Sécurité des implémentations pour la cryptographie

Partie 2 : Architecture produit et APIs de sécurité

Benoît Gérard

29 novembre 2017

Plan du cours

Étape 1

Définition du besoin et de l'architecture au niveau système.

Étape 2

Définition de l'interface carte/terminal : API exposée par la carte.

Étape 3

Implémentation d'une version résistante aux attaques non-crypto.

Étape 4

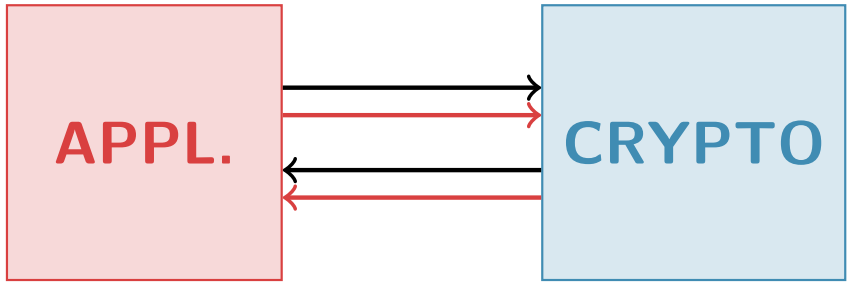
Implémentation d'algo. crypto. résistante aux attaques distantes.

Étape 5

Implémentation d'algo. crypto. résistante aux attaques locales.

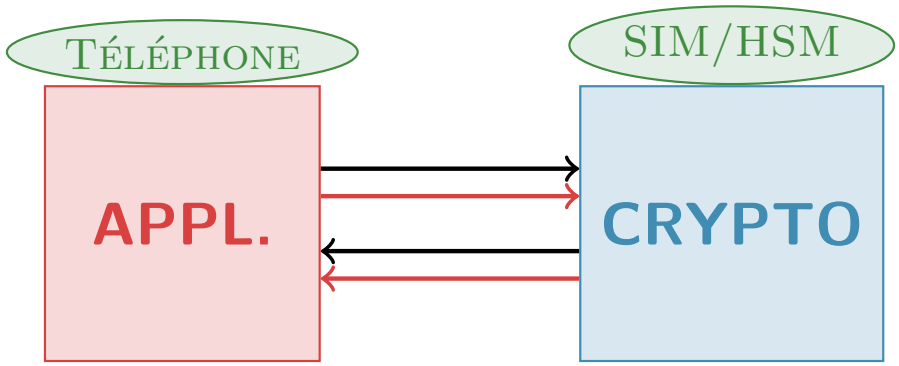
Sommaire de la session

Sommaire de la session



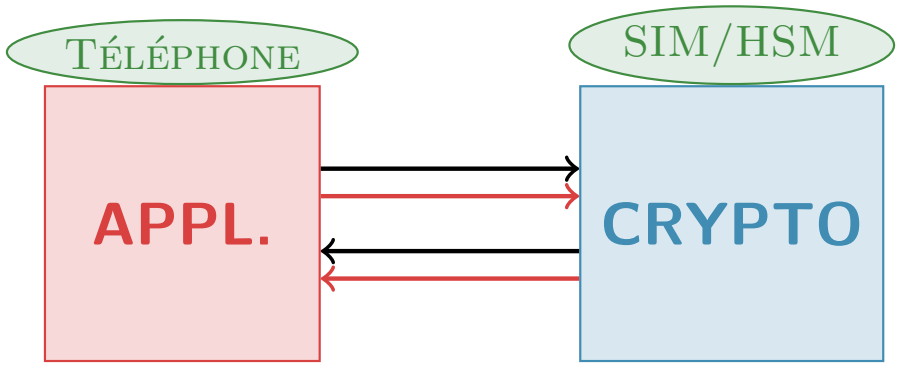
Cryptographie en mode *ressource*.

Cryptographie en mode *coupure*.

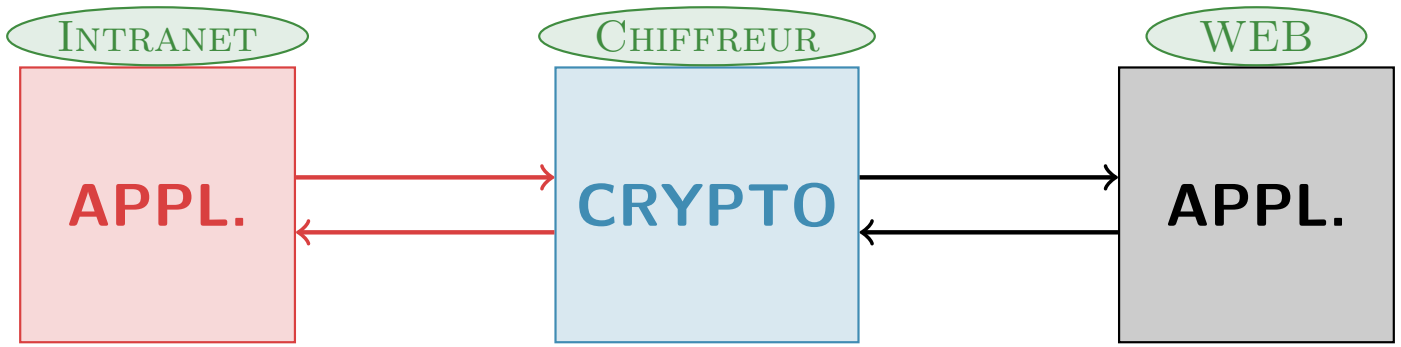


Cryptographie en mode *ressource*.

Cryptographie en mode *coupure*.



Cryptographie en mode *ressource*.



Cryptographie en mode *coupure*.

Différentes mémoires.

Différents types d'unité de calcul :

- processeur,
- micro-controlleur,
- FPGA (*Field Programmable Gate Array*),
- FPGA avec micro-controlleur intégré,
- ASIC (*Application-Specific Integrated Circuit*).

Un module peut être composé de plusieurs types (e.g. processeur avec accélérateur FPGA).

Qui fait quoi, qui parle à qui ? Comment ? Qui voit quoi ?

