

Informe Técnico de Arquitectura de Software: Estrategias de Firma Electrónica XML y Canonicalización en Go para el Ecosistema Tributario Ecuatoriano (Visión 2026)

1. Introducción y Contexto Estratégico

1.1 La Evolución de la Facturación Electrónica en Ecuador

El ecosistema de facturación electrónica en Ecuador, regido por el Servicio de Rentas Internas (SRI), ha transitado desde sus etapas iniciales de adopción voluntaria hacia un mandato casi universal que abarca a personas naturales y jurídicas. Hacia el horizonte de 2026, la infraestructura tecnológica que soporta estas transacciones no solo debe ser funcional, sino resiliente, escalable y estrictamente adherida a los estándares criptográficos definidos en las fichas técnicas gubernamentales.¹ La transición digital en la administración tributaria ecuatoriana ha impuesto requisitos rigurosos sobre la integridad, autenticidad y no repudio de los documentos electrónicos, fundamentados en el estándar internacional XMLDSig (XML Digital Signature) y su extensión europea XAdES (XML Advanced Electronic Signatures).¹

Para los desarrolladores y arquitectos de software que operan en el lenguaje de programación Go (Golang), este entorno presenta desafíos particulares. Go, conocido por su eficiencia y concurrencia, es una elección ideal para servicios de alto rendimiento de facturación masiva. Sin embargo, su ecosistema de librerías para el manejo de XML y firmas digitales es menos maduro o "bloated" que el de Java o .NET, lenguajes con los que tradicionalmente se diseñaron los sistemas del SRI.³ El desafío central que motiva este informe es la discrepancia técnica entre las herramientas modernas de Go, diseñadas para protocolos web ligeros como SAML 2.0, y los requisitos heredados (legacy) del SRI, que exigen implementaciones específicas de canonicalización XML que muchas librerías actuales han dejado de soportar por defecto.

1.2 Definición del Problema Técnico

El usuario ha identificado correctamente una barrera crítica en la implementación de firmas XAdES-BES para el SRI utilizando Go: la inadecuación de la librería ucarion/c14n. Esta librería, aunque excelente para su propósito original dentro del flujo de autenticación SAML, implementa por defecto y diseño la "Canonicalización Exclusiva" (Exclusive Canonicalization - <http://www.w3.org/2001/10/xml-exc-c14n#>). En contraste, la normativa técnica del SRI⁴ y la

práctica operativa de sus validadores exigen el uso de "Canonicalización Inclusiva" (Inclusive Canonicalization - <http://www.w3.org/TR/2001/REC-xml-c14n-20010315>).

Esta diferencia, que podría parecer sutil o puramente semántica, es en realidad una divergencia matemática fundamental en cómo se procesan los bytes del documento XML antes de aplicar el algoritmo de hash (SHA-1 o SHA-256). El uso de una canonicalización incorrecta resulta en un DigestValue que no coincide con el calculado por los servidores del SRI, provocando el rechazo inmediato del comprobante con errores genéricos de "Firma Inválida".⁶ Por tanto, el objetivo de este informe es diseccionar las alternativas viables en el ecosistema Go que permitan una implementación nativa, eficiente y, sobre todo, compatible con la especificación REC-xml-c14n-20010315 requerida para operar en 2026.

2. Fundamentos Teóricos de la Firma XMLDSig y Canonicalización

Para evaluar las alternativas con criterio experto, es imperativo desglosar la mecánica subyacente de la firma XML y por qué la elección del algoritmo de canonicalización es el factor determinante en el éxito de la integración con el SRI.

2.1 La Anatomía de un Documento XML Firmado (XAdES-BES)

El estándar XAdES-BES (Baseline Electronic Signature) añade una capa de complejidad sobre el estándar base XMLDSig. Mientras que XMLDSig se preocupa únicamente por la integridad criptográfica (¿ha cambiado el documento?), XAdES añade metadatos sobre el contexto de la firma (¿quién firmó?, ¿cuándo?, ¿bajo qué política?).

