buy it, use it, break it ... fix it : caml crush, un proxy pkcs11 filtrant

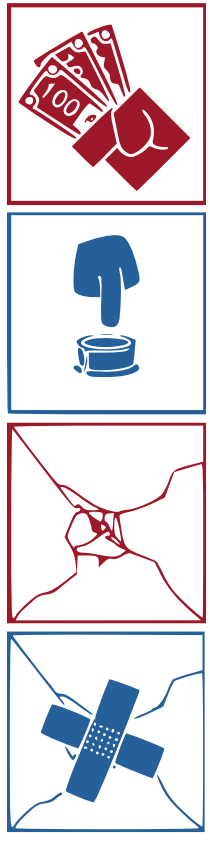
sstic 2014

R. BENADJILA, T. CALDERON, M. DAUBIGNARD

Agence **N**ationale de la
Sécurité des **S**ystèmes
d'**I**nformation

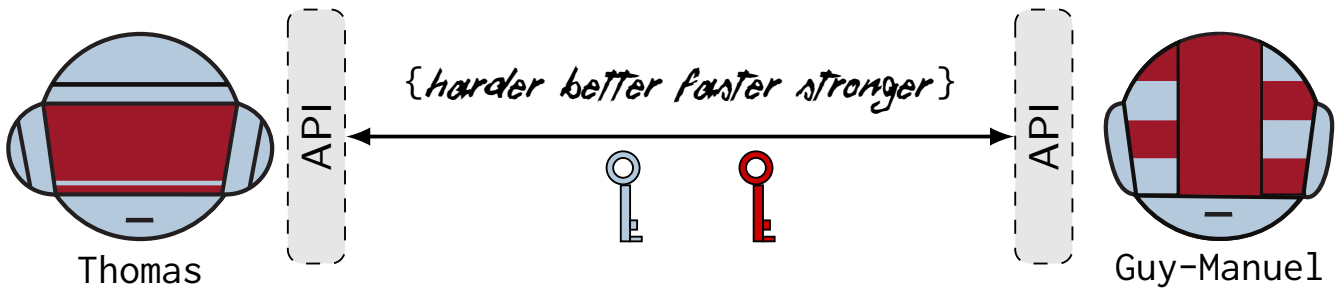
5 juin 2014

ANSSI



Contexte

- Thomas et Guy-Manuel veulent :
  - Échanger des secrets
  - Réaliser des opérations cryptographiques
  - Partager des clés et les utiliser
  - Utiliser une interface standardisée



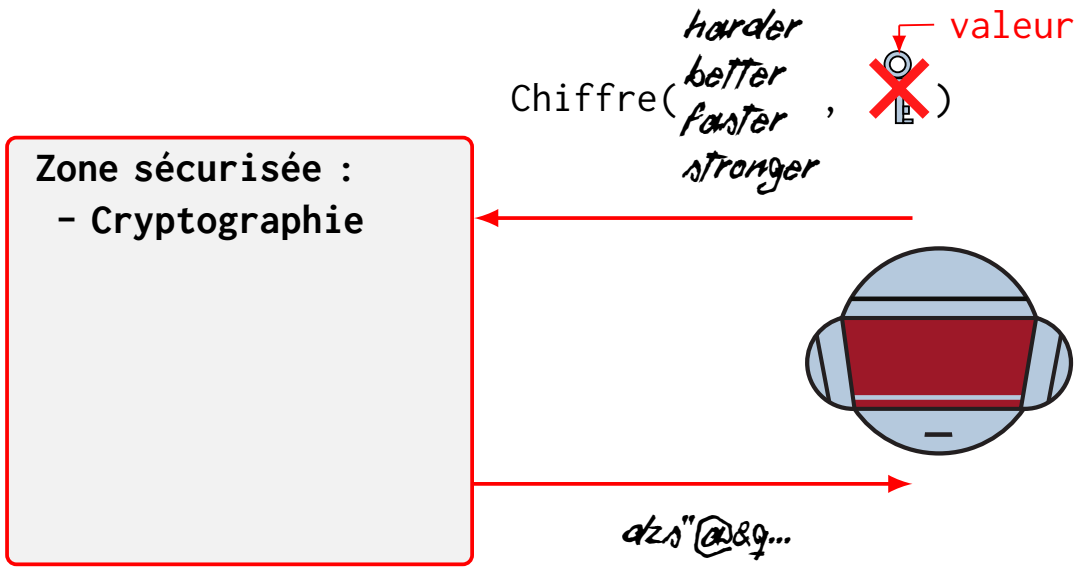
- Éviter les vulnérabilités et faiblesses.
- Caml Crush : une solution robuste et flexible de proxification de l'API PKCS#11.

Vous avez dit API de sécurité ?

- Une API de sécurité est une interface de programmation qui permet de réaliser des opérations cryptographiques et des opérations de gestion des clés.

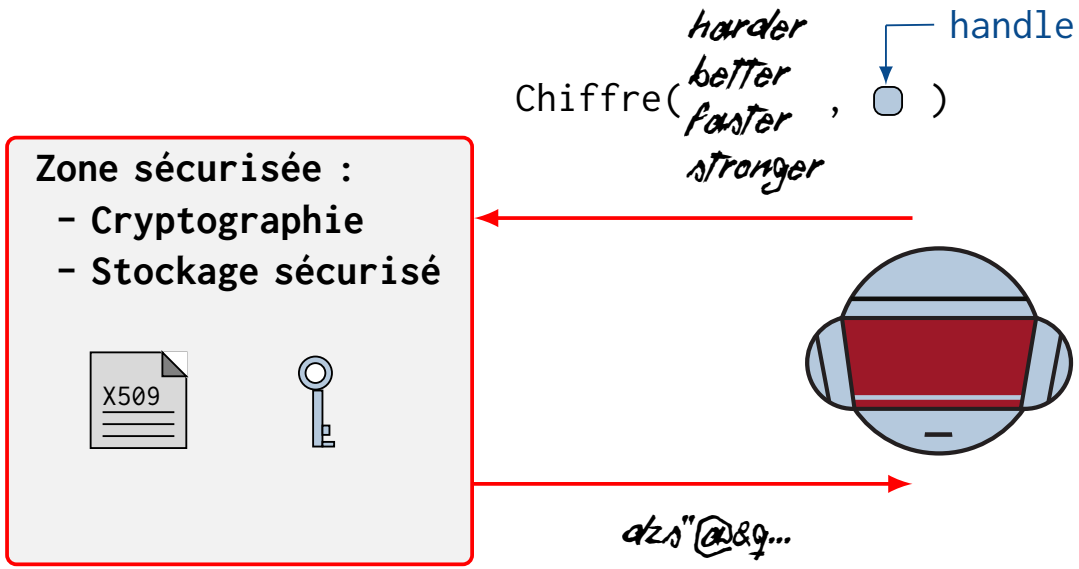
Vous avez dit API de sécurité ?

- Une API de sécurité est une interface de programmation qui permet de réaliser des opérations cryptographiques et des opérations de gestion des clés.
- Les valeurs de clés ne sont pas censées être manipulées.



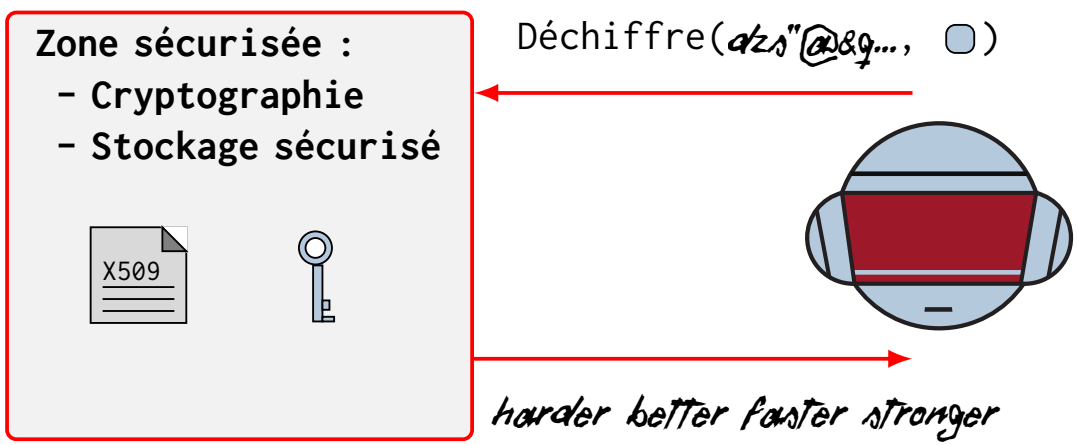
Vous avez dit API de sécurité ?

- Une API de sécurité est une interface de programmation qui permet de réaliser des opérations cryptographiques et des opérations de gestion des clés.
- Les valeurs de clés ne sont pas censées être manipulées.
- Les clés sont utilisées via des références («handles»).



Vous avez dit API de sécurité ?

- Une API de sécurité est une interface de programmation qui permet de réaliser des opérations cryptographiques et des opérations de gestion des clés.
- Les valeurs de clés ne sont pas censées être manipulées.
- Les clés sont utilisées via des références («handles»).

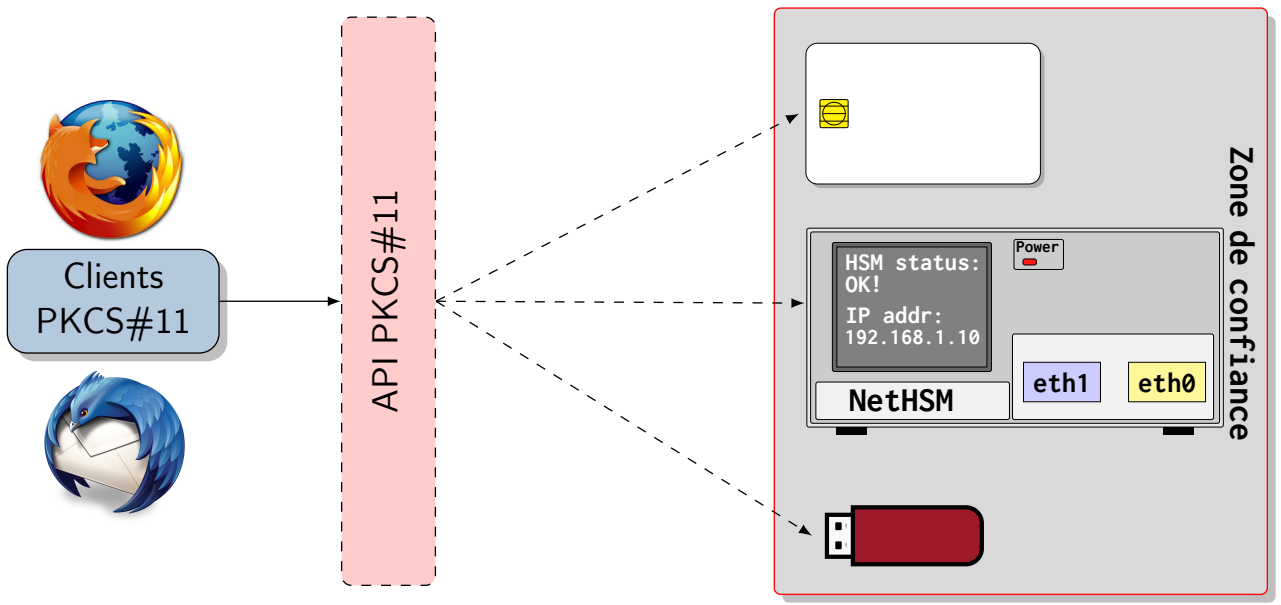


Le standard PKCS#11

- PKCS#11 = sous-ensemble de Public Key Cryptography Standards produit par RSA labs. L'API est aussi appelée **Cryptoki** (Cryptographic Token Interface).

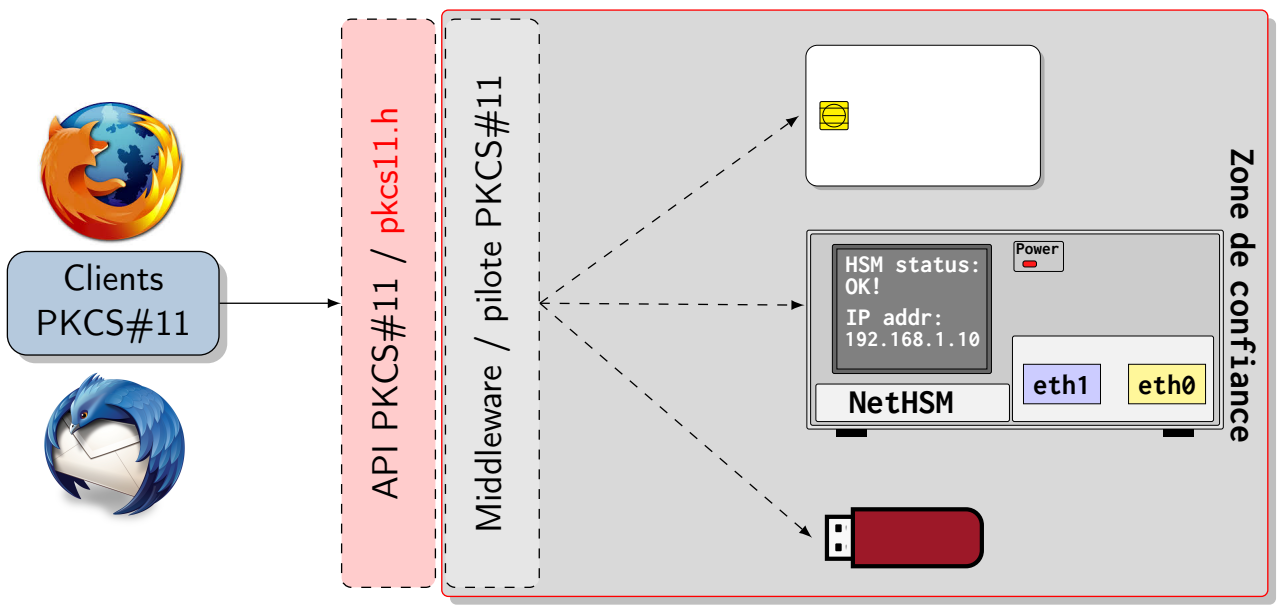
Le standard PKCS#11

- PKCS#11 = sous-ensemble de Public Key Cryptography Standards produit par RSA labs. L'API est aussi appelée **Cryptoki** (Cryptographic Token Interface).
- Pourquoi un standard ?
  - Limiter les interfaces propriétaires des terminaux qui implantent de la cryptographie.
  - Unicité et portabilité du code.



