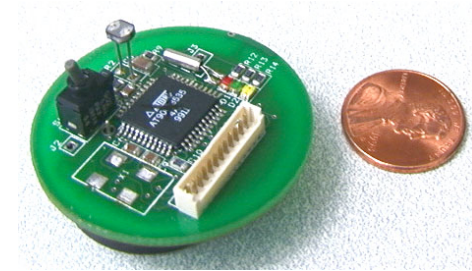


quelques problèmes de sécurité dans les réseaux de capteurs

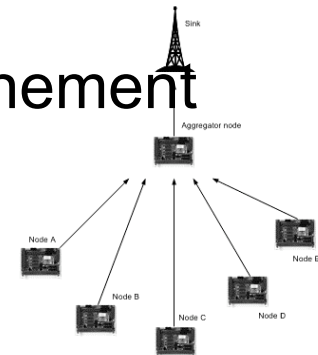
Marine MINIER
Laboratoire CITI
INSA de Lyon

Introduction



■ Réseaux sans fil multi-sauts

- ☐ utilisés en général pour monitorer un environnement
- ☐ Remontée des données à un nœud « puit »



■ Capteurs :

- ☐ Petites unités avec peu de ressources (énergie, calculs,..)
- ☐ Il faut donc préserver l'énergie (en calculs et en transmission) et le stockage des données et du code



Quelques attaques sur les réseaux de capteurs

- Ecoutes passives : mise à mal de la confidentialité
- Falsifications : mise à mal de l'intégrité des données
 - Modifications, rejeux, envoi de bruits
- Déni de service : liée à la qualité de service
- Compromission de nœuds par capture : mise à mal de l'authentification



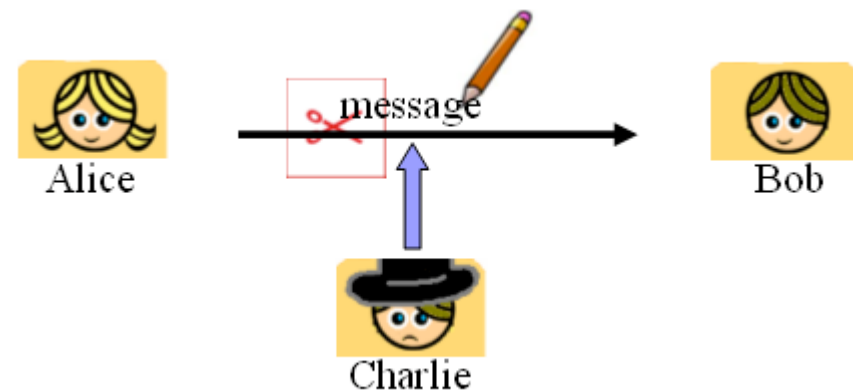
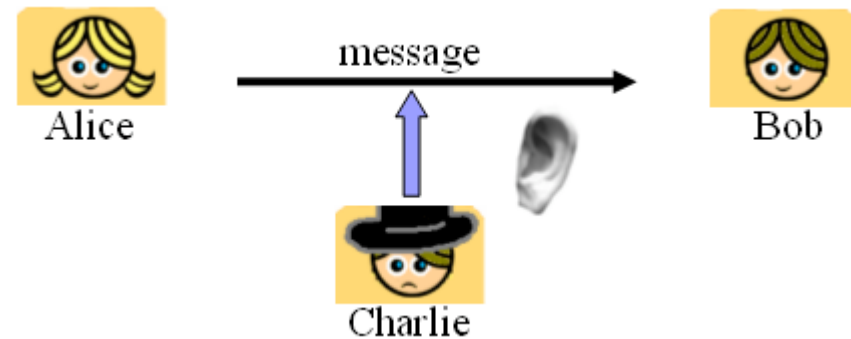
Quelles solutions apportées ?

