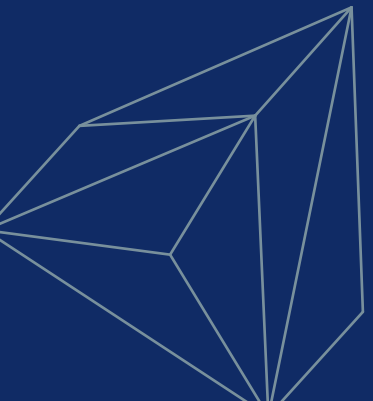
CIA

Chiffrement Authentifié pour l'IIoT

Loïc Rouquette

Sommaire

L’impératif du chiffrement authentifié



L’usine chimique

Problème

Un capteur de température envoie ses relevés chiffrés à un contrôleur.

L’usine chimique

Problème

Un capteur de température envoie ses relevés chiffrés à un contrôleur.

Confidentialité assurée : Un attaquant ne peut pas lire la température.

L'usine chimique

Problème

Un capteur de température envoie ses relevés chiffrés à un contrôleur.

Confidentialité assurée : Un attaquant ne peut pas lire la température.

Mais… que peut-il faire ?

L'approche classique : Confidentialité seule

$$\text{Nonce} + \text{Compteur} \rightarrow \text{AES}(\text{Clé}) \rightarrow \text{Flux de clés}$$

$$\text{Texte clair} + \text{Flux de clés} \rightarrow \text{Texte chiffré}$$

L'approche classique : Confidentialité seule

$$\text{Nonce} + \text{Compteur} \rightarrow \text{AES}(\text{Clé}) \rightarrow \text{Flux de clés}$$

$$\text{Texte clair} + \text{Flux de clés} \rightarrow \text{Texte chiffré}$$

Problème : Le chiffrement est malléable. Un attaquant peut "flipper" des bits dans le texte clair en modifiant le texte chiffré.

L'approche classique : Confidentialité seule

$$\text{Nonce} + \text{Compteur} \rightarrow \text{AES}(\text{Clé}) \rightarrow \text{Flux de clés}$$

$$\text{Texte clair} + \text{Flux de clés} \rightarrow \text{Texte chiffré}$$

Problème : Le chiffrement est malléable. Un attaquant peut "flipper" des bits dans le texte clair en modifiant le texte chiffré.

L'attaquant intercepte : $C = P \oplus \text{Flux de clés}$.

Il calcule = $C' = C \oplus \Delta$ (où Δ est un masque de bits).

Le destinataire déchiffre :

$$\begin{aligned} P' &= C' \oplus \text{Flux de Clé} \\ &= (P \oplus \text{Flux de Clés} \oplus \Delta) \oplus \text{Flux de Clés} \\ &= P \oplus \Delta \end{aligned}$$

L’usine chimique

L'usine chimique

Attaque par rejeu (Replay Attack) :

- L'attaquant intercepte un message chiffré valide pour `25°C`.
- Il le renvoie en boucle.
- Le contrôleur croit que tout est normal, alors que la température réelle monte en flèche.

L'usine chimique

Attaque par rejeu (Replay Attack) :

- L'attaquant intercepte un message chiffré valide pour `25°C`.
- Il le renvoie en boucle.
- Le contrôleur croit que tout est normal, alors que la température réelle monte en flèche.

Attaque par manipulation (Bit-flipping Attack) :

- L'attaquant modifie quelques bits du message chiffré.
- Le texte déchiffré devient une valeur absurde ou dangereuse (*e.g.* `-50°C` ou `330°C`).
- Le contrôleur déclenche une action potentiellement catastrophique.

L'usine chimique

Attaque par rejeu (Replay Attack) :

- L'attaquant intercepte un message chiffré valide pour `25°C`.
- Il le renvoie en boucle.
- Le contrôleur croit que tout est normal, alors que la température réelle monte en flèche.

Attaque par manipulation (Bit-flipping Attack) :

- L'attaquant modifie quelques bits du message chiffré.
- Le texte déchiffré devient une valeur absurde ou dangereuse (*e.g.* `-50°C` ou `330°C`).
- Le contrôleur déclenche une action potentiellement catastrophique.

Conclusion : Sans **intégrité** et **authenticité**, la **confidentialité** est une illusion de sécurité.

Idée : Chiffrer ET Authentifier Séparément

Idée : On chiffre le message, puis on calcule un MAC (Message Authentication Code) pour l'intégrité.

Idée : Chiffrer ET Authentifier Séparément

Idée : On chiffre le message, puis on calcule un MAC pour l'intégrité.

Mais… dans quel ordre ?

