Internet Engineering Task Force
Internet-Draft
Intended status: Informational
Expires: April 30, 2021

T. Hardjono
MIT
M. Hargreaves
Quant Network
N. Smith
Intel

October 27, 2020

An Interoperability Architecture for Blockchain Gateways
draft-hardjono-blockchain-interop-arch-01

Abstract

With the increasing interest in the potential use of blockchain systems for virtual assets, there is a need for these assets to have mobility across blockchain systems. An interoperability architecture for blockchain systems is needed in order to permit the secure flow of virtual assets between blockchain systems, satisfying the properties of transfer atomicity, consistency and durability. The architecture must recognize that there are different blockchain systems, and that the interior constructs in these blockchains maybe incompatible with one another. Gateway nodes perform the transfer of virtual assets between blockchain systems while masking the complexity of the interior constructs of the blockchain that they represent.

Table of Contents

1. Introduction

Currently there is little technical interoperability between blockchain systems. This results in the difficulty in transferring or migrating virtual assets associated with a public-key (address) in one blockchain system to another blockchain system.

The existing solutions involve a third party that mediates the transfer. This mediating third party is typically an asset-exchange entity (i.e. crypto-exchange) operating in a centralized hub-spoke fashion. This reliance on a third party results not only delays in transfers, but also in the need for key-holders to open accounts at the third party entity. It diminishes the autonomy of a blockchain system and limits its scalability.

This document describes an architecture for blockchain gateways that perform the unidirectional transfer of virtual assets between two autonomous blockchain systems which employ the gateways.

The purpose of this architecture document is to provide technical framework within which to discuss the various aspects of a transfer between two gateways, including security aspects and transfer commitment aspects.

2. Terminology

There following are some terminology used in the current document:

- Blockchain Domain: The collection of resources and entities participating within a blockchain system.

- Interior Resources: The various interior protocols, data structures and cryptographic constructs that are a core part of a blockchain system. Examples of interior resources include the ledger (blocks of confirmed transaction data), public keys on the ledger, consensus protocol, incentive mechanisms, transaction propagation networks, etc.

- Exterior Resources: The various resources that are outside a blockchain system, and are not part of the operations of a blockchain. Examples include data located at third parties such as asset registries, ledgers of other blockchains, PKI infrastructures, etc.

- Blockchain nodes: The nodes of the blockchain system which form the peer-to-peer network, which collectively maintain the shared ledger in the blockchain by following a consensus algorithm.

- Gateway nodes: The nodes of the blockchain system that are functionally capable of acting as a gateway in an asset transfer. Gateway nodes implement the gateway-to-gateway asset transfer protocol. Being a node on the blockchain system, a gateway has read/write access to the interior resources (e.g. ledger) of the blockchain. It participates in the consensus mechanism deployed for the blockchain system. Depending on the blockchain system implementation, some or all of the nodes may be gateway-capable.

- Blockchain address: This is the public-key of an entity as known within a blockchain system, employed to transact on the blockchain network and recorded on the ledger of the blockchain. Also referred to as the transaction signing public-key pair.

- Entity public-key pair: This the private-public key pairs of an entity used for interactions outside the blockchain system (e.g. TL1.3). The term is used to distinguish this public-key from the blockchain address.

- Asset transfer protocol: The gateway-to-gateway technical protocol used by two gateway nodes to perform a unidirectional transfer of a virtual asset.

- Asset profile: The prospectus of a regulated asset that includes information and resources describing the virtual asset. This includes the asset name/code, issuing authority, denomination, jurisdiction, and the URLs and mechanisms to validate the information. The asset profile is independent from the specific instantiation of the asset (on a blockchain or otherwise) and independent from its instance-ownership information.

- Virtual Asset: A virtual asset is a digital representation of value that can be digitally traded, or transferred as defined by the FATF [FATF].

- Virtual Asset Service Provider (VASP): Legal entity handling virtual assets as defined by the FATF [FATF].

