Contents

Démonstration/TP attaque par canaux auxiliaires

Organisation du dossier

- `arduino` : contient les programmes à mettre sur les cartes Arduino
  - `EEPROMverifyPIN` : l'Arduino possède deux commandes, une pour mettre un PIN et une pour vérifier le PIN. Lorsque le PIN est mis à jour, il est stocké dans la mémoire EEPROM de la carte ce qui permet, par exemple, à l'enseignant de mettre un PIN que l'étudiant devra trouver.
  - `EEPROMverifyPINcstTime` : dans ce programme, la vérification de PIN est réalisée en temps constant.
  - `EEPROMverifyRandomPIN` : dans ce programme, la carte va générer à chaque démarrage un PIN aléatoire qu'elle va stocker en mémoire. La seule façon de retrouver le PIN est donc d'attaquer la carte.
  - `NoTrigEEPROMverifyPIN` : dans ce programme, aucun signal électrique n'est levé/baissé avant/après la vérification du PIN.
  - `softwareSerial` : ce programme permet d'utiliser une carte Arduino comme intermédiaire pour attaquer une seconde carte qui ne fournit pas de signal électrique autour de la vérification de PIN.
- `canaux_auxiliaires.pdf` : présentation
- `montages` : contient les différents montages possibles pour réaliser les attaques.
- `scripts` : contient les scripts pour reproduire les attaques
  - `0_timing_diff.py` : script qui permet d'observer une différence de temps lors de la vérification d'un code PIN avec différents digits à la bonne valeur.
  - `1_timing_attack.py` : script qui retrouve le PIN mis dans la carte en réalisant une attaque basée sur la variation de temps d'exécution de la vérification du code PIN.

- `2_power_analysis.py` : script qui permet d'observer une fuite d'information en calculant la différence des moyennes de consommation entre le bon PIN et le mauvais PIN.
- `board_utils.py` : fichier Python instanciant une classe qui permet de communiquer avec une carte Arduino qui réalise la vérification de code PIN.
- `scope_utils.py` : fichier Python qui définit des fonctions ou des classes pour se connecter à un oscilloscope, récupérer des traces et les afficher.

A préparer

Développement pour communiquer avec l'oscilloscope

Afin de pouvoir faire fonctionner les scripts, il faut pouvoir communiquer avec un oscilloscope et récupérer la courbe qu'il affiche. Pour cela il faut développer dans le fichier `scope_util.py` une classe scope qui permet de récupérer cette trace via une fonction qui s'appelle `get_waveform()` et qui prend en paramètre le canal sur lequel récupérer la courbe.

Bien évidemment, il est possible d'utiliser une autre API, celle si étant celle qui correspond à mes propres bibliothèques. Il faut juste adapter les scripts en conséquence.

Modification de la carte Arduino pour réaliser l'analyse en consommation

Résistance au niveau de la masse Afin de pouvoir observer le courant tiré par le composant il faut souder une résistance entre la masse du composant et la masse de la carte. Sur les Arduino Uno R3, le composant est un ATMEGA328P. Les pins de la masses sont les 8 et 22.

Il faut donc retirer ces pins du socle, en le retournant ou en les découpant par exemple. Ensuite, il faut souder une résistance (1 Ohm par exemple) à l'une de ces pins puis de brancher la résistance à la masse de la carte.

La seconde résistance (celle qui est soudée derrière le composant) sert à rattraper une erreur que j'ai faite. Vous avez une résistance déjà présente à cet endroit sur la carte.

(Je sais, ce n'est pas très propre !)