Une histoire de compositions fragiles

1. MAC-then-encrypt (SSL/TSL < 1.2)

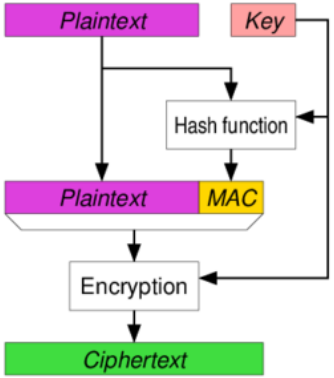


Figure 1: MAC-then-encrypt (MtE). Source : Wikipedia

- `chiffrer(clé, message || MAC(clé_mac, message))`
- **Faille** : Le destinataire doit déchiffrer AVANT de vérifier le MAC.

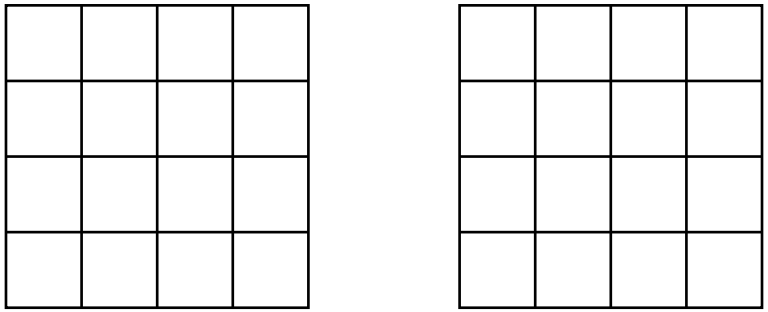
- Ouvre la porte aux **attaques par oracle de remplissage (Padding Oracle Attacks)**.

Focus : L'attaque par Oracle de Remplissage

Contexte : Les chiffrements par bloc (*e.g.*, AES) nécessitent que la taille du message soit un multiple de la taille du bloc (16 octets pour AES). Si ce n'est pas le cas, on ajoute des octets de remplissage (padding).

L'Oracle : Après déchiffrement, le serveur vérifie si le padding est valide. Sa réponse ("Padding Valide" ou "Padding Invalide") suffit à un attaquant pour déchiffrer le message, octet par octet, sans jamais connaître la clé.

L'Attaque "Padding Oracle" 1/4

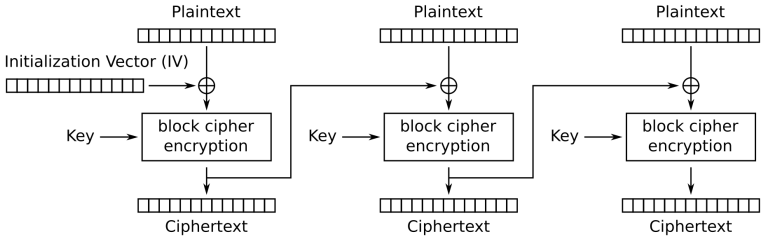


C_1 C_2

L'attaquant veut attaquer C_2.

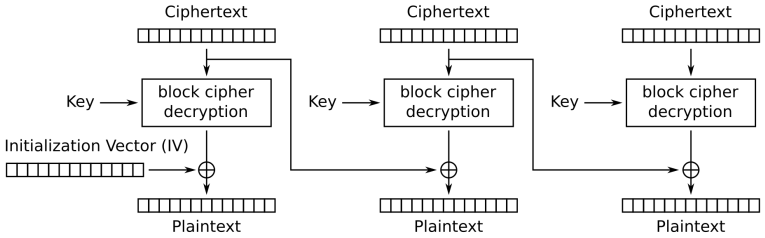
Étape 1 : Le Setup. L'attaquant intercepte C_1 et C_2. Il va forger un nouveau block C_1' et l'envoyer au serveur avec C_2.

L'Attaque "Padding Oracle" 1/4



Cipher Block Chaining (CBC) mode encryption

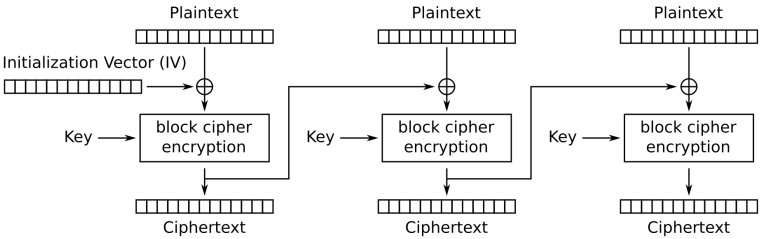
Figure 2: Chiffrement avec le mode d'opération CBC. Source : Wikipedia