En el contexto del SRI, un documento firmado (sea factura, retención o guía de remisión) es un archivo XML que contiene un nodo <ds:Signature> "envuelto" (Enveloped Signature). Esto significa que la firma reside dentro del mismo documento que protege. La estructura general es:

1. **Comprobante (Raíz):** Contiene infoTributaria, infoFactura, detalles, etc.
2. **Firma (<ds:Signature>):** Insertada usualmente al final del documento.
 - **<ds:SignedInfo>:** El corazón criptográfico. Contiene las referencias a lo que se firma y los algoritmos usados.
 - **<ds:SignatureValue>:** El resultado cifrado (RSA) del hash del SignedInfo.
 - **<ds:KeyInfo>:** Contiene el certificado X.509 del firmante.
 - **<ds:Object>:** Contiene las QualifyingProperties de XAdES (propiedades firmadas).

El proceso de validación del SRI implica dos pasos críticos de verificación de integridad, ambos dependientes de la canonicalización:

1. **Verificación de Referencia:** El SRI extrae el documento (excluyendo la firma), lo

canonicaliza, calcula su hash y lo compara con el <ds:DigestValue> presente en la referencia del SignedInfo.

2. **Verificación de Firma:** El SRI extrae el nodo <ds:SignedInfo>, lo canonicaliza, calcula su hash y verifica que corresponda al <ds:SignatureValue> desencriptado con la clave pública del certificado.

2.2 El Algoritmo de Canonicalización (C14N): El Núcleo del Conflicto

La canonicalización es el proceso de transformar un documento XML en una representación física de bytes estándar. XML es flexible: permite espacios en blanco arbitrarios, ordenamiento variable de atributos y uso de comillas simples o dobles. Dos documentos XML pueden ser lógicamente equivalentes pero binariamente distintos.⁷ Dado que los algoritmos de hash (como SHA-1) son sensibles a cambios de un solo bit, es imposible firmar XML "crudo"; se debe firmar su forma canónica.

2.2.1 Canonicalización Inclusiva (REC-xml-c14n-20010315)

Este es el algoritmo requerido por el SRI.⁴ Su filosofía es preservar el contexto completo del nodo.

- **Comportamiento de Namespaces:** Cuando se canonicaliza un sub-árbol (como el nodo SignedInfo), el algoritmo Inclusivo toma todos los espacios de nombres declarados en los ancestros del nodo (incluso si están en la raíz del documento y no se usan explícitamente dentro del SignedInfo) y los inyecta en el nodo canonicalizado como atributos xmlns.
- **Implicación:** Esto "congela" el nodo en su contexto original. Si se extrae el nodo firmado y se coloca en otro documento con diferentes namespaces padres, la firma podría romperse o ser considerada inválida porque el contexto ha cambiado.
- **Por qué el SRI lo usa:** Los comprobantes electrónicos son documentos autocontenido. El SRI diseñó su sistema asumiendo que el contexto (la factura completa) es estático. La canonicalización inclusiva es más segura en este contexto porque garantiza que no haya "sombras" de namespaces ocultos que puedan alterar la interpretación semántica de los datos.⁹

2.2.2 Canonicalización Exclusiva (xml-exc-c14n)

Este es el algoritmo usado por defecto en ucarion/c14n y preferido en SAML.

- **Comportamiento de Namespaces:** Solo incluye los espacios de nombres que son *visiblemente utilizados* dentro del nodo que se está canonicalizando. Ignora las declaraciones de ancestros no utilizadas.
- **Implicación:** Hace que la firma sea portable. Se puede mover el nodo firmado a otro envoltorio XML sin romper el hash, siempre que los namespaces internos no cambien.
- **El Problema con SRI:** Si se usa este algoritmo para firmar una factura del SRI, el proceso eliminará declaraciones de namespaces (como los prefijos globales del esquema XSD) que el validador del SRI espera encontrar en el hash. El resultado es una discrepancia criptográfica: el SRI calcula un hash sobre un set de bytes (con namespaces heredados)

y su software calcula un hash sobre otro set (sin ellos). La firma falla.