- Originator: Person or organization seeking the transfer of virtual asset to a beneficiary

- Beneficiary: Person or organization receiving the transferred virtual asset from an originator.

- Travel Rule information: Data regarding the VASPs, originators and beneficiaries involved in an asset transfer, as defined by the FATF [FATF] and as required by the jurisdiction of operations of the VASPs.

- Gateway device identity: The identity of the device implementing the gateway functions. The term is used in the sense of IDevID (IEEE 802.1AR) or EK/AIK (in TPM1.2 and TPM2.0) [IDevID].

- Gateway owner: The VASP who legally owns and operates a gateway node within a blockchain system.

- Passive transaction: A transaction aimed at recording some state metadata information on the ledger that does not affect assets recorded on the ledger. A passive transaction can be self-addressed (or has null as destination address) and can be used to signal implicitly to other nodes regarding an state-change of the metadata pertaining to an entity or an asset on the ledger.

Further terminology definitions can be found in [NIST] and [ISO]. The term 'blockchain' and 'distributed ledger technology' (DLT) are used interchangeably in this document.

3. Assumptions and Principles

The following assumptions and principles underlie the design of the current interoperability architecture, and correspond to the design principles of the Internet architecture.

3.1. Design Principles

- Opaque blockchain resources: The interior resources of each blockchain system is assumed to be opaque to (hidden from) external entities. Any resources to be made accessible to an external entity must be made explicitly accessible by a gateway node with proper authorization.

- Externalization of value: The gateway protocol is agnostic (oblivious) to the economic or monetary value of the virtual asset being transferred.

The opaque resources principle permits the interoperability architecture to be applied in cases where one (or both) blockchain systems are permissioned (private). It is the analog of the autonomous systems principle in IP networking [Clar88], where interior routes in local subnets are not visible to other external autonomous systems.

The value-externalization principle permits asset transfer protocols to be designed for efficiency, speed and reliability - independent of the changes in the perceived economic value of the virtual asset. It is the analog of the end-to-end principle in the Internet architecture [SRC84], where contextual information (economic value)

is placed at the endpoints of the transaction. In the case of a transfer of virtual assets, the originator and beneficiary at the respective blockchain systems are assumed to have a common agreement regarding the economic value of the asset.

3.2. Operational Assumptions

The following conditions are assumed to have occurred, leading to the invocation of the asset transfer protocol between two gateway nodes:

- Application layer transfer request: The transfer request from an originator in the origin blockchain is assumed to have occurred prior to the execution of the asset transfer protocol.
- Identification of originator and beneficiary: The originator and beneficiary are assumed to have been identified and that consent has been obtained from both parties regarding the asset transfer.
- Identification of origin and destination blockchain: The origin and destination blockchain systems is assumed to have been identified.
- Selection of gateway nodes: The two gateway nodes at the origin and destination blockchain systems respectively is assumed to have been selected.
- Identification of gateway-node owners (VASP): The VASP operating the gateway nodes are assumed to have been identified and their legal status verified.

4. Architecture

4.1. Goal of Architecture

The goal of the interoperability architecture is to permit two (2) gateway nodes belonging to distinct blockchain systems to conduct a virtual asset transfer between them, in a secure and non-repudiable manner while ensuring the asset does not exist simultaneously on both blockchains (double-spend problem).

The virtual asset as understood by the two gateway nodes is a digital representation of value, expressed in an standard digital format in a way meaningful to the gateway nodes syntactically and semantically.

The syntactic representation of the virtual asset between the two gateways need not bear any resemblance to the syntactic asset representation within their respective blockchain systems.

The architecture recognizes that there different blockchain systems currently in operation and evolving, and that in many cases the interior technical constructs in these blockchains maybe incompatible with one another.

The architecture therefore assumes that certain types of nodes (gateway nodes) will be equipped with an asset transfer protocol and other relevant resources that permits greater interoperability across these incompatible blockchain systems.