Cipher Block Chaining (CBC) mode decryption

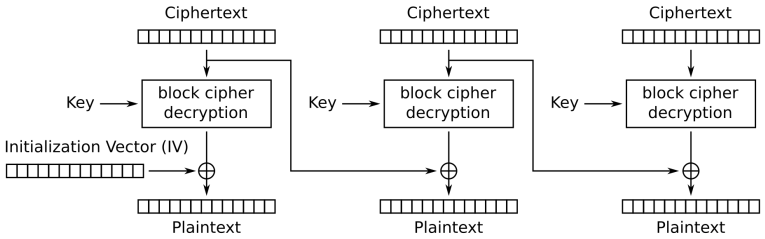
Figure 3: Déchiffrement avec le mode d'opération CBC. Source : Wikipedia

L'Attaque "Padding Oracle" 1/4



Cipher Block Chaining (CBC) mode encryption

Figure 4: Chiffrement avec le mode d'opération CBC. Source : Wikipedia



Cipher Block Chaining (CBC) mode decryption

Figure 5: Déchiffrement avec le mode d'opération CBC. Source : Wikipedia

Le serveur déchiffrera C_2 et le XORera avec C'_1 pour obtenir le texte clair P_2.

L'Attaque "Padding Oracle" 2/4

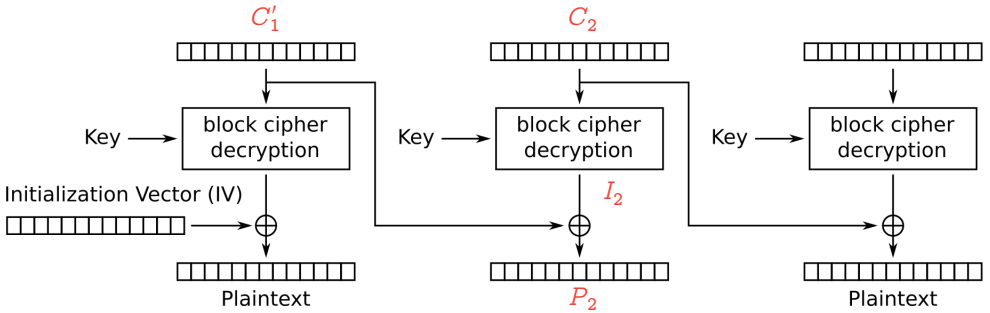


Figure 6: Déchiffrement avec le mode d'opération CBC. Source : Wikipedia

Étape 2 : L'objectif. L'attaquant veut que le dernier octet de P_2 soit `0x01` (padding valide d'un octet). Il doit donc faire en sorte que $\text{DernierOctet}(I_2) \oplus \text{DernierOctet}(C'_1) =$ `0x01`.

$$I_2 = \text{Dec}(K, C_2)$$
$$P_2 = I_2 \oplus C'_1$$

L'Attaque "Padding Oracle" 3/4

L'attaquant essaye les 256 valeurs possibles pour le dernier octet de C'_1.

Étape 3 : La recherche. L'attaquant envoie jusqu'à 256 requêtes au serveur. Pour chaque requête, il change la valeur du dernier octet de C'_1.

Quand le serveur répond "OK" – implicitement, car il ne renvoie pas d'erreur de padding" – l'attaquant a trouvé.

L’Attaque “Padding Oracle” 4/4

Étape 4: La Déduction

L'Attaque "Padding Oracle" 4/4

Étape 4: La Déduction

- L'attaquant a trouvé C'_1 qui donne un padding valide.

L'Attaque "Padding Oracle" 4/4

Étape 4: La Déduction

- L'attaquant a trouvé C'_1 qui donne un padding valide.
- Il connaît la valeur de $\mathrm{DernierOctet}(C'_1)$.

L'Attaque "Padding Oracle" 4/4

Étape 4: La Déduction

- L'attaquant a trouvé C'_1 qui donne un padding valide.
- Il connaît la valeur de $\text{DernierOctet}(C'_1)$.
- Il résout l'équation : $\text{DernierOctet}(I_2) = \texttt{0x01} \oplus \text{DernierOctet}(C'_1)$.

L'Attaque "Padding Oracle" 4/4

Étape 4: La Déduction

- L'attaquant a trouvé C'_1 qui donne un padding valide.
- Il connaît la valeur de $\text{DernierOctet}(C'_1)$.
- Il résout l'équation : $\text{DernierOctet}(I_2) = \texttt{0x01} \oplus \text{DernierOctet}(C'_1)$.
- Il connaît maintenant un octet d'état intermédiaire ! Il répète le processus pour trouver le deuxième octet (en visant un padding `0x02 0x02`), et ainsi de suite.