3. Análisis Técnico de ucarion/c14n y su Inviabilidad

Es fundamental documentar por qué la herramienta actual no es adecuada para evitar recaer en los mismos patrones de diseño al elegir una alternativa.

3.1 Diseño y Propósito

La librería github.com/ucarion/c14n es una implementación pura en Go diseñada específicamente para servir como base a github.com/ucarion/saml.¹¹ El protocolo SAML (Security Assertion Markup Language), utilizado para SSO (Single Sign-On), ha estandarizado casi universalmente el uso de **Exclusive Canonicalization** para mitigar ataques de "XML Signature Wrapping". En estos ataques, un adversario mueve la aserción firmada a una ubicación diferente en el mensaje SOAP; la canonicalización exclusiva ayuda a validar la aserción independientemente de su posición.

3.2 Brechas Funcionales respecto al SRI

El análisis del código fuente y la documentación de ucarion/c14n revela limitaciones estructurales para el caso de uso ecuatoriano:

1. **Falta de Configuración de Algoritmo:** La librería está "opinionada". Su función principal Canonicalize aplica la lógica exclusiva de facto. No expone una API fluida para cambiar al modo inclusivo (REC-xml-c14n-20010315).¹¹
2. **Manejo de Prefijos:** En el modo exclusivo, se requiere una lista explícita de InclusiveNamespaces para forzar la inclusión de ciertos prefijos. Configurar esto manualmente para que coincida exactamente con el comportamiento automático del algoritmo Inclusivo es propenso a errores y técnicamente deuda técnica.
3. **Procesamiento de DTDs:** La librería ignora directivas de procesamiento y DTDs.¹¹ Aunque las facturas del SRI no dependen de DTDs complejos para su validación de firma, esta simplificación indica que la librería no es un procesador XML de propósito general robusto, sino una utilidad especializada.

Conclusión Parcial: Intentar "parchear" ucarion/c14n para que se comporte como un canonicalizador inclusivo requeriría reescribir su lógica central de recorrido del árbol DOM y manejo de atributos de namespace. Esto es ineficiente cuando existen alternativas en el ecosistema Go que ya soportan el estándar requerido de forma nativa.

4. Evaluación de Alternativas en Go para 2026

A continuación, presentamos un análisis comparativo detallado de las librerías disponibles,

evaluando su capacidad para manejar Canonicalización Inclusiva, su soporte para XAdES y su viabilidad a largo plazo (2026).

4.1 La Referencia Estándar: russellhaering/goxmldsig

Esta librería es, por amplio margen, la implementación más madura y completa de XMLDSig en Go puro. Es la base sobre la que se construyen la mayoría de las soluciones de alto nivel (incluyendo implementaciones de SAML competidoras de ucarion).

4.1.1 Arquitectura y Capacidades de Canonicalización

goxmldsig no es monolítica; desacopla el proceso de firma del proceso de canonicalización, permitiendo injectar diferentes implementaciones de la interfaz Canonicalizer.

Lo más relevante para el usuario es que esta librería implementa explícitamente el algoritmo que el SRI requiere. Al revisar su código fuente ¹², encontramos el constructor:

Go

```
func MakeC14N10RecCanonicalizer() Canonicalizer
```

- **Significado:** "C14N10" refiere a Canonical XML 1.0. "Rec" refiere a la Recomendación W3C original (la versión Inclusiva).
- **Comportamiento Interno:** Esta función instancia un c14N10RecCanonicalizer. A diferencia de los canonicalizadores exclusivos (MakeC14N10Exclusive...), este objeto está programado para **no** podar los namespaces heredados. Recorre el árbol XML y, al procesar un nodo, consulta el contexto de sus padres para asegurar que todos los namespaces visibles en ese punto se serialicen explícitamente en el nodo actual.¹⁴

