SignXML Documentation

Release 0.0.1

Andrey Kislyuk

May 19, 2019

Contents

SignXML is an implementation of the W3C XML Signature standard in Python. This standard (also known as XMLDSig and RFC 3275) is used to provide payload security in SAML 2.0 and WS-Security, among other uses. Two versions of the standard exist (Version 1.1 and Version 2.0). *SignXML* implements all of the required components of the standard, and most recommended ones. Its features are:

- Use of defusedxml.lxml to defend against common XML-based attacks when verifying signatures
- Extensions to allow signing with and verifying X.509 certificate chains, including hostname/CN validation
- Support for exclusive XML canonicalization with inclusive prefixes (InclusiveNamespaces PrefixList, required to verify signatures generated by some SAML implementations)
- Modern Python compatibility (2.7-3.6+ and PyPy)
- Well-supported, portable, reliable dependencies: lxml, defusedxml, cryptography, eight, pyOpenSSL
- Comprehensive testing (including the XMLDSig interoperability suite) and continuous integration
- Simple interface with useful defaults
- Compactness, readability, and extensibility

CHAPTER 1

Installation

```
pip install signxml
```

Note: SignXML depends on lxml and cryptography, which in turn depend on OpenSSL, LibXML, and Python tools to interface with them. You can install those as follows:

OS	Python	Command
Ubuntu	Python 2	`apt-get install python-dev python-cffi libxml2-dev libxslt1-dev libssl-dev libffi-dev python-lxml python-cryptography python-openssl python-certifi python-defusedxml build-essential`
Ubuntu	Python 3	`apt-get install python3-dev python3-cffi libxml2-dev libxslt1-dev libssl-dev libffi-dev python3-lxml python3-cryptography python3-openssl python3-certifi python3-defusedxml build-essential`
Red Hat	Python 2	`yum install python-devel python-cffi libxml2-devel libxslt1-devel openssl-devel`
Red Hat	Python 3	`yum install python3-devel python3-cffi libxml2-devel libxslt1-devel openssl-devel`
Mac OS	Homebrew Python	`xcode-select --install; brew install openssl; export LDFLAGS="-L$(brew --prefix openssl)/lib" CFLAGS="-I$(brew --prefix openssl)/include"`

CHAPTER 2

Synopsis

SignXML uses the ElementTree API (also supported by lxml) to work with XML data.

```
from signxml import XMLSigner, XMLVerifier

cert = open("example.pem").read()
key = open("example.key").read()
root = ElementTree.fromstring(data_to_sign)
signed_root = XMLSigner().sign(root, key=key, cert=cert)
verified_data = XMLVerifier().verify(signed_root).signed_xml
```

2.1 Verifying SAML assertions

Assuming `metadata.xml` contains SAML metadata for the assertion source:

```
from lxml import etree
from base64 import b64decode
from signxml import XMLVerifier

with open("metadata.xml", "rb") as fh:
    cert = etree.parse(fh).find("//ds:X509Certificate").text

assertion_data = XMLVerifier().verify(b64decode(assertion_body), x509_cert=cert).
↪signed_xml
```

Signing SAML assertions

The SAML assertion schema specifies a location for the enveloped XML signature (between `<Issuer>` and `<Subject>`). To sign a SAML assertion in a schema-compliant way, insert a signature placeholder tag at that location before calling XMLSigner: `<ds:Signature Id="placeholder"></ds:Signature>`.

See what is signed

It is important to understand and follow the best practice rule of "See what is signed" when verifying XML signatures. The gist of this rule is: if your application neglects to verify that the information it trusts is what was actually signed, the attacker can supply a valid signature but point you to malicious data that wasn't signed by that signature. Failure to follow this rule can lead to vulnerability against attacks like SAML signature wrapping.

In SignXML, you can ensure that the information signed is what you expect to be signed by only trusting the data returned by the `verify()` method. The `signed_xml` attribute of the return value is the XML node or string that was signed.