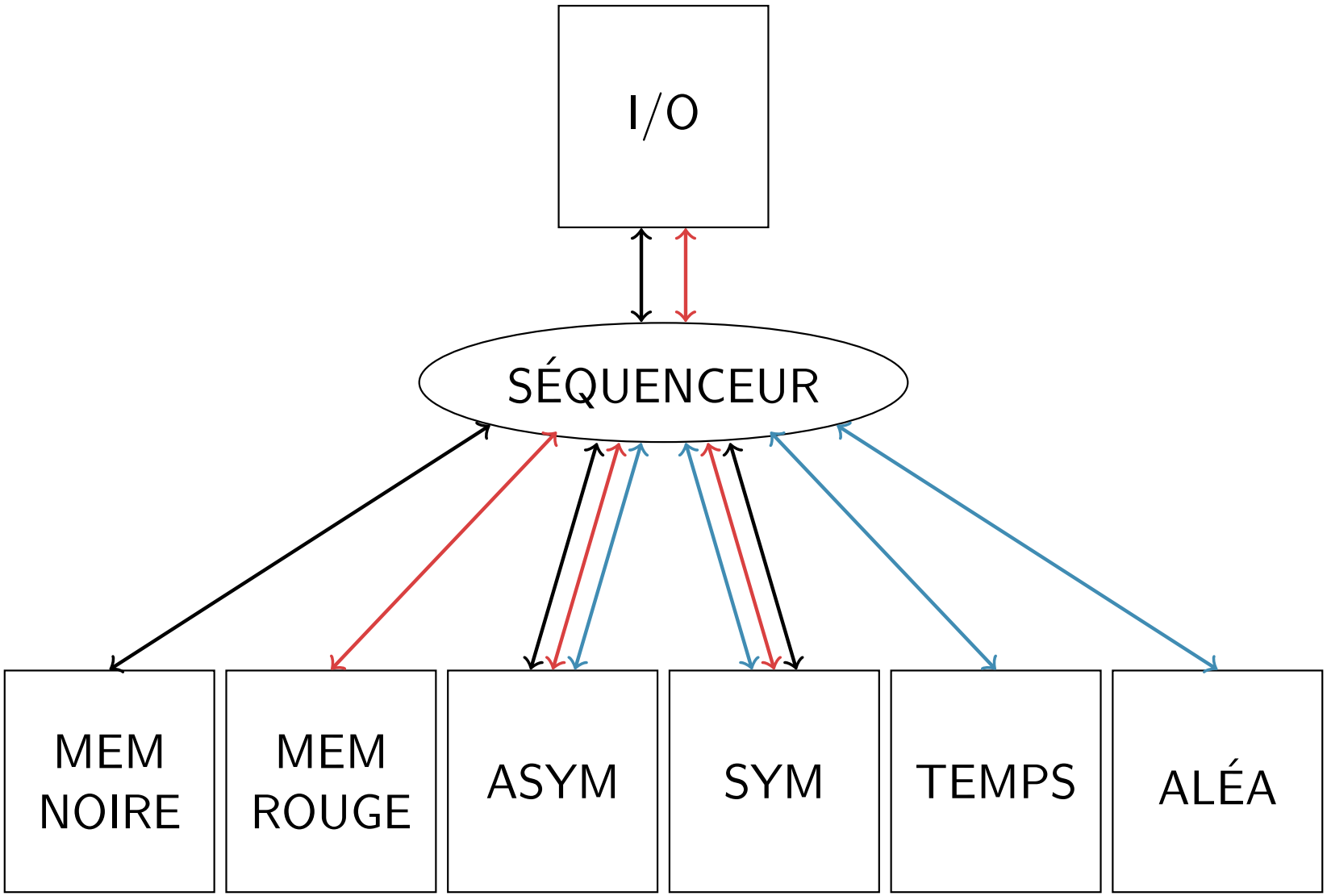
Évidemment

Taille, temps d'exécution, débit, coût.

- Rétro-compatibilité.
- Faible bande passante.
- Communications avec les autres éléments sporadiques (et non maîtrisées).
- Puissance ou énergie (électrique) limitée.
- Conditions climatiques extrêmes.
- Rayonnements ionisants (espace).
- ...

Plan

Module cryptographique
Vue générale



Briques cryptographiques

- Hachage
- Chiffrement symétriques
- Modes d'opération
- MAC

Implantations

- Bloc matériel total
- Accélération matérielle (e.g. instructions AES-NI)
- Implémentations logicielles

Briques cryptographiques

- Signature
- Chiffrement
- Authentification
- Crypto. basée sur l'identitée
- Chiffrement homomorphe

Implantations

- Bloc matériel seulement (rare)
- Accélérateur de calculs arithmétiques
- Implémentations logicielles

Génération d'aléa

- Source d'entropie
  - physique (e.g. anneaux d'oscillateurs)
  - autre (e.g. mouvements de souris)
- Pseudo-aléa
  - primitives cryptographiques (e.g. hachage)
  - *FSR (e.g. Mersenne twister)

Test de l'aléa

- Caractérisation et tests hors ligne
- Tests en ligne

Besoins

- cryptopériode
- validité de certificat
- protocoles de type *distance-bounding*

Heure de sécurité

- pas forément disponible selon la plateforme ciblée,
- doit être cohérente avec les autres éléments du système.

Adapter les solutions cryptographiques à la technologie disponible.
Il est préférable d'y avoir pensé lors de la conception au niveau système.

Besoins

- mémoire de travail (volatile)
- mémoire spécifique durcie (volatile ou non)
- mémoire exécutable (volatile ou non)
- mémoire de stockage (non volatile)

Technologies

- volatiles (RAM)
- persistantes (ROM,PROM,EEPROM,NVRAM,FLASH . . .)
- durcies (scrambling, effacement total . . .)

Attention aux problématiques d'effacement !

- Interface utilisateur
  - e.g. clavier sur carte à puce
- PUF (*Physical Unclonable Function*)
  - pour génération de clef ou d'aléa
- Cryptographie boîte blanche
  - stockage de clef dans le code
- Fusibles (OTP pour *One-Time Programmable*)
  - pour stockage, cycle de vie

Plan

Besoin de sécurité

- Protéger les données
- Protéger les secrets industriels
- Empêcher le piégeage

Deux arborescences de clefs

1. Clefs utilisées dans les services cryptographiques
   - ce pour quoi le système est mis en place
2. Clefs utilisées pour la sécurité locale
   - protection du secret industriel
   - garantie que les services sont bien mis en oeuvre

Menace

Piégeage du produit

- modifications des fonctionnalités du matériel
- exécution de code malicieux

Principe

1. Démarrage d'un petit coeur confiance
2. Lancement des autres modules par le coeur après vérification

Exemple

Chargement de bitstream d'un FPGA avec chiffrement et authentification.

Mémoire persistante non disponible en interne $\Rightarrow$ externe (donc vulnérable)
$\Longrightarrow$ protection des secrets avant stockage.

Protection

- Confidentialité
- Intégrité
  - intégrité temporelle,
  - intégrité spatiale.

Alors d'où proviennent les secrets de ces protections ?

- stockés en dur dans le code $\longrightarrow$ sécurité de type obfuscation
- de l'extérieur $\longrightarrow$ sécurité cryptographique

Intégrité temporelle

- Date comprise dans le bloc intègre.
- Ajout d'un compteur anti-rejeu.