PKCS#11 et la portabilité

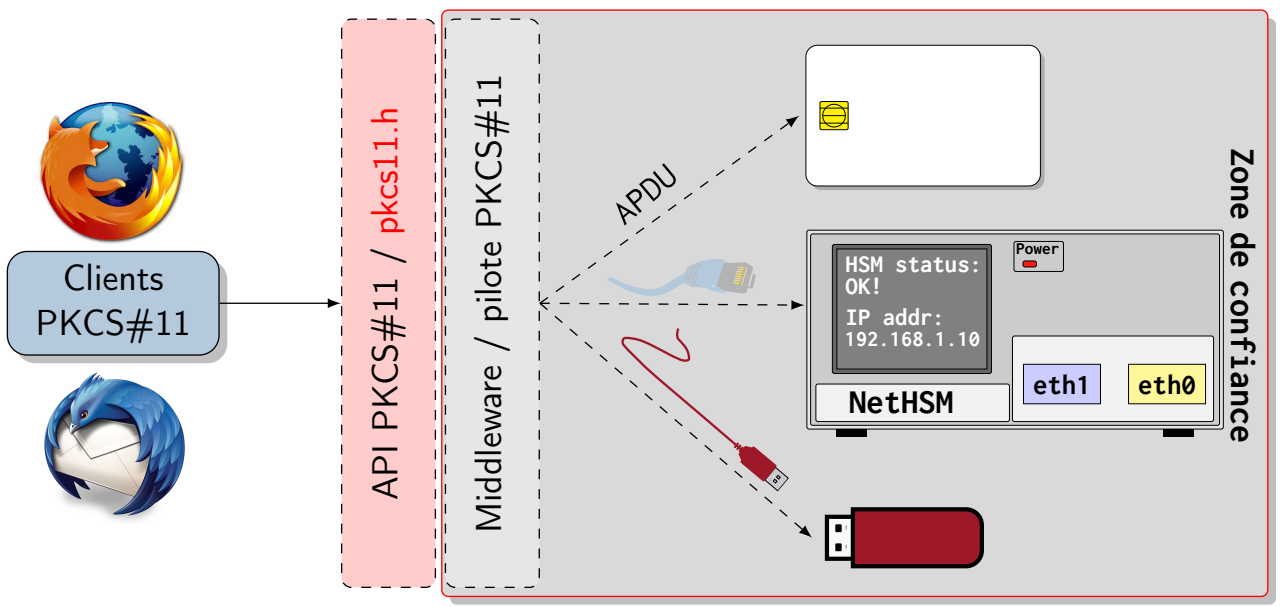
- Comment cacher la couche propriétaire ?
  - RSA labs fourni pkcs11.h
  - Les fabricants fournissent une bibliothèque partagée («middleware»)



PKCS#11 et la portabilité

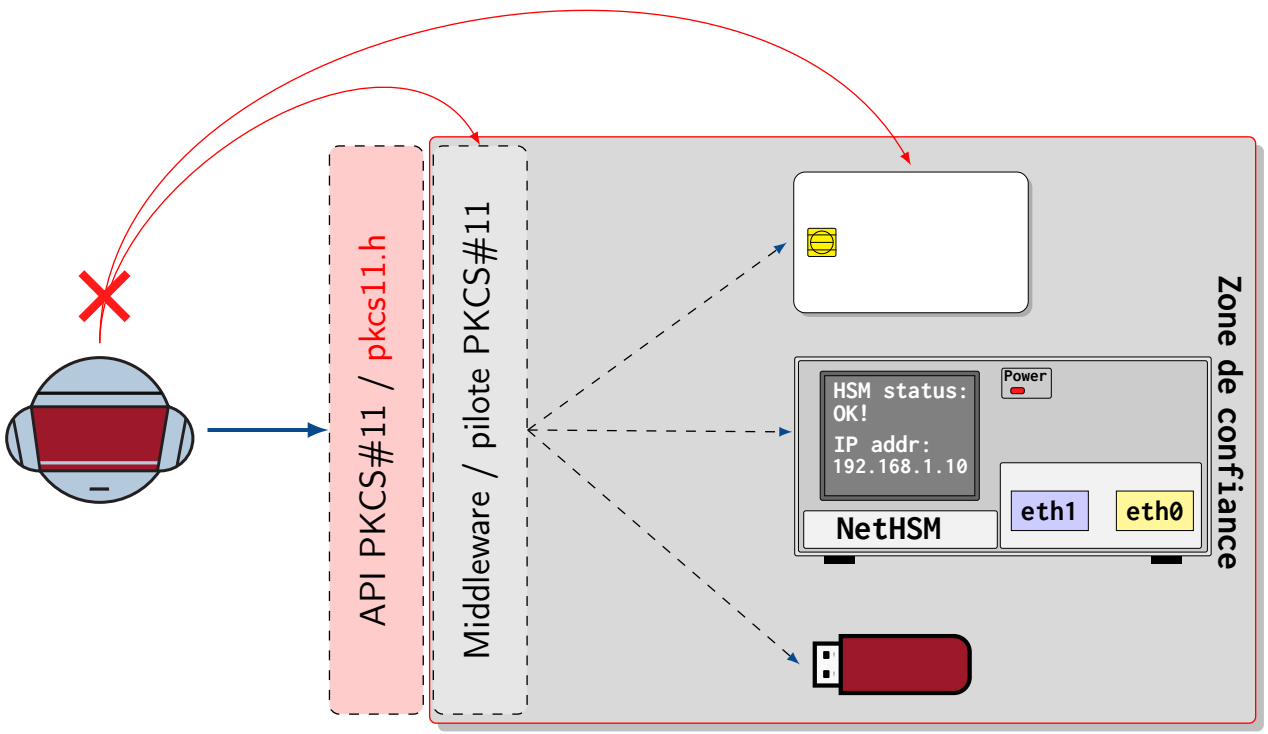
- Comment cacher la couche propriétaire ?
  - RSA labs fourni pkcs11.h
  - Les fabricants fournissent une bibliothèque partagée («middleware»)

- La bibliothèque gère le matériel :
  - Envoi d'APDU (via USB, port série ...) aux cartes à puce
  - Envoi de trames réseau aux NetHSM
  - Envoi de trames USB aux dongles



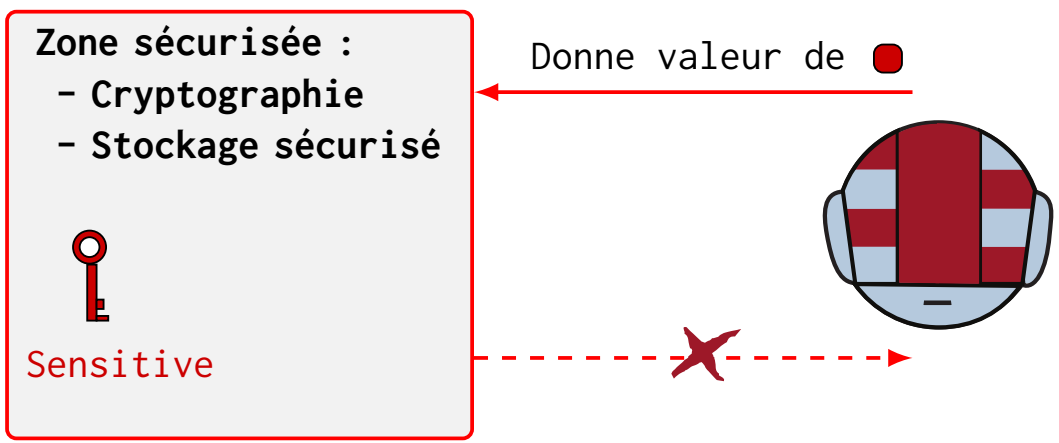
Modèle d'attaquant

- L'attaquant n'effectue que des requêtes PKCS#11.



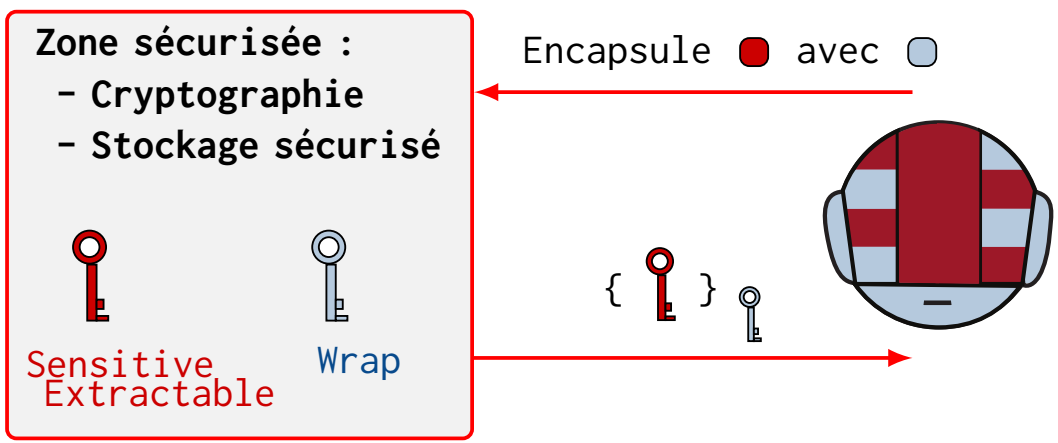
Confidentialité des clés

- PKCS#11 traduit l'usage des clés par des attributs.
  Ex. : Encrypt, Decrypt, Sensitive, . . .
- L'API de sécurité ne doit pas laisser sortir en clair les valeurs de clés confidentielles (Sensitive).



Mécanisme d'encapsulation (Wrap)

- L'API permet l'export (encapsulation ou Wrap) de clés.
- Les attributs qui gèrent cela :
  - Wrap/Unwrap = la clé peut en exporter/importer d'autres
  - Extractable = la clé peut être exportée

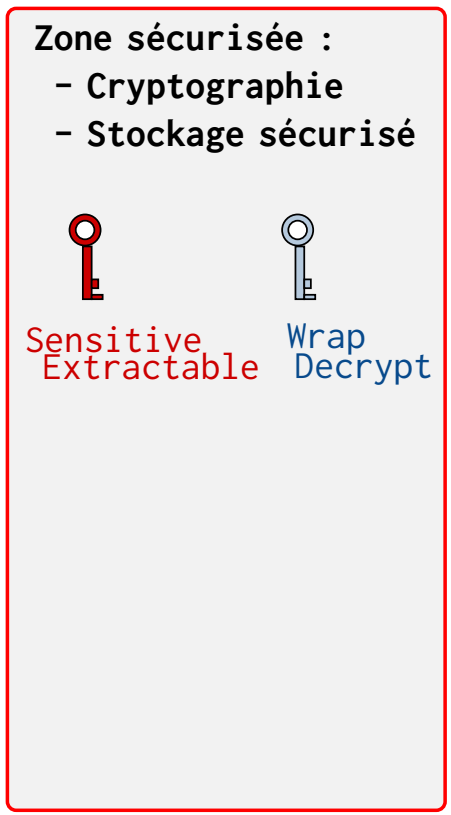


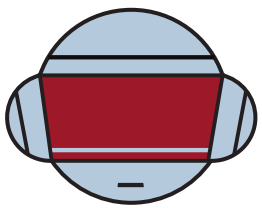
Attaque Wrap/Decrypt

- L'attaque utilise la confusion que fait l'API entre fonctions d'encapsulation et de chiffrement.

Attaque Wrap/Decrypt

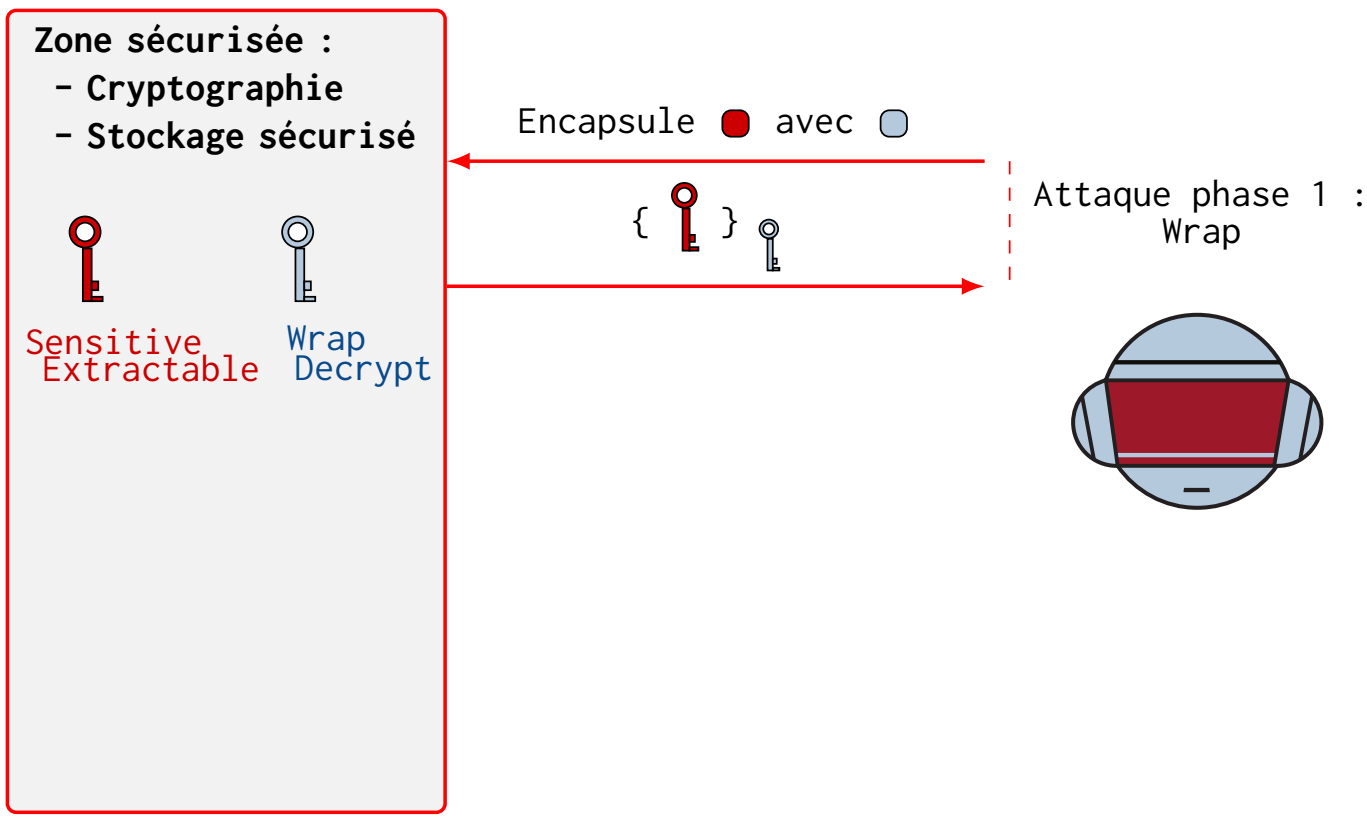
- L'attaque utilise la confusion que fait l'API entre fonctions d'encapsulation et de chiffrement.





