Power Week 2025

#pw2025

18 - 19 - 20 novembre 2025

IBM Innovation Studio Paris

S51 : XML Signature - Principes et mise en œuvre sur IBM i

19 novembre 14:45 - 15:45

Florian GRADOT
GAIA MINI SYSTEMES
florian.gradot@gaia.fr

IBM

common FRANCE

Présentation

Florian GRADOT

IBM i depuis 2009

Consultant IBM i

GAIA / VOLUBIS

Formation (débutant, perfectionnement)

Expertise IBM i

Centre de Services

IBM Influencer 2025

Awarded through the program

IBM Champions

GAIA

VOTRE EXPERT IBM i

VOLUBIS

DEVENEZ UN EXPERT

common FRANCE

MEMBER OF common EUROPE

Sommaire

- Introduction et contexte
- Concepts fondamentaux
- Exemple de mise en œuvre étape par étape
- Conclusion

Power Week

18 -19 - 20 novembre 2025

IBM

common FRANCE

IBM

Introduction

Introduction

- Pourquoi signer des XML ?
  - **Authenticité :** prouver l’émetteur
  - **Intégrité :** détecter les altérations
  - **Non-répudiation :** éviter le déni d'envoi

Rappels

- Les clés (couple)
  - Clé privée
    - Sert à signer ou déchiffrer
    - Gardée secrète, connue uniquement par le système
  - Clé publique
    - Sert à vérifier une signature ou chiffrer un message
    - Partagée librement avec tout le monde

A ne jamais stocker ensemble !

Rappels

- Le certificat (Carte d’identité de la clé publique)
  - Il contient :
    - La clé publique
    - L’identité du propriétaire (nom, organisation…)
    - La signature d’une Autorité de certification (ou auto-signé)
    - Période de validité
  - Sert à prouver que la clé publique appartient bien à l’entité mentionnée dans le certificat, grâce à la signature qui atteste cette liaison.

Rappels

- Autorité de certification (CA)
  - Organisme (public ou privé)
    - Vérifie l'identité d'une personne ou d'une entreprise
    - Valide sa clé publique
    - Produit un certificat signé
  - Sa signature permet de dire :
    - J'atteste que cette clé publique appartient vraiment à cette personne / entreprise

Power Week

18 -19 - 20 novembre 2025

IBM

common FRANCE

IBM

Concepts

Signature numérique vs chiffrement

- Chiffrement
  - Sert à cacher un message pour que seul le destinataire puisse le lire
    - L'expéditeur chiffre le message avec une clé
    - Le destinataire déchiffre avec la clé correspondante

- Signature numérique
  - Prouve qui a signé et garantit qu'il n'a pas été modifié
    - Signataire crée une empreinte (hash) du document
    - Il signe l'empreinte avec sa clé privée
    - Toute personne peut vérifier avec sa clé publique

Signature numérique vs chiffrement

- Le chiffrement protège le contenu
- La signature numérique prouve qui a créé ou envoyé le document

Fonction	Signature numérique	Chiffrement
Confidentialité	Non	Oui
Authenticité	Oui	Non
Intégrité	Oui	Non
Non-répudiation	Oui	Non
Lisibilité du message	Visible	Illisible

XML Signature (XMLDSIG)

- Norme du W3C (https://www.w3.org/TR/xmldsig-core/)
- Eléments clés : <Signature>, <SignedInfo>, <SignatureValue>, <KeyInfo>

```
<Signature>
  <SignedInfo>
    <CanonicalizationMethod/>
    <SignatureMethod/>
    <Reference URI="">
      <DigestMethod/>
      <DigestValue/>
    </Reference>
  </SignedInfo>
  <SignatureValue>...</SignatureValue>
  <KeyInfo>
    <X509Data>...</X509Data>
  </KeyInfo>
</Signature>
```

Types de signature

- Signature enveloppée
  - Elle est incluse dans le document XML qu'elle signe.

```
<Document>
    <Data>Exemple de données</Data>
    <Signature>
        <SignedInfo>...</SignedInfo>
        <SignatureValue>...</SignatureValue>
    </Signature>
</Document>
```