Intégrité spatiale

- Intégration de l'adresse mémoire dans le calcul du motif d'intégrité.
- Intégrité par blocs de taille à déterminer (en fonctions des contraintes).
- Utilisation d'arbres de Merkle pour réduire la mémoire nécessaire.

Plan

- On a une carte avec des secrets.
- On veut les utiliser
- sans les compromettre i.e.
  - les secrets restent secrets → *confidentialité*
  - les secrets ne sont pas altérés. → *intégrité*

Analogie bancaire

On veut un coffre fort !

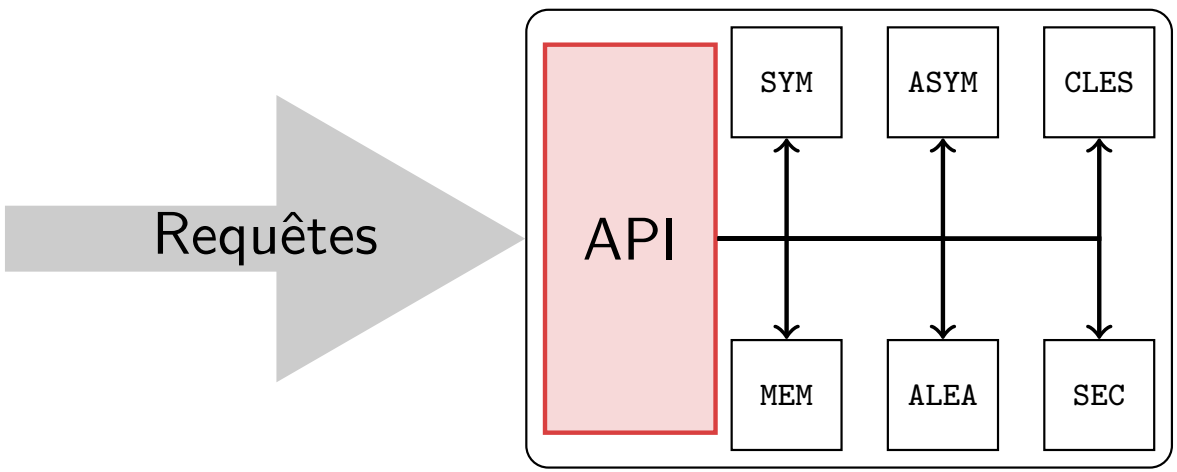
$$\text{infrastructure} + \text{règles d'accès}$$

$$\Downarrow$$

$$\text{politique de sécurité}$$

API de sécurité

Interface entre une zone sécurisée et le reste du monde.
Elle doit garantir la validité d'une ou plusieurs propriétés de sécurité pour toute suite d'appels à ses fonctions.



Propriété de sécurité classique

Un attaquant ne peut connaître que les secrets d'utilisateurs qu'il a corrompu.

Contraintes :

- besoin de standards flexibles,
- garantie des propriétés de sécurité.

Problème

Plus de flexibilité $\Rightarrow$ plus de difficultés à garantir la sécurité.

Exemple de PKCS#11

API de sécurité avec **70** fonctions (250 pages et 5 versions).

Une politique de sécurité pour une application cryptographique :

- Période de validité des certificats.
- Définition des droits de l'administrateur.
- Définition des droits des utilisateurs.
- Règles de manipulation des secrets :
  - impossible de supprimer le secret racine,
  - possibilité d'exporter certains secrets
  - ...

Tout ceci en fonction

1. de l'instant,
2. du type de secret,
3. de l'identité de l'utilisateur.

Plusieurs cycles (pouvant être indépendants)

- cycle de vie du système
- cycle de vie d'un élément (e.g. carte)
- cycle de vie de l'API

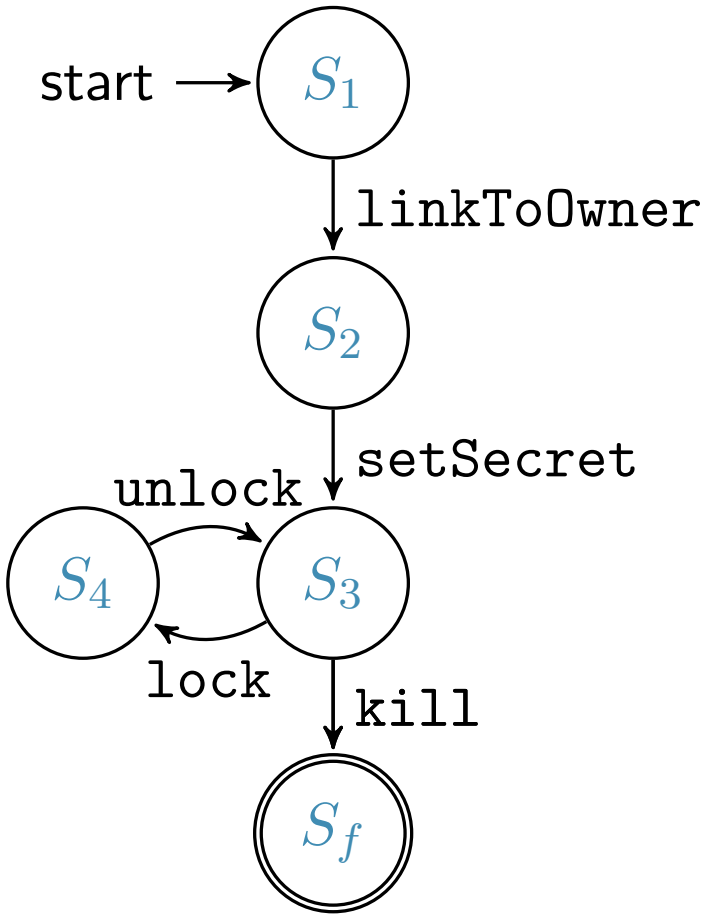
Changement de propriétaire d'une carte $\Rightarrow$ RAZ du cycle de l'API.
Renouvellement d'une carte $\Rightarrow$ RAZ du cycle de l'élément.

En général

Contient au moins les états :

1. Initialisation/Personnalisation (e.g. injection données personelles)
2. Utilisation
3. Fin de vie

À chaque état sa liste d'actions autorisées.



- Insertion des données biométriques seulement en S_1.
- La génération/injection du secret racine ne peut avoir lieu qu'en S_2.
- Utilisation des services uniquement dans l'état S_3.
- Dans l'état S_4 on peut uniquement déverouiller la carte.
- ...

- Reçoit les requêtes.
- Applique les règles du cycle de vie de l'API.
- Applique les règles de séquencement des sous-états
  - début/suite/fin du hachage
  - authentification avant un échange de clef DH
  - confirmation de l'échange de clef avant utilisation de celle-ci
  - ...