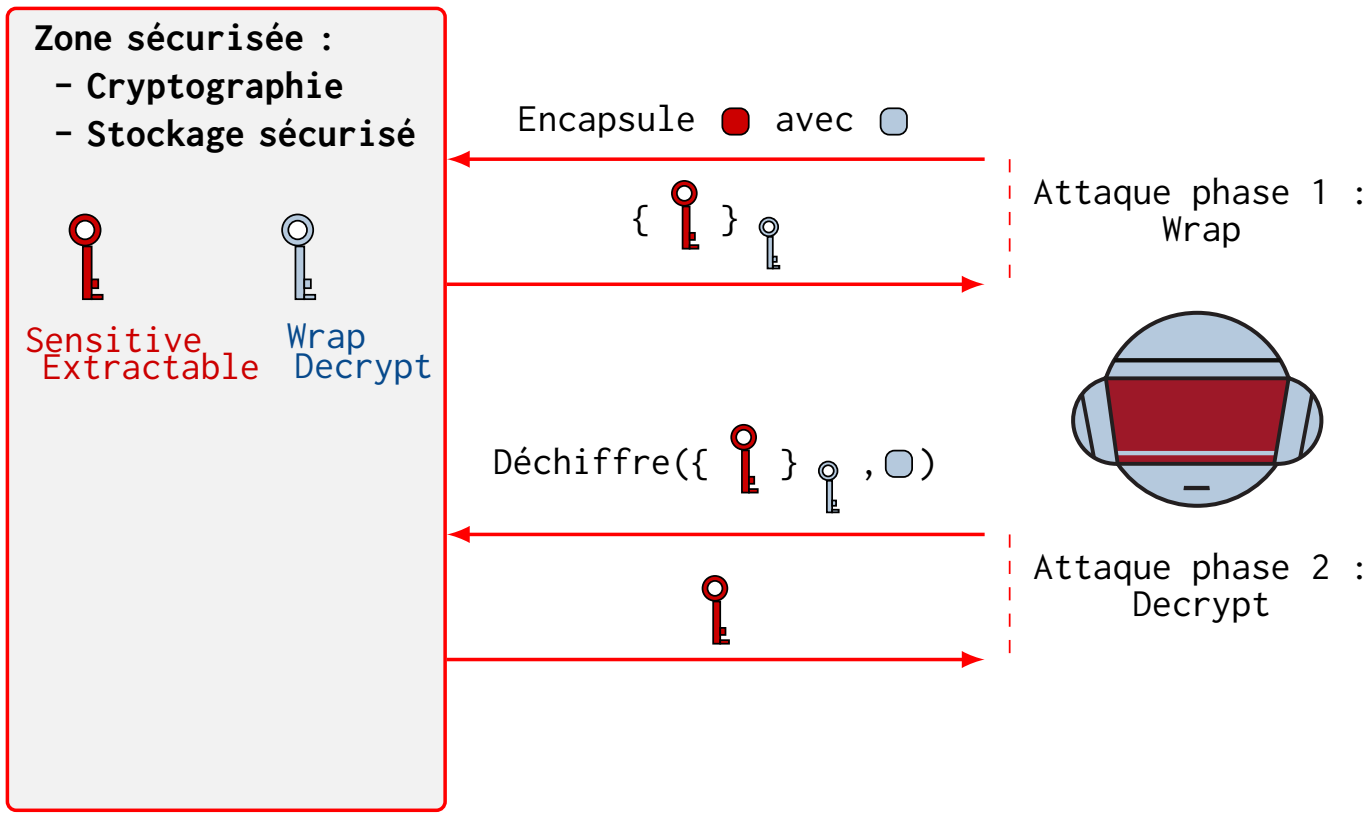
Attaque Wrap/Decrypt

- L'attaque utilise la confusion que fait l'API entre fonctions d'encapsulation et de chiffrement.



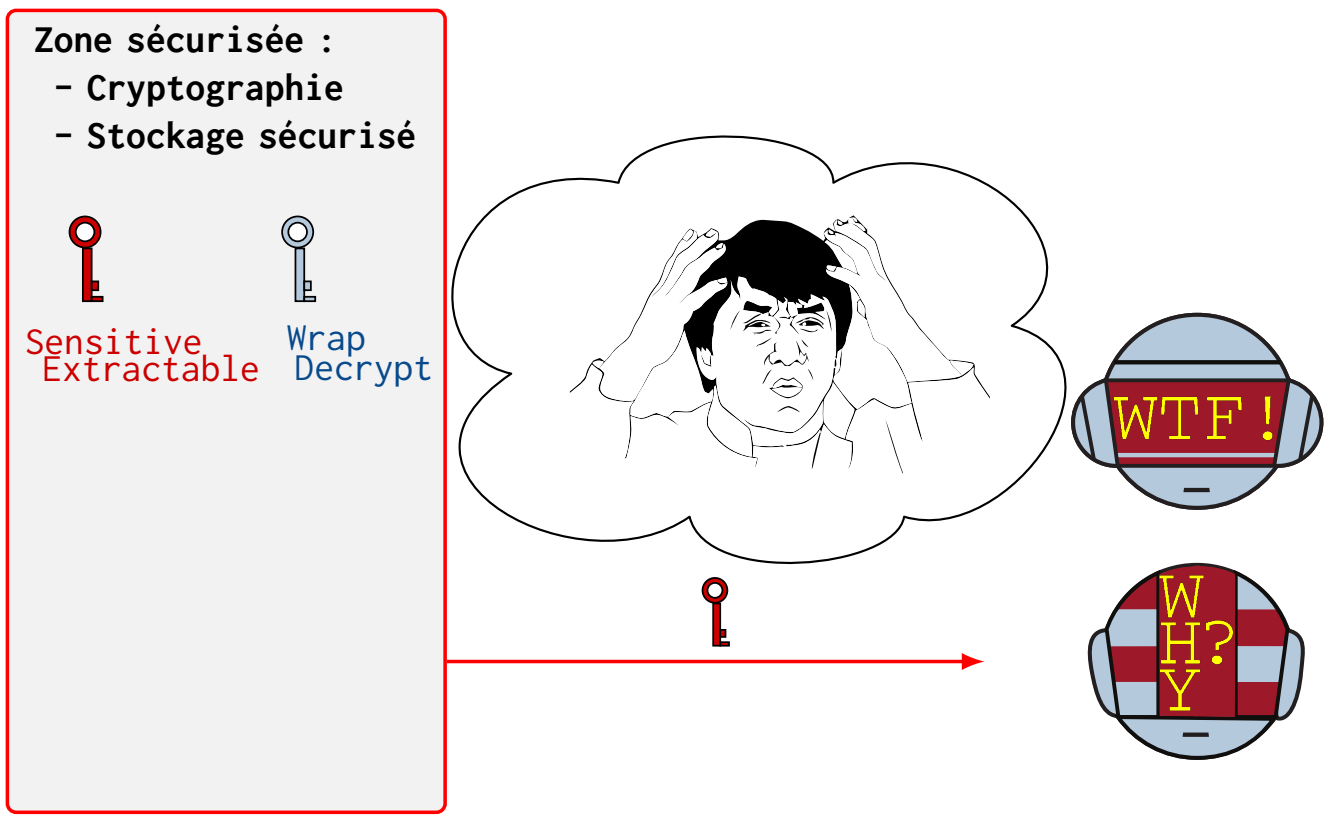
Attaque Wrap/Decrypt

- L'attaque utilise la confusion que fait l'API entre fonctions d'encapsulation et de chiffrement.



Attaque Wrap/Decrypt

- L'attaque utilise la confusion que fait l'API entre fonctions d'encapsulation et de chiffrement.



Veut-on vraiment PKCS#11 ?

Veut-on vraiment PKCS#11 ?

- Non, mais PKCS#11 est incontournable . . .

Veut-on vraiment PKCS#11 ?

Ⓣ Présenté à SSTIC 2011 par Graham Steel

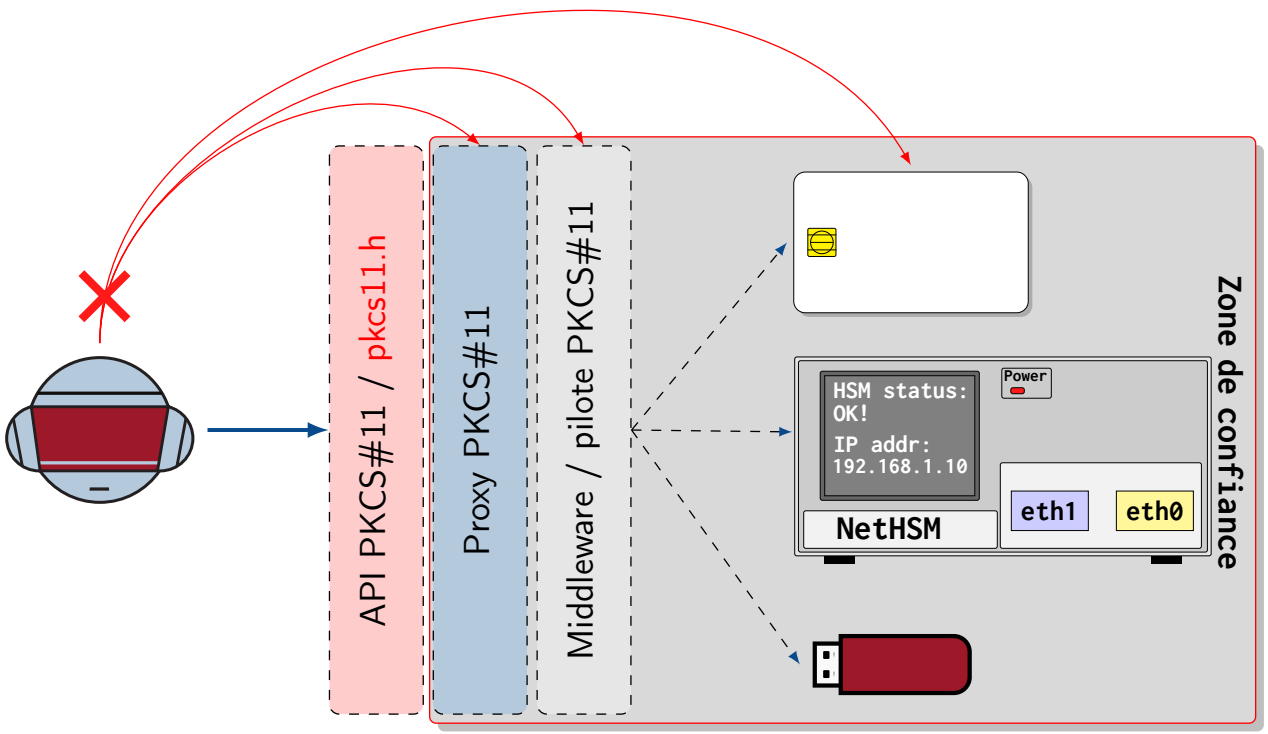
- Non, mais PKCS#11 est incontournable . . .
- Il existe des solutions pour détecter les faiblesses d'un token (Tookan Ⓣ , devenu Cryptosense Analyzer), mais pas vraiment de solution pour en protéger un réputé faible.

La proxification PKCS#11

- Idée : se mettre en coupure des requêtes PKCS#11 entre une application et un middleware.

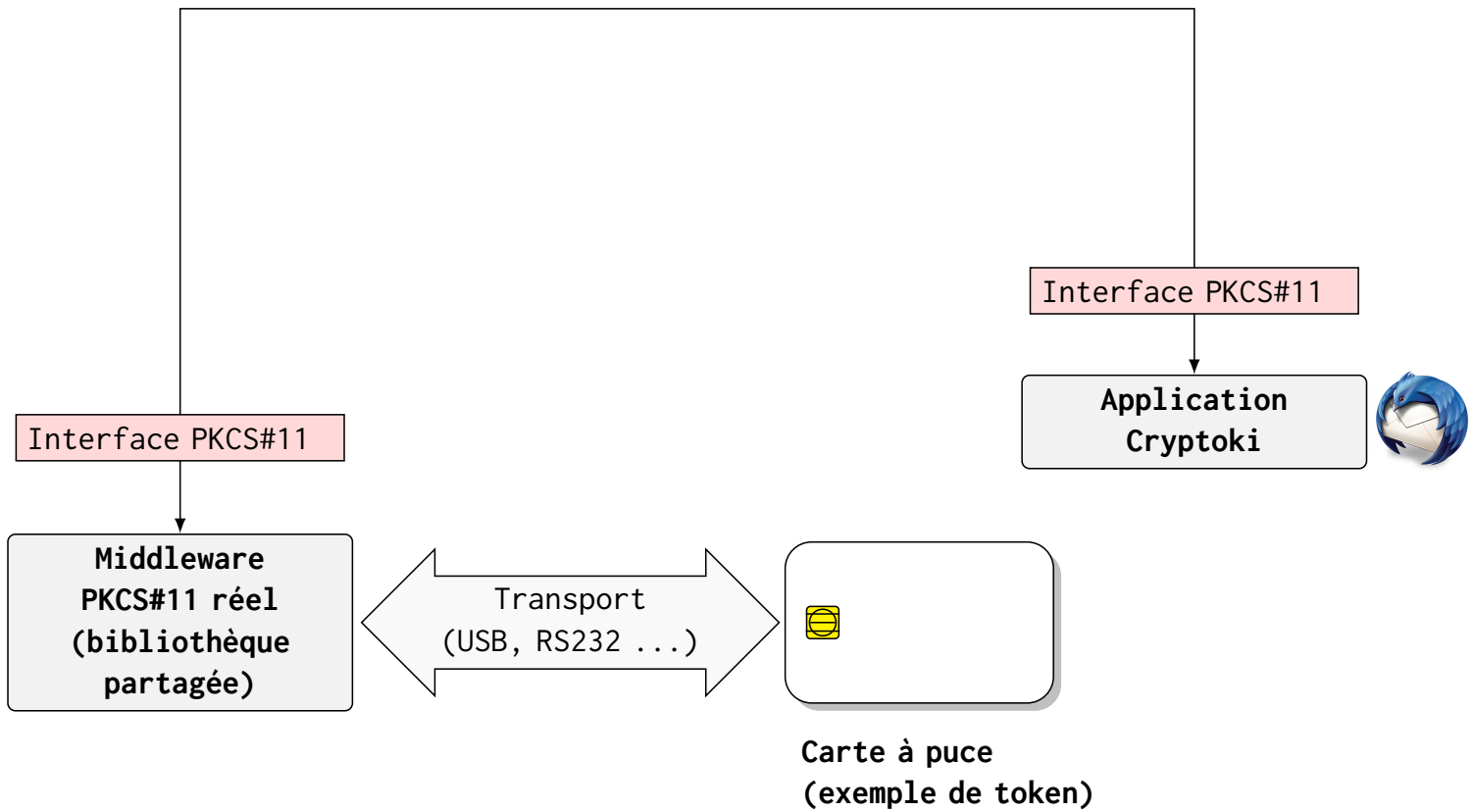
La proxification PKCS#11

- Idée : se mettre en coupure des requêtes PKCS#11 entre une application et un middleware.
- Adaptation du modèle d'attaquant :
  - Possibilité d'effectuer des requêtes PKCS#11
  - Impossibilité de contourner le proxy (c.f. Déploiement)