Une histoire de compositions fragiles

2. Encrypt-and-MAC (SSH)

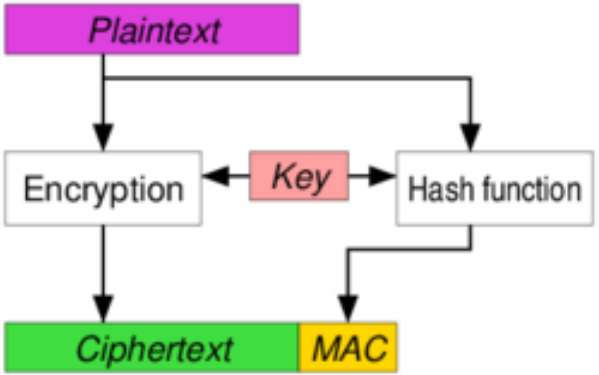


Figure 7: Encrypt-and-MAC (E&M). Source : Wikipedia

- `chiffrer(clé, message) || MAC(clé_mac, message)`
- **FAILLE** : Le MAC ne protège pas l'intégrité du *texte chiffré*.

Une histoire de compositions fragiles

3. Encrypt-then-MAC (TLS 1.2+, ISO Standard)

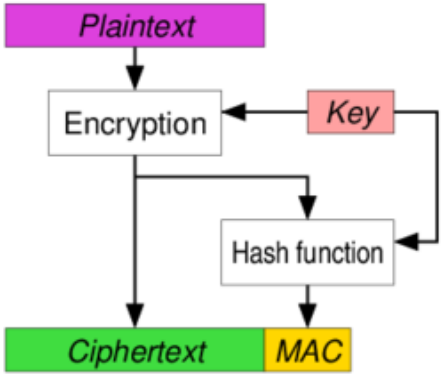


Figure 8: Encrypt-then-MAC (EtM). Source : Wikipedia

- `chiffrer(clé, message) || MAC(clé_mac, chiffrer(clé, message))`
- **Sécurisé** : Le destinataire vérifie le MAC d'abord. Si invalide, le paquet est rejeté SANS déchiffrement.

La solution moderne : AEAD (Authenticated Encryption with Associated Data)

Principe

Une seule opération atomique pour chiffrer ET authentifier. Réduit drastiquement les erreurs d'implémentation.

Interface unifiée :

- `chiffrer(Clé, Nonce, TextClair, Données Associées) → (TexteChiffré, Étiquette)`

Composants

- `Clé` : Le secret partagé
- `Nonce` : Numéro unique (public). **CRUCIAL** pour la sécurité.
- `Message` : Les données à protéger (confidentialité + intégrité)
- `DonnéesAssociées` : Données non chiffrées mais authentifiées (*e.g.*, en-têtes de paquets, ID de capteur)
- `Étiquette (Tag)` : Le MAC qui couvre le `TextClair` ET les `DonnéesAssociées`.

Le Pouvoir des Données Associées

Scénario IIoT : Un paquet réseau d'un capteur

En-tête (en clair) :

- ID Source : `0x1A`
- ID Destination : `0XB2`
- Numéro de Séquence : `1234`

Charge utile (chiffrée) :

- Température : `22.5°C`

Comment l'AEAD protège tout cela ?

1. L'en-tête est passée comme `DonnéesAssociées`.
2. La température est passée comme `TextClair`.
3. L'étiquette d'authentification est calculée sur **l'ensemble**.

Attaque bloquée : Un attaquant ne peut pas prendre une charge utile valide d'un paquet et la "greffer" sur un autre paquet avec un en-tête différent, car l'étiquette ne correspondrait pas.

Le Pouvoir des Données Associées

Scénario IIoT : Un paquet réseau d'un capteur

En-tête (en clair) :

- ID Source : `0x1A`
- ID Destination : `0XB2`
- Numéro de Séquence : `1234`

Charge utile (chiffrée) :

- Température : `22.5°C`

Comment l'AEAD protège tout cela ?

1. L'en-tête est passée comme `DonnéesAssociées`.
2. La température est passée comme `TextClair`.
3. L'étiquette d'authentification est calculée sur **l'ensemble**.

Attaque bloquée : Un attaquant ne peut pas prendre une charge utile valide d'un paquet et la "greffer" sur un autre paquet avec un en-tête différent, car l'étiquette ne correspondrait pas.

L'AEAD lie cryptographiquement le message à son contexte.

Le Processus de Validation

Les 2 sorties d'un AEAD

1. **Le Texte Chiffré (C) :** Le texte clair, chiffré.
2. **L'Étiquette d'Authentification (Tag, T) :** Une sorte de MAC qui couvre le nonce, les données associées et le texte chiffré.