Recommended reading: W3C XML Signature Best Practices for Applications, OWASP: On Breaking SAML: Be Whoever You Want to Be

Establish trust

If you do not supply any keyword arguments to `verify()`, the default behavior is to trust **any** valid XML signature generated using a valid X.509 certificate trusted by your system's CA store. This means anyone can get an SSL certificate and generate a signature that you will trust. To establish trust in the signer, use the `x509_cert` argument to specify a certificate that was pre-shared out-of-band (e.g. via SAML metadata, as shown in *Verifying SAML assertions*), or `cert_subject_name` to specify a subject name that must be in the signing X.509 certificate given by the signature (verified as if it were a domain name), or `ca_pem_file`/`ca_path` to give a custom CA.

2.2 XML signature methods: enveloped, detached, enveloping

The XML Signature specification defines three ways to compose a signature with the data being signed: enveloped, detached, and enveloping signature. Enveloped is the default method. To specify the type of signature that you want to generate, pass the `method` argument to `sign()`:

```
signed_root = XMLSigner(method=signxml.methods.detached).sign(root, key=key,␣
↪cert=cert)
verified_data = XMLVerifier().verify(signed_root).signed_xml
```

For detached signatures, the code above will use the `Id` or `ID` attribute of `root` to generate a relative URI (`<Reference URI="#value"`). You can also override the value of `URI` by passing a `reference_uri` argument to `sign()`. To verify a detached signature that refers to an external entity, pass a callable resolver in `XMLVerifier().verify(data, uri_resolver=...)`.

See the API documentation for more.

CHAPTER 3

Authors

- Andrey Kislyuk

CHAPTER 4

Links

- Project home page (GitHub)
- Documentation (Read the Docs)
- Package distribution (PyPI)
- Change log
- List of W3C XML Signature standards and drafts
- W3C Recommendation: XML Signature Syntax and Processing Version 1.1
- W3C Working Group Note: XML Signature Syntax and Processing Version 2.0
- W3C Working Group Note: XML Security 2.0 Requirements and Design Considerations
- W3C Working Group Note: XML Signature Best Practices
- XML-Signature Interoperability
- W3C Working Group Note: Test Cases for C14N 1.1 and XMLDSig Interoperability
- XMLSec: Related links
- OWASP SAML Security Cheat Sheet
- Okta Developer Docs: SAML

4.1 Bugs

Please report bugs, issues, feature requests, etc. on GitHub.

CHAPTER 5

License

Licensed under the terms of the Apache License, Version 2.0.

CHAPTER 6

API documentation

class signxml.**VerifyResult**

The results of a verification return the signed data, the signed xml and the signature xml

Parameters

- **signed_data** – The binary data as it was signed (literally)
- **signed_xml** (*ElementTree*) – The signed data parsed as XML (or None if parsing failed)
- **signature_xml** – The signature element parsed as XML

This class is a namedtuple representing structured data returned by `signxml.XMLVerifier.verify()`. As with any namedtuple, elements of the return value can be accessed as attributes. For example:

```
verified_data = signxml.XMLVerifier().verify(input_data).signed_xml
```

class signxml.**XMLSigner**(*method=<Methods.enveloped: 1>*, *signature_algorithm=u'rsa-sha256'*, *digest_algorithm=u'sha256'*, *c14n_algorithm=u'http://www.w3.org/2006/12/xml-c14n11'*)

Create a new XML Signature Signer object, which can be used to hold configuration information and sign multiple pieces of data.