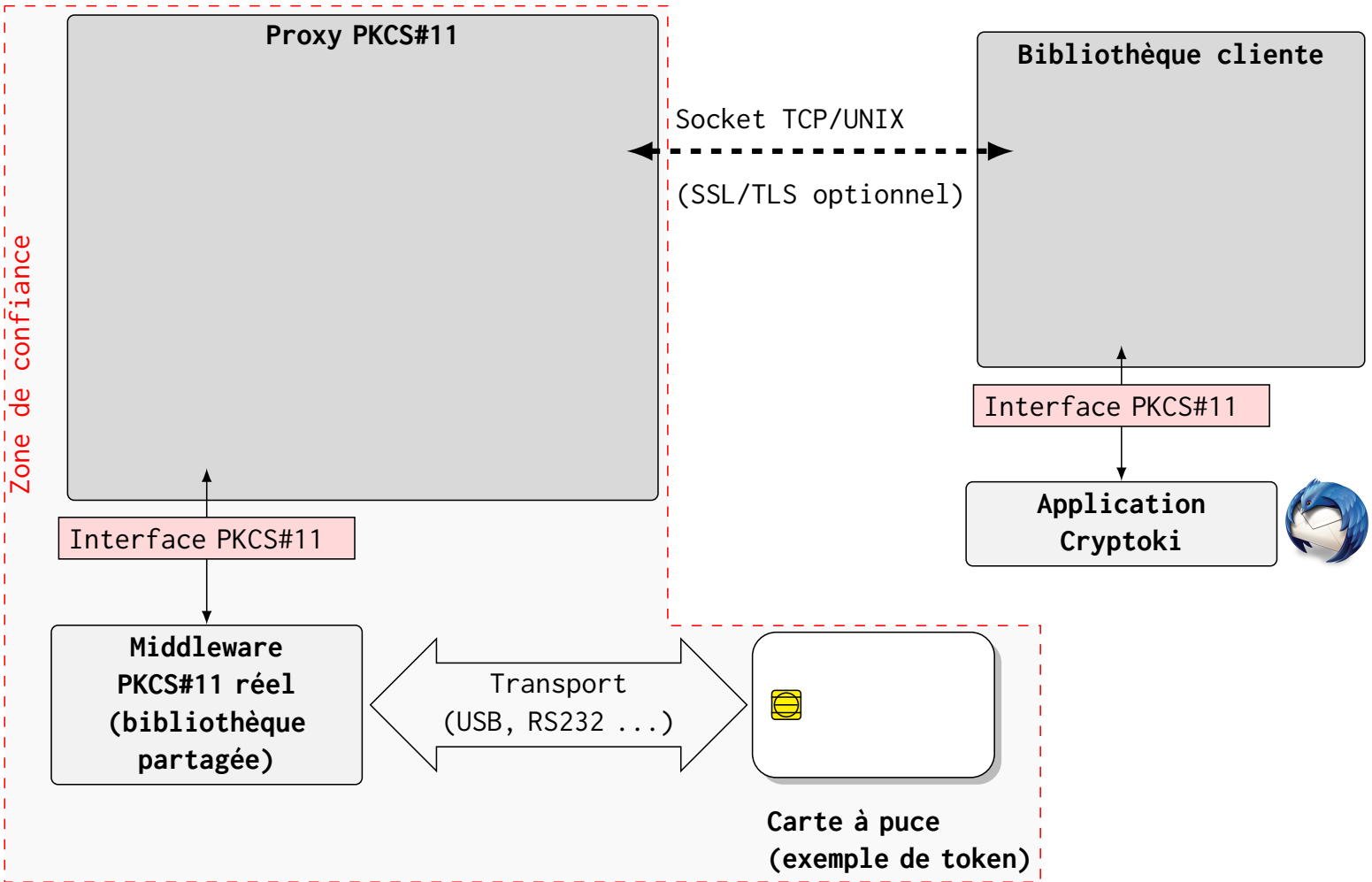
La proxification PKCS#11

- Idée : se mettre en coupure des requêtes PKCS#11 entre une application et un middleware.



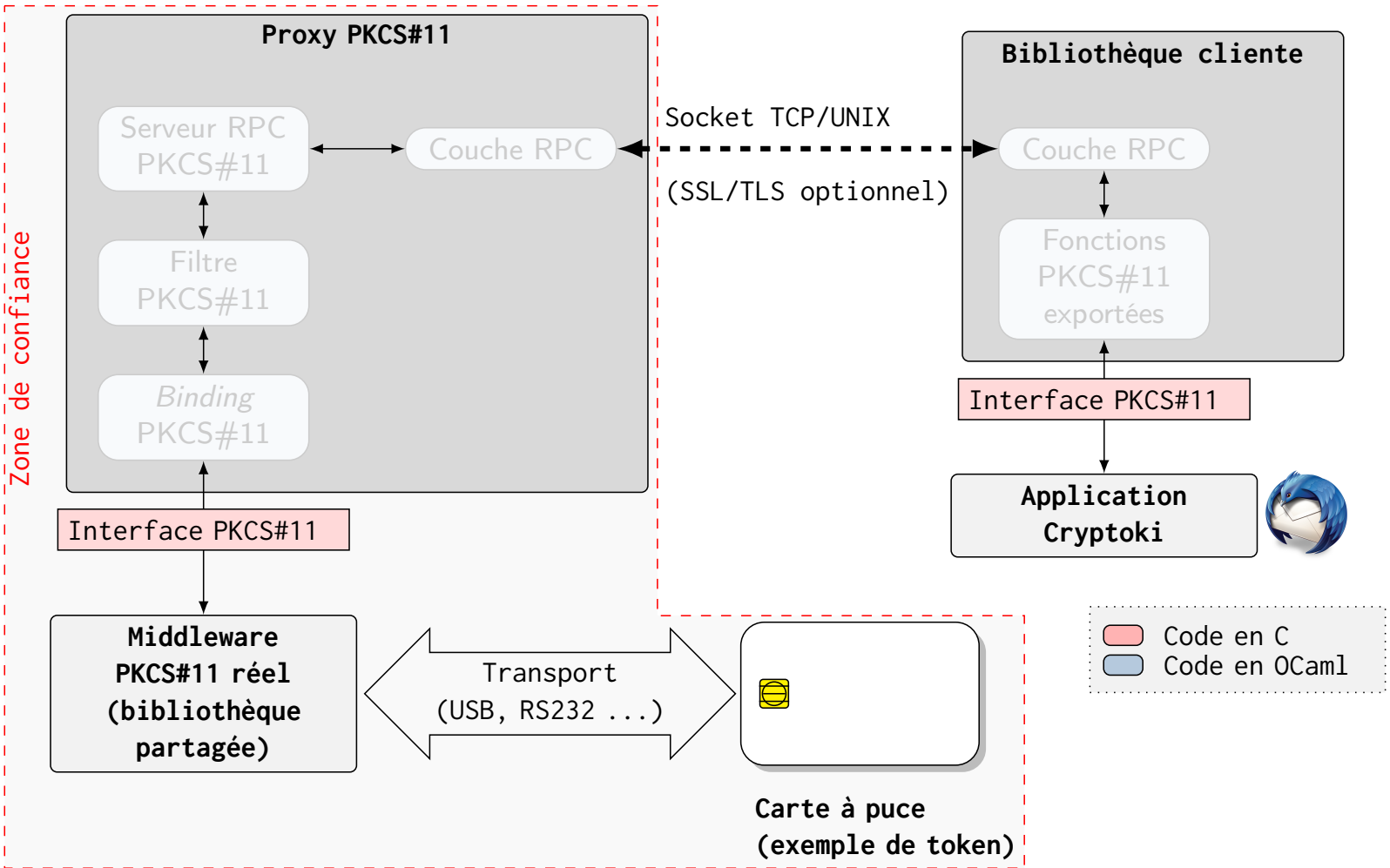
La proxification PKCS#11

- Idée : se mettre en coupure des requêtes PKCS#11 entre une application et un middleware.



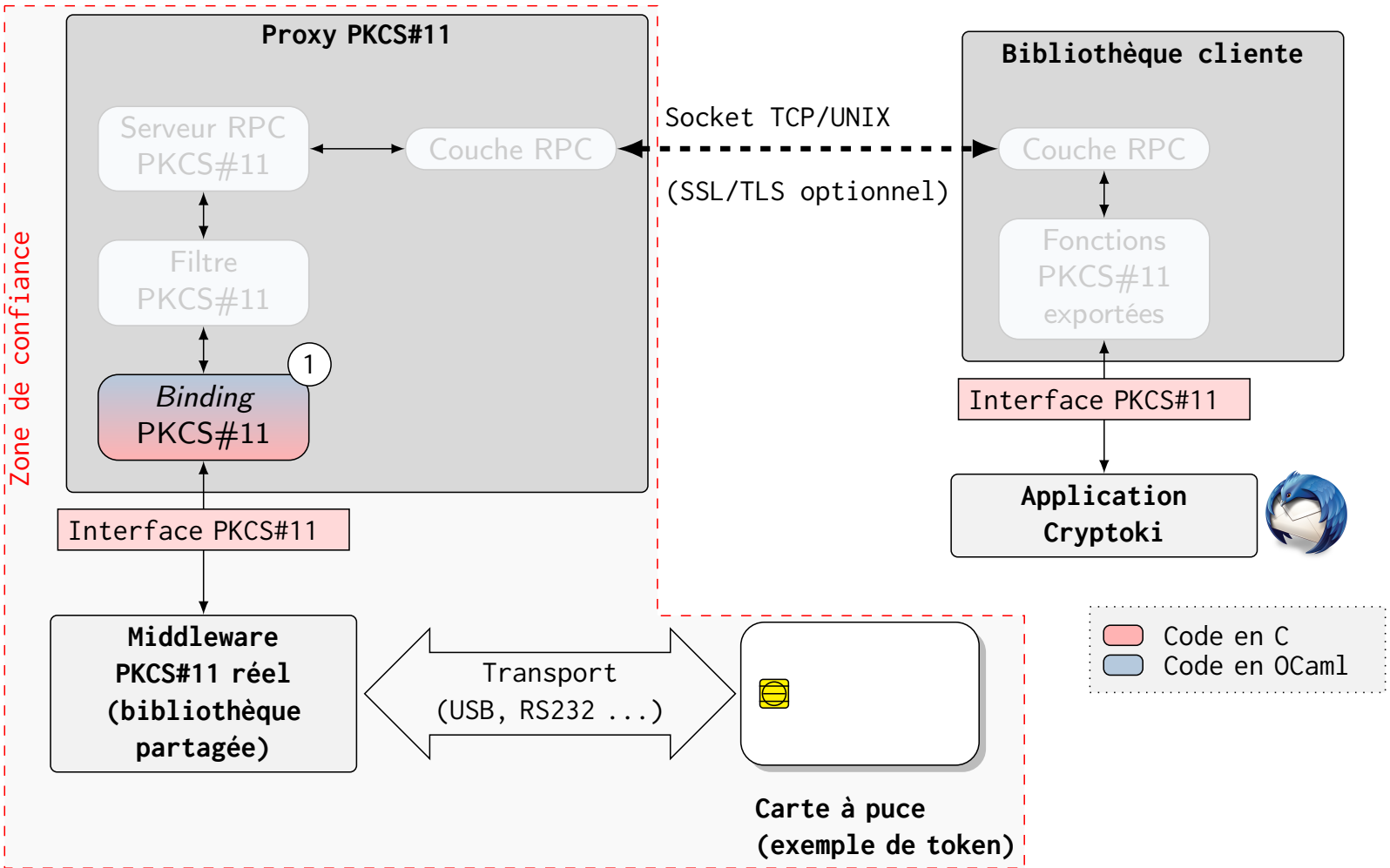
La proxification PKCS#11

- Un proxy modulaire avec des briques spécialisées.