Types de signature

- Signature enveloppante
  - Elle englobe les données qu'elle signe.

```
<Signature>
    <SignedInfo>...</SignedInfo>
    <SignatureValue>...</SignatureValue>
    <Object>
        <Data>Exemple de données</Data>
    </Object>
</Signature>
```

Types de signature

- Signature détachée
  - Elle est stockée séparément du document qu'elle signe.

```
<Signature>
    <SignedInfo>
        <Reference URI="http://exemple.com/data.xml"/>
    </SignedInfo>
    <SignatureValue>...</SignatureValue>
</Signature>
```

Canonisation C14N

- Deux documents XML peuvent sembler différents mais représenter la même information.

```
<person first="Power" last="Week"/>
<person last="Week" first="Power"></person>
```

- La canonisation convertit des documents en une version unique et standard.
  - Supprime les variables non significatives (espaces, ordre des attributs ..)
  - Uniformiser les noms et encodages des caractères
  - https://www.w3.org/TR/2001/REC-xml-c14n-20010315

Canonisation exclusive

- Variante de la canonisation C14N
- Elle ne propage pas les déclarations de namespaces qui ne sont pas utilisées par l'élément canonisé.
- https://www.w3.org/TR/xml-exc-c14n/

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
    <xs:element xmlns:xs="http://www.w3.org/2001/XMLSchema">Hello</xs:element>
</soap:Envelope>
```

Algorithmes et recommandations

- Hachage :
  - Transformer n'importe quelle donnée en empreinte numérique fixe
  - Un caractère qui change et le résultat sera différent
  - SHA-1, SHA256, SHA-3
  - Recommandations : SHA256

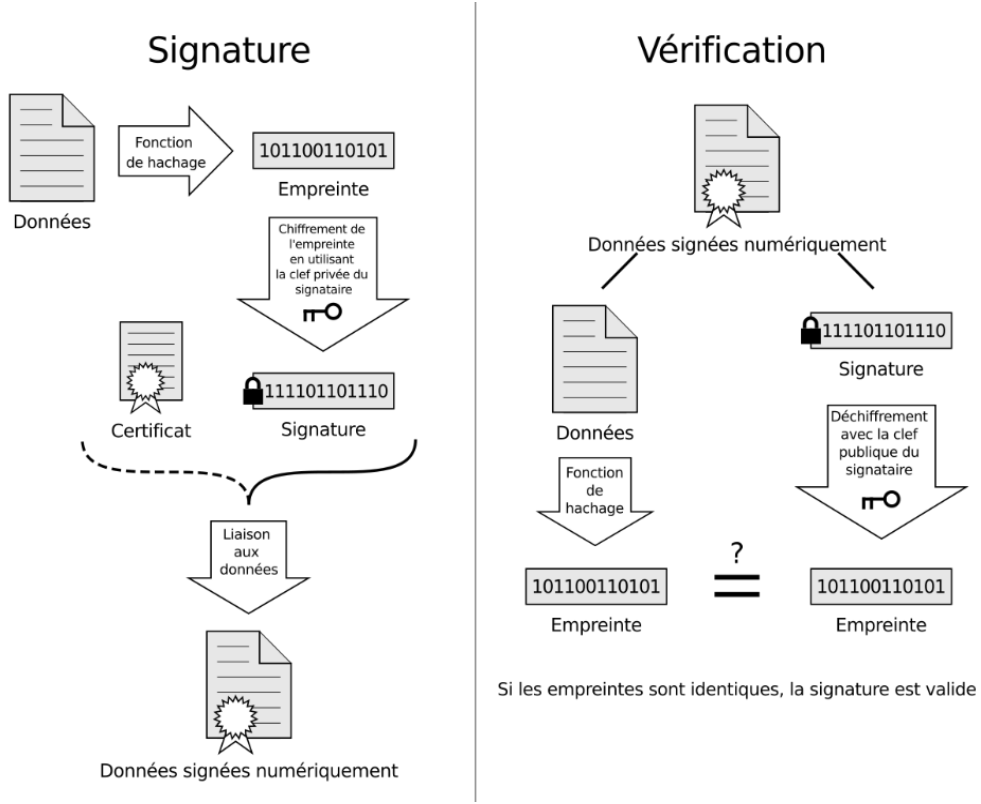
```
<DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
```

Algorithmes et recommandations

- Signature :
  - RSA-SHA1 → RSA pour la signature + SHA-1 pour le hash (moins sûr aujourd'hui)
  - RSA-SHA256 → RSA + SHA-256 (standard actuel et sécurisé)

