get('/bin/login'))
USAGE: /bin/login [user] [password]

>>> print(c.post('/bin/login', user='guest', password='guest'))
PAP login successful. Access to /home/guest granted.
[ You have new mail ]

>>> print(c.get('/home/guest'))
[...]
/home/guest/ISEC.bin
/home/guest/README.txt

>>> print(c.get('/home/guest/INBOX'))
----------------------------------------------------------------------
INBOX:

Message | Read | Sender                       | Subject
--------+------+------------------------------+-------------------------
   9    |   N  | shalondaclarice_bo@prai...   | Develop more size day by day
   8    |   N  | annalise3586453.2fba9@p...   | Get Ready For A New Company
   7    |   Y  | jmelendez@maine207.org       | Can you have all night long fun?

>>> print(c.get('/home/guest/INBOX/9'))
----------------------------------------------------------------------
Message #9.

Date: 15-01-21895 04:30:42
X-UGLIX-Spam: GUEST.MAIL:13:1@162622|3bd2d2a59a3b4546762631a5f30a134d
From: shalondaclarice_bo@prairieinet.net
Subject: Develop more size day by day

http://trio.nngoshdz.ru/ Are you ready to gratify your beloved one at night?

>>> c.get('/bin/login/CHAP')
{'challenge': '005fa36f6292475a9e49ac1fe6e96227'}        # ceci est un dict

>>> c.get('/bin/key-management/guest/pk')
# ServerError: ERREUR 404, guest hasn't uploaded a PK
```

(Eh oui, même dans UGLIX il y a du *spam* dans les boites mail).

Les étudiants ont donc un accès fondamentalement *non-interactif* au système : ils soumettent des requêtes et observent le résultat. L'application web ne peut pas leur envoyer quoi que ce soit sans avoir été sollicitée au préalable. Ceci est un inconvénient du point de vue du *gameplay* : l'expérience n'est pas très immersive et la différence avec un « vrai » OS est trop flagrante. Certains étudiants ont développé des sortes de *shell* qui émulent les commandes `ls`, `cd`, etc. pour compenser.

Mais ce fonctionnement a aussi un énorme avantage. Les requêtes peuvent renvoyer du texte ou bien un objet python arbitraire (tant qu'il est sérialisable en JSON). Ainsi, des requêtes peuvent renvoyer des listes, des tuples, des dictionnaires... Ceci facilite la programmation des interactions avec le système :

```
>>> tmp = c.get('/bin/login/CHAP')
>>> plaintext = 'toto-' + tmp['challenge']
>>> r = openssl.encrypt(plaintext, password)
>>> print(t.post('/bin/login/CHAP', username='toto', response=r)
CHAP login successful. Access to /home/toto granted.
```

Pour leur faciliter la vie, un module est mis à leur disposition, qui réalise le chiffrement symétrique seulement (en invoquant le programme `openssl` présent sur leur machine). Ils doivent l'étendre pour y ajouter les autres fonctionnalités.

Concrètement, la validation d'une tâche donne un *flag*, c'est-à-dire une chaîne de caractères opaque telle que :

