Sécurité des implémentations pour la cryptographie

Partie 6 : Résistance aux attaques locales invasives

Benoît Gérard

23 janvier 2018

Plan du cours

Étape 1

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

Étape 2

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

Étape 3

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

Étape 4

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

Sommaire de la session

Sommaire de la session

Plan

Objectif

Modifier le comportement nominal du composant pour attaquer.

Différentes techniques d'injection :

- laser (nécessite en général une décapsulation),
- champ magnétique,
- glitch sur l'horloge,
- sous-alimentation,
- glitch sur l'alimentation,
- conditions extérieures.

Injection de fautes
Effets

Différentes conséquences dont :

- bloquer un générateur d'aléa dans un état,
- modifier la valeur d'une cellule mémoire,
- écrire une donnée erronée dans un registre.

Injection de fautes
Différents modèles

Durée

- faute transitive,
- faute permanente.

Rayon

- modification d'un bit,
- modification d'un mot.

Contrôle

- écriture d'une donnée choisie,
- écriture d'une donnée aléatoire,
- mise à 0 ou 1.

Plan

RSA : $N = p\,q$ et

$$m = c^d \mod N$$

RSA-CRT :

$$m_p = c^{d \mod p-1} \mod p \quad , \quad m_q = c^{d \mod q-1} \mod q$$

Différentes techniques de recombinaison

Classique :

$$m = (m_q\,p^{-1} \mod q)\,p + (m_p\,q^{-1} \mod p)\,q \mod N$$

Garner :

$$m = m_q + q(q^{-1}(m_p - m_q) \mod p)$$

Faute : $m_p \longrightarrow \hat{m}_p$

$$\begin{aligned} m &= (m_q\, p^{-1} \mod q)\, p + (m_p\, q^{-1} \mod p)\, q \mod N \\ \hat{m} &= (m_q\, p^{-1} \mod q)\, p + (\hat{m}_p\, q^{-1} \mod p)\, q \mod N \end{aligned}$$

$$m - \hat{m} = \big((m_p - \hat{m}_p)\, q^{-1} \mod p\big)\, q \mod N$$

On a un multiple de q : on retrouve q avec

$$q = \mathrm{pgcd}(N, m - \hat{m})$$

Faute : $m_p \longrightarrow \hat{m}_p$

$$\begin{aligned} m &= m_q + q(q^{-1}(m_p - m_q) \mod p) \\ \hat{m} &= m_q + q(q^{-1}(\hat{m}_p - m_q) \mod p) \end{aligned}$$

$$m - \hat{m} = q\left(q^{-1}(m_p - \hat{m}_p) \mod p\right)$$

On a un multiple de q : on retrouve q avec

$$q = \mathrm{pgcd}(N, m - \hat{m})$$

Faute : $m_q \longrightarrow \hat{m}_q$

$$\begin{aligned} m &= m_q + q(q^{-1}(m_p - m_q) \mod p) \\ \hat{m} &= \hat{m}_q + q(q^{-1}(m_p - \hat{m}_q) \mod p) \end{aligned}$$

$$\begin{aligned} m - \hat{m} &= m_q - \hat{m}_q + q\left(q^{-1}(\hat{m}_q - m_q) \mod p\right) \\ &= 0 \mod p \end{aligned}$$

On a un multiple de p : on retrouve p avec

$$p = \mathrm{pgcd}(N, m - \hat{m})$$

Il existe des contre-mesures aux attaques précédentes.

On peut attaquer avec plus de calculs fautés ou plus de fautes durant un calcul.

Mais il existe aussi une attaque en faute sur N :

$$\begin{aligned} m &= (m_q\, p^{-1} \mod q)\, p + (m_p\, q^{-1} \mod p)\, q \mod N \\ \hat{m} &= (m_q\, p^{-1} \mod q)\, p + (m_p\, q^{-1} \mod p)\, q \mod \hat{N} \end{aligned}$$

On peut retrouver la factorisation de N par réduction de réseaux.