Partie de l'implémentation très critique et potentiellement complexe.

- Faire au plus simple.
- Être rigoureux et exhaustif.

Essentiel pour garantir que l'on se place bien dans le modèle des preuves cryptographiques (de protocoles ou autres).

Les données à protéger sont :

- des clefs,
- des données sensibles.

Mais on a aussi des certificats.

On a donc des données

- de sensibilités $\neq$,
- de tailles $\neq$,
- ayant des périodes de validité $\neq$,
- pour des usages $\neq$.

Différents types/structures pour différents objets contenant la valeur utile et des **attributs**.

Les attributs permettent de formaliser la politique de sécurité.

Cas multi-utilisateurs

Des attributs OWNER et/ou GROUP permettent de donner des droits quant à l'usage d'une clef.

Transport

Un attribut EXPORTABLE permet de savoir si l'on peut sortir une clef du coffre (chiffrée bien entendu).

Validité

Un attribut VALIDITY permet de donner une date limite de validité à un certificat.

Les propriétés de sécurité reposent sur le fait que l'API sache quel utilisateur est connecté.

La sécurité ne peut jamais entièrement se baser sur de la cryptographie !

- Vol de carte/badge/dongle.
- Récupération de mot de passe.
- Piratage des lecteurs d'empreintes.
- ...

Et en cas de perte d'un élément d'authentification ?

Authentification forte

Renforcer l'assurance que la bonne personne se connecte.

Combinaison de techniques :

- dispositif électronique (clef, carte, dongle . . .),
- mot de passe (ou code PIN),
- envoi d'un code par messagerie (SMS, internet),
- biométrie (empreinte digitale, rétine, ECG)
- . . .

Compromis entre complexité de mise en oeuvre et sécurité.
Permet un mode dégradé en cas de perte.

L'authentification permet de déterminer

- l'ensemble des objets possédés,
- le rôle de l'utilisateur.

En fonction, l'utilisateur aura différents droits.

- Droit de créer/supprimer
  - un utilisateur,
  - un objet possedé,
  - un objet autre.
- Droit d'exporter/importer
  - un objet possedé,
  - un objet autre.
- Droit d'effectuer certaines opérations spéciales
  - transition entre deux états du cycle de vie,
  - mise à jour.

Rappel des faits

Le composant pirate répond `OK` pour le code pin à la place de la carte.

Analyse

La carte n'a donc pas validé de code PIN avant la transaction.

Solution possible

Ajout d'un séquenceur qui vérifie qu'un code PIN a été testé avant une transaction.

Rappel des faits

Le composant pirate répond `OK` pour le code pin à la place de la carte.

Analyse

La carte n'a donc pas validé de code PIN avant la transaction.

Solution possible

Ajout d'un séquenceur qui vérifie qu'un code PIN a été testé **avec succès** avant une transaction.

Rappel des faits

Le composant pirate répond `OK` pour le code pin à la place de la carte.

Analyse

La carte n'a donc pas validé de code PIN avant la transaction.

Solution possible

Ajout d'un séquenceur qui vérifie qu'un code PIN a été testé **avec succès** avant une transaction.

Variantes liées aux contraintes commerciales

- Ajout d'un compteur : max. 3 opérations sans code PIN par mois.
- Ajout d'un plafond : max. 100 euros dépensés avant code PIN obligatoire.

Plan

La génération de clef peut avoir lieu :

- dans le périmètre de sécurité.
- hors du périmètre (import dans le périmètre par la suite).

Attention dans les deux cas !

Interne :

- Disposer d'un bon aléa.

Externe :

- Aucune maîtrise de l'aléa utilisé.
- Aucune garantie de non diffusion de la clef.

Périmètre de sécurité

Contient toutes les implémentations de l'API qui sont de confiance.

Key Wrap

Stockage/transport de clefs hors périmètre de sécurité ⇒ besoin

- de confidentialité,
- d'intégrité/d'authenticité.

Handles

Permet l'utilisation des clefs sans les manipuler directement (identifiants).

Cryptopériode

- usure de la clef,
- impact d'une compromission.

