

Vérifier un fichier ISOF

Sans IsoFind, sans réseau, sans nous faire confiance

Ce qu'un fichier signé prouve, ce qu'il ne prouve pas,
et comment le contrôler soi-même.

Sommaire

Comprendre

1. À qui s'adresse ce guide	3
2. Ce qu'une signature atteste	3
3. Anatomie d'un fichier	4
4. Ce qui est vérifiable	6

Vérifier

5. Les règles de sérialisation	8
6. Niveau 1, l'intégrité	9
7. Niveau 2, la signature de laboratoire	10
8. La provenance	11
9. Les attestations d'origine	13
10. Un fichier chiffré	15

Interpréter

11. Lire un verdict	16
12. Douze pièges	17

Annexes

13. Le vérificateur autonome	18
14. Obtenir un certificat	19
15. Références	20

Comprendre

Ce qu'est un fichier ISOF, ce que sa signature établit, et ce qu'il faut avoir sous la main pour le contrôler.

1. À qui s'adresse ce guide

À celui qui reçoit un fichier et doit décider s'il peut s'appuyer dessus : un auditeur, un client, un relecteur, un service qualité, un expert judiciaire, un confrère à qui l'on transmet des mesures.

Vous n'avez pas besoin d'IsoFind pour lire ce guide, ni pour faire ce qu'il décrit. Un fichier ISOF est un fichier JSON, lisible dans n'importe quel éditeur de texte, et toutes les vérifications se font hors ligne avec des algorithmes publics. C'est la raison d'être du format : une preuve qui exigerait le logiciel de celui qui l'a produite n'en serait pas une.

Ce qu'il vous faut

Pour	Il faut
Lire le contenu	Un éditeur de texte. Rien d'autre.
Vérifier l'intégrité	De quoi calculer un SHA-256 : Python, Node, ou un utilitaire en ligne de commande.
Vérifier la signature d'un laboratoire	Une bibliothèque cryptographique courante. Le script fourni n'utilise que <code>cryptography</code> , en Python.
Vérifier l'émetteur du certificat	Le certificat racine IsoFind, à obtenir une fois par le canal de votre choix, et à conserver.
Déchiffrer un fichier chiffré	La clé privée du destinataire. Sans elle, personne ne peut.

Le raccourci. Si vous voulez seulement une réponse, la section 13 donne un script d'une seule commande qui produit un verdict lisible. Le reste du guide explique ce que ce verdict recouvre, et c'est ce qui vous permettra de le défendre devant quelqu'un.

Ce que ce guide ne couvre pas

L'interprétation scientifique des mesures, la validité des méthodes analytiques, et la question de savoir si le laboratoire signataire est compétent ou honnête. Aucun dispositif cryptographique ne répond à ces questions, et un document qui laisserait croire le contraire vous desservirait.

2. Ce qu'une signature atteste

C'est la section la plus importante du guide, et la plus courte à résumer : une signature parle de l'histoire du fichier, jamais de la justesse de ce qu'il contient.

Ce qui est établi

Que le laboratoire nommé dans le certificat, porteur d'un certificat émis par l'autorité IsoFind, a transmis exactement ces données. Retirer une mesure, en ajouter une, changer une décimale, corriger une date : la signature ne colle plus, et le fichier est refusé.

Quand le fichier porte un bloc de provenance, la déclaration s'étend à l'histoire des modifications : l'échantillon a subi exactement ces changements, dans cet ordre, par ces acteurs. Une entrée retirée ou retouchée casse la chaîne.

Ce qui ne l'est pas

Que les mesures sont justes. Aucune signature ne peut l'établir. Un laboratoire peut signer des valeurs fausses, et sa signature sera parfaitement valide : elle attestera qu'il les a bien envoyées, ce qui est exactement ce qu'on lui demande.

Que le laboratoire est compétent, accrédité, ou de bonne foi. Le certificat atteste une identité, pas une qualité.

Que les données sont complètes. Un signataire choisit ce qu'il exporte et ce que sa signature couvre. Voir la section 12, où c'est le premier piège.

Cette distinction n'est pas une précaution rhétorique. Elle décide de ce que vous pouvez écrire dans un rapport. « Le fichier est intègre et provient de tel laboratoire » est une affirmation défendable ; « les valeurs sont certifiées » ne l'est pas, et personne ne l'a jamais promis.

Pourquoi cela vaut quand même

Parce que la question qu'un auditeur pose n'est presque jamais « ces nombres sont-ils exacts », à laquelle il ne peut de toute façon répondre qu'en refaisant les mesures. Elle est « ces nombres sont-ils ceux que le laboratoire a produits, ou ont-ils été arrangés en chemin ». À celle-là, un fichier ISOF signé répond, et il y répond sans vous demander de faire confiance au logiciel qui l'a écrit.