- Des protocoles de sécurité doivent donc permettre de garantir les propriétés suivantes :
 - Opération de routage (authentification, intégrité)
 - Confidentialité des données échangées (requis dans les applications sensibles)
 - Pas de nœuds malicieux (authentification sécurisée) ou des moyens rapides de les détecter (localisation, révocation)

Ce que garantit la cryptographie

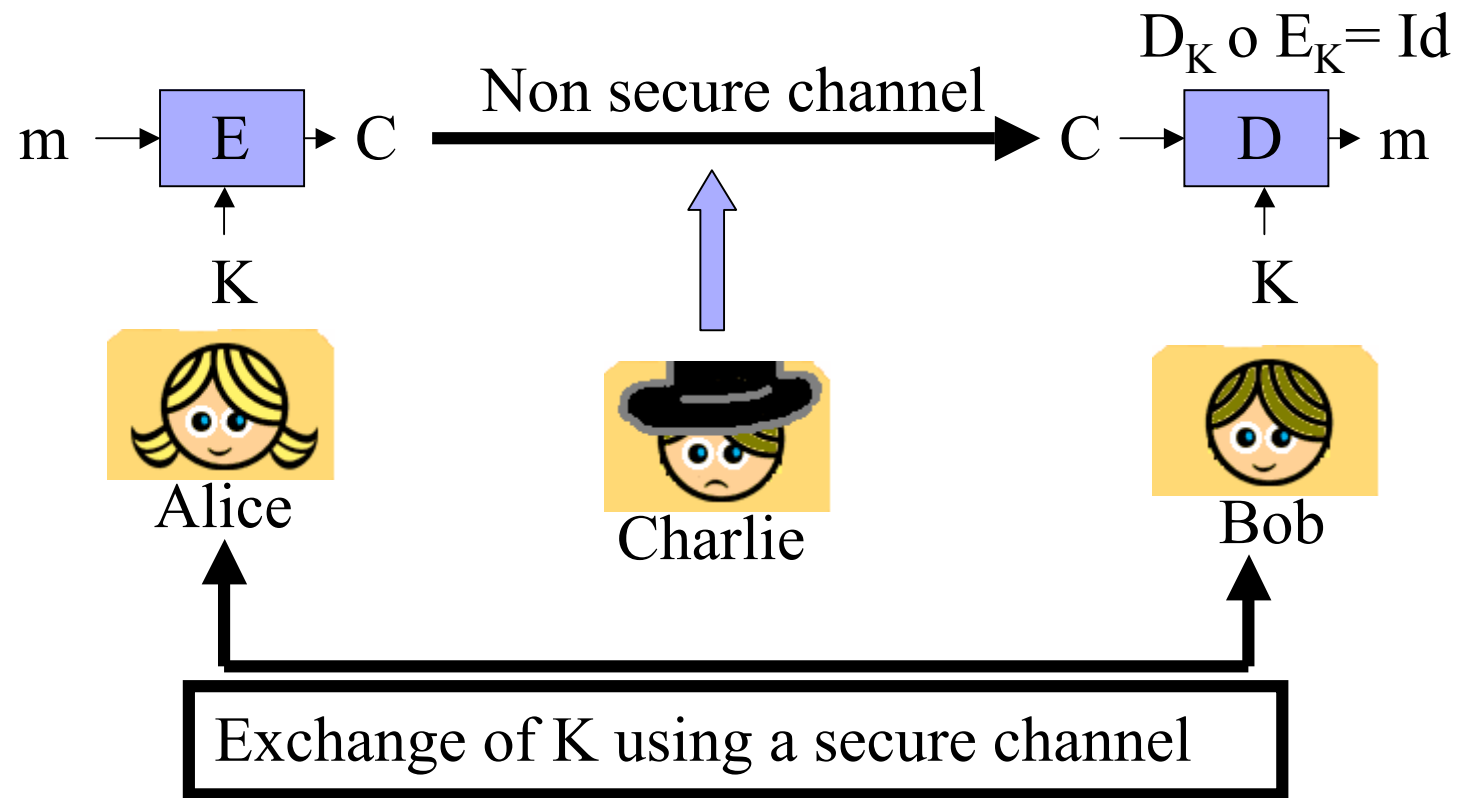
■ Permet de préserver

- ☐ Confidentialité (chiffrement)
- ☐ Intégrité (hachage)
- ☐ Authenticité (authentification, identification)
- ☐ Signature (non répudiation)