Le Processus de Vérification

Entrées : `Key, Nonce, Ciphertext, Tag, AssociatedData`. **Processus :** La bibliothèque crypto recalcule le Tag attendu à partir des autres entrées. **Sortie :**

- Si `Tag_calculé == Tag_reçu`, la sortie est `Plaintext`.
- Si `Tag_calculé ≠ Tag_reçu`, la sortie est une **ERREUR**. Le programme doit rejeter les données.

Le Pouvoirs des Données Associées (AD)

Pourquoi ne pas tout mettre dans le “Text Clair” ?

Le Pouvoirs des Données Associées (AD)

Pourquoi ne pas tout mettre dans le “Text Clair” ?

Certaines données doivent rester lisibles (non chiffrées) pour le routage ou le traitement, mais leur intégrité doit être garantie.

Le Pouvoirs des Données Associées (AD)

Pourquoi ne pas tout mettre dans le "Text Clair" ?

Certaines données doivent rester lisibles (non chiffrées) pour le routage ou le traitement, mais leur intégrité doit être garantie. Ce sont les méta-données du message.

Exemple d'AD en IIoT

Paquet : `| Header | Payload |`

Header : `{ID_Source: 0x1A, ID_Dest: 0x02, Num_Seq: 1234}`

Payload : `{Temp: 45.2, Pression: 1.5}`

- **Utilisation AEAD :**
  - `AssociatedData = Header`
  - `Plaintext = Payload`

Le routeur peut lire le `Header` pour diriger le paquet, mais ne peut pas le modifier sans invalider le Tag.

L'attaque par "Découpage" (Splicing Attack)

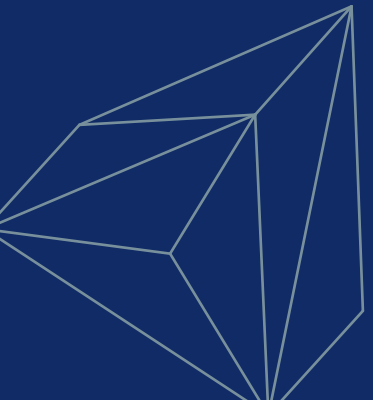
Scénario : L'attaquant capture deux messages valides.

- Msg1 : `{Header1, Payload1, Tag1}`
- Msg2 : `{Header2, Payload2, Tag2}`

Attaque : L'attaquant crée un faux message `{Header1, Payload2, ???}`.

- **Sans AD :** Si le Header n'était pas authentifié, l'attaquant pourrait peut-être réutiliser `Tag2` et le destinataire accepterait `Payload2` comme venant de la source de `Header1`.
- **Avec AD :** Le Tag dépend du Header. `Tag2` n'est valide que pour `{Header2, Payload2}`. Le faux message sera rejeté.

Le Nonce - Un point de défaillance unique



Le “Nombre Utilisé Une Seule Fois (Nonce)”

Nonce (Number used once) ≠ IV (Initialization Vector) : Un IV pour le mode CBC doit être **imprevisible**. Un **nonce** doit être unique.

Règles d’or : Un couple `(Clé, Nonce)` ne doit **JAMAIS** être réutilisé pour chiffrer deux messages différents.

Propriété : Garantit la sécurité sémantique. Chiffrer deux fois le même message avec des nonces différents.

Conséquence d’une réutilisation : Pas une simple “mauvaise pratique”. C’est une **défaillance catastrophique** de la sécurité du système.

Retour dans l'histoire

"Les cryptanalystes de Bletchley Park ont compris le fonctionnement de la machine en janvier 1942 sans jamais en avoir vu un seul exemplaire. Cela fut possible à cause d'une erreur commise par un opérateur allemand. Le 30 août 1941, un message de 4 000 caractères fut transmis ; cependant, le message n'ayant pas été reçu correctement à l'autre bout, celui-ci fut retransmis avec la même clé (une pratique formellement interdite par la procédure). De plus, la seconde fois le message fut transmis avec quelques modifications, comme l'utilisation de certaines abréviations. À partir de ces deux textes chiffrés, John Tiltman a été en mesure de reconstituer à la fois le texte en clair et le chiffrement. D'après le chiffrement, toute la structure de la machine fut reconstituée par W. T. Tutte."

— Wikipédia

Anatomie d'une catastrophe : Réutilisation dans AES-GCM

AES-GCM combine deux mécanismes :

1. **Chiffrement :** Mode Compteur (CTR).
   - KeyStream = AES(Clé, Nonce || Compteur)
   - TexteChiffré = TexteClair $\oplus$ KeyStream
2. **Authentification :** Hachage polynomial (GHASH).