Gestion des clefs

Exemple de non utilisation d'un handle

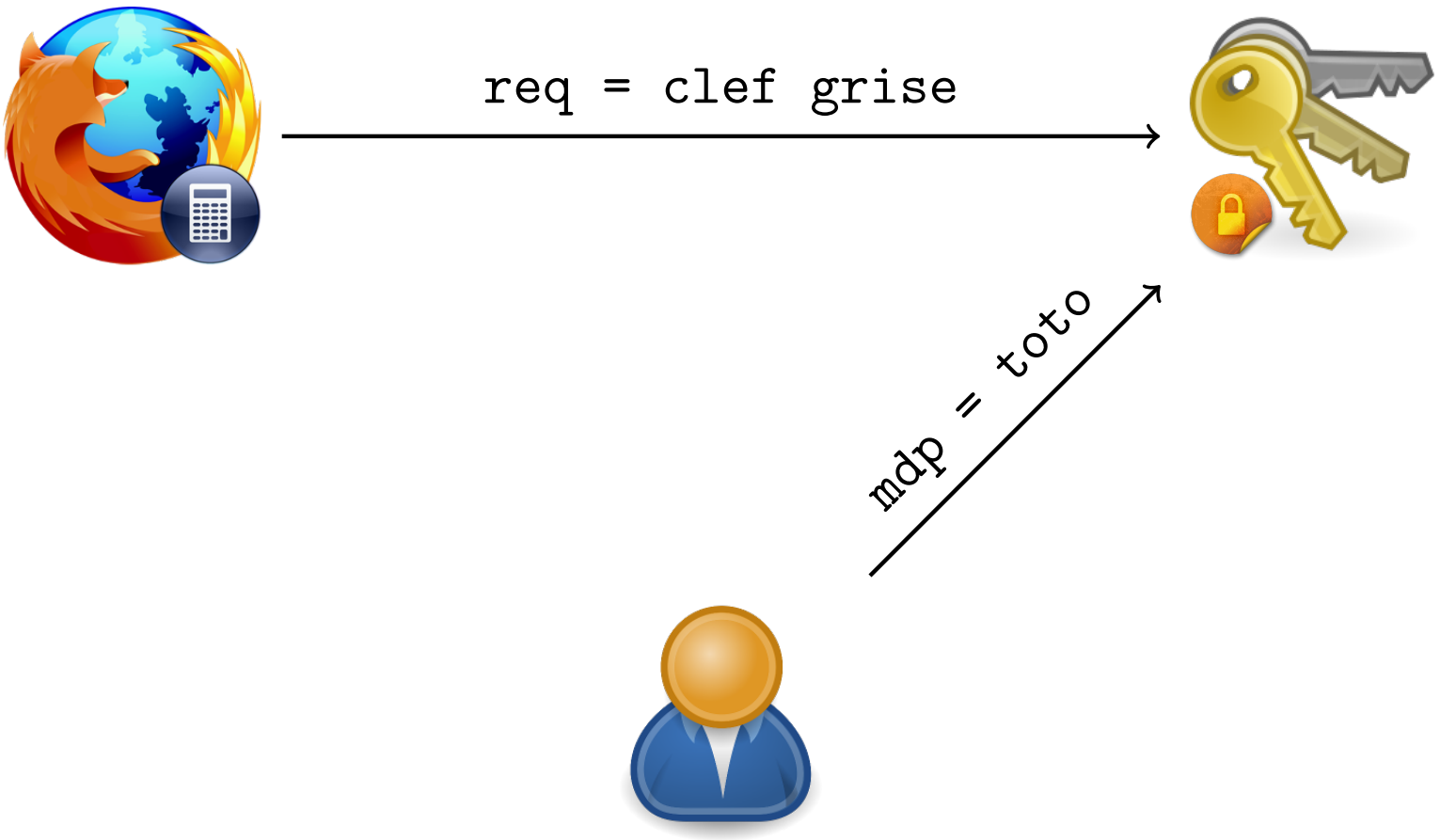
Gestion des clefs

Exemple de non utilisation d'un handle

`req = clef grise`

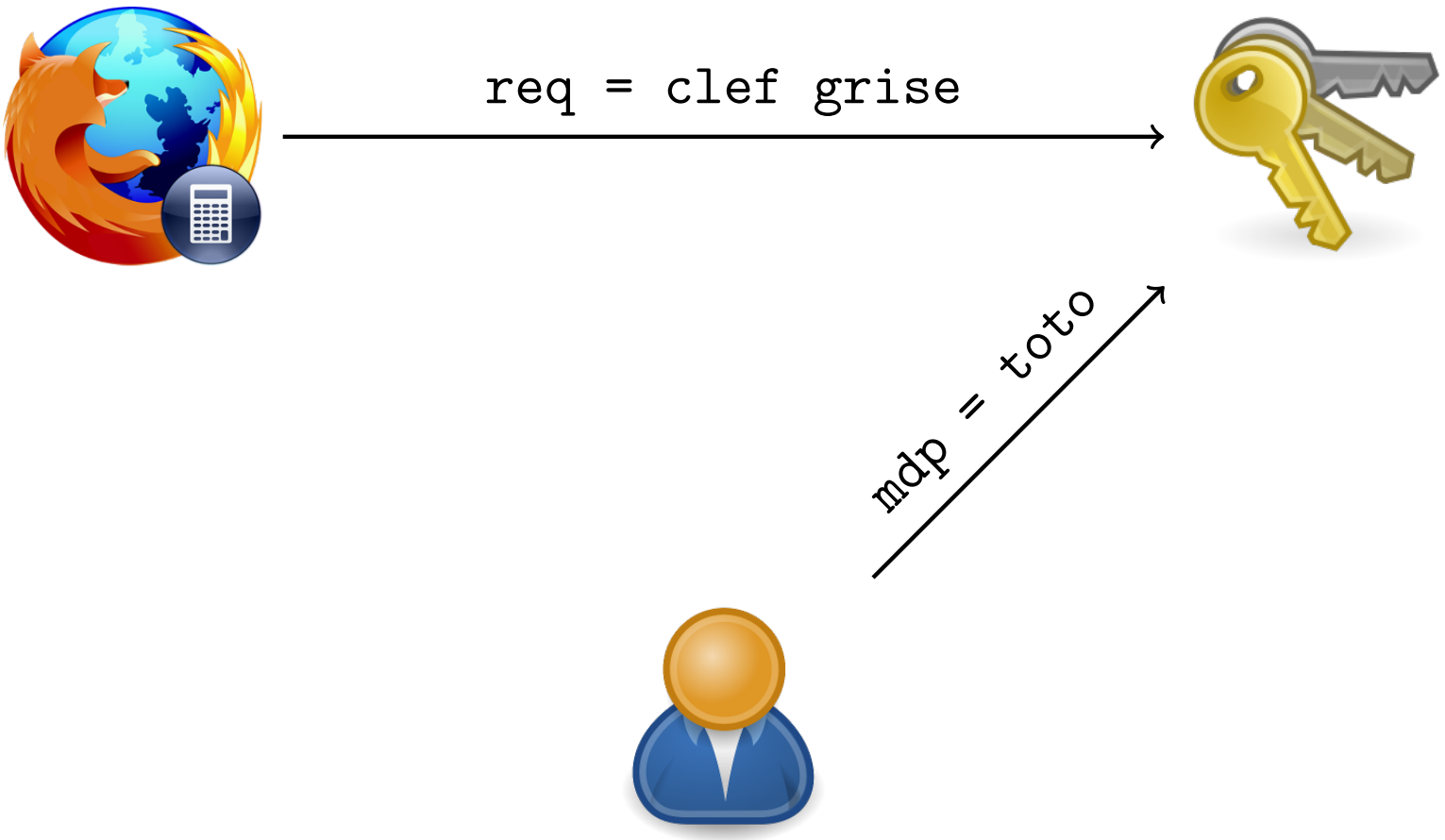
Gestion des clefs

Exemple de non utilisation d'un handle



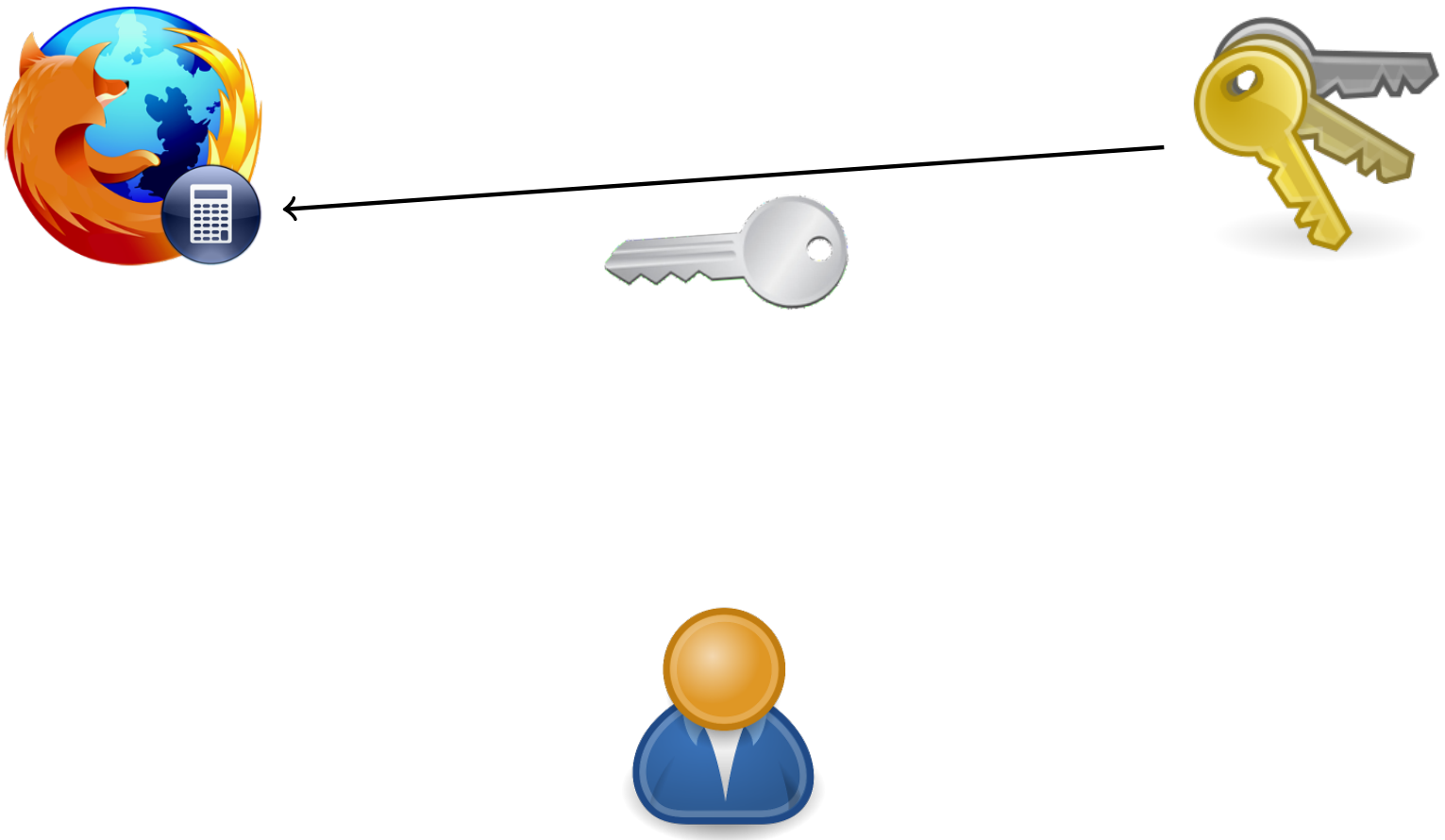
Gestion des clefs

Exemple de non utilisation d'un handle



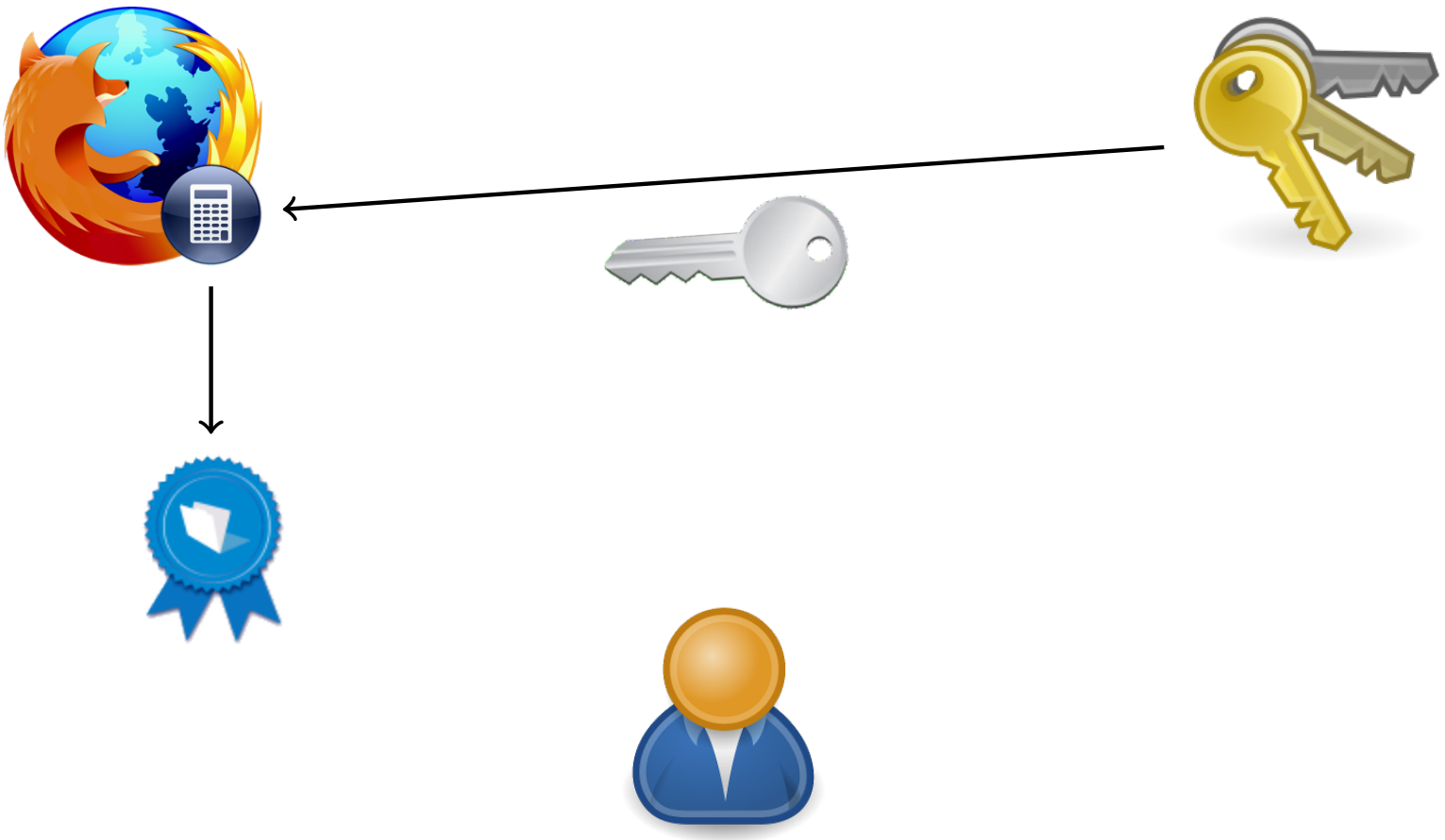
Gestion des clefs

Exemple de non utilisation d'un handle



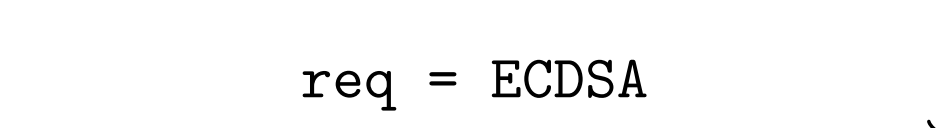
Gestion des clefs

Exemple de non utilisation d'un handle



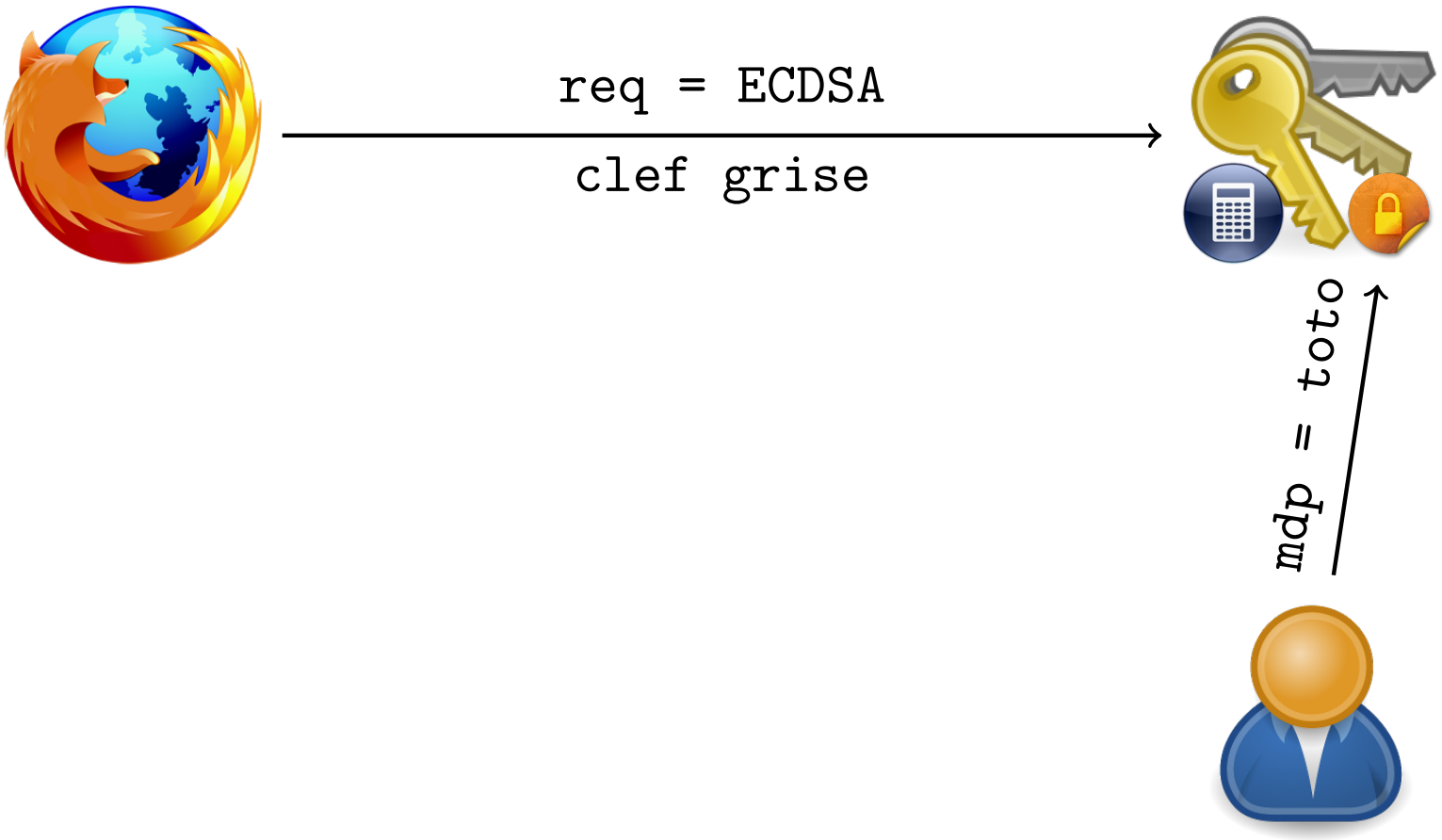
Gestion des clefs

Exemple d'utilisation d'un handle


req = ECDSA
clef grise

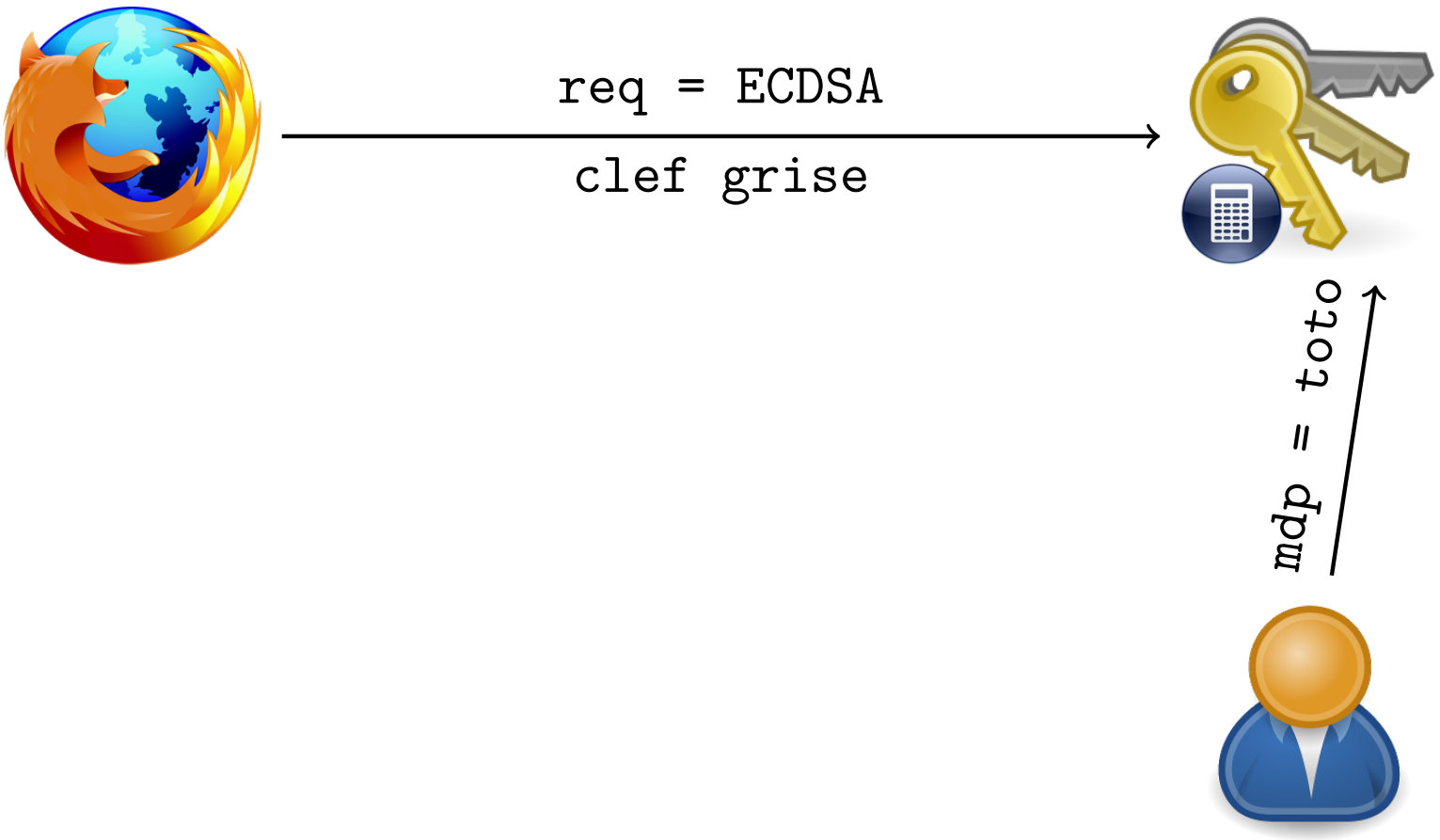
Gestion des clefs

Exemple d'utilisation d'un handle



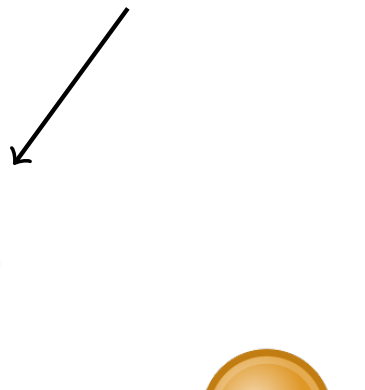
Gestion des clefs

Exemple d'utilisation d'un handle



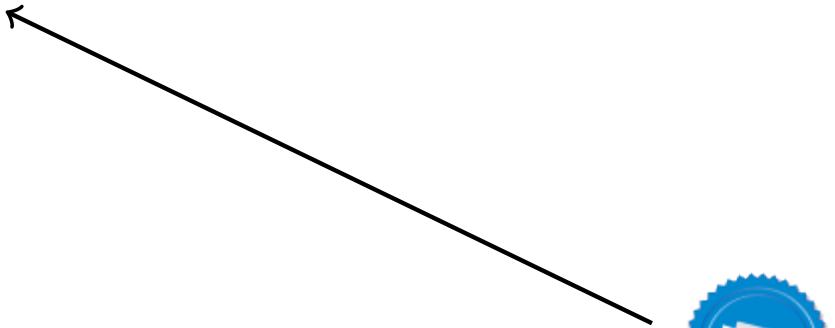
Gestion des clefs

Exemple d'utilisation d'un handle



Gestion des clefs

Exemple d'utilisation d'un handle



Bonne pratique