3. Anatomie d'un fichier

Un objet JSON, quelques blocs de premier niveau, et un bloc de signature qui dit lesquels il couvre.

```
{
  "isof_version": "1.2",
  "created_at": "2026-08-26T08:00:00+00:00",
  "created_by": {
    "software": "IsoFind", "software_version": "1.0.0",
    "operator": "...", "organisation": "..."
  },
  "project": { "name": "..." },
}
```

```

"samples":      [ ... ],    // echantillons, mesures comprises
"methods":      { ... },    // protocoles employes
"pipelines":    { ... },    // enchainements de methodes
"purification": { ... },    // rendements
"assignments":  [ ... ],    // quelle methode sur quel echantillon

"provenance":   { ... },    // historique signe, facultatif
"origin_claims":[ ... ],    // attestations d'origine, facultatif
"encryption":   { ... },    // present si le contenu est chiffre

"signature":    { ... }     // null si le fichier n'est pas signe
}

```

Un échantillon

Chaque échantillon porte son identité, sa localisation quand elle est renseignée, et jusqu'à quatre familles de données : isotopie, géochimie élémentaire, paramètres physico-chimiques, molécules. Un exportateur peut n'en transmettre qu'une partie.

```

{
  "id": "1", "name": "PZ-01", "material_type": "eau souterraine",
  "isotope_data": [
    {
      "element": "Sr", "system": "87Sr/86Sr",
      "ratio":      0.70912,      // LA MESURE, toujours
      "ratio_2se": 0.00002,
      "standard": "NBS 987", "instrument": "Neptune Plus",

      "derived": {                // present seulement s'il y a une correction
        "value": 0.70909, "nature": "blanc",
        "formule": "correction_blanc", "motif": "Blanc de session",
        "superseded": [ ... ]    // les corrections remplacees
      }
    }
  ]
}

```

ratio est la mesure, et le reste toujours. Quand une correction a été appliquée, elle vit dans **derived**, à côté, jamais à la place. Un lecteur qui ignore **derived** obtient donc une valeur vraie mais incomplète, et jamais une valeur calculée qu'il prendrait pour un instrument. Si vous reprenez des chiffres dans un rapport, regardez si **derived** est là, et dites laquelle des deux valeurs vous citez.

Le bloc de signature

```

// Niveau 1 : une empreinte, recalculable par n'importe qui
"signature": {
  "level": 1, "algorithm": "SHA-256",
  "scope": ["created_by", "samples", "methods", "purification"],

```

```

    "hash": "a3f1...", "signed_at": "...", "signed_by": "...
  }

  // Niveau 2 : une signature ECDSA, et l'identite qui va avec
  "signature": {
    "level": 2, "algorithm": "ECDSA-P256", "hash_algorithm": "SHA-256",
    "signed_scope": ["created_by", "samples", "methods", "purification", "provenance"],
    "signature_b64": "MEUCIQ...",
    "certificate_chain": ["<labo, PEM en base64>", "<autorite emettrice>"],
    "signed_at": "...", "signed_by": "...",
    "revocation": {
      "crl_url": "https://pki.isofind.io/crl/issuing-ca.crl",
      "crl_embedded_b64": "...", "crl_snapshot_date": "..."
    }
  }
}

```

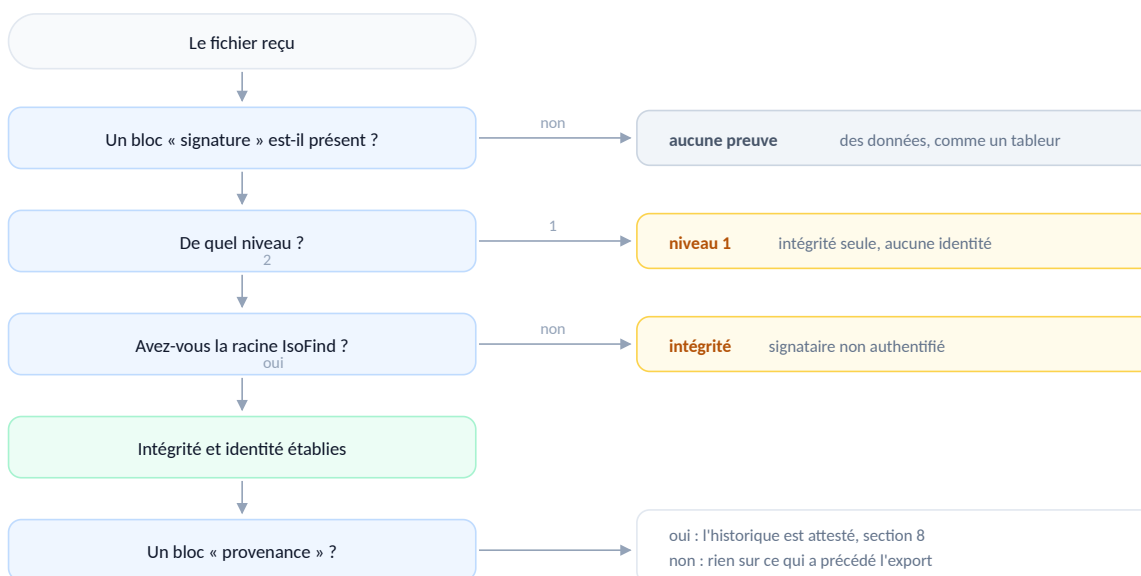
Le champ le plus important des deux est `scope`, ou `signed_scope` au niveau 2 : il énumère les blocs que la signature couvre. Ce qui n'y figure pas n'est pas protégé, et peut avoir été modifié après coup sans que rien ne le signale.