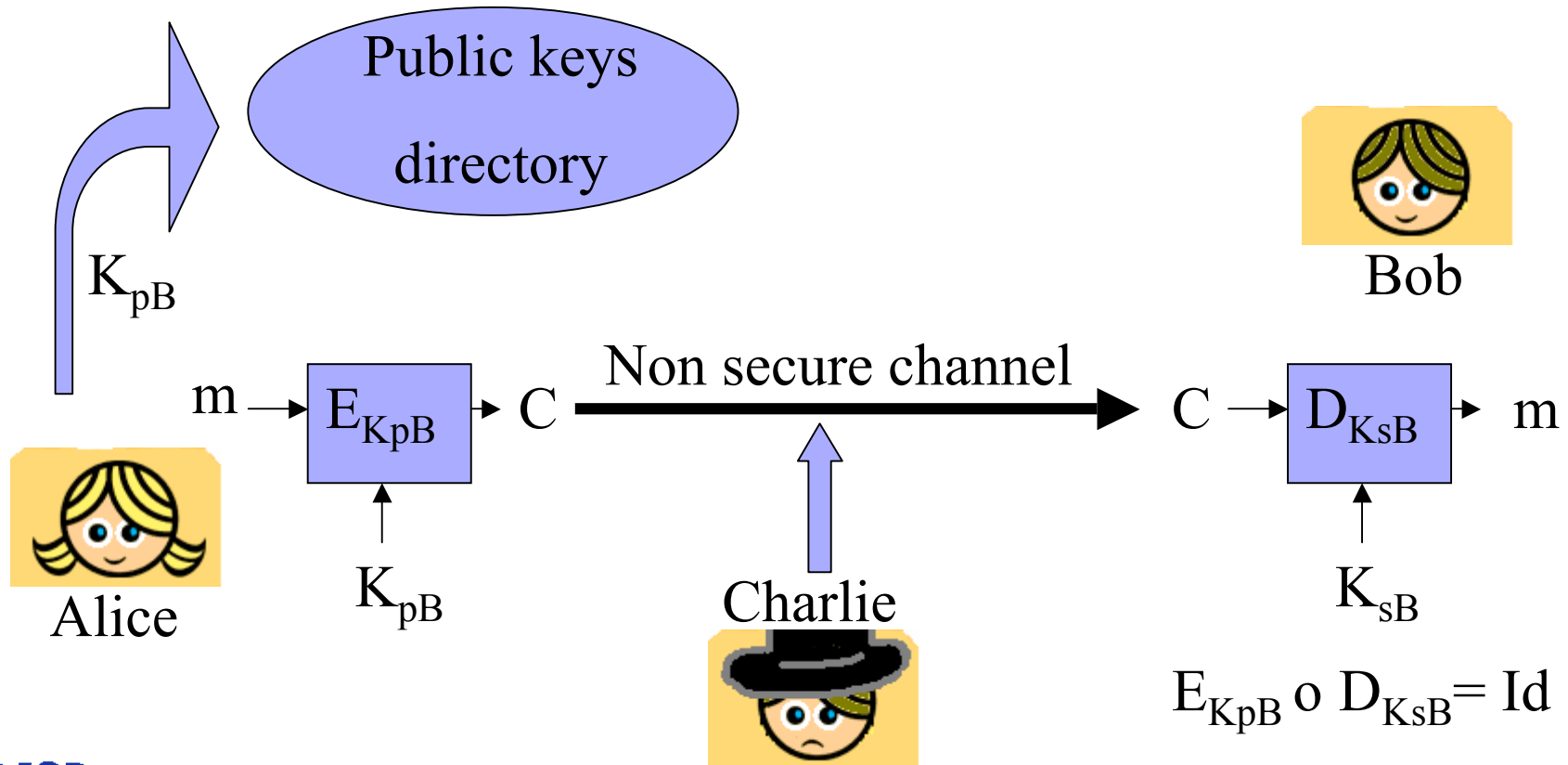
Deux méthodes pour chiffrer (1/2)

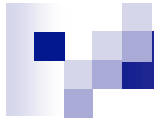
■ Cryptographie symétrique :



Deux méthodes pour chiffrer (2/2)

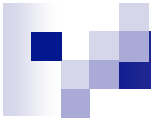
■ Cryptographie à clé publique :





Que peut-on utiliser sur un capteur ?