The resources within a blockchain system (e.g. ledgers, public-keys, consensus protocols, etc.) are assumed to be opaque to external entities in order to permit a resilient and scalable protocol design that is not dependent on the interior constructs of particular blockchain systems. This ensures that the virtual asset transfer protocol between gateways is not conditioned or dependent on these local technical constructs. The role of a gateway therefore is also to mask (hide) the complexity of the interior constructs of the blockchain system that it represents. Overall this approach ensures that a given blockchain system operates as a true autonomous system.

The current architecture focuses on unidirectional asset transfers, although the building blocks in this architecture can be used to support protocols for bidirectional transfers (conditional two unidirectional transfers).

For simplicity the current architecture employs two (2) gateway nodes in the respective blockchains, but collective multi-node transfers (i.e. multiple nodes at each side) [HS2019] may be developed based on the building blocks and constructs identified in the current architecture.

4.2. Overview of Asset Transfer

An asset transfer between two blockchain systems is carried out by two (2) gateway nodes in a direct interaction (unmediated), where the gateway represents the two respective blockchain systems.

A successful transfer results in the asset being extinguished (deleted) or marked on the origin ledger by the origin-gateway, and for the asset to be introduced by the destination-gateway into the destination ledger.

The mechanism to extinguish or introduce an asset from/into a ledger is dependent on the specific blockchain system. The task of the respective gateway is to implement the relevant mechanism to modify the ledger of their corresponding blockchain system in such a way that together the two blockchains maintain consistency from the asset

perspective, while observing the design principles of the architecture.

An asset transfer protocol that can satisfy the properties of atomicity and consistency in the case of two private blockchain systems, should also do in the case when one or both are public blockchain systems.

4.3. Desirable Properties of Asset Transfer

The desirable features of asset transfers between two gateway nodes include, but not limited, to the following:

- Atomicity: Transfer must either commit or entirely fail (failure means no change to asset ownership).
- Consistency: Transfer (commit or fail) always leaves both blockchains in a consistent state (asset located in one blockchain only).
- Isolation: While transfer occurring, asset ownership cannot be modified (no double-spend).
- Durability: Once a transfer has been committed, must remain so regardless of gateway crashes.
- Verifiable by authorized third parties: With proper authorization to access relevant interior resources, third party entities must be able at any time to perform audit-validation of the two respective ledgers for asset transfers across the corresponding blockchain systems.
- Containment of side-effects: Any effects due to errors or security/integrity breaches in a blockchain system during an asset transfer must be contained within that blockchain.

An implementation of the asset transfer protocol should satisfy these properties, independent of whether the implementation employs stateful messaging or stateless messaging between the two gateways.

4.4. Event log-data, crash recovery and backup gateways

Implementations of gateway nodes should maintain event logs and checkpoints for the purpose of gateway crash recovery. The log-data generated by a gateway should be considered as an interior resource accessible to other authorized gateway nodes within the same blockchain system

Mechanisms used to select or elect a gateway node in a blockchain system for a given asset transfer could be extended to include the selection of a backup gateway node. The primary gateway and the backup gateway may or may not belong to the same owner (VASP).

Some blockchain systems may utilize the ledger itself as means to retain the log-data, allowing other nodes in the blockchain to have visibility and access to the gateway log-data. In these cases, the gateway node may employ its transaction-signing key pair to issue a passive transaction (e.g. self-addressed, no asset) on the ledger, incorporating details of the transfer event.

Other blockchain systems may employ off-chain storage that is accessible to all gateway nodes in the blockchain domain. In such cases, to provide event-sequencing integrity the gateway may store a hash of the log-data on the ledger of the blockchain (passive transaction) prior to writing the log-data to the off-chain storage.

The mechanism used to provide gateway crash-recovery is dependent on the blockchain system and the gateway implementation. For interoperability purposes the information contained in the log and the format of the log-data should be standardized, permitting vendors of gateway products to reduce development costs over time.

The resumption of an interrupted transfer (e.g. due to gateway crash, network failure, etc.) should take into consideration the aspects of secure channel establishment and the aspects of the transfer protocol resumption. In some cases, a new secure channel (e.g. TLS session) must be established with the backup gateway node, within which the asset transfer protocol could be continued from the last checkpoint prior to the interruption. However, in other cases both the secure channel and the transfer protocol must be started completely afresh (no resumption).