4. Ce qui est vérifiable

Un fichier porte de zéro à quatre objets vérifiables, indépendants les uns des autres. Commencez par regarder lesquels sont là.

Objet	Présent si	Ce qu'il établit
Signature de document	<code>signature</code>	Que les blocs du scope n'ont pas bougé. Au niveau 2, par qui ils ont été envoyés.
Provenance	<code>provenance</code>	Que l'historique des modifications est complet et dans l'ordre.
Attestations d'origine	<code>origin_claims</code>	Que le raisonnement menant à une conclusion d'origine est cohérent et rattaché aux données du fichier.
Chiffrement	<code>encryption</code>	Rien en soi : il détermine ce que vous pouvez lire, donc ce que vous pouvez vérifier.

Par où commencer



Les branches de droite ne sont pas des échecs : ce sont des niveaux de preuve différents, et ce qu'il faut savoir avant de citer le fichier quelque part.

Un fichier non signé n'est pas une preuve. Il reste des données utiles, échangées dans un format lisible, exactement comme un tableur. Ce n'est pas un défaut du fichier : la plupart des échanges scientifiques n'ont pas besoin d'être opposables. Cela devient un problème le jour où on le cite comme s'il l'était.

Vérifier

Une procédure par objet vérifiable, avec la règle de calcul exacte et de quoi la refaire soi-même.

5. Les règles de sérialisation

Toute vérification revient à recalculer une empreinte sur du texte. Deux façons d'écrire le même objet JSON donnent deux empreintes différentes, donc la règle d'écriture fait partie du format. Lisez cette section avant les suivantes.

ISOF v1.2 en emploie deux, selon le bloc. Ce n'est pas élégant, et la version majeure suivante les unifiera sur RFC 8785 en rupture annoncée. En attendant, les deux sont reproductibles à partir du seul fichier, et les voici.

Règle	Employée par	Clés
A	La signature du document, niveaux 1 et 2	Dans l'ordre du fichier, non triées
B	La provenance, les attestations d'origine	Triées par ordre alphabétique, à tous les niveaux

Les deux partagent le reste : JSON compact, sans espace après les séparateurs, sans échappement des caractères non ASCII, encodé en UTF-8.

```
# Regle A : la signature du document
json.dumps(payload, separators=(",", ":"), ensure_ascii=False)

# Regle B : provenance et attestations
json.dumps(payload, separators=(",", ":"), ensure_ascii=False, sort_keys=True)
```

La règle A dépend de l'ordre, et c'est vérifiable quand même

Ne pas trier semble fragile, mais un analyseur JSON qui conserve l'ordre des clés du fichier, ce que font Python depuis la 3.7 et JavaScript, rend exactement l'ordre dans lequel le signataire a écrit. Lisez le fichier, resérialisez, vous retrouvez son texte. En Python, chargez avec

`object_pairs_hook=OrderedDict` pour que ce soit explicite plutôt que tacite.

Ce qu'il ne faut surtout pas faire, c'est trier vous-même en croyant bien faire : l'empreinte ne tombera jamais juste.

Les nombres sont le vrai piège. Un langage n'écrit pas les flottants comme un autre : Python sérialise `1.0`, JavaScript écrit `1`. Une vérification écrite dans un langage différent de celui du signataire peut donc échouer sur un fichier intact. Les fichiers ISOF sont écrits en Python ; vérifiez-les en Python, ou vérifiez très soigneusement le formatage des nombres de votre langage.

Ce qui entre dans le calcul

Jamais le fichier entier. Toujours les blocs listés par le scope, dans l'ordre du scope, et rien d'autre.

```
payload = OrderedDict((bloc, doc[bloc]) for bloc in scope if bloc in doc)
```

Une exception, pour les fichiers chiffrés. Quand le scope vaut exactement

`["encrypted_metadata"]`, le payload n'est pas construit à partir du fichier mais fixé par la spécification : `created_by`, `project`, `created_at`, dans cet ordre. Voir la section 10.

6. Niveau 1, l'intégrité

Une empreinte SHA-256 sur les blocs couverts. Elle répond à une seule question : le fichier a-t-il changé depuis qu'on a calculé cette empreinte.