Attaques sur RSA

Safe-error sur Square and Multiply + instruction inutile

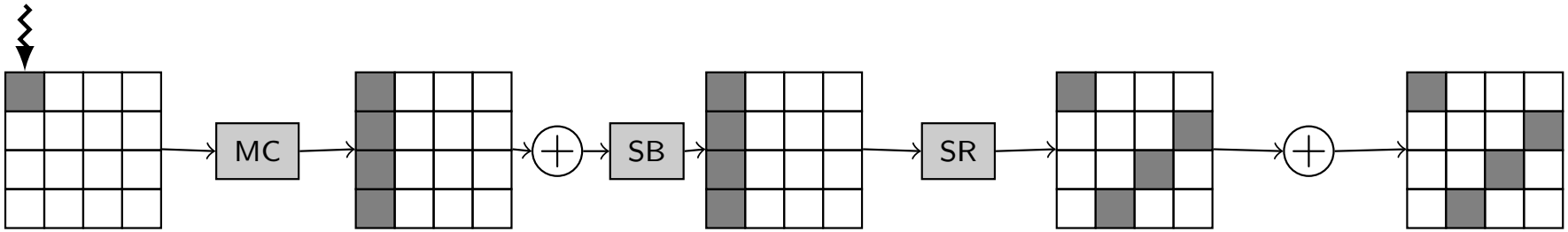
```
BIGINT modExp(BIGINT m, bool d[]) {
  BIGINT t = m, z = m;
  for ( INT i ... ) {
    t = t * t;
    if ( d[i] == 1 )
    t = t + m;
    else
    z = z + m;
  }
  return t;
}
```

1. Injection d'une faute lors de l'addition à l'étape i.
2. Si le résultat est bon alors d[i] vaut 0,
3. sinon d[i] vaut 1.

Plan

Attaques sur AES

Faute sur le tour 9 (Piret et Quisquater)

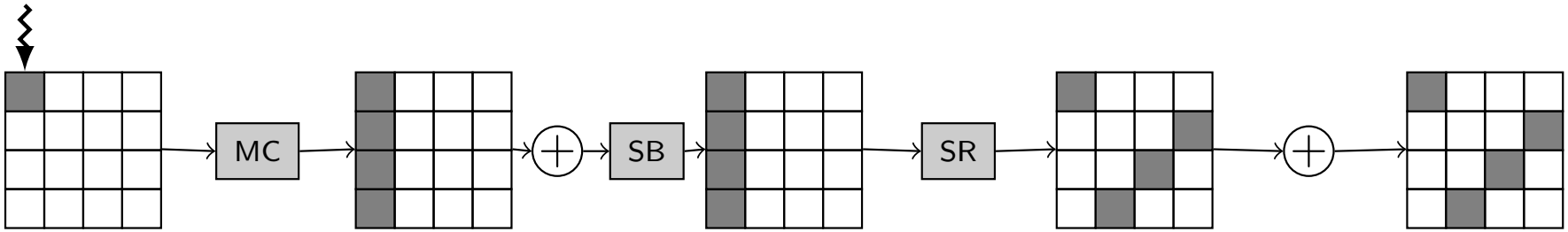


Case noire : il y a une différence entre deux exécutions.

Case blanche : il n'y a pas de différence entre deux exécutions.

Attaques sur AES

Faute sur le tour 9 (Piret et Quisquater)



Seulement 2^8 valides

sur 2^{32}.

MixColumn est linéaire :

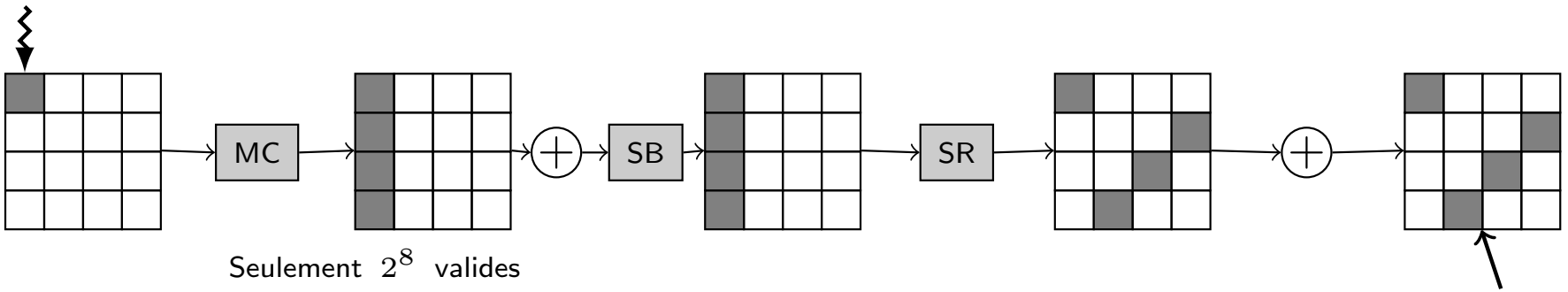
$$\texttt{MC}(P) \oplus \texttt{MC}(P \oplus E) = \texttt{MC}(E).$$