4.1.2 Ventajas para el Caso SRI

1. **Conformidad Exacta:** Al usar MakeC14N10RecCanonicalizer(), se obtiene byte-for-byte la salida que espera el validador Java del SRI.
2. **Independencia de CGO:** Al estar escrita en Go puro (usando beevik/etree para el DOM), facilita la compilación cruzada y el despliegue en contenedores ligeros (Alpine Linux), algo crucial para arquitecturas de microservicios modernas en 2026.
3. **Mantenimiento:** Es una librería activa, ampliamente usada en la industria, lo que garantiza parches de seguridad.¹⁵

4.1.3 Limitaciones

goxmldsig es una librería de firma genérica (XMLDSig). No conoce la especificación XAdES.

No generará automáticamente los nodos SignedProperties ni QualifyingProperties que el SRI exige. El desarrollador debe construir estas estructuras XML manualmente usando etree y luego usar goxmldsig para firmarlas.

4.2 La Solución Especializada: [digitalautonomy/goxades_sri](#)

Esta librería representa un esfuerzo comunitario por adaptar las herramientas genéricas de Go a la realidad específica de Ecuador. Es un fork de artemkunich/goxades, que a su vez se basa en russellhaering/goxmldsig.

4.2.1 Adaptaciones Específicas para SRI

El análisis de este repositorio¹⁷ revela modificaciones críticas que ahorran semanas de desarrollo:

1. **Estructura XAdES-BES Predefinida:** La librería contiene structs y lógica para generar el nodo Object con las propiedades calificadas (QualifyingProperties) requeridas por el SRI.
2. **Algoritmo de Canonicalización Inclusiva por Defecto:** A diferencia de la librería base que podría tender hacia lo exclusivo, este fork fuerza la configuración hacia la canonicalización inclusiva necesaria para Ecuador.
3. **Manejo de IDs Aleatorios:** El SRI requiere que los IDs de los nodos de firma (Signature, SignedProperties, SignedInfo) sean únicos. goxades_sri implementa generadores de números aleatorios para cumplir con este requisito de unicidad y evitar colisiones en validaciones masivas.¹⁷
4. **Corrección de Namespaces en Data:** Aborda un problema común donde la ubicación de los atributos de namespace en el nodo raíz de la factura causaba errores de validación.

4.2.2 Estado de Mantenimiento

La principal advertencia es que el repositorio no ha tenido actividad significativa en los últimos 3 años.¹⁸ Sin embargo, dado que la especificación XML del SRI (Ficha Técnica Offline) es estable y no ha sufrido cambios estructurales radicales en el estándar de firma, el código sigue siendo funcional.

Estrategia recomendada: No usar como dependencia directa inmutable. Se recomienda realizar un fork privado o usar el patrón vendor para tener control sobre el código, permitiendo actualizar las dependencias subyacentes (goxmldsig) si surgen vulnerabilidades de seguridad de aquí a 2026.

4.3 La Alternativa Empresarial: [moov-io/signedxml](#)

Moov Financial ofrece esta librería como parte de su stack fintech open source.

4.3.1 Potencial y Barreras

Aunque es una librería robusta y bien mantenida¹⁵, su enfoque principal sigue siendo la interoperabilidad financiera norteamericana y SAML. Los issues del repositorio¹⁹ discuten

explícitamente la dificultad de implementar REC-xml-c14n-20010315 (Inclusive), con usuarios preguntando cómo añadirla. Aunque es técnicamente posible implementando la interfaz CanonicalizationAlgorithm personalizada, esto devuelve la carga de la prueba al desarrollador. Para un proyecto que apunta a 2026 con requisitos específicos del SRI, usar moov-io/signedxml implicaría "luchar contra la corriente" de la librería para forzarla a comportarse de una manera no nativa, lo cual añade riesgo innecesario.