The log-data collected by a gateway node acts also as a checkpoint mechanism to assist the backup gateway node in continuing the transfer. The point at which to re-start the transfer protocol flow is dependent on the implementation of the gateway. Some owners (VASPs) of gateway nodes may choose to start afresh the transfer of the asset, and not to resume partially completed transfers.

4.5. Overview of the Phases in Asset Transfer

The interaction between two gateways in an asset transfer is summarized in Figure 1, where the origin blockchain is B1 and the destination blockchain is B2. The gateways are denoted as G1 and G2 respectively.

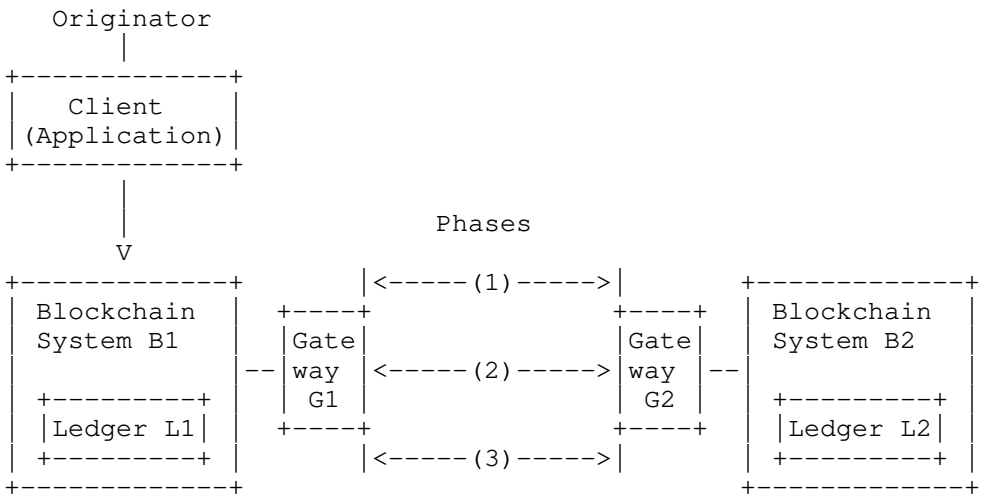

Figure 1

The three phases are summarized as follows.

- Phase 1: Pre-transfer Verification of Asset and Identities. In this phase the gateways G1 and G2 must mutually identify themselves and authenticate that both possess gateway-capabilities. Gateway G1 must communicate to G2 the asset-profile of the asset to be transferred, and that consent have been obtained from the beneficiary regarding accepting the transfer.

- Phase 2: Evidence of asset locking or escrow. In this phase, gateway G1 must provide gateway G2 with sufficient evidence that the asset on blockchain B1 is in a locked state (or escrowed) under the control of G1 on ledger L1, and safe from double-spend on the part of its current owner (the originator).

- Phase 3: Transfer commitment. In this phase gateways G1 and G2 commits to the transaction

These phases will be further discussed below.

5. Pre-transfer Verification of Asset and Identities (Phase 1)

The primary purpose of the first phase is to verify the various information relating to the asset to be transferred, the identities of the originator and beneficiary, the identity and legal status of the entities (VASPs) who own and operate the gateways, and the device-identities of the gateways.

This phase starts with the assumption that in blockchain B1 the gateway to process the asset transfer to B2 has been selected (namely gateway G1). It also assumes that the destination blockchain B2 has been identified where the beneficiary address is located, and that gateway G2 in blockchain B2 has been identified that will peer with G1 to perform the transfer.