La procédure

1. Lire `signature.scope`, la liste des blocs couverts.
2. Construire le payload : ces blocs, dans cet ordre, ceux qui existent.
3. Sérialiser selon la règle A, encoder en UTF-8, calculer le SHA-256.
4. Comparer à `signature.hash`.

```
import json, hashlib
from collections import OrderedDict

doc = json.load(open("fichier.iso", encoding="utf-8"),
                object_pairs_hook=OrderedDict)
sig = doc["signature"]
scope = sig["scope"]

payload = OrderedDict((b, doc[b]) for b in scope if b in doc)
canonique = json.dumps(payload, separators=(",", ":"), ensure_ascii=False)
calculé = hashlib.sha256(canonique.encode("utf-8")).hexdigest()

print("intègre" if calculé == sig["hash"] else "altéré")
```

Une empreinte n'est pas une signature. N'importe qui peut modifier le fichier, recalculer l'empreinte, la réécrire dans le bloc, et changer `signed_by` pour le nom qu'il veut. Le niveau 1 protège contre l'altération accidentelle, la corruption en transit, la modification maladroite. Il ne protège contre personne de décidé. Pour cela, il faut le niveau 2.

Le cas des fichiers signés avant août 2026

Jusqu'à cette date, le signataire ajoutait le bloc `created_by` au payload sans le déclarer dans le scope. Ces fichiers existent et restent vérifiables : si le calcul ci-dessus ne tombe pas juste, et seulement dans ce cas, recommencez en ajoutant `created_by` à la fin du payload.

```
if calcule != sig["hash"] and "created_by" not in scope:
    payload["created_by"] = doc["created_by"] # en DERNIER
    # puis recalculer. Si cela tombe juste, le fichier est intact,
    # au format antérieur.
```

Cette seconde tentative doit rester la seule variante essayée. Un vérificateur qui essaierait plusieurs formes jusqu'à en trouver une qui marche finirait par valider un fichier modifié.

7. Niveau 2, la signature de laboratoire

ECDSA sur la courbe P-256, avec SHA-256, et un certificat qui remonte à l'autorité IsoFind. C'est ce niveau qui répond à « qui a envoyé ça ».

La procédure, en cinq contrôles

Contrôle	Ce qu'il établit	Sans lui
1. Signature ECDSA	Les blocs du scope n'ont pas bougé depuis la signature.	Rien n'est établi.
2. Chaîne de certification	Le certificat a été émis par l'autorité IsoFind.	La signature vaut, mais par quelqu'un que rien n'identifie.
3. Validité temporelle	Le certificat n'était pas expiré à la signature.	Une signature posée avec un certificat périmé.
4. Révocation	Le certificat n'était pas révoqué à la date de la CRL jointe.	Un certificat compromis peut avoir servi.
5. Identité déclarée	Que le sujet du certificat est bien le laboratoire attendu.	Un certificat valide, mais d'un autre laboratoire.

Le cinquième est le vôtre, et aucun programme ne le fera : comparez le sujet du certificat, affiché par le script, au laboratoire dont vous attendiez le fichier.

```
import base64
from cryptography import x509
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.primitives.asymmetric import ec

sig = doc["signature"]
scope = sig["signed_scope"]
```

```

chaine = [x509.load_pem_x509_certificate(base64.b64decode(c))
          for c in sig["certificate_chain"]]
labo    = chaine[0]                                # feuille d'abord, puis l'emettrice

payload = OrderedDict((b, doc[b]) for b in scope if b in doc)
canonique = json.dumps(payload, separators=(",", ":"), ensure_ascii=False)

labo.public_key().verify(
    base64.b64decode(sig["signature_b64"]),
    canonique.encode("utf-8"),
    ec.ECDSA(hashes.SHA256()),
) # leve InvalidSignature si le contenu a bouge

print(labo.subject.rfc4514_string())
print(labo.fingerprint(hashes.SHA256()).hex())

```

La chaine de certification

Le fichier embarque le certificat du laboratoire et celui de l'autorité émettrice. Il n'embarque pas la racine, et c'est volontaire : une racine fournie par le document qu'elle sert à valider ne prouverait rien. Vous devez l'obtenir une fois, par un canal distinct, et la conserver.

```
python verify_isof.py fichier.isof --root isofind_root_ca.pem
```

Sans la racine, ne concluez pas trop. Une signature dont l'émetteur n'a pas été vérifié établit que le document n'a pas bougé depuis qu'il a été signé, par quelqu'un. C'est utile, ce n'est pas une identité. Le script le dit explicitement plutôt que de rendre un verdict vert.

Révocation, et ce qu'une CRL embarquée vaut

Le bloc `revocation` contient une copie de la liste de révocation prise au moment de la signature. Elle permet de conclure hors ligne, ce qui est le point, mais elle est datée : elle établit que le certificat n'était pas révoqué à cette date, et rien sur aujourd'hui. Pour un contrôle courant, comparez la date de l'instantané à celle de la signature. Pour un dossier contentieux, allez chercher la CRL à jour à l'adresse indiquée par `crl_url`.

8. La provenance

La signature établit que le fichier n'a pas bougé après l'export. La provenance établit ce qui s'est passé avant : chaque modification subie par l'échantillon, dans l'ordre, avec son auteur.

Pourquoi deux chaines, et pourquoi celle-ci est vérifiable

En interne, IsoFind chaine son historique par HMAC-SHA256, avec une clé qui ne quitte jamais le poste. C'est ce qui empêche quelqu'un ayant accès à la base de recalculer une empreinte valide, et c'est aussi ce qui rend cette chaine invérifiable par vous : sans la clé, vous ne pouvez rien en faire.