Différencier (fortement) les éléments sensibles.

Mauvais exemple

Attribut *sensible*.

1. K crée déclarée *sensible*
2. Export(K) refusé
3. Attribut *sensible* désactivé
4. Export(K) autorisé

1. K crée
2. Export(K) autorisé
3. K déclarée *sensible*

Leçon

Type défini à la création d'un élément.
Types/attributs liés à la sécurité non modifiables (par l'utilisateur).

Buts

1. Limiter l'impact d'une compromission.
2. Éviter de tendre le bâton à l'attaquant.

Buts

1. Limiter l'impact d'une compromission.
2. Éviter de tendre le bâton à l'attaquant.

Exemple : sortir une clef en clair

Clef K qui protège en confidentialité les clefs et les messages.

1. $C = E(K, K')$
2. $E^{-1}(K, C) \rightarrow K'$

Exemple : injecter une clef maîtrisée

Clef K chiffrement + import de clefs.

1. $C = E(K, 0)$
2. $K' = \texttt{Import}(K, C) = E^{-1}(K, C) \rightarrow K' = 0$

Il existe d'autres attaques plus ou moins réalisables en pratique :

- key conjuring (génération d de clefs non autorisées),
- key binding (découpe de clefs longues en sous-clefs),
- injection clef troyenne.