- Uniquement de la cryptographie symétrique car la cryptographie asymétrique est trop coûteuse en terme de ressources et trop lente
- => **La clé K doit être partagée par les deux parties**
- Règles du jeu : aujourd'hui, les algorithmes sont connus de tous, la sécurité repose sur le secret de la clé



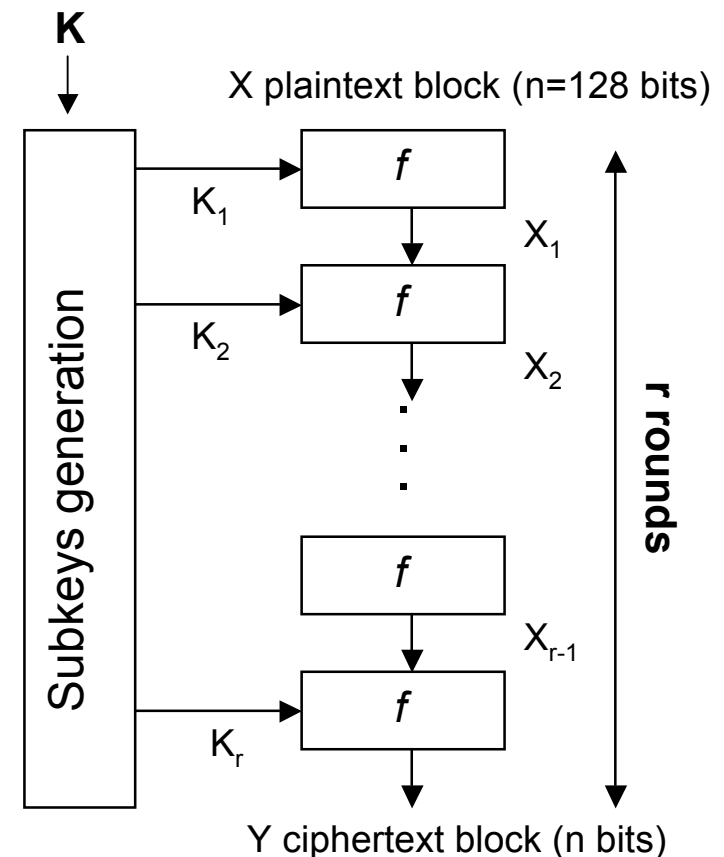
Les outils de la cryptographie symétrique

- Chiffrements par blocs ou chiffrements à flot (confidentialité)
- Fonction de hachage (intégrité)
- Chaîne de hachage (authenticité)
- MAC (Message Authentication Code) (intégrité avec preuve de la source)

Chiffrements par blocs

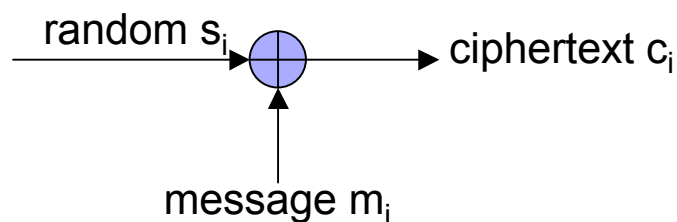
- Alice et Bob partagent une même clé K (128, 192 ou 256 bits)
- On chiffre par blocs :
 - Le texte clair m est divisé en blocs de longueur fixe
 - Chaque bloc est chiffré séparément
- La clé K génère r sous-clés
- La fonction f est une permutation des blocs
- Exemples: DES, AES, IDEA,...