```
GUEST.MAIL:13:1@162622|3bd2d2a59a3b4546762631a5f30a134d
```

Les étudiants doivent les saisir dans une interface web pour marquer les points correspondants. Il s'agit donc, peu ou prou, d'un « *capture-the-flag* » maintenant devenu standard dans le domaine des formations à la cybersécurité. Pour chaque *flag* soumis (donc chaque tâche réalisée), l'interface web indique le nombre d'autres étudiants qui l'ont accomplie. Ceci donne aux étudiants une indication utile sur leur progression (la liste des tâches à réaliser n'est pas révélée à l'avance pour entretenir un peu le suspense).

Les *flags* contiennent une description de la tâche accomplie, le numéro du compte qui l'a réalisée, le tout accompagné d'un MAC. Une des dernières tâche révèle une capture d'écran montrant le code de vérification des flags. La clef du MAC est inopportunément masquée par une fenêtre mal placée, mais les deux pixels du bas des 32 caractères hexadécimaux qui la composent est, lui, visible. Cela permet aux étudiants les plus motivés de « hacker le jeu » et de la retrouver en $\approx 2^{40}$ essais.

À très peu d'exceptions près, les étudiants plébiscitent le choix qui consiste à présenter le tout comme une sorte de jeu vidéo rétro avec une vague intrigue et une ambiance *geek*. Ils sont nombreux à dire qu'ils apprécient le côté ludique, et « *plus concret* »qu'un TP classique (sujet / liste de questions / compte-rendu). Un étudiant a dit : « *on ne travaille pas, on s'amuse* », un autre : « *Le fait de progresser dans une histoire m'a beaucoup motivé.* »

J'ai utilisé cette application pendant 7 ans. Elle se compose de $\approx$ 30 000 lignes de code python (1Mo) ainsi que 370Ko de documentation : c'est le plus gros programme que j'ai écrit. Je ne suis pas capable de quantifier le temps que cela m'a pris. Le gros du développement a eu lieu lors de sa mise en place. Relancer l'application pour une nouvelle promotion d'étudiants ne prend que quelques minutes.

Le code source n'est pas public car cela révélerait largement la solution des problèmes aux étudiants. En effet, entre un tiers et un quart du code est constitué de tests unitaires qui vérifient le comportement de l'application tant sur de mauvaises entrées que sur les bonnes. L'existence de cette suite de tests a été un élément clef du bon fonctionnement de l'ensemble. Mais je suis prêt à tout partager avec des collègues !

5 Vers un système plus interactif

En 2019, j'ai changé d'établissement. Le moment était venu d'écrire une nouvelle application de TP, toujours basée sur la simulation d'un système d'exploitation à explorer.

Pour rendre l'expérience plus immersive et plus interactive, les étudiants se connectaient à cette nouvelle version via le protocole telnet ou bien via une version maison du protocole ssh plutôt que via des requêtes HTTP. L'application simule alors un *shell* sur un système UNIX-like qui s'appelle encore `UGLIX`. Ce *shell* offre la TAB-complétion et certains étudiants ont cru pendant plusieurs semaines être connecté à un « vrai » système d'exploitation...

Cette fois, un client `telnet` quelconque est suffisant. Le tout est plus interactif car le serveur peut réagir instantanément à l'appui sur une touche du clavier, et transmettre des caractères aux terminaux des étudiants à n'importe quel moment : on peut leur envoyer des « notifications », on peut jouer des séquences animées (en mode texte), etc.

Cette nouvelle application pour des étudiants de M2 (26 000 lignes de python) n'a servi qu'un an. Elle avait de gros inconvénients : simuler le *shell* d'un système UNIX complet, avec son système de fichier, son système de permission, ses signaux, etc. n'est pas facile ! L'application avait des bugs difficiles à corriger. En outre, le caractère plus fluide et interactif de son utilisation par les étudiants, que je percevais comme un avantage, a pour conséquence que l'échange de données entre client et serveur est moins structuré. Il n'y a plus de requêtes et de réponses, juste un flux d'octets. Ceci rend l'application difficile à tester de manière automatique, alors que c'est très facile avec le système requête / réponse (le code de test peut simplement émettre des requêtes comme un utilisateur normal).

Reconnaissant que j'avais été victime du classique « effet du 2ème système » [2], j'ai donc renoncé à simuler un OS complet et j'ai mis au rancart cette deuxième tentative.