Anatomie d'une catastrophe : Réutilisation dans AES-GCM

AES-GCM combine deux mécanismes :

1. **Chiffrement :** Mode Compteur (CTR).
   - KeyStream = AES(Clé, Nonce || Compteur)
   - TexteChiffré = TexteClair $\oplus$ KeyStream
2. **Authentification :** Hachage polynomial (GHASH).

Que se passe-t-il si le même Nonce est réutilisé avec la même Clé ?

Anatomie d'une catastrophe : Réutilisation dans AES-GCM

Si la même paire `(Clé, Nonce)` est utilisée pour P_1 et P_2 :

- Le `KeyStream` est **identique**.
- Un attaquant peut calculer : $C_1 \oplus C_2 = (P_1 \oplus \text{KeyStream}) \oplus (P_2 \oplus \text{KeyStream}) = P_1 \oplus P_2$.
- **Impact 1 :** L'attaquant obtient le XOR des deux textes en clair. Si la structure est connue (JSON, en-têtes, etc.), il peut retrouver P_1 et P_2.

La Fonction GHASH

Principe : C'est une multiplication polynomiale dans le corps de Galois $\mathbb{F}_{2^{128}}$.

Clé d'authentification H : Dérivée de la clé AES $K = \mathrm{AES}_{\mathrm{Enc}}(K, 0)$.

Le Tag est calculé en fonction de H, des données associées A et du texte chiffré C.

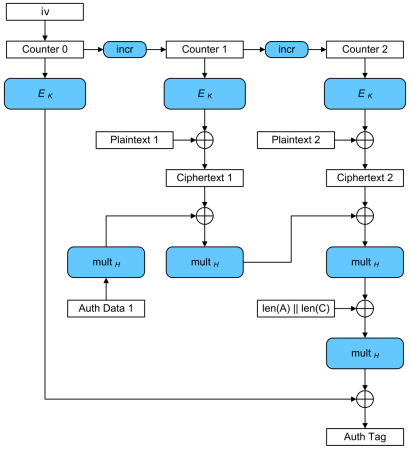


Figure 9: Galois Message Authentification Code (source: Wikipedia)

L'attaque "Interdite" de Joux 1/4

Le Scénario : Un développeur a fait une erreur. Le même nonce N est utilisé pour deux messages. L'attaquant observe les deux communications.

- Msg1 : $(A_1, P_1) \rightarrow (C_1, T_1)$;
- Msg2 : $(A_2, P_2) \rightarrow (C_2, T_2)$.

Objectif de l'attaquant : Retrouver la clé secrète H.

L'attaque "Interdite" de Joux 2/4

Équation du Tag (simplifiée) :

$$\text{Tag} = \text{GHASH}(H, A, C) \oplus \text{AES}(K, N_0) \text{ (où } N_0 \text{ est dérivé du nonce)}$$

Pour Msg1 : $T_1 = \text{GHASH}(H, A_1, C_1) \oplus \text{AES}(K, N_0)$

Pour Msg2 : $T_1 = \text{GHASH}(H, A_2, C_2) \oplus \text{AES}(K, N_0)$

L'attaque "Interdite" de Joux 2/4

Équation du Tag (simplifiée) :

$$\text{Tag} = \text{GHASH}(H, A, C) \oplus \text{AES}(K, N_0) \text{ (où } N_0 \text{ est dérivé du nonce)}$$

Pour Msg1 : $T_1 = \text{GHASH}(H, A_1, C_1) \oplus \text{AES}(K, N_0)$

Pour Msg2 : $T_1 = \text{GHASH}(H, A_2, C_2) \oplus \text{AES}(K, N_0)$

Le terme $\text{AES}(K, N_0)$ est le même dans les deux cas car le nonce est le même !

L'attaque "Interdite" de Joux 3/4

L'attaquant calcule $T_1 \oplus T_2$. Le terme $\mathrm{AES}(K, N_0)$ s'annule.

Il reste : $T_1 \oplus T_2 = \mathrm{GHASH}(H, A_1, C_1) \oplus \mathrm{GHASH}(H, A_2, C_2)$.

GHASH est **linéaire** donc : $T_1 \oplus T_2 = \mathrm{GHASH}(H, A_1 \oplus A_2, C_1 \oplus C_2)$.

C'est une équation polynomiale où la seule inconnue est H.

L’attaque “Interdite” de Joux 4/4

La Rupture : En résolvant cette équation (ce qui est mathématiquement faisable), l’attaquant retrouve H.