À retenir

Une bonne sécurité des clefs implique

1. de contrôler fortement les opérations effectuées,
2. de séparer les clefs selon leurs usages.

C'est une grande source de vulnérabilités (on n'a pourtant pas fait de cryptanalyse).

Plan

API C utilisée par :

- autorités de certification,
- HSM (*Hardware Security Module*)
- GnuTLS,
- OpenSSL/SSH/SC,
- Mozilla,
- OpenVPN,
- Truecrypt,
- ...

Implémentation conforme PKCS #11

⇳

implémente un sous-ensemble de règles

- Model checking
  - TOOKAN par Graham Steel.
  - Analyse l'implémentation pour déterminer le sous-ensemble de règles implantées,
  - fourni le modèle correspondant à un model checker,
  - fait tourner le checker jusqu'à trouver une faille.
  - Très peu d'implémentations résistantes ...

- Preuve formelle API générique proposée à CCS en 2012
  - gestion des clefs symétriques,
  - prise en compte de la révocation.

Comment s'assurer que l'implémentation finale est bien conforme aux hypothèses de la preuve ?

Problématique de la certification (Critères Communs) :

- Certification donnée par l'ANSSI suite à l'analyse par un CESTI.
- On ne peut plus toucher au produit une fois obtenue.
- EAL4 et plus : demande une conception méthodique.
- EAL5 et plus : nécessite des méthodes (semi-)formelles.

Contraintes coût/délai/performances :

- manque de temps et parfois de compétences,

$\Rightarrow$ utilisation de produits sur étagère fait par des spécialistes

- HSM (PCI, microSD),
- TPM *Trusted Platform Module* (composant pour carte mère),
- Globull (disque avec coeur crypto de chez Bull).

ANSSI : *Agence Nationale de la Sécurité des Systèmes d'Information*.
CESTI : *Centre d'Évaluation de la Sécurité des Technologies de l'Information*.

À retenir

Messages

- La mise en œuvre d'un produit de sécurité est complexe et ne peut être mené seul avec succès.
- Il est essentiel de prendre le temps de réfléchir en amont et de spécifier clairement et précisément les choses.

Bonnes pratiques

- Faire au plus simple (mais pas au plus simpliste).
- Découper le problème en sous-problèmes bien définis et plus simples.
- Prendre des précautions (tests, cloisonnement . . .).