■ Structure générale



Chiffrements à flot

- « one time pad » :



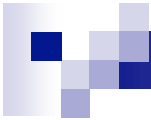
$$\begin{array}{r} m = m_0 \ m_1 \ m_2 \ m_3 \ \dots \\ \oplus \quad s = s_0 \ s_1 \ s_2 \ s_3 \ \dots \\ \hline = c = c_0 \ c_1 \ c_2 \ c_3 \ \dots \end{array}$$

- s_i est généré en utilisant un chiffrement à flot (dit aussi générateur pseudo-aléatoire)
 - Initialisé avec la clé partagée K et un IV qui doit être différent à chaque message
 - La sécurité repose sur les qualités mathématiques du générateur (grande période, pas de biais statistiques,...)
 - Meilleur que les chiffrements par blocs en termes de contraintes hardware, ne propagent pas les erreurs mais que des pairwise communications

Fonction de Hachage



- Préserve l'intégrité
- Calcule l'empreinte h d'un message M :
$$h = H(M)$$
- Doit être résistante aux collisions : il est difficile de trouver deux messages M et M' / $H(M)=H(M')$
- L'empreinte doit avoir au moins 160 bits afin d'éviter les attaques du type paradoxe des anniversaires
- Exemples: MD5 (broken), SHA-0 (broken), SHA-1, SHA-256,...



MACs: Message authentication Codes

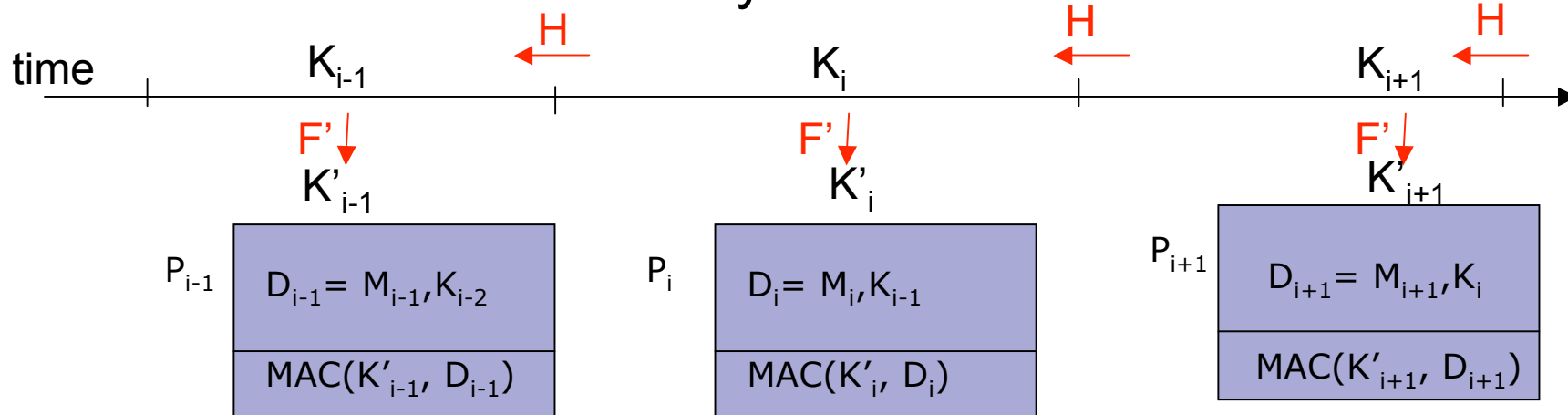
- Fonction de hachage à clés
- Intégrité + signature entre les deux personnes partageant la même clé K
- Exemple:
 - $H(K,p,m,K)$: MAC avec doublement de la clé
 - p = padding = on complète m en utilisant des 0 ou des 1 ou une valeur aléatoire

Authentification de paquets : TESLA

■ TESLA

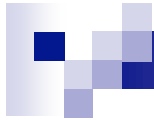
- Authentification de paquets utilisant des HASH et des MACs
- L'envoyeur choisit un aléa K_n et génère une chaîne de hachage :

$$K_n, \dots, K_0 / K_j = H(K_{j+1})$$
- Une seconde fonction F' calcule des clés MACs
- Les clés K_i sont révélés dans l'ordre inversé de leur construction selon un temps T dépendant du délai de transmission et de la synchronisation entre les noeuds



Authenticated with P_i
and K_{i-1}

Will be authenticated
with P_{i+2}



Applications :

■ Protections pour :

- ☐ La transmission des données
- ☐ L'intégrité des données
- ☐ L'authentification saut à saut et bout à bout