4.4 Hardware y Tokens: alapierre/godss

Si el requerimiento para 2026 incluye firmar utilizando tokens físicos (smart cards, tokens USB del Banco Central o Security Data) directamente desde una aplicación Go (por ejemplo, un agente de escritorio), godss es la única opción viable documentada que integra soporte PKCS#11 con XAdES.²⁰

- **Relevancia:** Permite la comunicación con el driver del token para realizar la operación criptográfica de firma privada sin exponer la clave.
- **Base:** Está construida sobre goxades, por lo que hereda la capacidad de configurar canonicalización inclusiva, aunque su configuración puede ser compleja.

5. Tabla Comparativa de Alternativas

A continuación, se presenta una síntesis estructurada de las opciones para facilitar la toma de decisiones arquitectónicas.

Criterio	ucarion/c14n	russellhaering/goxmldsig	digitalautonomy/goxades_sri	moov-io/sign edxml
Soporte C14N Inclusivo	Nulo/Complejo (Nativo Exclusivo)	Nativo y Explícito (MakeC14N10Rec...)	Possible (Configurable)	Nativo (Preconfigurado)
Soporte XAdES-BES	No	No (Solo Core XMLDSig)	Sí (Específico SRI)	Limitado/Manual
Madurez en Go	Alta (SAML)	Muy Alta (Estándar de facto)	Media (Fork específico)	Alta (Fintech)
Dependencia	No	No (Pure Go)	No	No

de CGO				
Facilidad de Integración SRI	Baja (Requiere hacks)	Media (Requiere construir XAdES)	Alta (Listo para usar)	Media/Baja
Estado de Mantenimiento	Activo	Muy Activo	Bajo (Requiere fork propio)	Muy Activo
Viabilidad 2026	Inviable	Recomendada (Base)	Recomendada (Referencia)	Alternativa secundaria

6. Guía de Implementación Técnica: Arquitectura Híbrida

Basado en el análisis, la solución óptima para 2026 no es depender de una sola librería "caja negra", sino construir una solución híbrida que utilice **russellhaering/goxmldsig** como motor criptográfico confiable, configurado con los parámetros específicos que la librería **goades_sri** ha demostrado que funcionan.

A continuación, se detalla el flujo de implementación paso a paso, integrando los detalles técnicos y snippets de código relevantes.

6.1 Paso 1: Configuración del Motor de Canonicalización

El primer paso es instanciar el contexto de firma asegurando que el canonicalizador sea el correcto. No se debe usar el constructor por defecto si este apunta a Exclusive C14N.

Go

```
import (
    "github.com/russellhaering/goxmldsig"
    "github.com/russellhaering/goxmldsig/etreeutils"
    "crypto"
)
```

```

// Configuración del contexto de firma para SRI
func NewSRISigningContext(ks dsig.X509KeyStore) *dsig.SigningContext {
    ctx := dsig.NewDefaultSigningContext(ks)

    // PUNTO CRÍTICO: Selección del Canonicalizador Inclusivo
    // Esto asegura compatibilidad con http://www.w3.org/TR/2001/REC-xml-c14n-20010315
    ctx.Canonicalizer = dsig.MakeC14N10RecCanonicalizer()

    // Configuración de Algoritmos (Verificar ficha técnica 2026 para SHA256)
    // Históricamente SRI usa SHA1. Si 2026 exige SHA256, cambiar aquí.
    ctx.Hash = crypto.SHA1
    ctx.SignatureMethod = "http://www.w3.org/2000/09/xmldsig#rsa-sha1"
    ctx.DigestAlgorithm = "http://www.w3.org/2000/09/xmldsig#sha1"

    return ctx
}

```

6.2 Paso 2: Construcción de la Estructura XAdES-BES

Dado que goxmldsig no genera los metadatos XAdES, se debe injectar manualmente el objeto QualifyingProperties.