```
<CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
<SignatureMethod Algorithm="http://www.w3.org/2001/04/xmldsig-more#rsa-sha256"/>
```

Schéma explicatif



Power Week

18 -19 - 20 novembre 2025

IBM

common FRANCE

IBM

Cas pratique

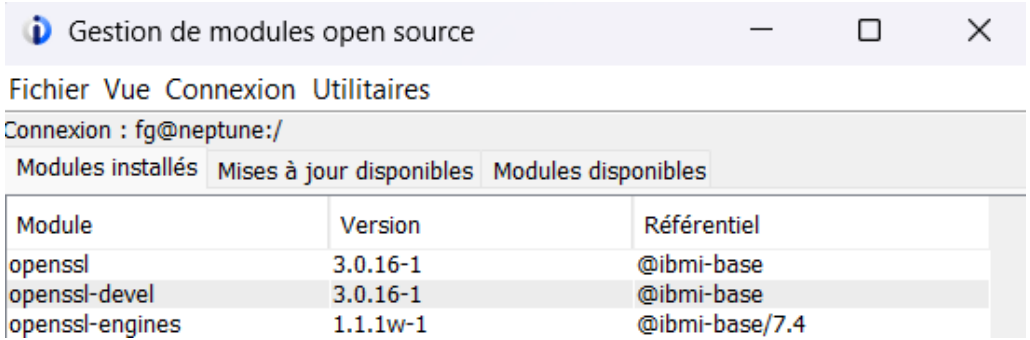
Cas pratique

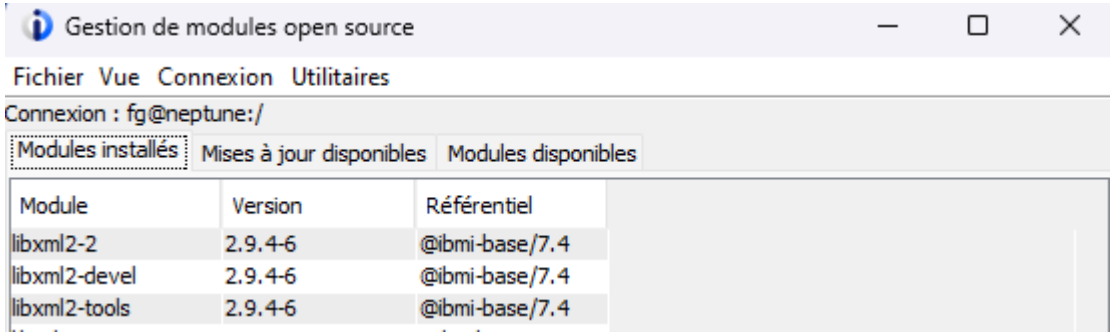
- A partir des commandes OpenSource
  - Openssl
  - xmllint

- Génération de clé privée / publique manuellement pour le cas pratique

Pré-requis

- Avoir les modules OpenSource installés





Fichiers d’exemple utilisés

- **Key.pem** = clé privée
- **Key.pub** = clé publique
- **pw2025.xml** = XML à signer
- **pw2025_c14n.xml** = XML canonisé
- **hash.bin** = hash du fichier xml
- **pw2025.signature** = signature du fichier

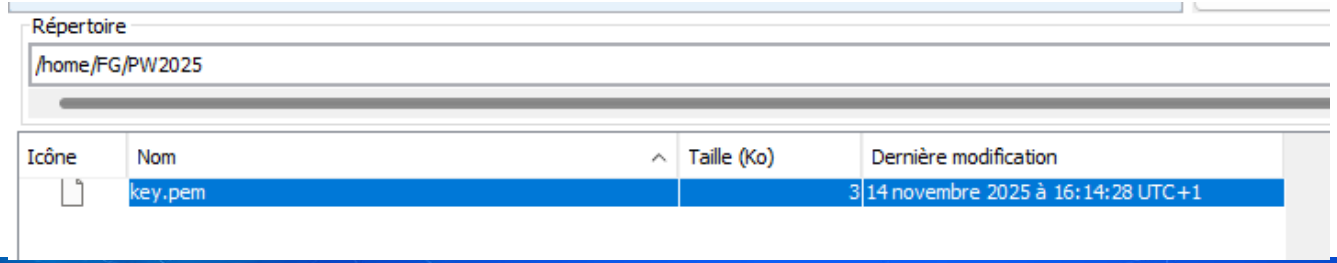
Etape 1 : Génération de la clé privée

- Commande :

/QOpenSys/pkgs/bin/openssl genrsa -out key.pem 4096

  - **genrsa :** commande pour générer une clé RSA privée
  - **-out key.pem :** fichier de sortie
  - **4096 :** taille de la clé en bits