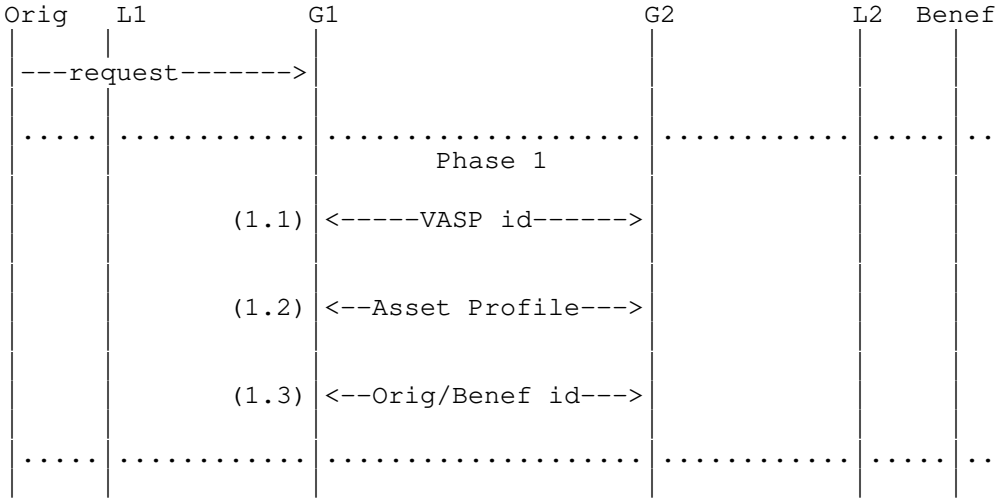

Figure 2

There are several steps that may occur in Phase 1 (see Figure 2):

- Secure channel establishment between G1 and G2: This includes the mutual verification of the gateway device identities and the exchange of the relevant parameters for secure channel establishment. In cases where device attestation [RATS] is required, the mutual attestation protocol must occur between G1 and G2 prior to proceeding to the next phase.

- Validation of the gateway ownership: There must be a means for gateway G1 and G2 to verify their respective ownerships (i.e. VASP1 owning G1 and VASP2 owning G2 respectively). Examples of ownership verification mechanism include the chaining of the gateway-device X.509 certificate up to the VASP entity certificate, directories of gateways and VASPs, and others.

- Validation of VASP status: In some jurisdictions limitations may be placed for regulated VASPs to transact only with other

similarly regulated VASPs. Examples of mechanisms used to validate a VASP legal status include VASP directories, Extended Validation (EV) X.509 certificates for VASPs, and others.

- Delivery and validation of asset profile: Gateway G1 must deliver to G2 the asset-profile for the virtual asset to be transferred. Gateway G2 must validate the authenticity of the statements (claims) found in the asset profile. The policies governing blockchain B2 with regards to permissible incoming assets must be enforced by G2.

- Exchange of Travel Rule information: In jurisdictions where the Travel Rule policies regarding originator and beneficiary information is enforced, the gateways G1 and G2 must exchange this information [FATF].

- o Negotiation of asset transfer protocol parameters: Gateway G1 and G2 must agree on the parameters to be employed within the asset transfer protocol. Examples include endpoints definitions for resources, type of commitment flows (e.g. 2PC or 3PC), lock-time durations, and others [ODAP].

6. Evidence of asset locking or escrow (Phase 2)

The asset transfer protocol can commence when both gateways G1 and G2 have completed the verifications in Phase 1.

The steps of Phase 2 is shown in Figure 3, and broadly consists of the following:

- Commencement (2.1): Gateway G1 indicates the start of the asset transfer protocol by sending a transfer-commence message to gateway G2. Among others, the message must include a cryptographic hash of the information agreed-upon in Phase 1 (e.g. asset profile, gateway identities, originator/beneficiary public keys).

- Acknowledgement (2.2): The gateway G2 must send an explicit acknowledgement of the commence message, which should include a hash of commencement message and other relevant session parameters.

- G1 lock/escrow asset (2.3): Gateway G1 proceeds to lock or escrow the asset belonging to the originator on ledger L1. The lock may take the form of passive transaction that signals to other nodes that the asset is temporarily inaccessible. This signals to other nodes in the blockchain system to ignore other transactions

pertaining to the asset until such time the lock by G1 is finalized or released.