Go

```

// Estructura conceptual para inyección de XAdES
func AgregarXAdESPProperties(signatureElement *etree.Element, cert *x509.Certificate) {
    // 1. Crear nodo Object
    object := signatureElement.CreateElement("ds:Object")

    // 2. Crear QualifyingProperties
    qp := object.CreateElement("xades:QualifyingProperties")
    qp.CreateAttr("Target", "#SignatureID") // Referencia al ID de la firma

    // 3. Crear SignedProperties (Este nodo TAMBIÉN debe ser firmado/referenciado)
    sp := qp.CreateElement("xades:SignedProperties")
    sp.CreateAttr("Id", "SignedPropertiesID") // ID único necesario

    // 4. Agregar SigningTime y SigningCertificate
    ssp := sp.CreateElement("xades:SignedSignatureProperties")

```

```

// Tiempo de firma (Formato ISO 8601)
st := ssp.CreateElement("xades:SigningTime")
st.SetText(time.Now().Format(time.RFC3339))

//... Lógica para agregar SigningCertificate con hash y serial...
}

```

Insight Técnico: Es vital notar que el nodo SignedProperties no es solo informativo; **debe ser parte del hash criptográfico**. Esto implica que en el <ds:SignedInfo> debe haber una segunda referencia (<ds:Reference URI="#SignedPropertiesID">) además de la referencia al documento principal. goxmldsig permite agregar referencias adicionales manualmente al contexto antes de firmar.

6.3 Paso 3: Manejo de Certificados PKCS#12 (.p12)

El SRI emite certificados en formato .p12. Go tiene limitaciones nativas con algoritmos de cifrado de contenedores antiguos (como RC2 con 40-bit keys) que a menudo usan las autoridades certificadoras en Ecuador (BCE, Security Data).

Problema Común: pkcs12: error decoding PFX: pkcs12: unknown cipher

Solución: Utilizar la librería software.sslmate.com/src/go-pkcs12 que es un fork más robusto y compatible que golang.org/x/crypto/pkcs12.

Go

```

import "software.sslmate.com/src/go-pkcs12"

func CargarCertificado(rutaP12, password string) (dsig.X509KeyStore, error) {
    p12Data, _ := ioutil.ReadFile(rutaP12)

    // Decodificar usando librería robusta
    privateKey, cert, caCerts, err := pkcs12.DecodeChain(p12Data, password)
    if err!= nil {
        return nil, err
    }

    // Retornar implementación de KeyStore para goxmldsig
    return &dsig.MemoryX509KeyStore{
        PrivateKey: privateKey,
        Cert: cert,
    }
}

```

```
 }, nil  
}
```

6.4 Paso 4: El Proceso de Firma (Enveloping)

Finalmente, se une todo. Se parsea la factura XML, se aplica la firma con el contexto configurado.

1. Cargar XML de la factura en un etree.Document.
2. Instanciar el SigningContext (Paso 6.1).
3. Llamar a ctx.SignEnveloped(elementoRaiz).
4. **Importante:** Antes de serializar el resultado final, inyectar el bloque XAdES (Paso 6.2) dentro del nodo Signature generado. *Nota:* Esto requiere cuidado, ya que modificar la firma después de generarla invalidaría el hash del SignedInfo si el XAdES fuera parte de lo firmado.
 - **Corrección:** La forma correcta es construir el SignedProperties, calcular su hash, agregarlo como referencia al SigningContext **antes** de llamar a SignEnveloped. De esta forma, goxmldsig incluirá el hash de las propiedades en el SignedInfo y generará una firma válida que cubra tanto la factura como las propiedades.

7. Desafíos y Consideraciones para 2026

7.1 Migración de SHA-1 a SHA-256

Aunque la ficha técnica actual menciona SHA-1⁴, este algoritmo es considerado inseguro globalmente. Es previsible que para 2026 el SRI exija SHA-256.