Parameters

- **method** (*methods*) – `signxml.methods.enveloped`, `signxml.methods.enveloping`, or `signxml.methods.detached`. See the list of signature types under XML Signature Syntax and Processing Version 2.0, Definitions.
- **signature_algorithm** (*string*) – Algorithm that will be used to generate the signature, composed of the signature algorithm and the digest algorithm, separated by a hyphen. All algorithm IDs listed under the Algorithm Identifiers and Implementation Requirements section of the XML Signature 1.1 standard are supported.
- **digest_algorithm** (*string*) – Algorithm that will be used to hash the data during signature generation. All algorithm IDs listed under the Algorithm Identifiers and Implementation Requirements section of the XML Signature 1.1 standard are supported.

sign(*data*, *key=None*, *passphrase=None*, *cert=None*, *reference_uri=None*, *key_name=None*, *key_info=None*, *id_attribute=None*)

Sign the data and return the root element of the resulting XML tree.

Parameters

- **data** (*String, file-like object, or XML ElementTree Element API compatible object*) – Data to sign
- **key** (string, `cryptography.hazmat.primitives.interfaces.RSAPublicKey`, `cryptography.hazmat.primitives.interfaces.DSAPublicKey`, or `cryptography.hazmat.primitives.interfaces.EllipticCurvePublicKey` object) – Key to be used for signing. When signing with a certificate or RSA/DSA/ECDSA key, this can be a string containing a PEM-formatted key, or a `cryptography.hazmat.primitives.interfaces.RSAPublicKey`, `cryptography.hazmat.primitives.interfaces.DSAPublicKey`, or `cryptography.hazmat.primitives.interfaces.EllipticCurvePublicKey` object. When signing with a HMAC, this should be a string containing the shared secret.
- **passphrase** (*string*) – Passphrase to use to decrypt the key, if any.
- **cert** (*string, array of strings, or array of OpenSSL.crypto.X509 objects*) – X.509 certificate to use for signing. This should be a string containing a PEM-formatted certificate, or an array of strings or OpenSSL.crypto.X509 objects containing the certificate and a chain of intermediate certificates.
- **reference_uri** (*string or list*) – Custom reference URI or list of reference URIs to incorporate into the signature. When `method` is set to `detached` or `enveloped`, reference URIs are set to this value and only the referenced elements are signed.
- **key_name** (*string*) – Add a KeyName element in the KeyInfo element that may be used by the signer to communicate a key identifier to the recipient. Typically, KeyName contains an identifier related to the key pair used to sign the message.
- **key_info** (`lxml.etree.Element`) – A custom KeyInfo element to insert in the signature. Use this to supply `<wsse:SecurityTokenReference>` or other custom key references.
- **id_attribute** (*string*) – Name of the attribute whose value `URI` refers to. By default, SignXML will search for "Id", then "ID".

Returns A `lxml.etree.Element` object representing the root of the XML tree containing the signature and the payload data.

To specify the location of an enveloped signature within **data**, insert a `<ds:Signature Id="placeholder"></ds:Signature>` element in **data** (where "ds" is the "http://www.w3.org/2000/09/xmldsig#" namespace). This element will be replaced by the generated signature, and excised when generating the digest.

class signxml.**XMLVerifier**

Create a new XML Signature Verifier object, which can be used to hold configuration information and verify multiple pieces of data.

verify(*data*, *require_x509=True*, *x509_cert=None*, *cert_subject_name=None*, *ca_pem_file=None*, *ca_path=None*, *hmac_key=None*, *validate_schema=True*, *parser=None*, *uri_resolver=None*, *id_attribute=None*, *expect_references=1*)

Verify the XML signature supplied in the data and return the XML node signed by the signature, or raise an exception if the signature is not valid. By default, this requires the signature to be generated using a

valid X.509 certificate. To enable other means of signature validation, set the **require_x509** argument to *False*.

See what is signed

It is important to understand and follow the best practice rule of "See what is signed" when verifying XML signatures. The gist of this rule is: if your application neglects to verify that the information it trusts is what was actually signed, the attacker can supply a valid signature but point you to malicious data that wasn't signed by that signature.