Une seconde chaine est donc exportée, en SHA-256 nu, recalculable à partir des seules données présentes dans le fichier. C'est elle que la signature scelle. Signer l'empreinte interne aurait produit une attestation qui a l'apparence d'une preuve sans en être une : vous auriez lu « signé par le laboratoire X », constaté que la signature est valide, et n'auriez eu aucun moyen de contrôler ce qu'elle couvre.

Le calcul

Chaque évènement est haché avec l'empreinte du précédent, en partant de soixante-quatre zéros. Les champs qui entrent dans le calcul et leur ordre sont figés par le format.

```
CHAMPS = ("seq", "timestamp", "operation", "table", "row_id",
          "before", "after", "actor", "agent")

precedent = "0" * 64

for e in evenements:
    corps = {c: e.get(c) for c in CHAMPS}
    # REGLE B : clefs triées
    msg = precedent + "|" + json.dumps(corps, separators=(",", ":"),
                                       ensure_ascii=False, sort_keys=True)
    h = hashlib.sha256(msg.encode("utf-8")).hexdigest()

    assert e["seq"] == attendu      # pas de chainon manquant
    assert e["prev"] == precedent   # pas de rupture
    assert e["hash"] == h           # pas de retouche

    precedent = h

assert precedent == echantillon["head"]  # la tete recolle
```

Les trois contrôles attrapent trois fraudes distinctes. Le rang détecte un évènement retiré, le chainage détecte un évènement inséré, l'empreinte détecte une valeur modifiée.

Les deux modes

mode	Contenu	Ce que vous pouvez conclure
complet	Les évènements entiers, avec les valeurs avant et après.	Tout : ce que la valeur était, ce qu'elle est devenue, et que rien n'a été ajouté ni retiré.
resume	Les métadonnées et une empreinte du contenu, sans les valeurs.	Que l'historique est complet et non retouché. Pas ce qu'il contient.

Le mode résumé n'est pas une dérobade. Un fournisseur peut devoir prouver à son client qu'il n'a pas retouché ses mesures sans lui révéler ses corrections internes. La chaîne reste vérifiable, et si le contenu vous est révélé plus tard, vous pourrez confirmer qu'il est bien celui qui était couvert. En mode résumé, le corps haché est composé de `seq`, `timestamp`, `operation`, `table`, `row_id`, `actor`, `agent` et `content_sha256`.

Un aveu à ne pas manquer. Chaque échantillon porte un champ `internal_chain` qui dit si la chaîne HMAC interne était intacte au moment de l'export. Un laboratoire qui exporte une chaîne rompue le déclare donc lui-même, sous sa signature, et il ne peut pas le taire. Si vous lisez `valid: false`, la vérification cryptographique peut très bien passer : ce n'est pas une erreur de vérification, c'est une déclaration, et elle appelle une question au signataire.

La signature du bloc

La provenance est couverte de l'une des deux façons suivantes. Soit elle figure dans le scope de la signature du document, ce qu'IsoFind impose dès qu'un bloc de provenance est présent. Soit elle porte sa propre signature ECDSA, calculée sur la sérialisation réglée B du bloc entier, signature exclue.

Signer le bloc entier plutôt que la seule empreinte de tête n'est pas un détail : la tête ne couvre que le chaînage, et l'on pourrait changer un nom d'acteur ou une date sans invalider quoi que ce soit.

9. Les attestations d'origine

Un bloc `origin_claims` porte des conclusions du type « cet échantillon est compatible avec ce gisement ». Ce n'est plus de l'intégrité, c'est un raisonnement, et il se vérifie autrement.

Ces attestations suivent le schéma ISOF v1.3. Le vérificateur ne juge pas la conclusion, il contrôle que le raisonnement se tient : que les données citées existent dans le fichier, qu'elles n'ont pas changé depuis l'émission, que le verdict est compatible avec le type d'attestation, et qu'aucune référence ne tourne en rond.

Ce que porte une attestation

Champ	Contenu
<code>claim_type</code>	<code>origin</code> , <code>non_origin</code> , <code>compatibility</code> , <code>exclusion</code>
<code>subject</code>	L'échantillon visé, et son <code>sample_hash</code> au moment de l'émission
<code>target</code>	Ce à quoi on le compare : gisement, fournisseur, zone géographique, lot, procédé
<code>verdict</code>	La décision, <code>match</code> , <code>no_match</code> ou <code>inconclusive</code> , et son modèle d'incertitude
<code>evidence</code>	Les éléments invoqués, chacun pointant une donnée du fichier ou une autre attestation

Champ	Contenu
<code>method_chain</code>	La méthode employée, et le moteur qui l'a exécutée
<code>limitations</code>	Obligatoire, et non vide. Le format refuse une conclusion qui ne dit pas ce qu'elle ne couvre pas.

Les combinaisons refusées

Un verdict doit être compatible avec le type d'attestation, faute de quoi le sens s'inverse sans qu'on s'en aperçoive.