- **Impacto en Go:** La librería goxmldsig soporta SHA-256 nativamente. Solo requiere cambiar ctx.Hash = crypto.SHA256 y actualizar las URLs de los algoritmos (<http://www.w3.org/2001/04/xmldsig-more#rsa-sha256>).
- **Estrategia:** Diseñar la aplicación con configuración dinámica del algoritmo de hash, permitiendo un "switch" rápido sin re-desplegar código si la normativa cambia abruptamente.

7.2 Validación de Espacios de Nombres (Namespace Propagation)

Un error frecuente en el ambiente de pruebas del SRI es la "pérdida" de namespaces cuando el XML se genera con librerías que no manejan bien la herencia.

- **Insight de Segundo Orden:** Al usar MakeC14N10RecCanonicalizer(), se mitiga este riesgo en la firma, pero se debe asegurar que la generación inicial de la factura (antes de firmar) incluya correctamente los atributos xmlns. Si el XML base está mal formado (ej. prefijos no declarados), el canonicalizador fallará o producirá un XML inválido. Se recomienda validar el XML contra el XSD del SRI **antes** de intentar firmarlo.

7.3 Rendimiento y Concurrencia

Para sistemas de facturación masiva (miles de facturas por minuto), la canonicalización es costosa en CPU.

- **Optimización:** goxmldsig es eficiente, pero el parsing de XML es pesado. Reutilizar instancias de etree.Document o implementar pools de trabajadores para el proceso de firma puede mejorar el throughput.
 - **Evitar CGO:** Mantenerse en librerías Go puro (goxmldsig lo es) evita el overhead de llamadas al sistema y simplifica la gestión de memoria en entornos concurrentes de alta carga.
-

8. Conclusiones y Recomendación Final

El análisis exhaustivo de las alternativas disponibles en el ecosistema Go confirma que **ucarion/c14n no es una opción viable** para la firma electrónica del SRI debido a su incompatibilidad fundamental con la Canonicalización Inclusiva (REC-xml-c14n-20010315).

Para garantizar el éxito del proyecto hacia 2026, se recomienda la siguiente estrategia de arquitectura:

1. **Motor Criptográfico Principal:** Adoptar **russellhaering/goxmldsig** como la librería central. Es la única que ofrece soporte nativo, mantenido y correcto para la Canonicalización Inclusiva requerida, a través del constructor MakeC14N10RecCanonicalizer().
2. **Capa de Lógica de Negocio (XAdES):** Utilizar el código del repositorio **digitalautonomy/goxades_sri** como referencia de implementación o "blueprint". No importarlo ciegamente, sino extraer su lógica de construcción de estructuras XAdES (SignedProperties, manejo de IDs aleatorios) e integrarla sobre una versión actualizada de goxmldsig.
3. **Gestión de Certificados:** Implementar una carga robusta de archivos .p12 utilizando software.sslmate.com/src/go-pkcs12 para asegurar compatibilidad con todos los proveedores de certificación ecuatorianos.

Esta combinación ofrece el equilibrio óptimo entre cumplimiento normativo estricto (Canonicalización Inclusiva + XAdES-BES), seguridad criptográfica moderna (soporte SHA-256 latente) y mantenibilidad del software a largo plazo.

Referencias de Investigación Integradas

- ¹ Requisitos del SRI y Ficha Técnica.
- ¹¹ Limitaciones de ucarion/c14n.

- ¹² Capacidades de russellhaering/goxmldsig y MakeC14N10RecCanonicalizer.
- ¹⁷ Implementación específica SRI en goxades_sri.
- ⁷ Teoría de Canonicalización XML W3C.