In SignXML, you can ensure that the information signed is what you expect to be signed by only trusting the data returned by the `verify()` method. The return value is the XML node or string that was signed. Also, depending on the signature settings used, comments in the XML data may not be subject to signing, so may need to be untrusted.

Recommended reading: http://www.w3.org/TR/xmldsig-bestpractices/#practices-applications

---

Establish trust

If you do not supply any keyword arguments to `verify()`, the default behavior is to trust **any** valid XML signature generated using a valid X.509 certificate trusted by your system's CA store. This means anyone can get an SSL certificate and generate a signature that you will trust. To establish trust in the signer, use the `x509_cert` argument to specify a certificate that was pre-shared out-of-band (e.g. via SAML metadata, as shown in *Verifying SAML assertions*), or `cert_subject_name` to specify a subject name that must be in the signing X.509 certificate given by the signature (verified as if it were a domain name), or `ca_pem_file`/`ca_path` to give a custom CA.

Parameters

- **data** (*String, file-like object, or XML ElementTree Element API compatible object*) – Signature data to verify
- **require_x509** (*boolean*) – If `True`, a valid X.509 certificate-based signature with an established chain of trust is required to pass validation. If `False`, other types of valid signatures (e.g. HMAC or RSA public key) are accepted.
- **x509_cert** (*string or OpenSSL.crypto.X509*) – A trusted external X.509 certificate, given as a PEM-formatted string or OpenSSL.crypto.X509 object, to use for verification. Overrides any X.509 certificate information supplied by the signature. If left set to `None`, requires that the signature supply a valid X.509 certificate chain that validates against the known certificate authorities. Implies **require_x509=True**.
- **ca_pem_file** (*string or bytes*) – Filename of a PEM file containing certificate authority information to use when verifying certificate-based signatures.
- **ca_path** (*string*) – Path to a directory containing PEM-formatted certificate authority files to use when verifying certificate-based signatures. If neither **ca_pem_file** nor **ca_path** is given, the Mozilla CA bundle provided by `certifi` will be loaded.
- **cert_subject_name** (*string*) – Subject Common Name to check the signing X.509 certificate against. Implies **require_x509=True**.
- **hmac_key** (*string*) – If using HMAC, a string containing the shared secret.
- **validate_schema** (*boolean*) – Whether to validate **data** against the XML Signature schema.

- **parser** (`lxml.etree.XMLParser` compatible parser) – Custom XML parser instance to use when parsing **data**.
- **uri_resolver** (*callable*) – Function to use to resolve reference URIs that don't start with "#".
- **id_attribute** (*string*) – Name of the attribute whose value `URI` refers to. By default, SignXML will search for "Id", then "ID".
- **expect_references** (*int or boolean*) – Number of references to expect in the signature. If this is not 1, an array of VerifyResults is returned. If set to a non-integer, any number of references is accepted (otherwise a mismatch raises an error).

Raises `cryptography.exceptions.InvalidSignature`

Returns VerifyResult object with the signed data, signed xml and signature xml

Return type *VerifyResult*

`signxml.`**methods**
alias of `signxml.Methods`

CHAPTER 7

Release Notes

CHAPTER 8

Changes for v2.6.0 (2019-01-10)

- Update dependencies to baseline on Ubuntu 18.04
- Clarify documentation of Ubuntu installation dependencies
- List ipaddress as a dependency
- Strip PEM header from OpenSSL.crypto.X509 cert
- Doc updates: dependency versions, standard links
- Fix cryptography deprecation warnings. Closes #108
- Allow URI attribute of Reference to be absent (#102)

CHAPTER 9

Changes for v2.5.2 (2017-12-07)

- Fix release

CHAPTER 10

Changes for v2.5.1 (2017-12-07)

Fix release

CHAPTER 11

Changes for v2.5.0 (2017-12-07)

- Relax dependency version constraints.
- Drop Python 3.3 support.
- Support for PEM files with CR+LF line endings (#93).