<code>claim_type</code>	Décisions admises
<code>origin</code>	<code>match</code> , <code>inconclusive</code>
<code>non_origin</code>	<code>no_match</code> , <code>inconclusive</code>
<code>compatibility</code>	<code>match</code> , <code>inconclusive</code>
<code>exclusion</code>	<code>no_match</code> , <code>inconclusive</code>

Les codes que rend le vérificateur

Code	Sens
<code>ISO-F-002</code>	Attestation malformée, elle ne suit pas le schéma
<code>ISO-F-200</code>	Échantillon visé introuvable, et aucune référence externe
<code>ISO-F-201</code>	<code>sample_hash</code> ne correspond plus : l'échantillon a changé depuis l'émission
<code>ISO-F-202</code>	Une donnée citée n'existe pas dans le document
<code>ISO-F-204</code>	Verdict incompatible avec le type d'attestation
<code>ISO-F-210</code>	Attestation citée introuvable, ou d'un type inattendu
<code>ISO-F-211</code>	Empreinte d'une attestation citée non concordante
<code>ISO-F-212</code>	Cycle dans le graphe des attestations
<code>ISO-F-213</code>	Profondeur au-delà du seuil recommandé, avertissement
<code>ISO-F-214</code>	Attestation composite sans bloc expliquant l'agrégation

Le `sample_hash` se recalcule comme le reste : règle B, sur l'échantillon entier privé de son propre champ `sample_hash`, préfixé de `sha256:`. Un écart n'est pas nécessairement une fraude, une attestation émise avant une mise à jour de l'échantillon produit le même symptôme, mais dans les deux cas l'attestation ne s'applique plus à ce qu'elle prétend viser.

Lisez `limitations`. C'est le seul champ obligatoire dont le contenu n'est vérifiable par aucun programme, et c'est celui qui vous dira ce que l'attestation ne couvre pas. Un émetteur sérieux y écrit quelque chose d'utile.

10. Un fichier chiffré

Un fichier peut être chiffré pour un destinataire précis. Vous pouvez alors vérifier son origine sans pouvoir en lire le contenu, et c'est délibéré.

Le bloc `encryption` porte `active: true`, et les blocs de données contiennent du texte chiffré au lieu des mesures.

```
"encryption": {
  "active": true,
  "algorithm": "X25519-HKDF-SHA256 + AES-256-GCM",
  "recipient_fingerprint": "...", // pour qui, sans le nommer
  "ephemeral_public_key": "...",
  "session_key_enc": "...", "wrap_nonce": "...", "nonce": "..."
}
```

La clé de session est tirée au hasard pour ce fichier, elle chiffre les données en AES-256-GCM, et elle est elle-même encapsulée pour la clé publique X25519 du destinataire. Personne d'autre ne peut la retrouver, y compris l'expéditeur une fois le fichier écrit.

Ce qui reste vérifiable

La signature d'un fichier chiffré porte le scope `["encrypted_metadata"]`, et couvre trois champs restés en clair : `created_by`, `project` et `created_at`, dans cet ordre. Vous pouvez donc établir qui a émis le fichier, pour quel projet et quand, sans rien savoir de son contenu.

Ce que cette signature ne couvre pas. Les blocs chiffrés eux-mêmes n'entrent pas dans le calcul. Quelqu'un pourrait remplacer le contenu chiffré par un autre sans invalider la signature ; le destinataire s'en apercevrait au déchiffrement, puisque l'authentification GCM échouerait, mais un tiers qui vérifie seulement la signature ne le verrait pas. Ne concluez donc jamais « le contenu chiffré est authentique », mais « l'enveloppe vient de tel émetteur ».

Le déchiffrement demande la clé privée du destinataire et ne fait pas partie de ce guide. Si le fichier vous est destiné, IsoFind ou le paquet `isof` s'en chargent ; une fois déchiffré, le document redevient un ISOF ordinaire, et les sections 6 à 9 s'appliquent à ce qu'il contient.

Interpréter

Ce qu'un verdict autorise à écrire, et les erreurs de lecture qui coutent le plus cher.

11. Lire un verdict

Une vérification rend rarement un simple oui ou non. Voici ce que chaque sortie autorise à écrire, et ce qu'elle appelle comme geste.