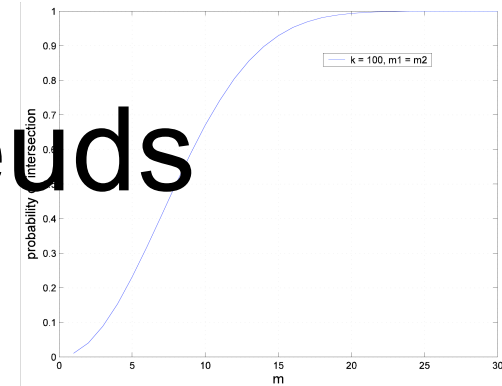
- ☐ => permet de protéger les protocoles de routage (LEAD, EAD,...)
en intégrant des chaînes de MAC et un hachage du message

■ Quelques problèmes :

- ☐ Comment alors authentifier les nœuds ?
- ☐ Comment agréger correctement des données ?

■ Ce que nous avons fait

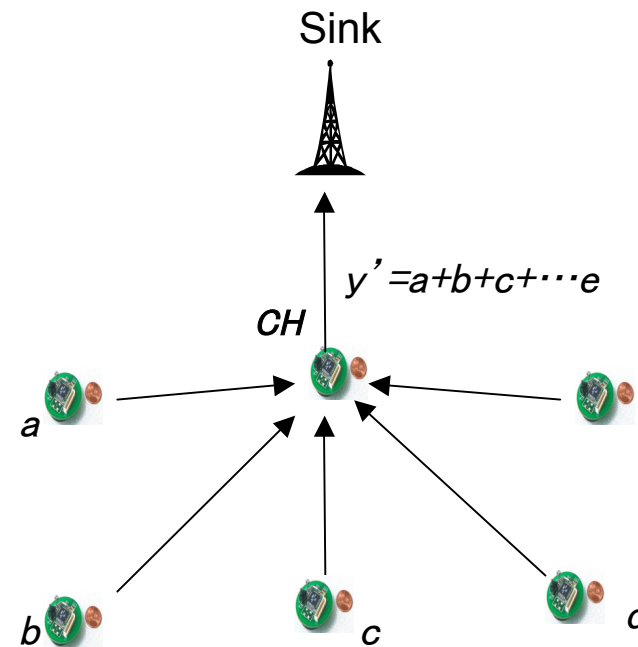
Authentification des noeuds

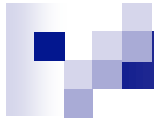


- Résolu par le problème du « Key establishment »
- Une clé partagée entre chacun des nœuds et le puits est pré-loadé dans chaque capteur pour authentification
- Vrai problème : comment établir des clés entre chaque paire de nœuds ?
 - Un large pool S de clés uniques est tiré au hasard
 - Chaque nœud reçoit m clés choisies aléatoirement dans S (key ring)
 - Avec une grande probabilité, deux nœuds voisins possèdent une clé commune (paradoxe des anniversaires)
 - Si deux voisins n'ont pas de clés communes, ils en construisent une à travers un chemin d'intermédiaires authentifiés (grâce à la phase d'établissement de clés)
- => Problème dans ce cas : la **compromission** de noeuds

Aggrégation de données (1/3)

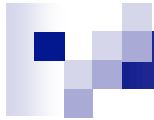
- Pour améliorer la quantité de données transmises
 - Entre les capteurs pour éviter une sur-consommation d'énergie : la transmission = très coûteux
 - Autres intérêts : calcul direct de moyennes,...
 - Aggrégation : au lieu de transmettre chaque valeur séparément, les données sont agrégées par des noeuds intermédiaires





Aggrégation de données (2/3)

- Principe général : Chaque nœud intermédiaire :
 - Mesure sa valeur
 - Reçoit d'autres valeurs de ses voisins
 - Ajoute sa valeur à celles-ci
 - Envoie le résultat
 - => les noeuds intermédiaires agrègent les données de leurs enfants
- Besoins de sécurité
 - Confidentialité et privacy
 - Générale : protection contre les écoutes non-autorisées
 - bout-en-bout: Possibilité de modifier les données sans y avoir accès
 - Authentification et intégrité
 - Protection contre les attaques d'injections ou d'altération de messages
 - Protection contre les captures de nœuds
 - Localisation rapide des noeuds victimes