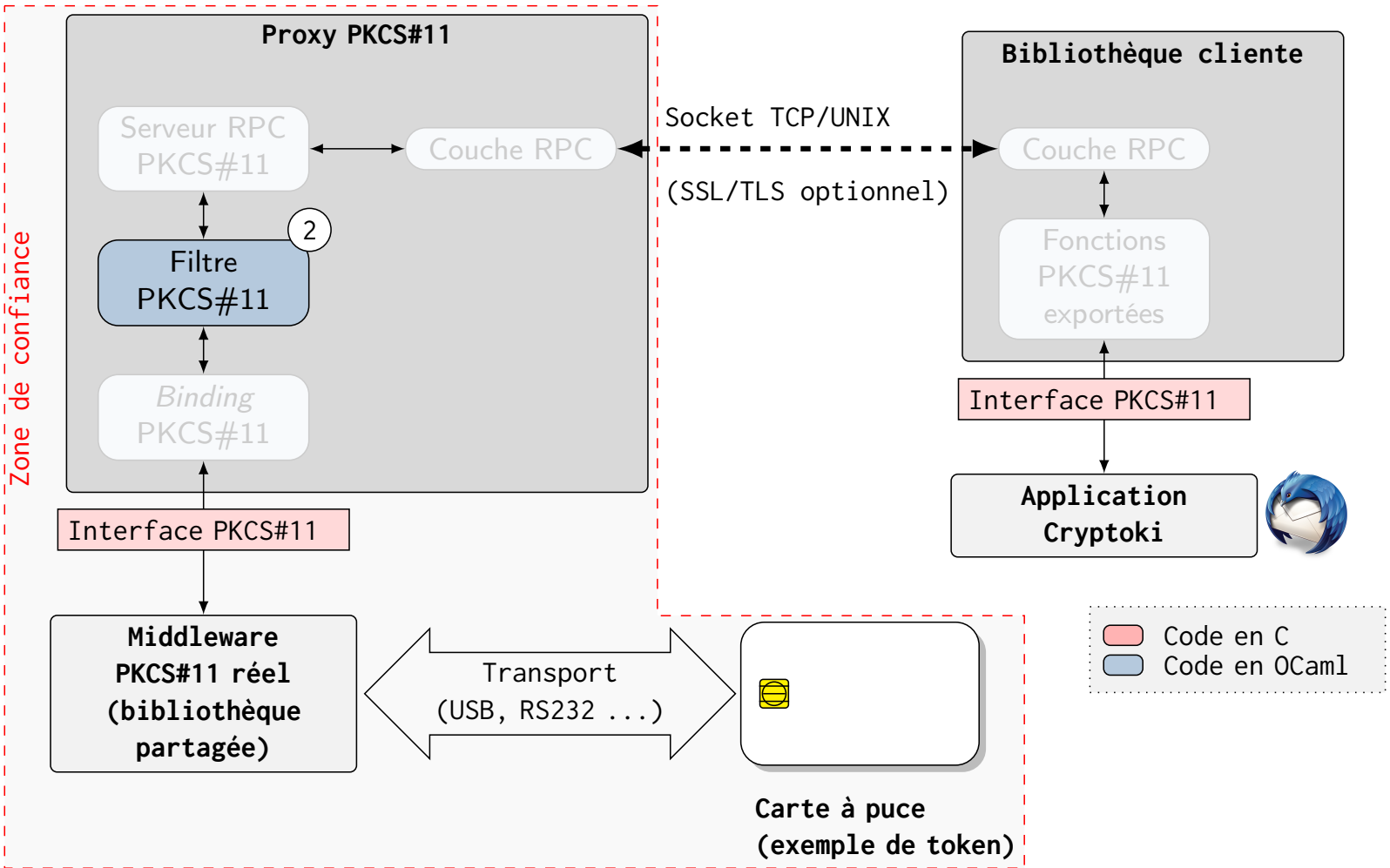
La proxification PKCS#11

- Binding OCaml/C nécessaire pour discuter avec le middleware d'origine.



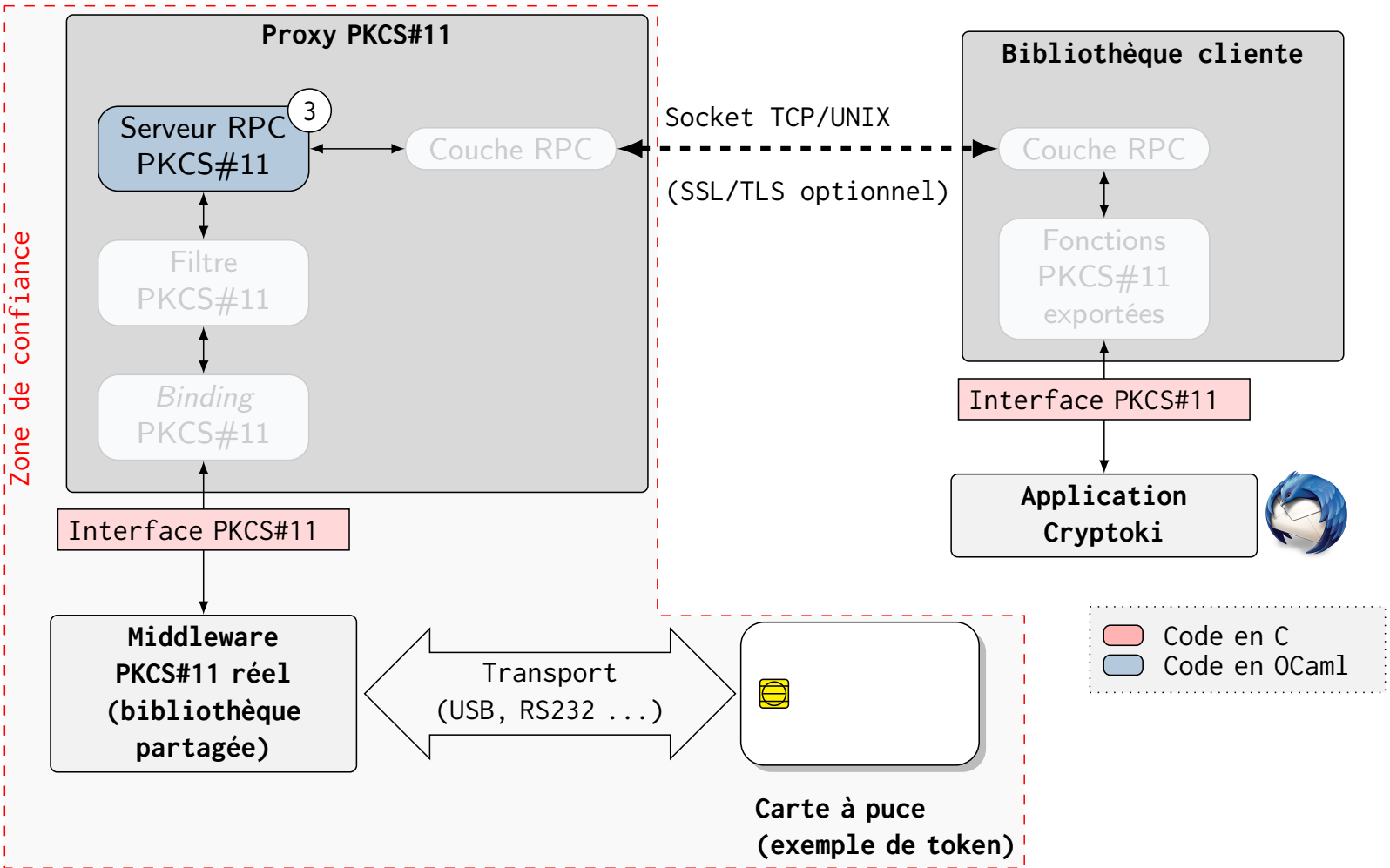
La proxification PKCS#11

- Un filtre en OCaml (contenant l'intelligence du proxy).



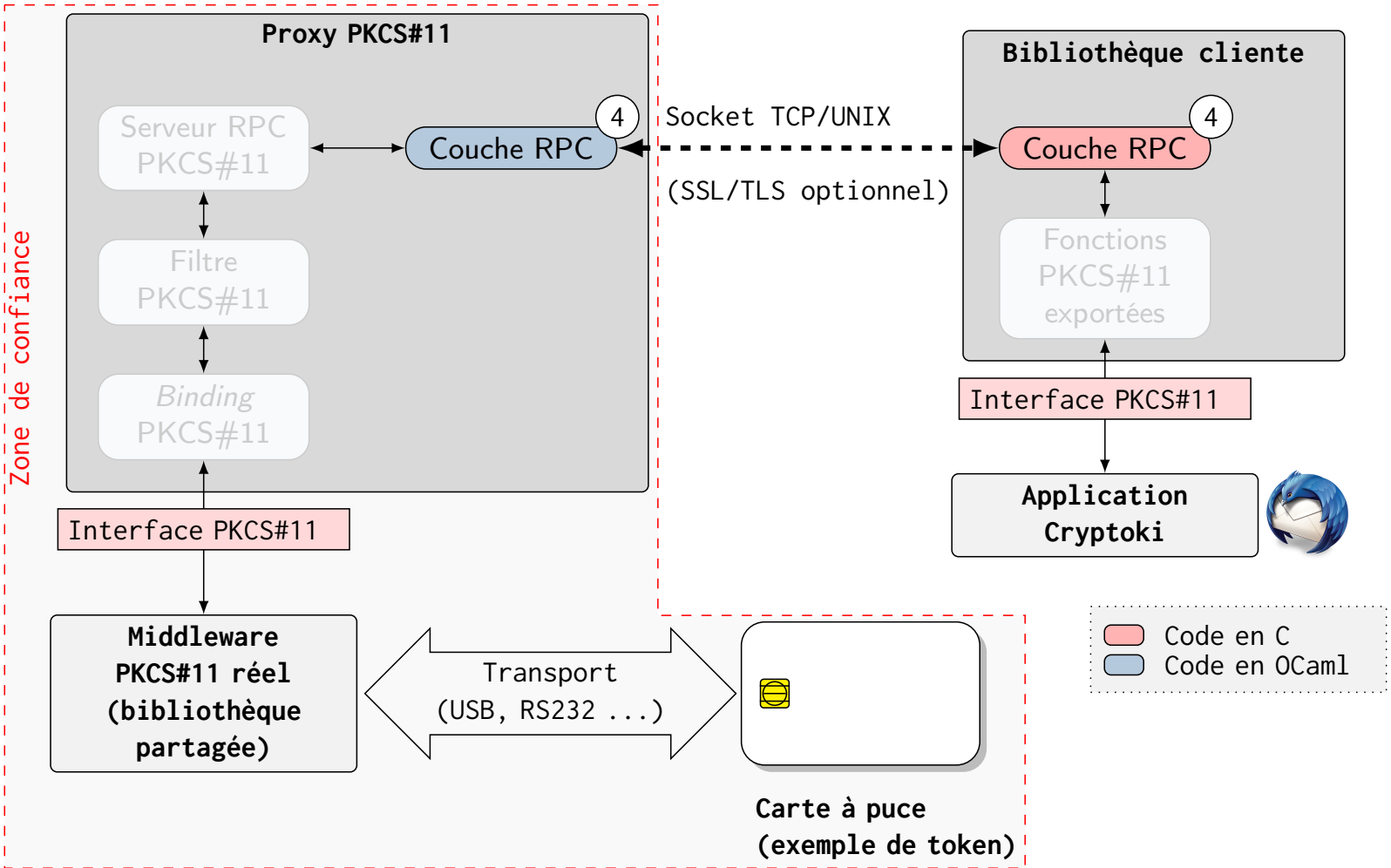
La proxification PKCS#11

- Un serveur RPC en OCaml fourni par la bibliothèque Netplex.



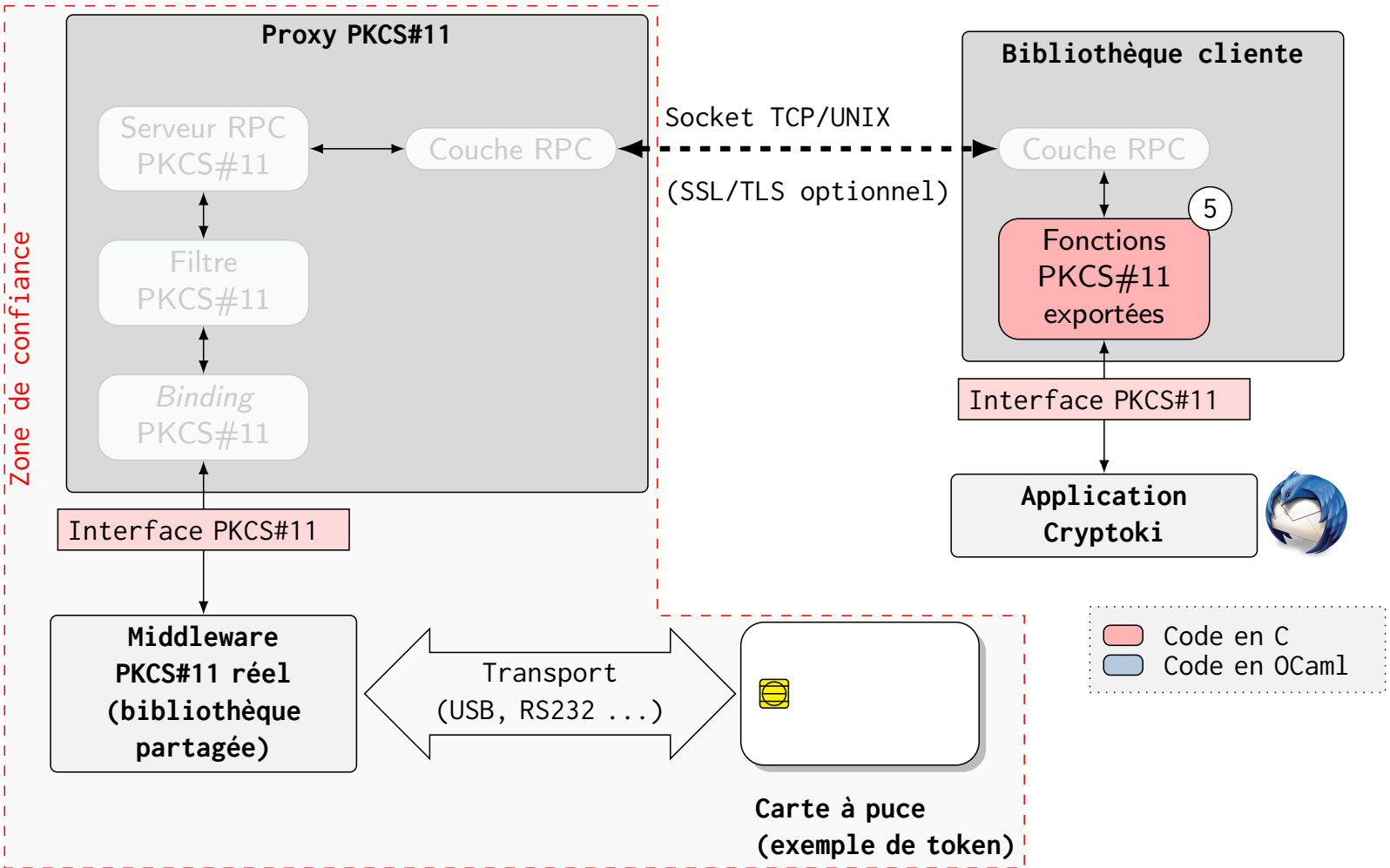
La proxification PKCS#11

- Une couche Sun RPC pour le transport des fonctions P11, en OCaml ou C via le standard XDR.