```
> cd /home/fg/PW2025
  $
> /QOpenSys/pkgs/bin/openssl genrsa -out key.pem 4096
  $
```



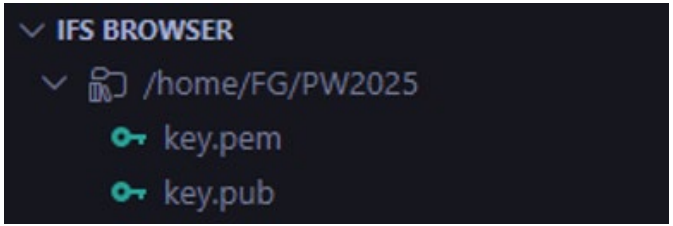
Etape 2 : Extraction clé publique

- Commande :

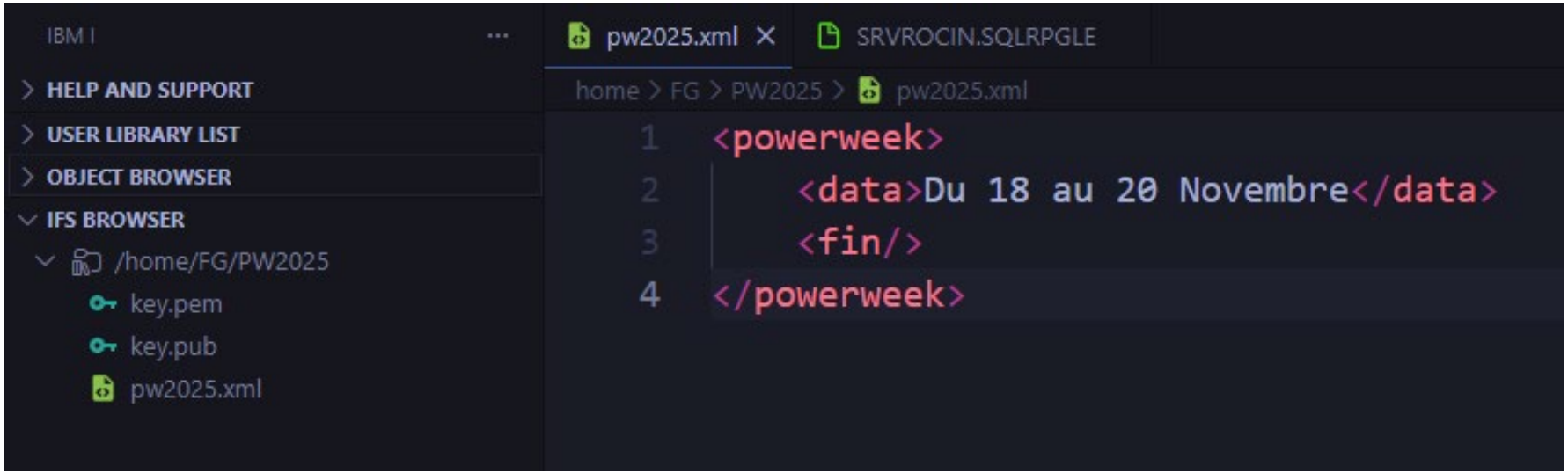
/QOpenSys/pkgs/bin/openssl rsa -in key.pem -pubout > key.pub

  - **RSA :** utilitaire OpenSSL pour manipuler les clés RSA
  - **-in key.pem :** clé privée en entrée
  - **-pubout > key.pub :** clé publique en sortie