Conséquence : Une fois que H est connu, l’attaquant peut calculer des Tags valides pour n’importe quel message de son choix. Il peut se faire passer pour le capteur et injecter des données forgées qui seront acceptées comme authentiques.

La sécurité du système s’est totalement effondrée.

Gestion pratique des Nonces

Comment garantir l'unicité ?

Gestion pratique des Nonces

Comment garantir l'unicité ?

- **Compteurs :**
  - **Stratégie :** Commencer à 0, incrémenter pour chaque message.
  - **Avantage :** Simple, garanti unique.
  - **Défi :** Maintenir l'état de manière fiable (redémarrages, systèmes distribués).

Gestion pratique des Nonces

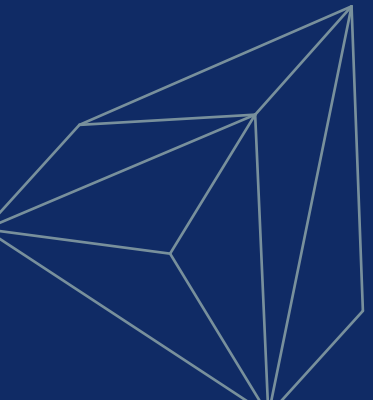
Comment garantir l'unicité ?

- **Compteurs :**
  - **Stratégie :** Commencer à 0, incrémenter pour chaque message.
  - **Avantage :** Simple, garanti unique.
  - **Défi :** Maintenir l'état de manière fiable (redémarrages, systèmes distribués).
- **Nonces Aléatoires :**
  - **Stratégie :** Générer un nonce aléatoire pour chaque message.
  - **Avantage :** Pas besoin de maintenir l'état.
  - **Défi :** Risque de collision. Pour 96 bits, la probabilité devient significative après 2^32 messages (paradoxe des anniversaires).

Gestion pratique des Nonces

- **AEAD Résistants à la Mauvaise Utilisation (Misuse-Resistant) :**
  - **Exemples :** SIV (RFC 5297), AES-GCM-SIV (RFC 8452).
  - **Principe :** Le nonce n'est pas utilisé directement. Une clé interne est dérivée du nonce ET du message.
  - **Avantage :** Si un nonce est réutilisé, la seule fuite est de savoir si les deux messages étaient identiques. Pas de perte de confidentialité ou d'authenticité.
  - **Défi :** Plus lents (deux passes sur les données).

Cryptographie pour environnements contraints



Le défi de l'IIoT

Cryptographie Légère (Lightweight Cryptography) : Pas une crypto "faible", mais une crypto **optimisée** pour des environnements aux ressources limitées.

Contraintes matérielles :

- **RAM :** Quelques Kilo-octets.
- **Puissance de calcul :** Fréquences basses (MHz).
- **Énergie :** Budgets en milliwats (mW) ou microwatts (µW), souvent sur batteries.

Caractéristique	Server (Intel Xeon)	MCU milieu de gamme (Cortex-M4)	MCU basse conso (Cortex-M0+)
Vitesse	3.0+ GHz	80 - 180 MHz	32 - 48MHz
RAM	64+ Go	40 - 384 Ko	20 –64 Ko
Consommation	150+ W	Milliwats (mW)	Microwatts (µW)
Crypto Mat.	AES-NI	Optionnel	Rare

La compétition Lightweight Crypto du NIST

- **Objectif :** Standardiser un ou plusieurs algorithmes de cryptographie légère.
- **Processus :** Public, sur plusieurs années (2019-2023).
- **Départ :** 56 candidats.
- **Gagnants (Février 2023):** La famille d'algorithmes ASCON (chiffrement et MAC).

Pourquoi ASCON ?

- Excellente performance globale (logiciel et matériel).
- Simplicité de conception.
- Bon potentiel de résistance aux attaques par canal auxiliaire.

Il a été standardisé dans la publication NIST SP 800-232.

Comparaison conceptuelle : AES-GCM vs ASCON

AES-GCM : Une construction composite

Combine deux primitives distinctes :

- Chiffrement par bloc (AES en mode CTR).
- Hachage polynomial (GHASH) pour l'authentification.

Nécesite d'implémenter deux logiques différentes.

ASCON : Une construction unifiée

- Basé sur une **unique** primitive : une **permutation** de 320 bits.
- Utilise un mode d'opération appelé **construction en éponge** (sponge construction).