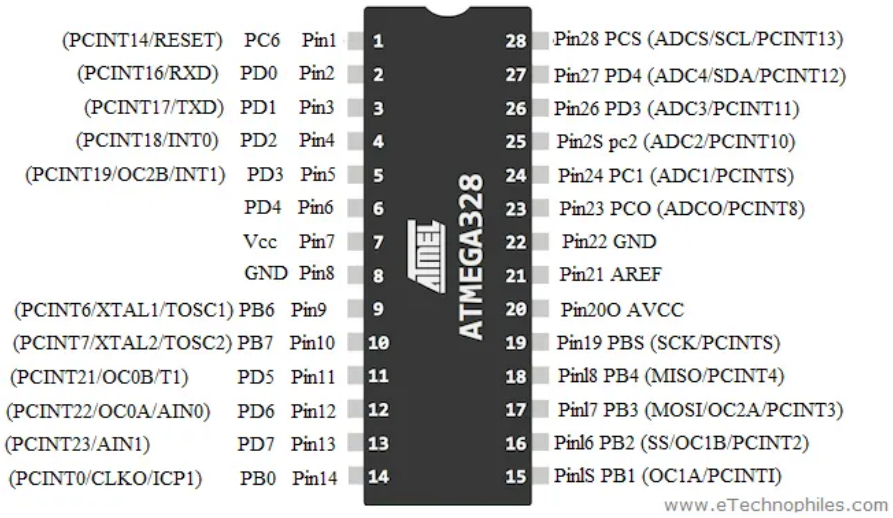

Figure 1: ATMEGA328P pinout

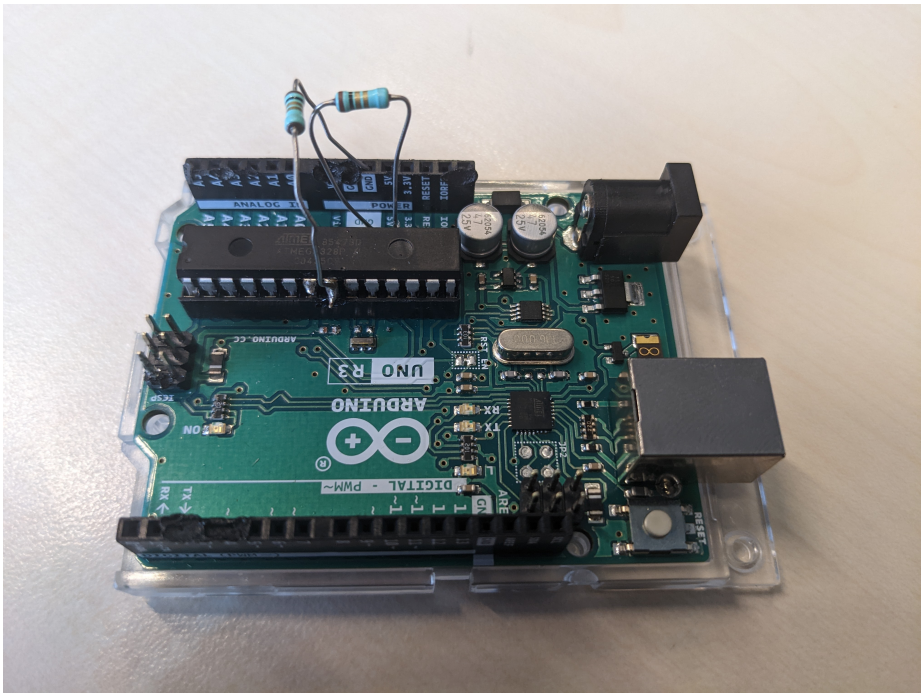

Figure 2: Carte Arduino modifiée avec une résistance en sortie de la masse

Retirer les capacités de découplage Une intervention intéressante à faire (je n'ai pas testé sans le faire) consiste à retirer les capacités de découplages. Ces capacités servent à filtrer les variation de l'alimentation afin d'éviter d'abîmer le circuit. Dans notre cas, ces capacités peuvent filtrer des signaux qui fuitent de l'information, il est donc intéressant de les retirer.