```
> /QOpenSys/pkgs/bin/openssl rsa -in key.pem -pubout > key.pub
writing RSA key
$
```



Etape 3 : Déposer le XML à signer



Etape 4 : Canoniser le XML

- Commande :

/QOpenSys/pkgs/bin/xmllint --exc-c14n pw2025.xml > pw2025_c14n.xml

  - **--exc-c14n :** canonisation exclusive
  - **pw2025.xml :** fichier à canoniser
  - **pw2025_c14n.xml :** fichier canonisé

```
> /QOpenSys/pkgs/bin/xmllint --exc-c14n pw2025.xml
  <powerweek>
      <data>Du 18 au 20 Novembre</data>
      <fin></fin>
  </powerweek>$
```

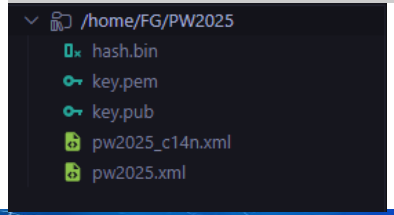
Etape 5 : Générer le hash

- Commande :

/QOpenSys/pkgs/bin/openssl dgst -binary -sha256 pw2025_c14n.xml > hash.bin

  - **Dgst :** commande pour calculer le hash (digest)
  - **–binary :** sortie au format binaire
  - **–sha256 :** Algorithme de hash
  - **pw2025_c14n.xml :** Fichier dont on calcule le hash
  - **> hash.bin :** Hash au format binaire