- G2 log incoming (2.4): Gateway G2 correspondingly writes a log (passive transaction) on ledger L2 indicating an imminent arrival of the asset to L2. This may act as a notification for the beneficiary regarding the asset transfer.

- Lock Evidence (2.5): Gateway G1 sends a digitally signed evidence regarding the lock (escrow) performed by G1 on the asset on ledger L1. The signature by G1 is performed using its entity public-key pair.

- Evidence receipt (2.6): If gateway G2 accepts the evidence, G2 then responds with a digitally signed receipt message which includes a hash of the previous lock-evidence message. Otherwise, if G2 declines the evidence then G2 can ignore the transfer and let it time-out (i.e. transfer failed).

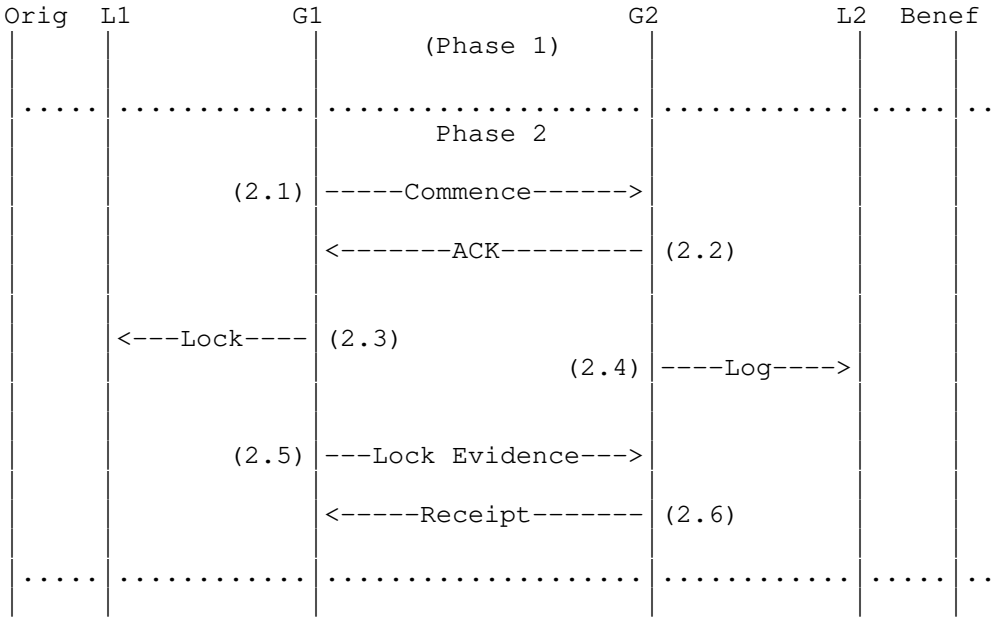

Figure 3

The precise form of the evidence in step 2.5 is dependent on the blockchain system in B1, and must be previously agreed upon between G1 and G2 in Phase 1.

The purpose of this evidence is for dispute resolution between G1 and G2 (i.e. entities who own and operate G1 and G2 respectively) in the case that double-spend is later detected.

The gateway G2 must return a digitally signed receipt to G1 of this evidence in order to cover G1 (exculpatory proof) in the case of later denial by G2.

7. Transfer Commitment (Phase 3)

In Phase 3 the gateways G1 and G2 finalizes to the asset transfer by performing a commitment protocol (e.g. 2PC or 3PC) as a process (sub-protocol) embedded within the overall asset transfer protocol.

Upon receiving the evidence-receipt message in the previous phase, G1 begins the commitment (see Figure 4):

- Commit-prepare (3.1): Gateway G1 indicates to G2 to prepare for the commitment of the transfer. This message must include a hash of the previous messages (message 2.5 and 2.6).

- Ack-prepare (3.2): Gateway G2 acknowledges the commit-prepare message.