Obras citadas

1. La factura electrónica en Ecuador - EDICOM ES, fecha de acceso: enero 13, 2026, <https://edicomgroup.es/factura-electronica/ecuador>
2. Ficha Técnica Comprobantes Electrónicos SRI | PDF | Transport Layer Security - Scribd, fecha de acceso: enero 13, 2026, <https://es.scribd.com/document/245005273/Ficha-Tecnica-Comprobantes-Electronicos-SRI>
3. Adjust the xades-bes signature to the SRI Ecuador requirement · Issue #100 - GitHub, fecha de acceso: enero 13, 2026, <https://github.com/PeculiarVentures/xadesjs/issues/100>
4. Diseño e implementación del sistema de facturación electrónica para Diario El Mercurio en APEX, con almacenamiento en Oracle - Repositorio Institucional de la Universidad Politécnica Salesiana, fecha de acceso: enero 13, 2026, <https://dspace.ups.edu.ec/bitstream/123456789/8948/1/UPS-CT005213.pdf>
5. Realizar una factura mayor a 50 dolares como consumidor final - Q&A - PCGerente, fecha de acceso: enero 13, 2026, <https://discourse.pcgerente.com/t/realizar-una-factura-mayor-a-50-dolares-como-consumidor-final/918>
6. Firmar xml con python usando fastApi (Firma electrónica Sri Ecuador) | by Omar Guanoluisa, fecha de acceso: enero 13, 2026, <https://medium.com/@omar.guanoluisa25/sri-ecuador-firmar-xml-api-creada-con-fastapi-7e99da90b004>
7. Firma digital XML - IBM, fecha de acceso: enero 13, 2026, <https://www.ibm.com/docs/es/was/8.5.5?topic=wsspmica-xml-digital-signature-1>
8. XML Canonicalization Requirements Document - W3C, fecha de acceso: enero 13, 2026, <https://www.w3.org/TR/NOTE-xml-canonical-req>
9. XML Signature Guidelines - OASIS Open, fecha de acceso: enero 13, 2026, <https://www.oasis-open.org/committees/security/docs/draft-sstc-xmlsig-guidelines-03.doc>
10. Canonical XML Version 2.0 - W3C, fecha de acceso: enero 13, 2026, <https://www.w3.org/TR/xml-c14n2/>
11. ucarion/c14n: A Golang implementation of XML canonicalization - GitHub, fecha de acceso: enero 13, 2026, <https://github.com/ucarion/c14n>
12. dsig package - github.com/russellhaering/goxmldsig - Go Packages, fecha de acceso: enero 13, 2026, <https://pkg.go.dev/github.com/russellhaering/goxmldsig>
13. c14n package - github.com/unix-world/smartgoext/xml-utils/c14n - Go Packages, fecha de acceso: enero 13, 2026, <https://pkg.go.dev/github.com/unix-world/smartgoext/xml-utils/c14n>
14. goxmldsigcanonicalize.go at main - GitHub, fecha de acceso: enero 13, 2026, <https://github.com/russellhaering/goxmldsig/blob/main/canonicalize.go>

15. moov-io/signedxml: pure go library for processing signed XML documents - GitHub, fecha de acceso: enero 13, 2026, <https://github.com/moov-io/signedxml>
16. russellhaering/goxmldsig: Pure Go implementation of XML Digital Signatures - GitHub, fecha de acceso: enero 13, 2026, <https://github.com/russellhaering/goxmldsig>
17. goxades_sri/goxades.go at main · digitalautonomy/goxades_sri ..., fecha de acceso: enero 13, 2026, https://github.com/digitalautonomy/goxades_sri/blob/main/goxades.go
18. digitalautonomy/goxades_sri: Implementation of XAdES signature in golang, with some updates for SRI compatibility - GitHub, fecha de acceso: enero 13, 2026, https://github.com/digitalautonomy/goxades_sri
19. Issues · moov-io/signedxml - GitHub, fecha de acceso: enero 13, 2026, <https://github.com/moov-io/signedxml/issues>
20. godss command - github.com/alapierre/godss - Go Packages, fecha de acceso: enero 13, 2026, <https://pkg.go.dev/github.com/alapierre/godss>