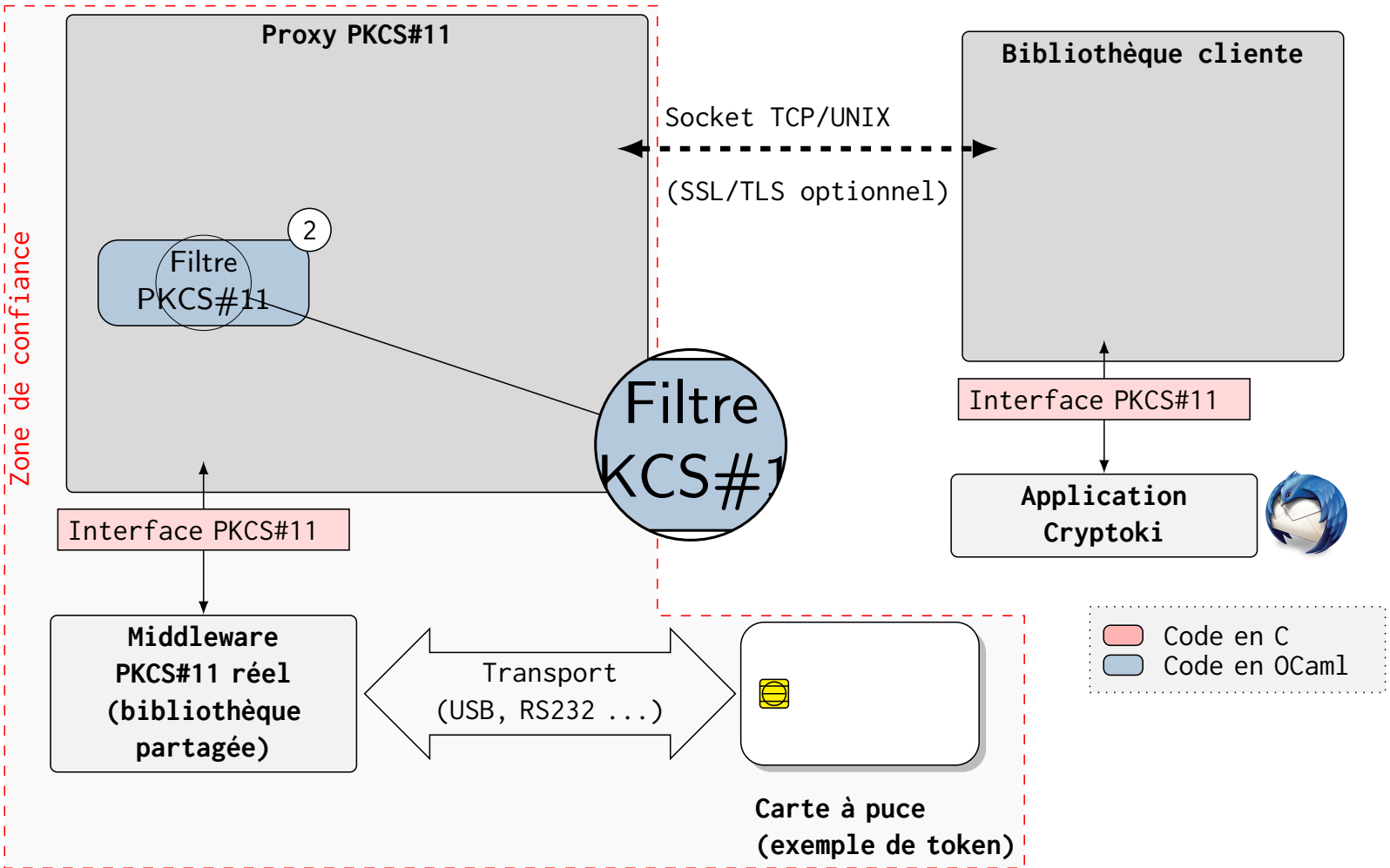
La proxification PKCS#11

- Une bibliothèque cliente qui exporte la P11 côté applicatif.



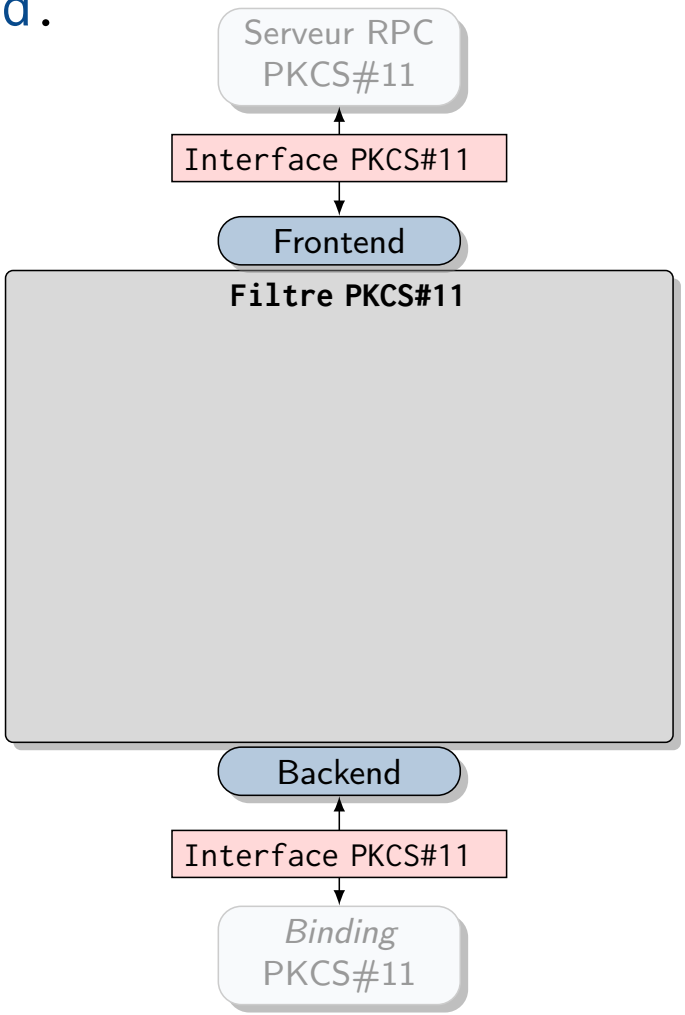
La proxification PKCS#11

- Zoom sur le filtre qui est au cœur de la solution.



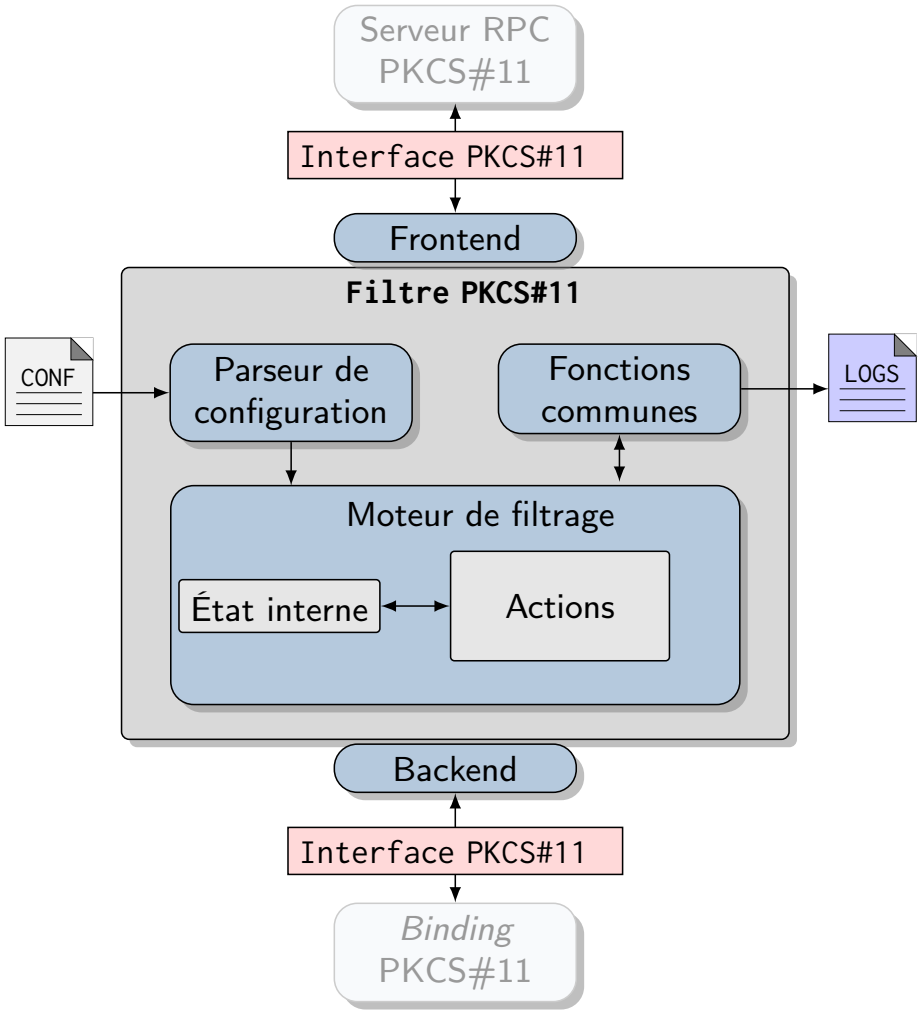
Filtrage PKCS#11

- Le filtre est isolé du reste via un frontend et un backend.



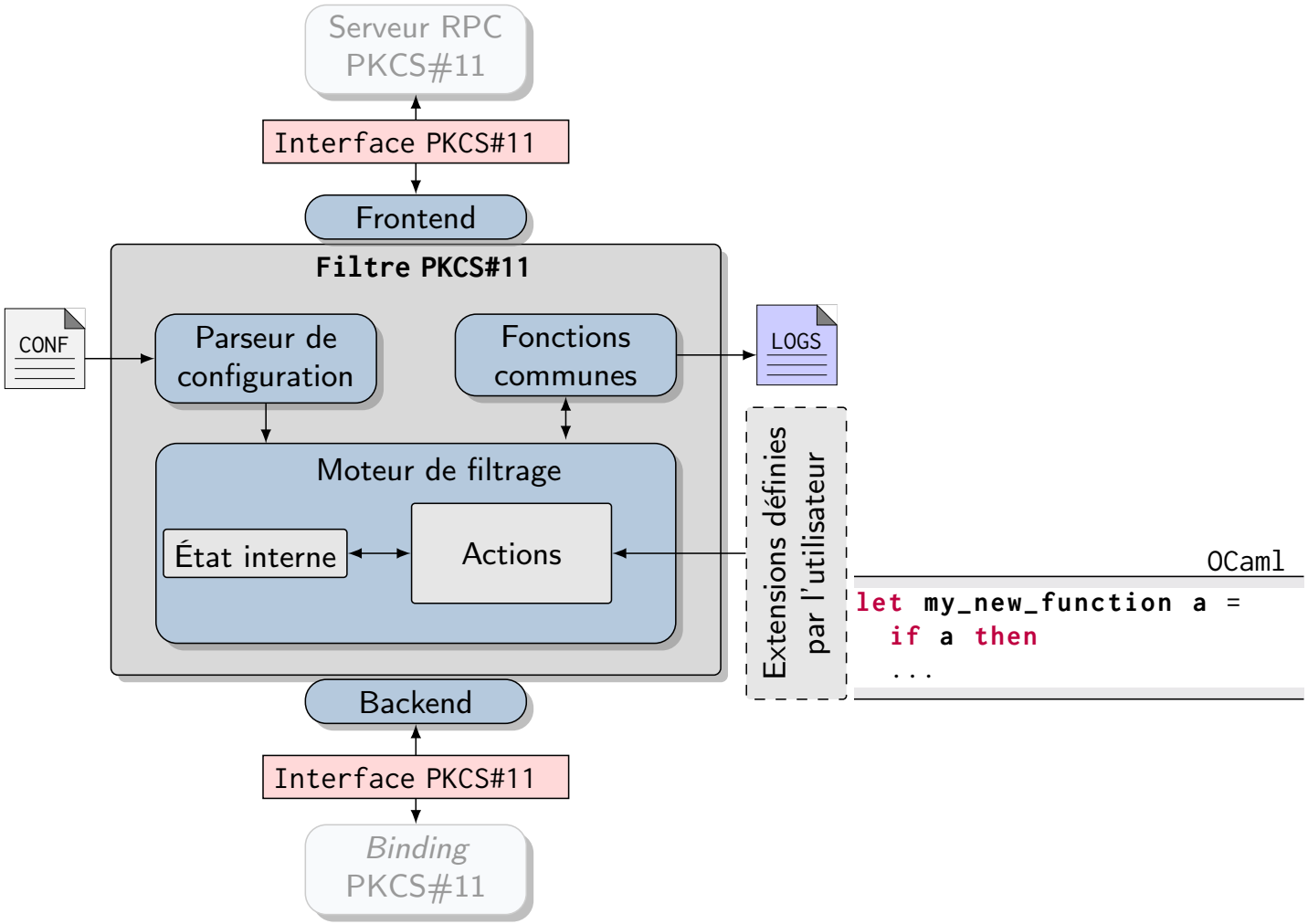
Filtrage PKCS#11

- Vue globale des composants du filtre.



Filtrage PKCS#11

- Actions de l'utilisateur extensibles.