```
> /QOpenSys/pkgs/bin/openssl dgst -binary -sha256 pw2025_c14n.xml > hash.bin
  $
```

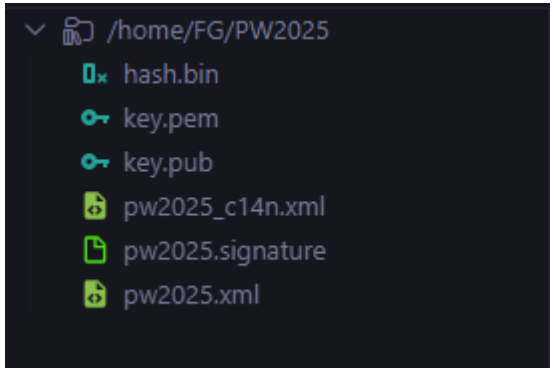


Etape 6 : Générer la signature

- Commande :

/QOpenSys/pkgs/bin/openssl pkeyutl -sign -in hash.bin -inkey key.pem -keyform PEM -pkeyopt digest:sha256 -out pw2025.signature

  - **openssl pkeyutl –sign :** signer des données
  - **-in hash.bin :** hash du fichier xml
  - **-inkey key.pem :** clé privée
  - **-keyform PEM :** Format de la clé privée
  - **-pkeyopt digest:sha256 :** Algorithme de hash
  - **-out pw2025.signature :** Signature en sortie



Etape 7 : Contrôler la signature

- Commande :

/QOpenSys/pkgs/bin/openssl dgst -verify key.pub -keyform PEM -sha256 -signature pw2025.signature -binary pw2025_c14n.xml

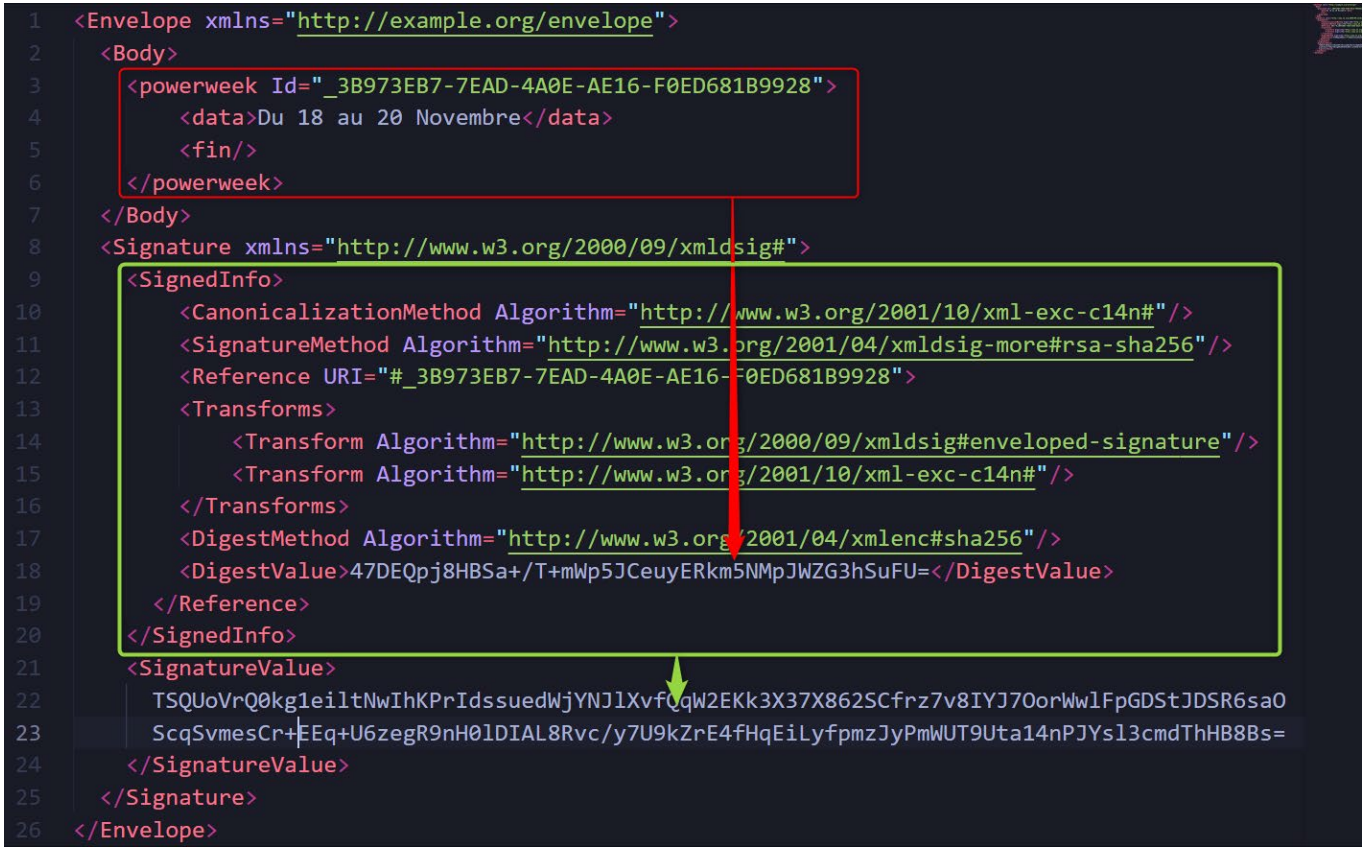
  - **Dgst :** Manipuler les digests (hash)
  - **-verify key.pub :** Clé publique pour vérifier la signature
  - **-keyform PEM :** format de la clé
  - **-sha256 -binary :** Hash utilisé pour générer le hash avant signature + format
  - **-signature pw2025.signature :** Fichier contenant la signature
  - **pw2025_c14n.xml :** fichier original

```
> /QOpenSys/pkgs/bin/openssl dgst -verify key.pub -keyform PEM -sha256 -signature pw2025.signature -binary pw2025_c14n.xml
Verified OK
$
```

Pour une signature enveloppée

- Les étapes :
  - Calculer le hash du XML
  - L'ajouter dans la balise <SignedInfo> + renseigner la référence
  - Calculer le hash et signer la balise <SignedInfo>
  - Ajouter la signature dans la balise <SignatureValue>
  - Incorporer tout le bloc <Signature> dans le XML d'origine

Exemple résultat final



Power Week

18 -19 - 20 novembre 2025

IBM

common FRANCE

IBM

Conclusion

Conclusion

- Il est possible sur l'IBM i de signer des XML grâce à l'OpenSource
- Bien faire attention au rajout de la signature dans le XML
- Attention au nombre de fichiers générés

- Alternative:
  - Utiliser les API IBM : Qc3EncryptData et Qc3DecryptData
  - JAVA ? PHP ? Autre ?


MERCI