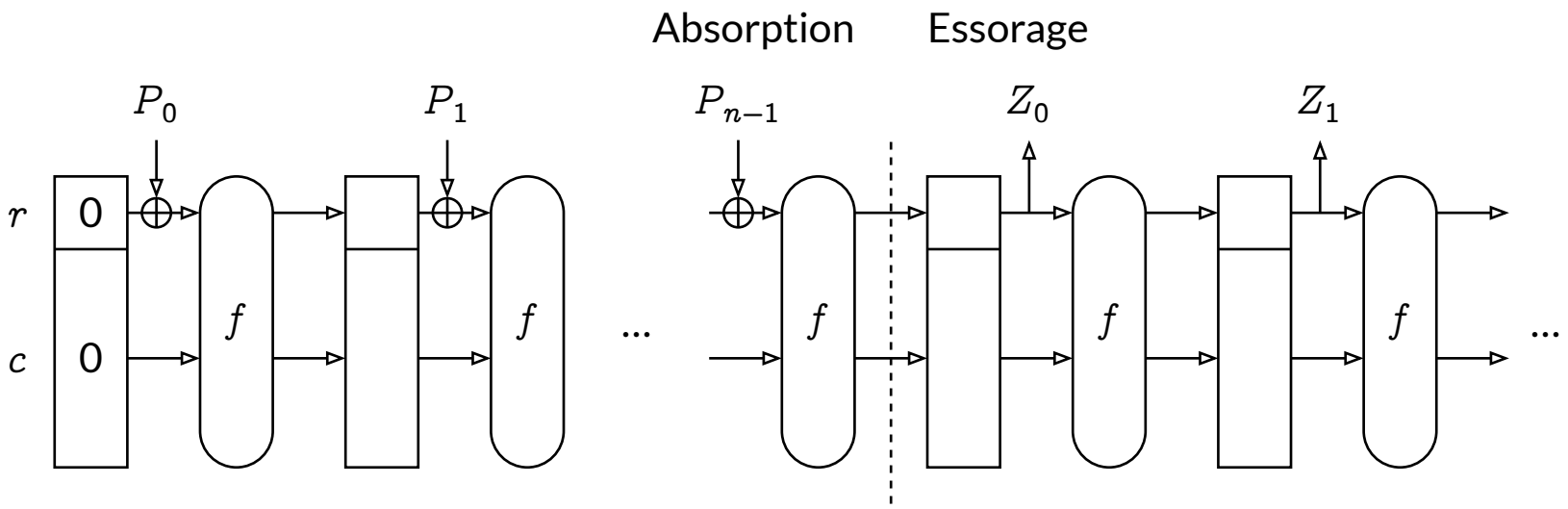


Figure 10: Construction en éponge. Source : Wikipedia

Limites et Points d'Attention

La Cryptographie ne résout pas tout !

Limites et Points d'Attention

La Cryptographie ne résout pas tout !

- **Gestion des clés :** Comment distribuer et mettre à jour les clés de manière sécurisée ? C'est un problème complexe.

Limites et Points d'Attention

La Cryptographie ne résout pas tout !

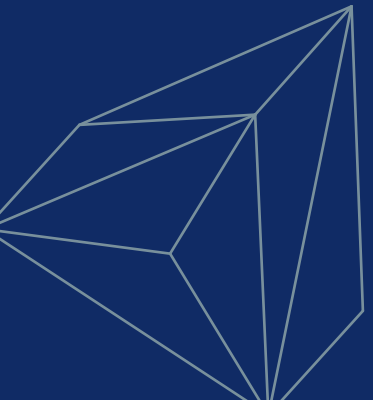
- **Gestion des clés :** Comment distribuer et mettre à jour les clés de manière sécurisée ? C'est un problème complexe.
- **Sécurité physique :** Que se passe-t-il si un attaquant vole un capteur et en extrait la clé ?

Limites et Points d'Attention

La Cryptographie ne résout pas tout !

- **Gestion des clés :** Comment distribuer et mettre à jour les clés de manière sécurisée ? C'est un problème complexe.
- **Sécurité physique :** Que se passe-t-il si un attaquant vole un capteur et en extrait la clé ?
- **Sécurité du Code :** Des bugs dans le reste du firmware peuvent contourner la meilleure des cryptographies.

Conclusion et Questions



Conclusion

L'**AEAD est non négociable :** La confidentialité sans intégrité et authenticité est **dangereuse**, surtout en IIoT.

Les **nonces de votre responsabilité :** une réutilisation est une faille catastrophique. Utilisez des compteurs ou des AEAD résistants (AES-GCM-SIV).

Le contexte est roi : Pour les appareils contraints de l'IIoT, les algorithmes "lourds" comme AES-GCM ne sont pas toujours la meilleure solution.

L'avenir est léger : La standardisation d'ASCON par le NIST marque une évolution vers des primitives unifiées, efficaces et plus simples à sécuriser, parfaitement adaptées aux contraintes de l'IIoT.

Questions ?