Sortie	Ce que cela veut dire	Ce qu'il y a à faire
Signature de niveau 2 valide, chaîne remontée jusqu'à la racine	Les blocs couverts sont exactement ceux qu'a envoyés le laboratoire nommé dans le certificat.	Vérifier que ce laboratoire est bien celui dont vous attendiez le fichier, et regarder le scope.
Signature valide, racine non fournie	Le fichier n'a pas bougé depuis sa signature, par quelqu'un dont l'identité n'est pas établie.	Se procurer le certificat racine et recommencer. Sans cela, n'écrivez pas le nom du laboratoire dans votre rapport.
Signature de niveau 1 valide	Le fichier n'a pas été altéré par accident. Rien de plus.	Si l'enjeu est opposable, demander un fichier signé au niveau 2.
Format antérieur signalé	Le fichier est intact ; il a été signé avant août 2026, quand <code>created_by</code> était couvert sans être déclaré.	Rien. C'est une note de compatibilité, pas un défaut du fichier.
Empreinte ou signature non concordante	Un bloc couvert a changé après la signature. Le fichier ne vaut plus rien comme preuve.	Ne pas chercher où : redemander le fichier à la source. Une différence peut aussi venir d'un outil qui a réenregistré le JSON en le reformatant.
Chaîne de provenance rompue	L'historique exporté a été retouché : un événement retiré, inséré, ou modifié.	Question au signataire. La signature du document peut échouer aussi, puisqu'elle couvre la provenance.
Chaîne interne déclarée rompue	Le laboratoire déclare lui-même, sous sa signature, que son historique local était incohérent à l'export.	Ce n'est pas une erreur de vérification, c'est un aveu. Demander l'explication avant d'utiliser les données.
Certificat expiré	À distinguer : expiré avant la signature, ou expiré depuis. Le second cas est normal pour un vieux fichier.	Comparer <code>signed_at</code> à la date de fin de validité du certificat.
Certificat révoqué	Le certificat a été retiré par l'autorité, souvent après une compromission.	Ne pas se fier au fichier. Vérifier la date de révocation par rapport à celle de la signature.
Aucune signature	Des données, rien d'autre.	Utilisables comme un tableur. Ne pas les citer comme opposables.

La phrase qui tient

Quand tout passe, ce que vous pouvez écrire est : « le fichier reçu le tant est intègre, il porte une signature valide du laboratoire X émise par l'autorité IsoFind, et couvre les blocs suivants ».

Ce que vous ne pouvez pas écrire est : « les mesures sont certifiées », « les données sont exactes », ou « le laboratoire est accrédité ». Rien de tout cela n'a été vérifié, et rien de tout cela n'était promis.

12. Douze pièges

Les erreurs de lecture qu'on voit revenir, dans l'ordre de ce qu'elles coutent.

1. **Le scope partiel.** Une signature ne couvre que les blocs qu'elle liste. Un fichier peut porter une signature parfaitement valide et un bloc `assignments` modifié après coup. Lisez le scope avant de conclure, et citez-le.
2. **Prendre le niveau 1 pour une signature.** C'est une empreinte, recalculable par quiconque modifie le fichier. Le nom dans `signed_by` est déclaratif et n'engage personne.
3. **Confondre `ratio` et `derived`.** La mesure reste toujours dans `ratio`. Quand `derived` existe, une correction a été appliquée, et c'est souvent elle que le laboratoire considère comme la valeur en vigueur. Citer l'une pour l'autre change le résultat sans que rien ne le signale.
4. **Croire que la racine est dans le fichier.** Elle n'y est pas, et elle n'y sera jamais. Une racine fournie par le document qu'elle valide ne prouverait rien.
5. **Trier les clés en recalculant.** La signature du document n'est pas triée. Un vérificateur qui trie n'obtiendra jamais la bonne empreinte, et conclura à une altération sur un fichier intact.
6. **Vérifier dans un autre langage sans y penser.** Le formatage des nombres diffère d'un langage à l'autre. Un `1.0` devenu `1` suffit à faire échouer une vérification.
7. **Reformater le fichier avant de le vérifier.** Ouvrir un ISOF dans un outil qui réindente et réenregistre ne change pas les données, mais peut changer l'ordre des clés ou l'écriture des nombres. Vérifiez le fichier tel qu'il est arrivé.
8. **Lire une CRL embarquée comme une réponse d'aujourd'hui.** C'est un instantané daté du jour de la signature. Pour un contentieux, allez chercher la liste à jour.
9. **Ignorer `internal_chain`.** Un laboratoire peut signer un historique dont il déclare lui-même qu'il était rompu. La vérification cryptographique passe, et pourtant quelque chose ne va pas.
10. **Prendre le mode résumé pour une provenance complète.** Elle prouve qu'il n'y a eu ni ajout ni retrait ni retouche. Elle ne montre pas ce que les valeurs étaient.
11. **Conclure sur le contenu d'un fichier chiffré.** La signature ne couvre que les métadonnées en clair. L'enveloppe est authentifiée, pas le contenu.
12. **Confondre « aucune attestation d'origine » et « origine non établie ».** La plupart des fichiers n'en portent pas, parce que la plupart des échanges ne concluent rien sur une origine. L'absence n'est pas un résultat négatif.

Annexes

Le vérificateur, l'obtention d'un certificat, et où aller ensuite.

13. Le vérificateur autonome

`verify_isof.py` accompagne ce guide. Une seule dépendance, aucune connexion, aucun composant IsoFind.

```
pip install cryptography
python verify_isof.py fichier.isof --root isofind_root_ca.pem
```

Il enchaîne ce que les sections 6 à 8 décrivent : la signature du document au niveau trouvé, la chaîne de certification quand une racine est fournie, l'expiration, la révocation par la CRL embarquée, puis la chaîne de provenance événement par événement. Il sort avec le code 0 quand tout ce que le fichier permettait de contrôler est passé, et 1 sinon, ce qui permet de l'appeler depuis une chaîne de traitement.