CHAPTER 12

Changes for v2.4.0 (2017-07-10)

- Import asn1crypto on demand
- Allow newer versions of cryptography library (#89)

CHAPTER 13

Changes for v2.3.0 (2017-04-24)

- Add explicit dependency on asn1crypto to setup.py (#87)
- Remove use of Exception.message for py3 compatibility. Closes #36 (#86)
- Use asn1crypto instead of pyasn1 to match cryptography lib (#85)
- Pin to major version of lxml instead of minor
- Allow newer versions of several requirements (#84)
- Allow newer version of eight library (#83)

CHAPTER 14

Changes for v2.2.4 (2017-03-19)

- Documentation and test fixes

CHAPTER 15

Changes for v2.2.3 (2016-12-20)

- Release automation: parse repo name correctly

CHAPTER 16

Changes for v2.2.2 (2016-12-20)

- Expand supported cryptography version range. Fixes #74
- Documentation and release automation improvements

CHAPTER 17

Changes for v2.2.1 (2016-09-26)

- Fix handling of reference URIs in detached signing
- Test infra fixes

CHAPTER 18

Changes for v2.2.0 (2016-09-25)

- Support custom key info when signing
- Initial elements of ws-security support
- Support signing and verifying multiple references

CHAPTER 19

Changes for v2.1.4 (2016-09-18)

- Only sign the referenced element when passed reference_uri (thanks to @soby).
- Add CN validation - instead of a full X.509 certificate, it is now possible to pass a common name that will be matched against the CN of a cert trusted by the CA store.
- Order-agnostic cert chain validation and friendlier ingestion of cert chains.
- Minor/internal changes; packaging fix for 2.1.0

CHAPTER 20

Changes for v2.1.0 (2016-09-18)

- Pre-release; see notes for v2.1.4

20.1 Version 2.0.0 (2016-08-05)

- Major API change: signxml.xmldsig(data).sign() -> signxml.XMLSigner().sign(data)
- Major API change: signxml.xmldsig(data).verify() -> signxml.XMLVerifier().verify(data)
- Signer and verifier objects now carry no data-specific state; instead carry system configuration state that is expected to be reused
- Signer and verifier objects should now be safe to reuse in reentrant environments
- Internal architecture changes to improve modularity and eliminate data-specific latent state and side effects

20.2 Version 1.0.2 (2016-08-01)

- Update xmlenc namespaces for downstream encryptxml support

20.3 Version 1.0.1 (2016-07-14)

- Packaging fix: remove stray .pyc file

20.4 Version 1.0.0 (2016-04-08)

- Major API change: Return signature information in verify() return value (#41, #50). Thanks to @klondi.

- Major API change: Excise signature node from verify() return value to avoid possibly returning untrusted data (#47). Thanks to @klondi.

20.5 Version 0.6.0 (2016-03-24)

- Remove signature nodes appropriately (#46). Thanks to @klondi.
- Expand Travis CI test to include flake8 linter.

20.6 Version 0.5.0 (2016-03-02)

- Add support for using a KeyName element within the KeyInfo block (#38). Thanks to @Pelleplutt.
- Update cryptography dependency
- Expand Travis CI test matrix to include OS X

20.7 Version 0.4.6 (2015-11-28)

- Python 3.5 compatibility fix: move enum34 into conditional dependencies (#37). Thanks to @agronholm.

20.8 Version 0.4.5 (2015-11-08)

- Support enveloped signatures nested at arbitrary levels beneath root element (#32, #33). Thanks to @jmindek.
- Update certifi, cryptography dependencies

20.9 Version 0.4.4 (2015-08-07)

- Handle xml.etree.ElementTree nodes as input (previously these would cause a crash, despite the documentation suggesting otherwise). Closes #19, thanks to @nickcash.