Les différences possibles sont donc :

$$\{\texttt{MC}(0), \texttt{MC}(1), \texttt{MC}(2), \ldots, \texttt{MC}(255)\}.$$

Attaques sur AES

Faute sur le tour 9 (Piret et Quisquater)



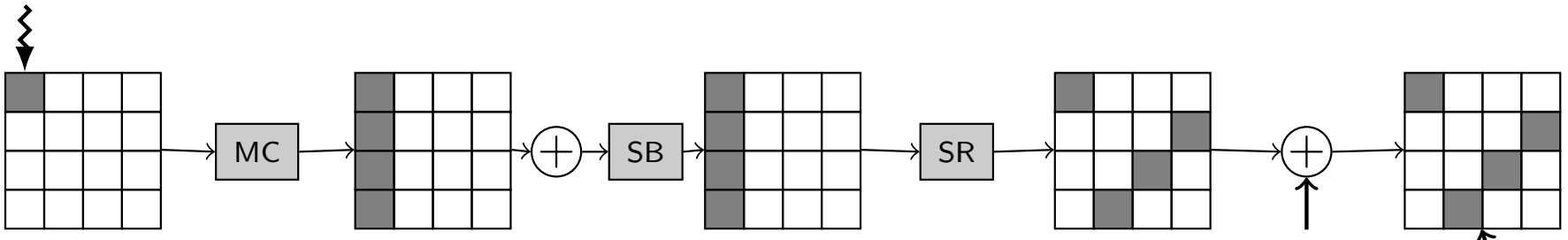
Seulement 2^8 valides

sur 2^{32}.

on connait les chiffrés

Attaques sur AES

Faute sur le tour 9 (Piret et Quisquater)



Seulement 2^8 valides

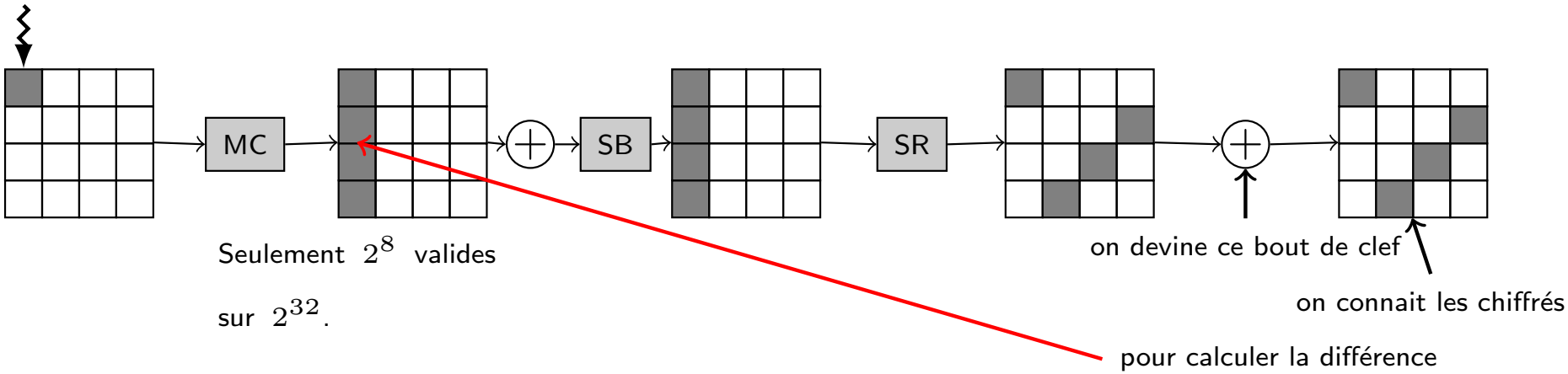
sur 2^{32}.