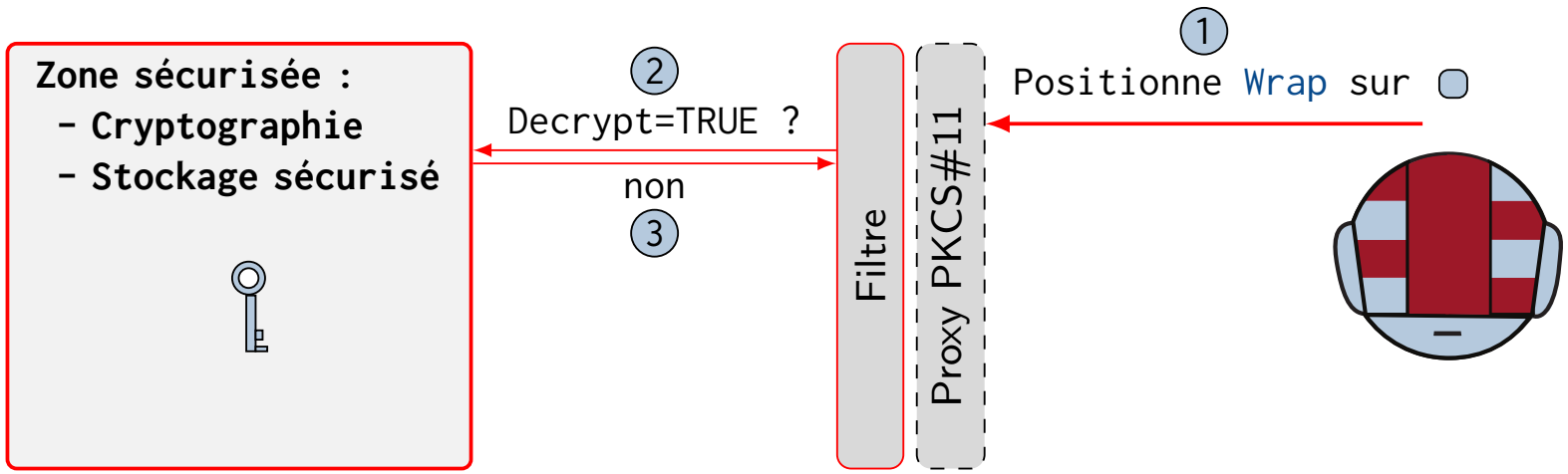


OCaml

```
let my_new_function a =
  if a then
  ...
```

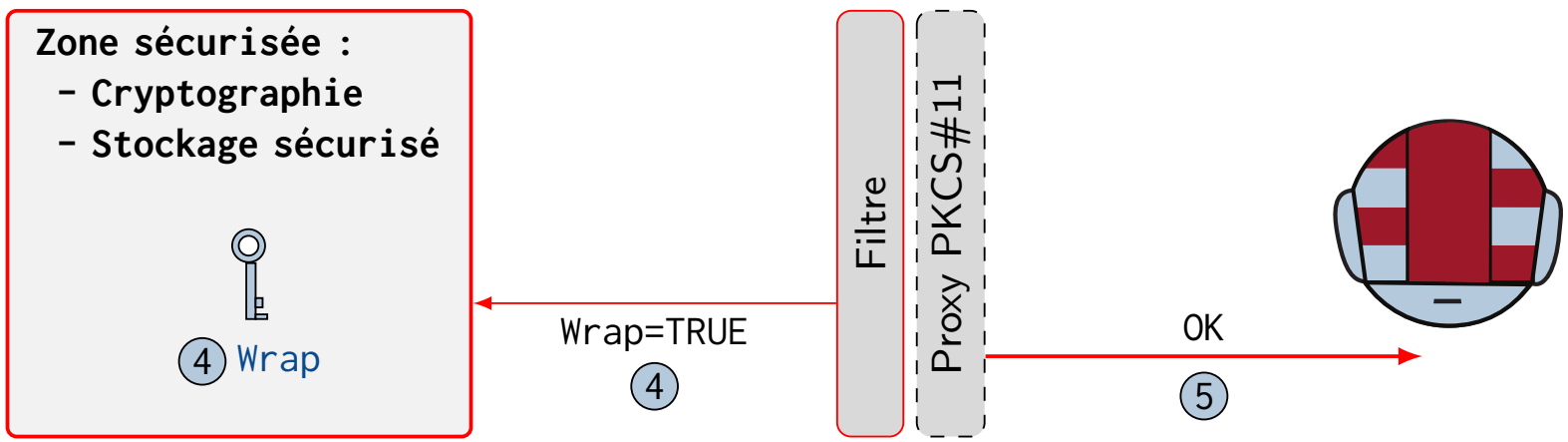
Aspects sécurité

- Implémente des contre-mesures dynamiques aux attaques sur PKCS#11 : par exemple les attributs conflictuels.



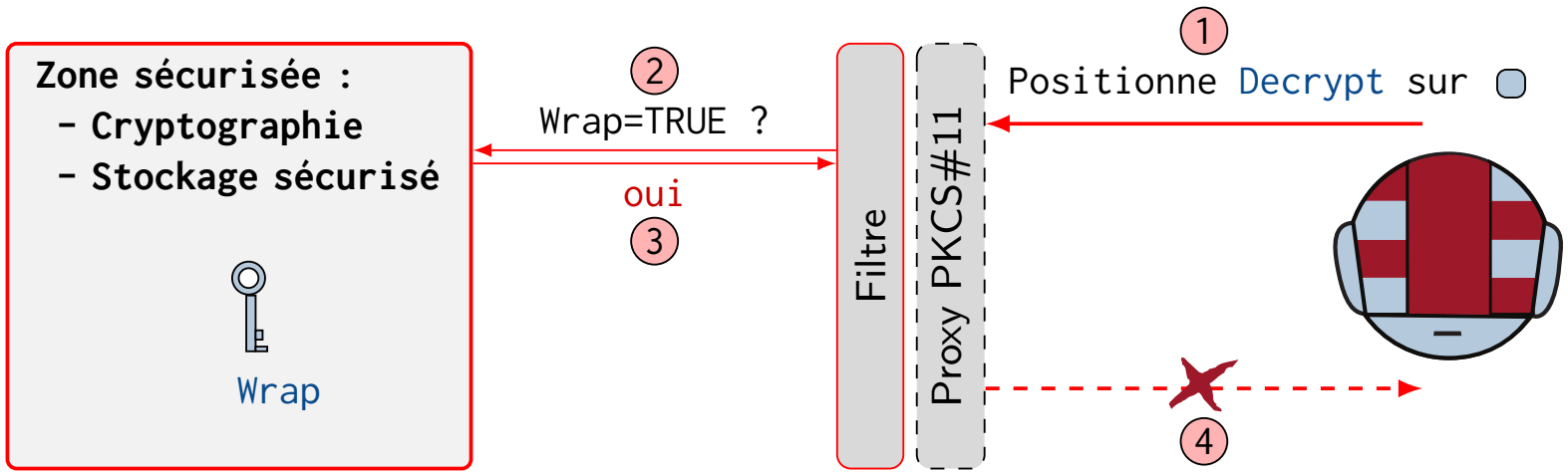
Aspects sécurité

- Implémente des contre-mesures dynamiques aux attaques sur PKCS#11 : par exemple les attributs conflictuels.



Aspects sécurité

- Implémente des contre-mesures dynamiques aux attaques sur PKCS#11 : par exemple les attributs conflictuels.



Aspects sécurité

- Implémente des contre-mesures dynamiques aux attaques sur PKCS#11 : par exemple les attributs conflictuels.
- Bloquer des attaques sur tout périphérique PKCS#11 :
  - Avec les patchs logiques de l'API
  - En interdisant les fonctions/mécanismes dangereux

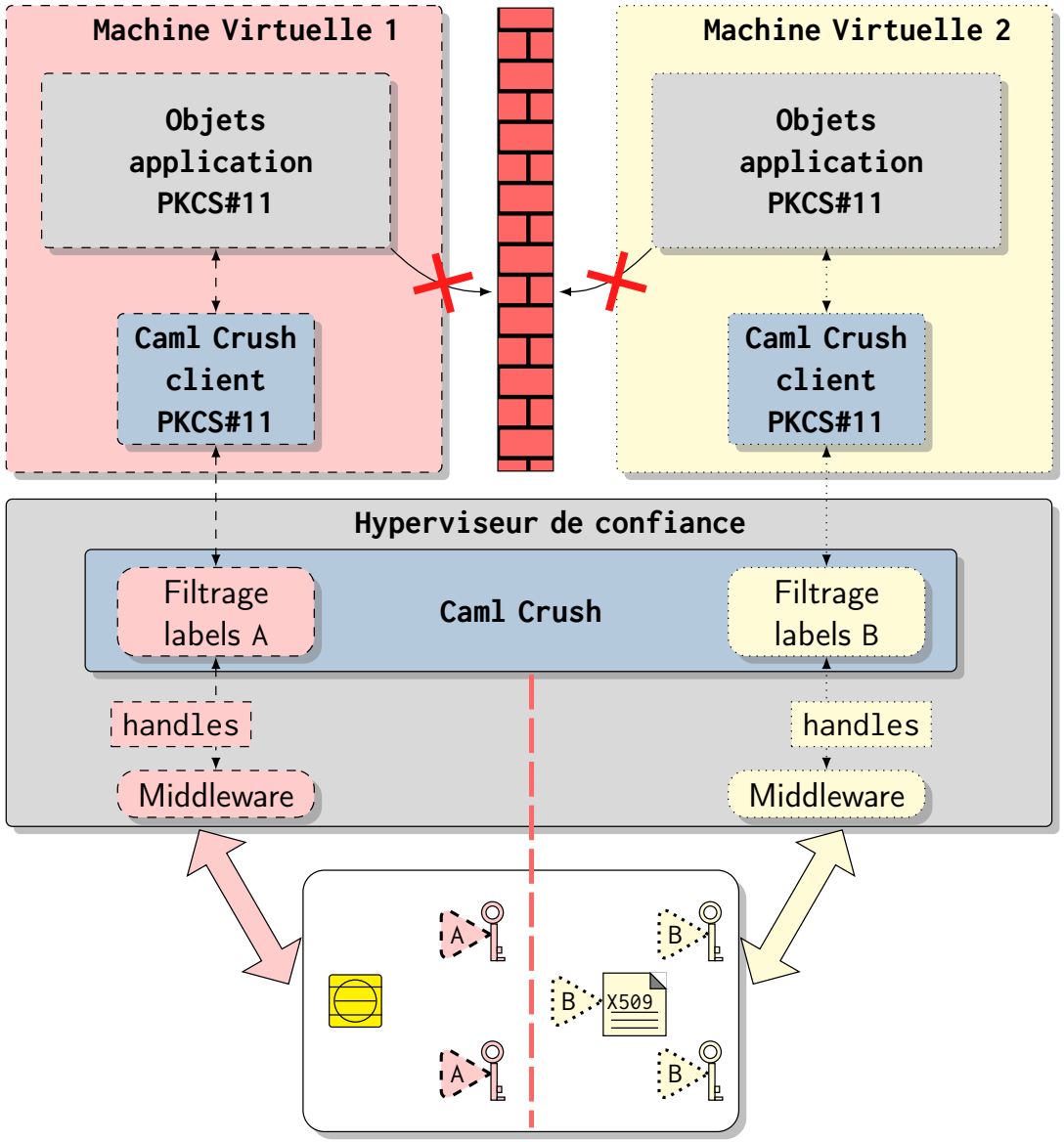
Aspects sécurité

- Implémente des contre-mesures dynamiques aux attaques sur PKCS#11 : par exemple les attributs conflictuels.
- Bloquer des attaques sur tout périphérique PKCS#11 :
  - Avec les patchs logiques de l'API
  - En interdisant les fonctions/mécanismes dangereux
- Offre une solution aux limitations de PKCS#11 :
  - Segmentation des logins SO (admin) et User
  - Filtrage d'objets sur une ressource
  - Forcer l'utilisation en lecture seule
  - Durcir la politique de complexité du PIN
  - . . .

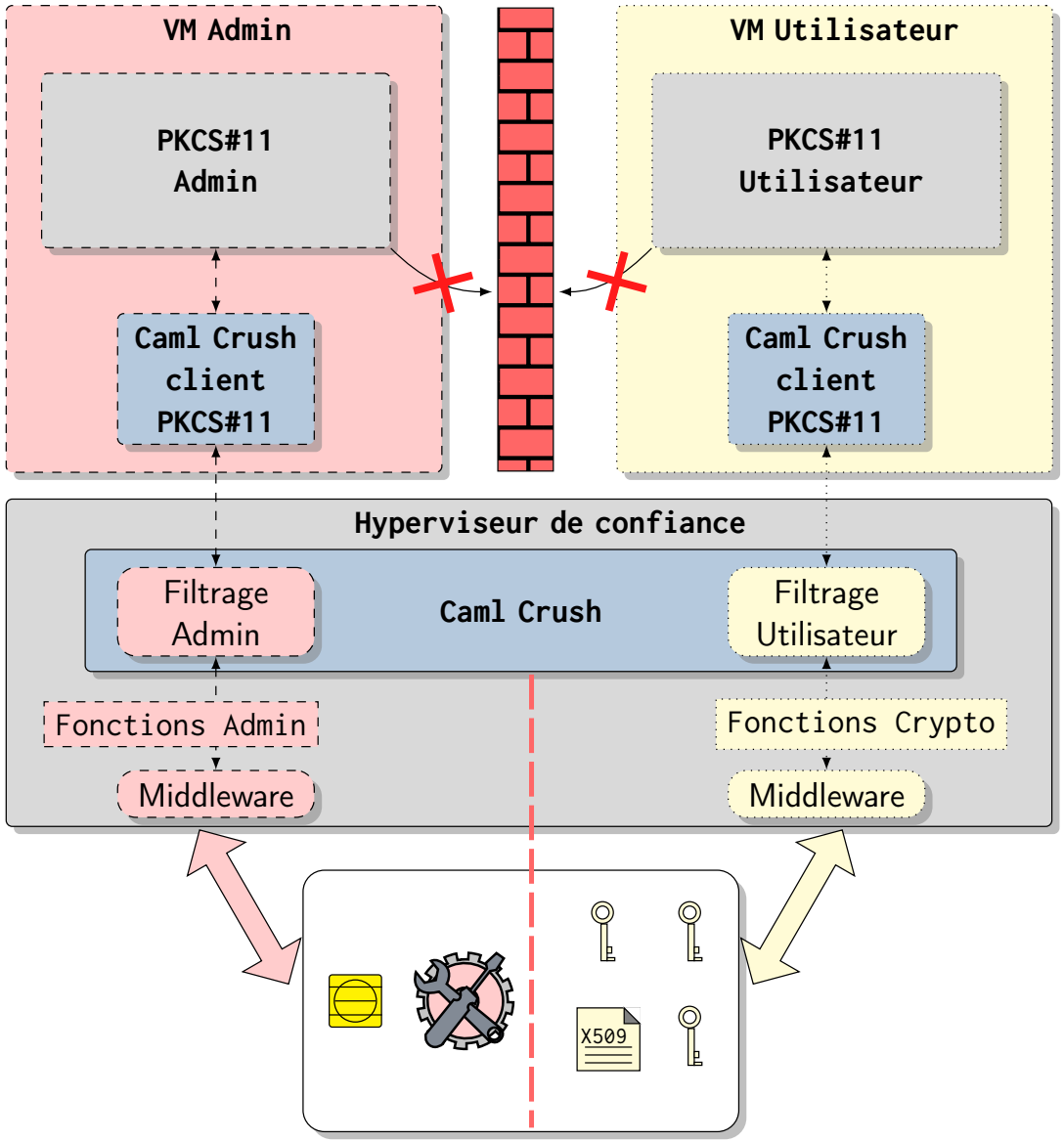
Aspects sécurité

- Implémente des contre-mesures dynamiques aux attaques sur PKCS#11 : par exemple les attributs conflictuels.
- Bloquer des attaques sur tout périphérique PKCS#11 :
  - Avec les patchs logiques de l'API
  - En interdisant les fonctions/mécanismes dangereux
- Offre une solution aux limitations de PKCS#11 :
  - Segmentation des logins SO (admin) et User
  - Filtrage d'objets sur une ressource
  - Forcer l'utilisation en lecture seule
  - Durcir la politique de complexité du PIN
  - . . .
- Contourner les problèmes (connus) liés à un «middleware».