Une sortie type

```
ISOFile file      : essai.isof
format version   : 1.2
created by       : HydroSciences Montpellier

Document signature
-----
PASS level 2 ECDSA signature matches
note covered blocks: created_by, samples, methods, purification, provenance
note certificate subject: O=HSM,CN=HydroSciences Montpellier
note certificate fingerprint: b4a2e1a765de1f70...
PASS certificate chains up to the supplied root
note CRL snapshot is 3 days old

Provenance
-----
note mode: complet
PASS provenance chain recomputed: 2 event(s) over 1 sample(s)

VERDICT: every check that this file allows has passed.
This says the file has not been altered since it was signed. It says
nothing about whether the measurements are correct.
```

Les lignes **WARN** et **skip** comptent autant que les **PASS**. Elles disent ce qui n'a pas pu être contrôlé : racine absente, fichier sans provenance, CRL manquante. Un verdict vert obtenu sans elles ne recouvre pas la même chose.

Le script est court et lisible : il est fait pour être relu, adapté, ou réimplémenté dans un autre langage à partir des règles de la partie II. Le paquet `isof`, publié sur PyPI sous licence MIT, fournit une implémentation de référence plus complète, chiffrement compris.

14. Obtenir un certificat

Cette section s'adresse à l'autre bord : celui qui veut émettre des fichiers signés au niveau 2, et non les vérifier.

Comment cela se passe

1. Le laboratoire génère une paire de clés ECDSA P-256 sur son poste, et une demande de certificat portant son identité. La clé privée ne quitte jamais la machine, et n'est jamais transmise.
2. La demande part vers l'autorité IsoFind, qui contrôle l'identité de l'entité déclarée.
3. Le certificat signé revient et se range dans le magasin local, chiffré au repos.
4. À partir de là, chaque export peut être signé au niveau 2, et chaque fichier emporte le certificat et celui de l'autorité émettrice.

Ce que la clé privée ne fait jamais

Elle ne quitte pas le poste, elle n'est pas déposée chez un tiers, elle n'est pas récupérable en cas de perte. Un certificat perdu se remplace, les fichiers déjà signés restent vérifiables, et c'est le comportement attendu d'une signature.

Les paliers

Entité signataire	Par an
Laboratoire académique ou organisme public	80 €
Entreprise jusqu'à 20 personnes	300 €
Entreprise de 20 à 250 personnes	700 €
Entreprise de plus de 250 personnes	1 500 €
Autorité de contrôle, défense	sur devis

Le tarif couvre le contrôle d'identité et la tenue de l'autorité, pas les données signées, dont IsoFind ne reçoit jamais copie. Vérifier un fichier reste gratuit et sans inscription, pour toujours : un format dont la vérification serait payante ne servirait à rien.

Le niveau 1 reste disponible sans certificat. Il suffit pour un échange entre collègues, une archive, un dépôt de données. Le certificat n'a d'intérêt que si vos fichiers doivent être opposables à quelqu'un.

15. Références

Les algorithmes employés

Usage	Algorithme
Empreinte de document, chainage de provenance	SHA-256
Signature de document et de provenance	ECDSA P-256, SHA-256
Chainage interne de l'historique, non exporté	HMAC-SHA256
Chiffrement d'un fichier pour un destinataire	X25519, HKDF-SHA256, AES-256-GCM
Empreinte d'échantillon et d'attestation	SHA-256, préfixée sha256:

Versions

Le document décrit ISOF v1.2, la version qu'IsoFind écrit dans `isof_version`. Le schéma des attestations d'origine porte son propre numéro, v1.3, et évolue séparément. La bascule des règles de sérialisation vers RFC 8785 est prévue pour la version majeure suivante, en rupture annoncée : les fichiers émis aujourd'hui resteront vérifiables selon les règles de ce guide.

Vocabulaire

Terme	Sens dans ce guide
Scope	La liste des blocs qu'une signature couvre. Ce qui n'y est pas n'est pas protégé.
Provenance	L'historique signé des modifications subies par un échantillon avant l'export.
Chaîne interne	Le chainage HMAC local, non exportable et non vérifiable par un tiers, dont l'état est déclaré dans le fichier.
Attestation d'origine	Une conclusion motivée sur la provenance géographique ou industrielle d'un échantillon.
Niveau 1, niveau 2	Empreinte seule, ou signature avec certificat.

Aller plus loin

- Le paquet `isof`, sur PyPI, licence MIT : implémentation de référence du format, lecture, écriture, signature et chiffrement.
- Le certificat racine de l'autorité IsoFind, à obtenir une fois et à conserver.
- IsoFind, le logiciel qui produit ces fichiers. Il n'est nécessaire ni pour les lire ni pour les vérifier, et c'est le point.

Un doute sur un fichier reçu. Le premier réflexe utile n'est pas cryptographique : demandez au signataire de vous renvoyer le fichier, et comparez les deux. Une différence entre deux envois se voit tout de suite, et elle en dit plus long qu'une empreinte.