on devine ce bout de clef

on connait les chiffrés

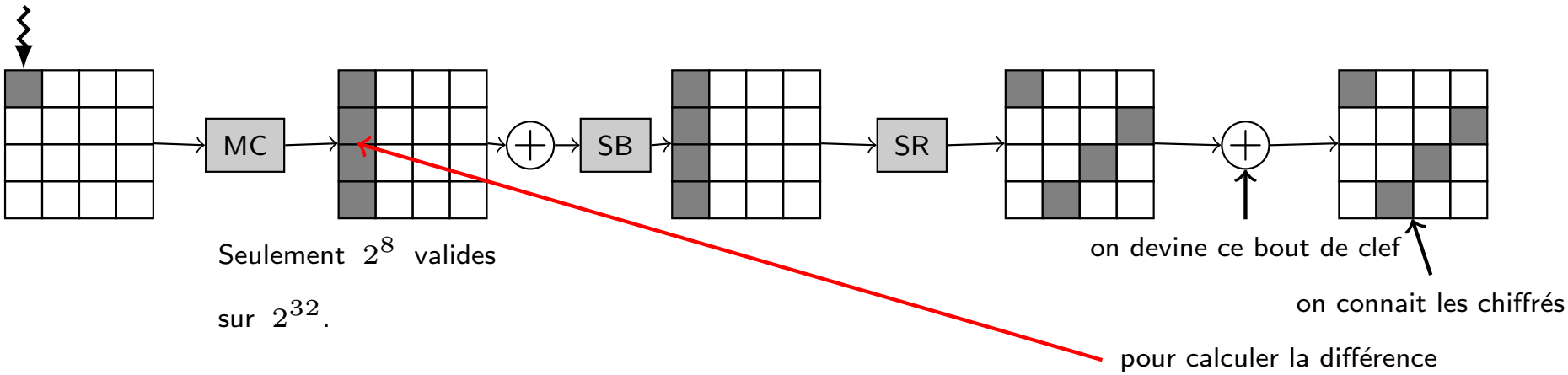
Attaques sur AES

Faute sur le tour 9 (Piret et Quisquater)



Attaques sur AES

Faute sur le tour 9 (Piret et Quisquater)

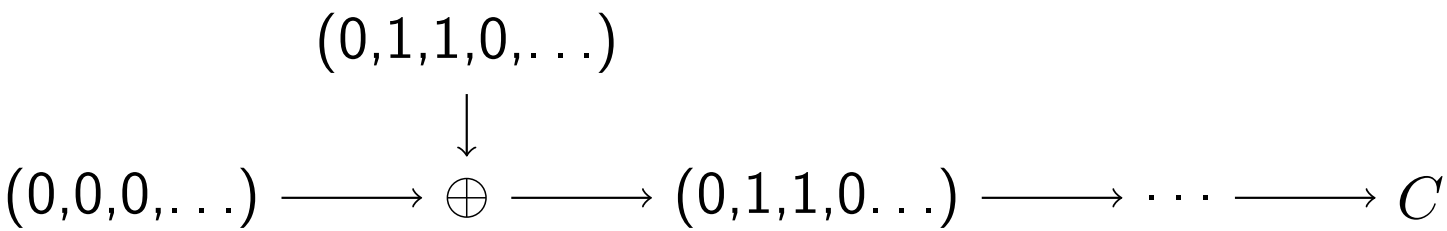


1. On précalcule l'ensemble des 2^8 "bonnes valeurs".
   - bonne clef : la différence appartient au bon ensemble,
   - mauvaise clef : la différence est *aléatoire* (et dnoc souvent hors de l'ensemble).
2. Il faut 2 fautes par colonne pour retrouver toute la clef.
3. On peut éviter de tester 2^{32} clefs car les calculs sont indépendents d'un octet à l'autre.

Modèle de faute

Collage à 0 d'un bit.

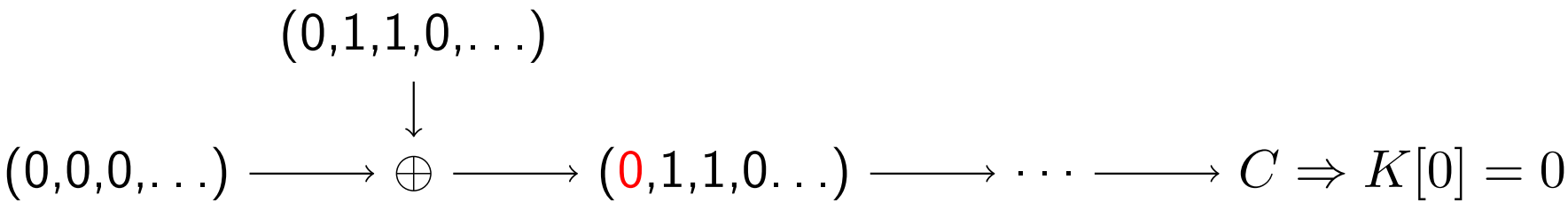
- Clair $P = (0, 0, \ldots, 0)$.
- Attaque un bit après XOR de la première sous-clef.



Modèle de faute

Collage à 0 d'un bit.