Pour garder un côté ludique, immersif et interactif, j'ai écrit une 3ème application en 2020 (https://crypta.sfpn.net). Elle est aussi entièrement en-ligne, interactive (`telnet` ou `ssh-like`) et se présente sous la forme d'un jeu d'aventure en mode texte (une autre idée éhontément volée à [10]). Les étudiants saisissent des commandes en « langage naturel » (aller à gauche, entrer, monter, prendre tel objet, utiliser tel autre, etc.). Le système affiche la description des lieux et des objets. L'utilisation d'un objet déclenche l'exécution d'une fonction arbitraire côté serveur. Sur le plan technique, ceci s'inspire du jeu de fiction interactive écrit en 1975–76 (en FORTRAN) par Will Crowther et Don Woods [4], et de la version en C réécrite par Knuth [5]. L'analyseur syntaxique très rudimentaire a été repris tel quel.

L'action se déroule sur le campus de Jussieu la faculté des sciences de Sorbonne Université. Les étudiants doivent réussir à sortir de la fac, déserte car évacuée à cause d'une épidémie... (ceci est bien commode pour justifier que le protagoniste ne rencontre personne dans le monde virtuel). Pour cela, il leur faudra bien sûr accomplir un certain nombre de tâches de nature cryptographique. En plus, un mécanisme mystérieux censure une partie de leurs perceptions. Après avoir « hacké le jeu » et gagné la possibilité de se téléporter dans des pièces qui leurs étaient jusqu'alors inaccessibles, les étudiants découvrent qu'une partie de leurs enseignants a été massacrée et que les évènements qui se sont déroulés sur le campus ont une tournure plus sombre que ce qui était suggéré jusque-là.

Ce système (fiction interactive en mode texte) réalise à mes yeux un compromis. Les étudiants sont immédiatement placés dans un « jeu », ce qui est un choix assumé. L'inconvénient majeur de ce système est qu'il est moins simple pour les étudiants d'automatiser leurs interactions avec l'application. Une solution potentielle consiste à leur donner la possibilité de faire des *Remote Procedure Calls* sur le serveur.

Les interactions entre les utilisateurs et le système sont cependant assez structurées, et le « monde » dans lequel se déroule l'action est assez simple (les étudiants explorent des lieux et interagissent avec des objets). Programmer l'application n'a pas posé de problème majeur et le tout redevient (assez) facilement testable. L'application a 17 500 lignes de code python. J'estime à 15 jours complets le temps de travail nécessaire à son écriture. Elle sert cette année pour la 3ème fois consécutive. Les doctorants de mon équipe de recherche ont été volontarisés pour la tester (c'est indispensable).

Sur le plan technique, l'application repose sur le *framework* twisted [8] et utilise massivement les *coroutines* disponibles depuis python 3.5.

6 Conclusion

Je recommande fortement la mise en place de plateformes en ligne pour l'enseignement de la cryptographie, au vu des bénéfices et des possibilités décrites ci-dessus.

L'expertise technique que ceci semble nécessiter peut s'acquérir « en faisant » (ça a été mon cas). L'investissement en temps que cela représente n'est pas négligeable mais peut en fait s'amortir sur plusieurs années, et il n'y a plus de rendus de TP à corriger. Je conseille fortement à ceux qui voudraient se lancer d'utiliser django.

Choisir de présenter les TPs comme un « jeu » est un choix qui n'a pas que des avantages ; au final le principal inconvénient est qu'on passe beaucoup de temps à décrire un monde virtuel avec plus ou moins de talent et à se demander comment intégrer telle ou telle tâche cryptographique dans le « scénario ». De plus, rendre le système à la fois interactif et facile à utiliser dans des scripts est non-trivial.

Références

[1] Bottle : Python web framework. https://bottlepy.org.

[2] Frederick P. Brooks. *The mythical man-month – Essays on Software-Engineering*. Addison-Wesley, 1975.

[3] django : The web framework for perfectionists with deadlines. https://www.djangoproject.com.

[4] Dennis Jerz. Somewhere nearby is colossal cave : Examining will crowther's original "adventure" in code and in kentucky. *Digit. Humanit. Q.*, 1(2), 2007.

[5] Donald E. Knuth. Adventure, 1998. http://www.literateprogramming.com/adventure.pdf.

[6] Rob Pike, David L. Presotto, Sean Dorward, Bob Flandrena, Ken Thompson, Howard Trickey, and Phil Winterbottom. Plan 9 from bell labs. *Comput. Syst.*, 8(2) :221–254, 1995.

[7] John H. Reppy and Julia Lawall, editors. *Proceedings of the 11th ACM SIGPLAN International Conference on Functional Programming, ICFP 2006, Portland, Oregon, USA, September 16-21, 2006*. ACM, 2006.

[8] Twisted : An event-driven networking engine. https://twisted.org/.

[9] Damien Vergnaud. *Exercices et problèmes de cryptographie — 3ème édition*. Dunod, 2018.

[10] Tom Murphy VII, Daniel Spoonhower, Chris Casinghino, , Daniel R. Licata, Karl Crary, and Robert Harper. The cult of the bound variable : The 9th annual icfp programming contest. Technical report CMU-CS-06-163, 2006. http://www.boundvariable.org/press/tr-06-163.pdf.