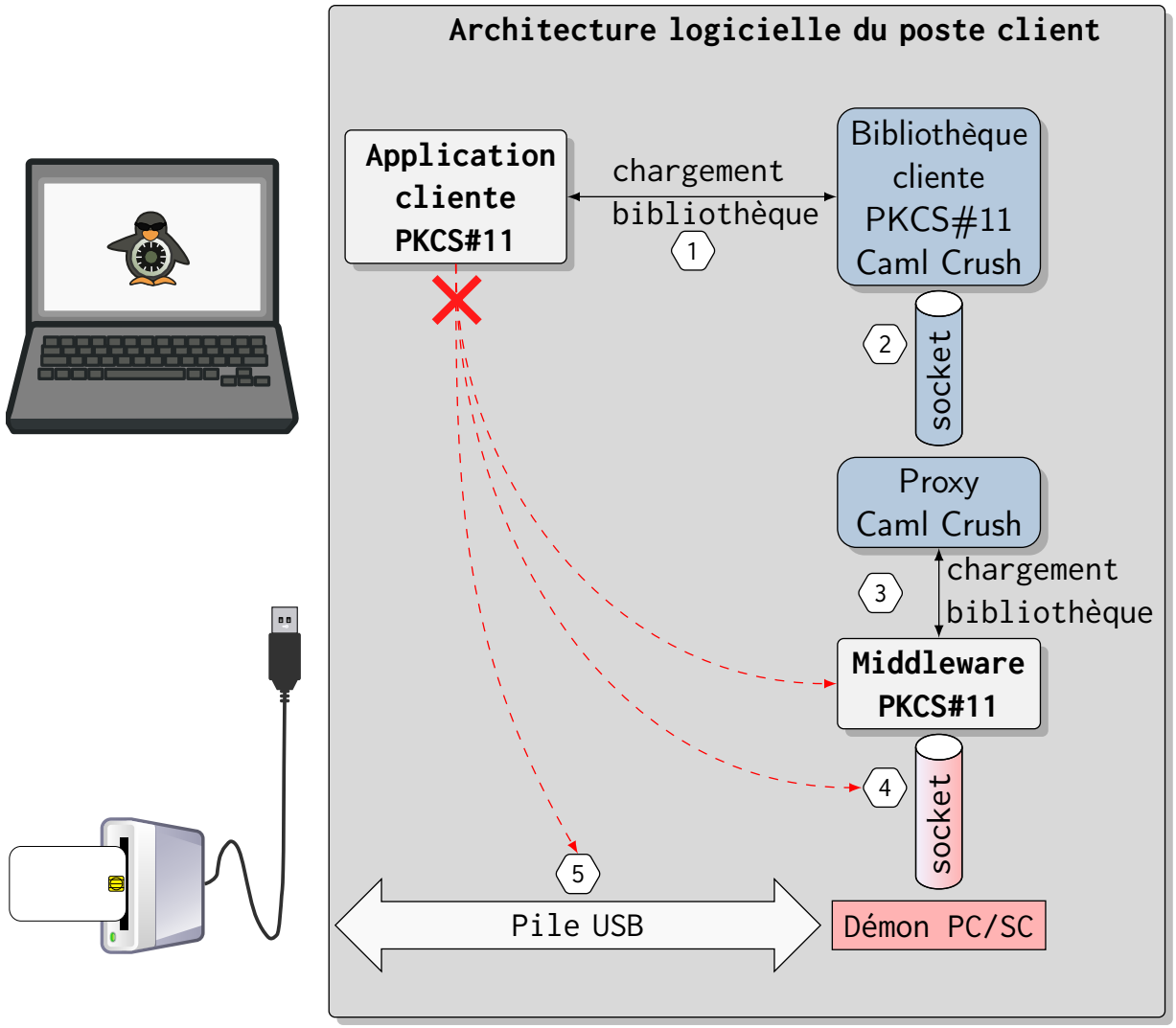
Partage de ressources



Cloisonnement des usages

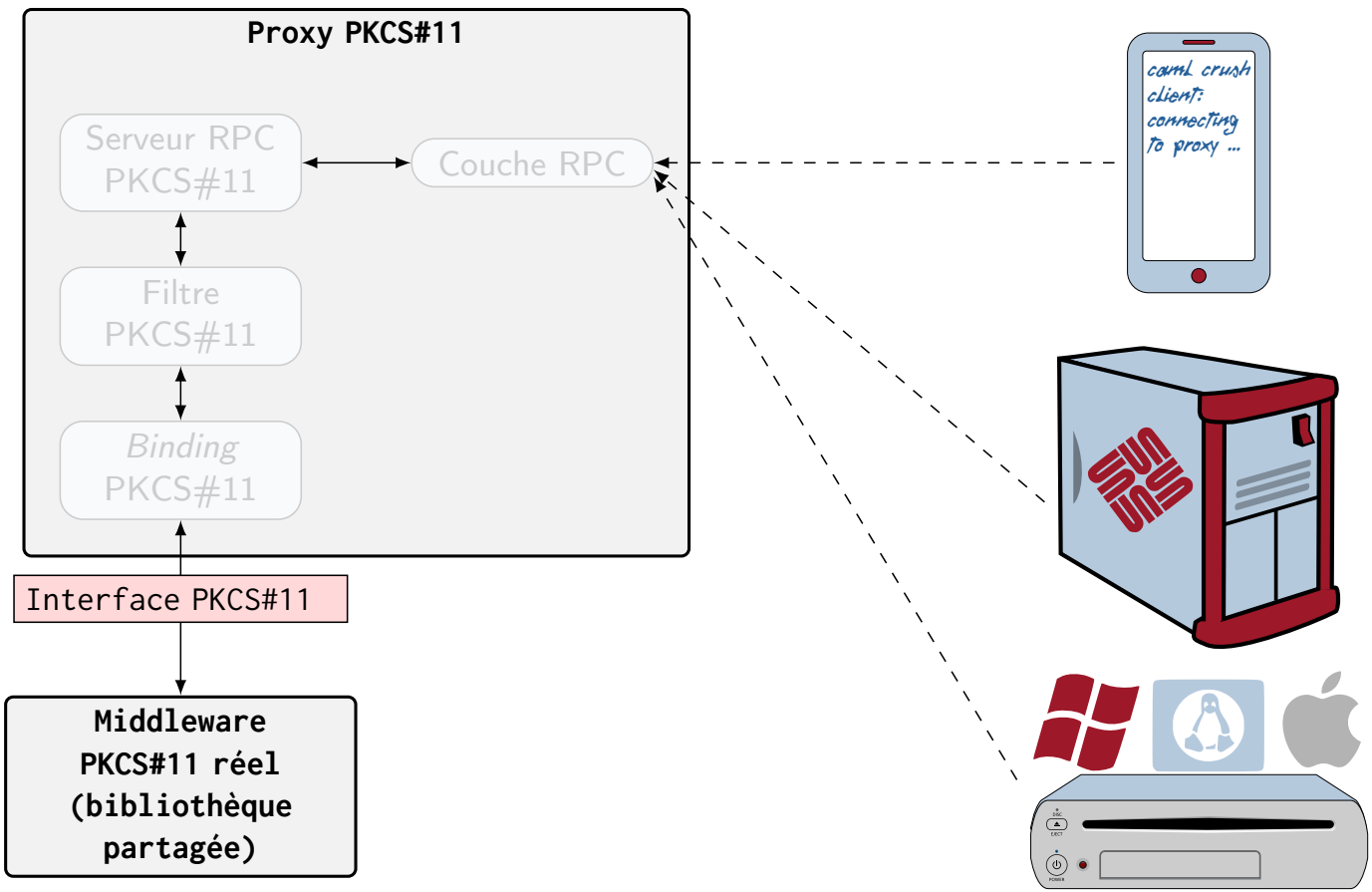


Poste durci


Architecture logicielle du poste client
Application cliente PKCS#11
chargement bibliothèque
1
Bibliothèque cliente PKCS#11 Caml Crush
2
socket
Proxy Caml Crush
3
chargement bibliothèque
Middleware PKCS#11
4
socket
5
Pile USB
Démon PC/SC

Portabilité de la bibliothèque cliente

- La bibliothèque cliente est portable sous divers CPU ainsi que divers OS.



Conclusion

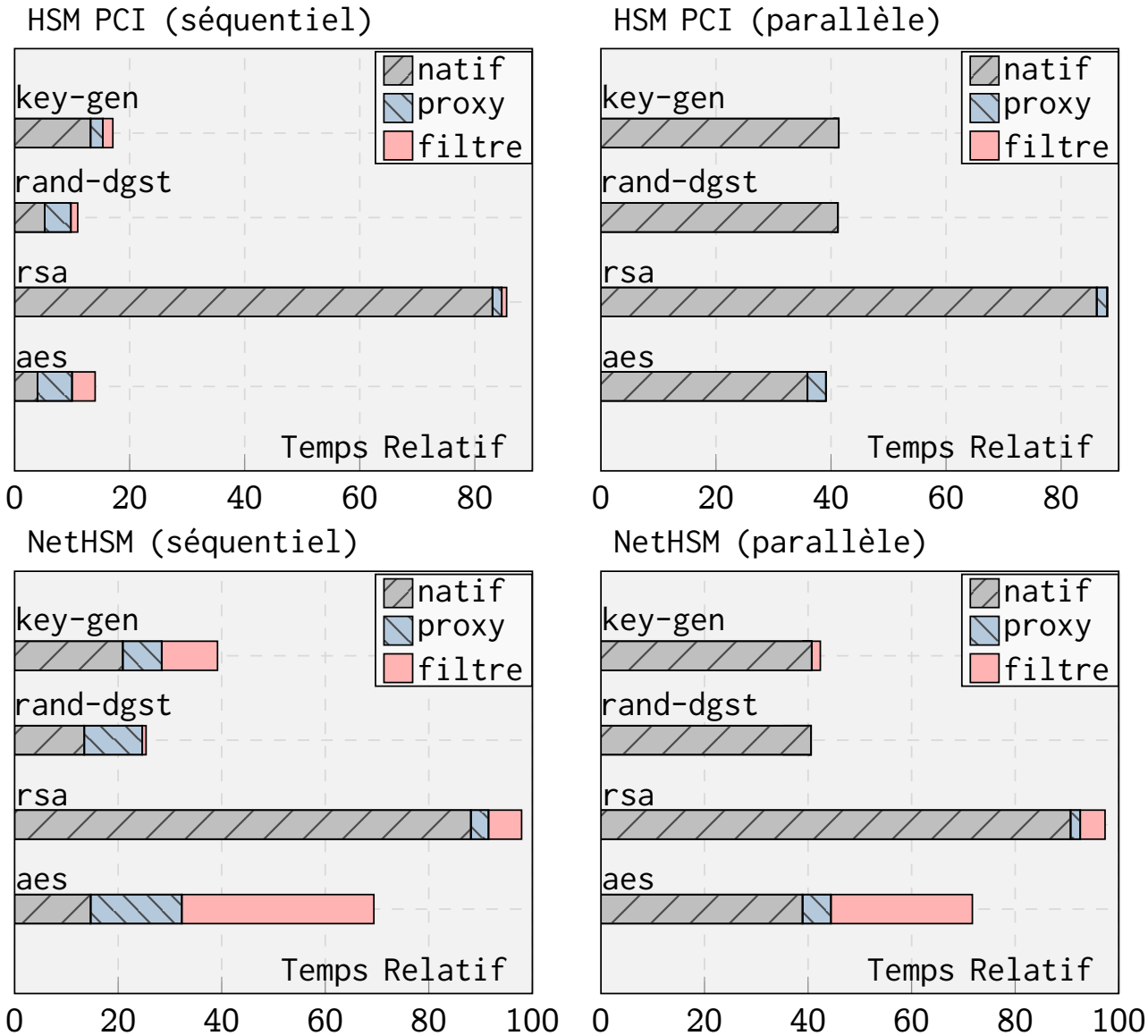
- Blocage dynamique des attaques PKCS#11.
- Apporte d'autres fonctionnalités utiles.
- Caml Crush est open source :
  - https://github.com/ANSSI-FR/caml-crush

Conclusion

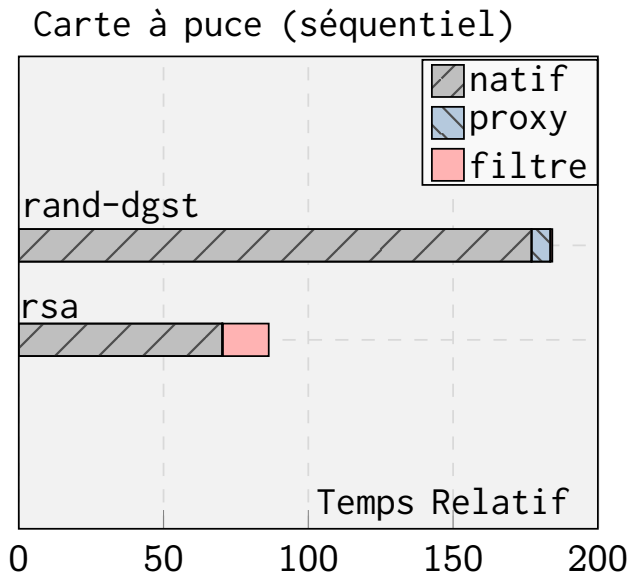
- Blocage dynamique des attaques PKCS#11.
- Apporte d'autres fonctionnalités utiles.
- Caml Crush est open source :
  - https://github.com/ANSSI-FR/caml-crush

? questions ?

Performances



Performances



- Coût négligeable en asymétrique.
- Impact plus élevé sur les opérations symétriques.
  - Carte à puce : peu de support
  - HSM : dépend du matériel mais le coût réduit dès lors que la charge du HSM augmente

Matrice de compatibilité

	Client C		Client Ocaml		pkcs11proxyd		SSL/TLS
	Unix	TCP	Unix	TCP	Unix	TCP	
Linux	✓	✓	✓	✓	✓	✓	✓
FreeBSD	✓	✓	✓	✓	✓	✓	✓
Mac OS X	✗	✓	✓	✓	✓	✓	✓
Win32 (natif)	✗	✓	✗	✗	✗	✗	
Win32 (cygwin)							

- Caml Crush fonctionne sur les plateformes Little/Big Endian (avec architectures hybrides entre client et serveur).

Choix d'OCaml

- Langage fonctionnel, fortement typé, gestion de la mémoire avec ramasse-miettes, possibilité de compilation native via ocamlopt :
  - Garanties fortes contre les vulnérabilités classiques (buffer overflows)
  - Garde-fous à la compilation grâce à l'inférence de type
  - Aspect fonctionnel compatible avec l'expressivité de règles de filtrage
  - . . . tout cela en gardant des performances raisonnables !