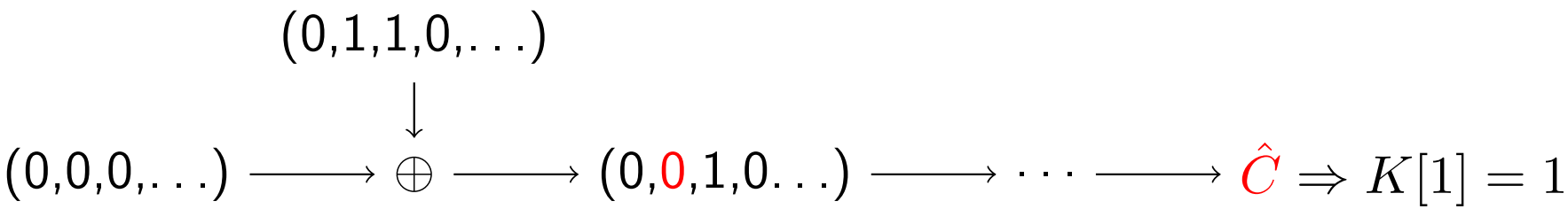
- Clair $P = (0, 0, \ldots, 0)$.
- Attaque un bit après XOR de la première sous-clef.



Modèle de faute

Collage à 0 d'un bit.

- Clair $P = (0, 0, \ldots, 0)$.
- Attaque un bit après XOR de la première sous-clef.



Plan

Dans un protocole de signature :

- la génération de la signature utilise un secret,
- la vérification de la signature utilise uniquement des données publiques.

On protège donc **uniquement la génération de signature** contre

- les attaques par canaux auxiliaires,
- les attaques en fautes.

Protection contre les canaux-auxiliaires

Vu au cours précédent.

Protection contre l'injection de fautes

L'attaque fonctionne si on récupère la signature erronée.

1. $s = sign(ks, m)$
2. if $verif(kp, m, s) ==$ `ok`
3. renvoyer s
4. sinon renvoyer `erreur`.

Attaque présentée en 2012 : combinaison faute/mesure de courant.

- Injection d'une faute à la signature $\rightarrow s' \neq s$.
- La vérification échoue donc on ne récupère pas s'.
- Durant la vérification on a mesuré la consommation de courant.
- Avec une attaque de type horizontale on récupère l'information dont on a besoin sur s' pour l'attaque en faute.
- C'est gagné !

Dans ce contexte, la vérification de signature est amenée à manipuler des données sensibles : s'.

Contre-mesures
Quelques grands principes

Il existe plusieurs types de contre-mesures toutes coûteuses.

- Répéter le calcul et vérifier que l'on obtient la même chose
  - ne résiste pas aux fautes permanentes ou multiples temporelles.
- Dupliquer (ou plus) le bloc matériel effectuant le calcul
  - ne résiste pas aux fautes multiples spatiales.
- Intégrer des vérifications en cours de calcul
  - différentes façons de faire.

Exemple d'idée : utilisation de RNS (Residue Number System).

- calculs effectués en parallèle sur des petits modules,
- résultat reconstruit à la fin (CRT),
- ajout de modules (redondance) pour détecter des erreurs/fautes.

Contre-mesure

Attention à aussi se protéger contre

Encore une fois : pas besoin d'aller attaquer la cryptographie.

On peut :

- sauter une/des instruction(s),
  - modification du program counter,
  - mise à 0 de l'opcode,
- modifier une valeur de retour,
  - mise d'un code d'erreur à 0 (ou juste modification),
  - mise à 0 du retour d'une comparaison.

Pour retrouver le “code” d’un composant :

1. découper les différentes “couches” du composant,
2. photographier au microscope électronique les lamelles,
3. analyser les images pour retrouver les transistors,
4. regrouper les transistors en portes logiques,
5. regrouper les portes en blocs puis en déduire les algorithmes.

On peut aussi juste retrouver le contenu d’une mémoire.

Attaque coûteuse en temps et matériel mais automatisable en partie.

Utilisation d'un FIB (*Focused Ion Beam* : outil d'analyse de défaillance)

- faisceau d'ion,
- atmosphère gazeuse modifiable.

Permet de :

- faire des "trous",
- déposer du métal.

On peut donc modifier le circuit pour

- altérer son fonctionnement,
- lire des bits qui transitent en interne.

Il existe là aussi des contre-mesures.

- Enfouir les “fils” sensibles.
- Résine de protection (attention à la dissipation de la chaleur).
- Ajouter une grille de détection d'intrusion.
- Mélanger les données (mémoire, bus).
- ...

À retenir

Messages

- Un attaquant motivé peut être très puissant.
- Savoir contre qui on se protège permet de mieux envisager les risques.

Bonnes pratiques

- Essayer de détecter les erreurs si l'attaquant peut être local
  - utilisation de codes correcteurs / bits de parité,
  - duplication du circuit ou du calcul,
  - vérification du résultat.