Aggrégation de données (3/3)

- Utilisation de fonctions homomorphiques

$$E(a \odot b) = E(a) \otimes E(b)$$

- ☐ Proposition d'utiliser des chiffrements à flot
[Castelluccia et al - 05]

- Utilisation d'arbre de Hachage [Perrig et al - 07]

- Etude des fonctions les plus résilientes contre la compromission de nœuds [Wagner - 04]



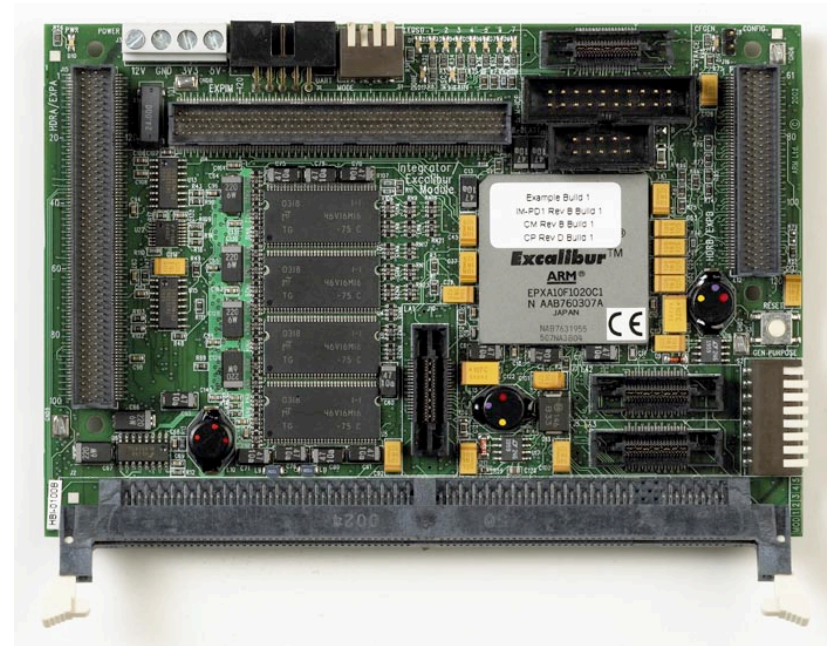
Ce que nous avons fait

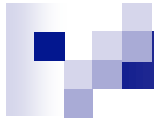
[Fournel et al - 07]

- Benchmarks de chiffrements à flot
 - Les classiques : RC4, SNOW v2.0, AES-CTR
 - Ceux sélectionnés pour la phase 2 du projet européen ECRYPT :
 - Profil software
 - Dragon, HC-128, HC-256, LEX, Phélix, Py, Pypy, Salsa20, SOSEMANUK

Réalisation des benchmarks

- Un capteur 32 bits
- La plateforme :
ARM922T, micro-
contrôleur 32 bit,
Altera Excalibur
EPXA10,
 - Avec deux mémoires
caches de 8 kB





Conclusion

- Actuellement nous préparons les benchmarks des chiffrements à flot choisis pour la phase 3 de ECRYPT sur un capteur ayant une architecture 16 bits
 - Il s'agit de : CryptMT (CryptMT Version 3), Dragon, HC (HC-128 and HC-256), LEX (LEX-128, LEX-192 and LEX-256) NLS (NLSv2, encryption-only), Rabbit, Salsa20, SOSEMANUK
- Quelques problèmes ouverts :
 - Mise en place de mécanismes de détection de compromission de nœuds peu coûteux
 - Sécurisation de protocoles d'auto-configuration et d'auto-organisation dans les réseaux de capteurs



Merci pour votre attention

Questions ?

Les chiffrements à flot

- Trois étapes:
 - Un état initial de longueur L (au moins deux fois la taille de la clé)
 - Une fonction f de mise à jour de l'état interne
 - Une fonction g qui filtre ou combine tout ou partie de l'état interne choisie pour ses bonnes propriétés
- Constructions les plus usitées:
 - Etat initial = clé (et/ou l'IV)
 - On construit un premier état interne à partir du précédent
 - Mise à jour à l'aide de f
 - La sortie est filtré par g