20.10 Version 0.4.3 (2015-07-26)

- Do not open schema file in text mode when parsing XML (closes #18, thanks to @nick210)
- Update cryptography dependency

20.11 Version 0.4.2 (2015-04-24)

- Add support for parameterizable signature namespace (PR #12, thanks to @ldnunes)
- Update cryptography dependency

20.12 Version 0.4.1 (2015-04-21)

- Add support for detached signatures (closes #3)
- Update pyOpenSSL dependency; use X509StoreContext.verify_certificate()

20.13 Version 0.4.0 (2015-03-08)

- Use pyasn1 for DER encoding and decoding, eliminating some DSA signature verification failures

20.14 Version 0.3.9 (2015-02-04)

- Do not distribute tests in source archive

20.15 Version 0.3.7 (2015-02-04)

- Configurable id attribute name for verifying non-standard internal object references, e.g. ADFS (closes #6)

20.16 Version 0.3.6 (2015-01-10)

- Python 3 compatibility fixes
- Fix test matrix (Python version configuration) in Travis

20.17 Version 0.3.5 (2014-12-22)

- Refactor application of enveloped signature transforms
- Support base64 transform
- Support application of different canonicalization algorithms to signature and payload (closes #1)

20.18 Version 0.3.4 (2014-12-14)

- Add support for exclusive canonicalization with InclusiveNamespaces PrefixList attribute

20.19 Version 0.3.3 (2014-12-13)

- Overhaul support of canonicalization algorithms

20.20 Version 0.3.2 (2014-12-11)

- Fix bug in enveloped signature canonicalization of namespace prefixes

20.21 Version 0.3.1 (2014-10-17)

- Fix bug in enveloped signature excision

20.22 Version 0.3.0 (2014-10-16)

- Allow location of enveloped signature to be specified

20.23 Version 0.2.9 (2014-10-14)

- Use exclusive c14n when signing

20.24 Version 0.2.8 (2014-10-13)

- Namespace all tags when generating signature

20.25 Version 0.2.7 (2014-10-13)

- Switch default signing method to enveloped signature

20.26 Version 0.2.6 (2014-10-13)

- Fix typo in ns prefixing code

20.27 Version 0.2.5 (2014-10-13)

- Fix handling of DER sequences in DSA key serialization
- Parameterize excision with ns prefix

20.28 Version 0.2.4 (2014-10-12)

- Fix excision with ns prefix

20.29 Version 0.2.3 (2014-10-12)

- Fixes to c14n of enveloped signatures
- Expand tests to use the XML Signature interoperability test suite

20.30 Version 0.2.2 (2014-10-04)

- Load bare X509 certificates from SAML metadata correctly

20.31 Version 0.2.1 (2014-10-04)

- Always use X509 information even if key value is present
- Internal refactor to modularize key value handling logic

20.32 Version 0.2.0 (2014-10-02)

- Use defusedxml when verifying signatures.
- Eliminate dependency on PyCrypto.
- Introduce support for ECDSA asymmetric key encryption.
- Introduce ability to validate xmldsig11 schema.
- Expand test suite coverage.

20.33 Version 0.1.9 (2014-09-27)

- Allow use of external X509 certificates for validation; add an example of supplying a cert from SAML metadata.

20.34 Version 0.1.8 (2014-09-25)

- Packaging fix.

20.35 Version 0.1.7 (2014-09-25)

- Packaging fix.

20.36 Version 0.1.6 (2014-09-25)

- Accept etree elements in verify.

20.37 Version 0.1.5 (2014-09-25)

- Packaging fix.

20.38 Version 0.1.4 (2014-09-25)

- Begin work toward conformance with version 1.1 of the spec.

20.39 Version 0.1.3 (2014-09-23)

- Require x509 for verification by default.

20.40 Version 0.1.2 (2014-09-22)

- Documentation fixes.

20.41 Version 0.1.1 (2014-09-22)

- Documentation fixes.

20.42 Version 0.1.0 (2014-09-22)

- Initial release.

CHAPTER 21

Table of Contents

Python Module Index

S

Index