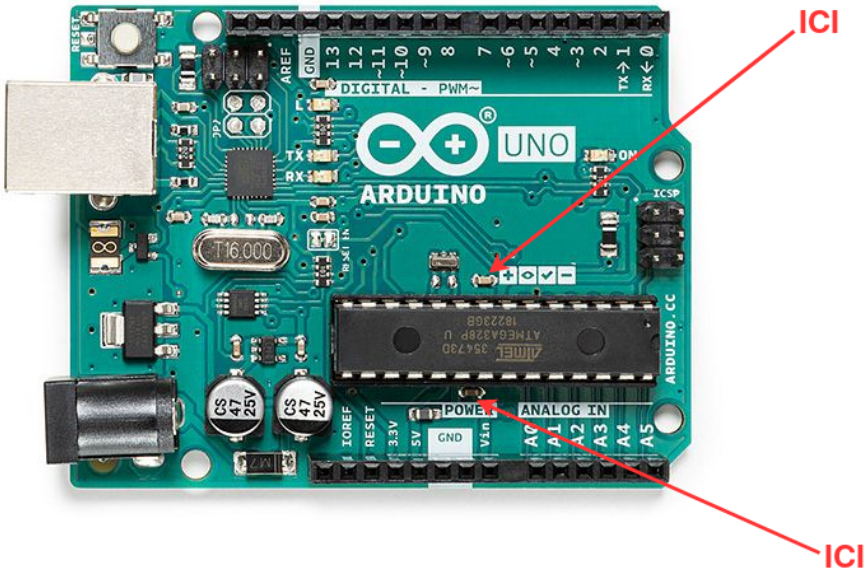

Figure 3: ATMEGA328P capacités de découplage

Montages

Montage simple

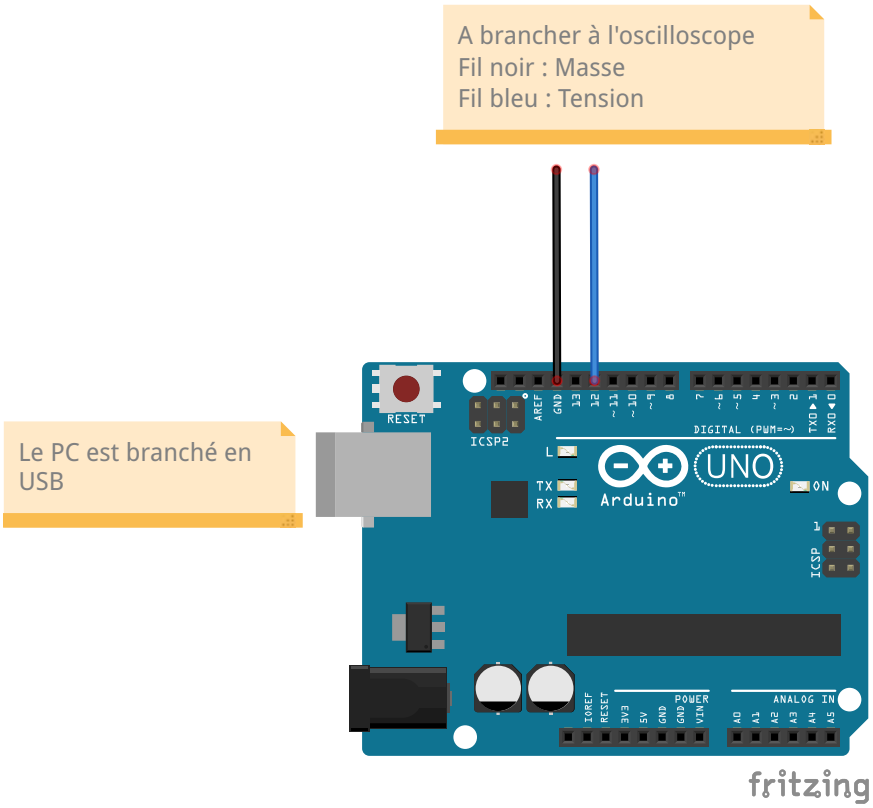

Figure 4: Montage simple pour l'analyse temporelle

Montage avec une carte attaquante

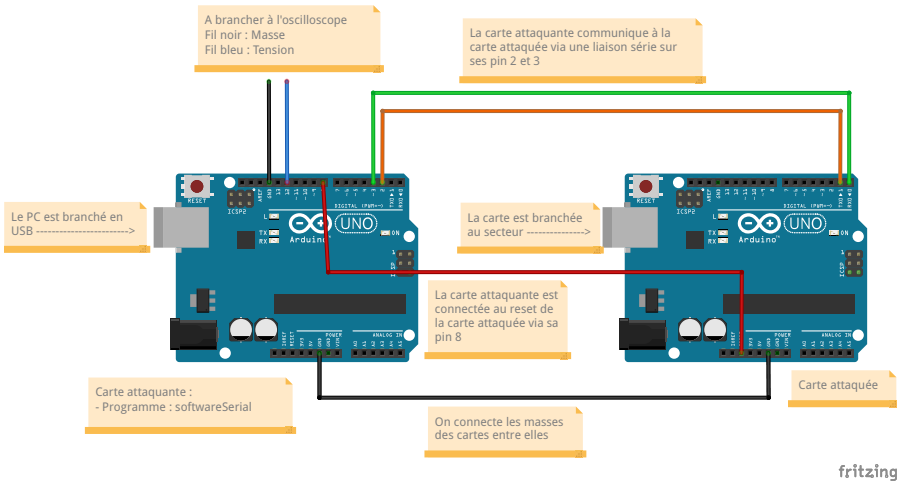

Figure 5: Montage avec une carte attaquante

Attaque temporelle avec différence des moyennes

Si jamais l'oscilloscope utilisé ne permet pas de réaliser la distinction en utilisant la différence des moyennes sur la consommation (car la fréquence d'échantillonage est trop faible par exemple), il est possible de la faire en utilisant le temps (normalement, je n'ai pas essayé).

De manière très similaire, dans la configuration avec deux cartes, le temps entre le front montant et le front descendant du signal de trigger envoyé par la carte attaquante est bruité.

On a quelque chose de la forme : $t_{obs} = F(d) + B$

et donc on moyenne, le temps observé est $\tilde{t}_{obs} = \tilde{F}(d) + C$

et donc en faisant la différence des moyennes on devrait distinguer le cas où d est le bon PIN du cas où c'est le mauvais PIN.