- Lock-final (3.3): Gateway G1 issues a lock-finalization transaction (passive) on ledger L1 that signals the permanent extinguishment of the asset. This transaction must include a hash reference to the lock transaction on L1 previously in step (2.3). This indicates that the asset is no longer associated with the public-key of its previous owner (originator) and that the asset instance is no longer recognized on the ledger L1.

- Commit-final (3.4): Gateway G1 indicates to G2 that G1 has performed a local lock-finalization on L1. This message must be digitally signed by G1 and should include the block number and transaction number (of the confirmed block) on ledger L1.

- Asset-create (3.5): Gateway issues a transaction on ledger L2 to create the asset, associated with the public-key of the beneficiary. This transaction must include a hash of the previous message (3.4) and hash reference to the log-incoming transaction on L2 previously in step (2.4). These hash references connects the newly created asset with the overall transfer event originating from gateway G1.

- Ack-final (3.6): Gateway G2 indicates to G1 that G2 has performed an asset-creation transaction on L2. This message must be

digitally signed by G2 and should include the block number and transaction number (of the confirmed block) on ledger L2.

- Location-record (3.7): Gateway G1 has the option to record the block number and transaction number (as reported by G2 in the previous step) to ledger L1, using a passive transaction. This transaction should include a hash reference to the confirmed lock-finalization transaction on L1 from step 3.3. This information may aid in future search, audit and accountability purposes from a legal perspective.

- Transfer complete (3.8): Gateway G1 must explicitly close the asset transfer session with gateway G2. This allows both sides to close down the secure channel established in Phase 1.

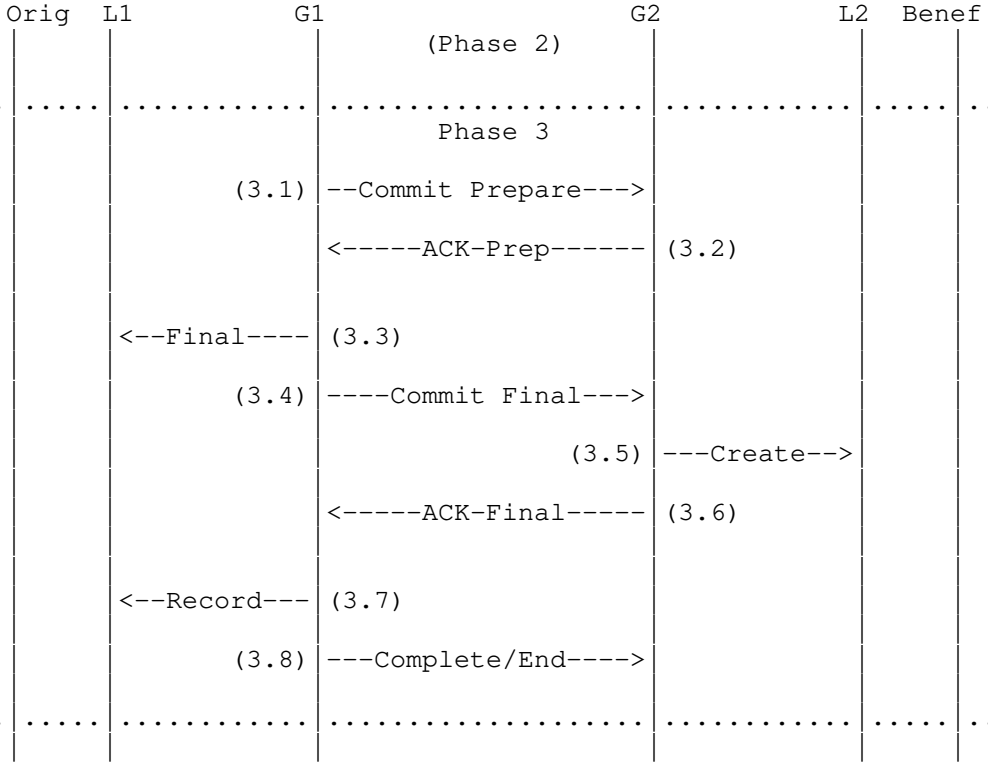

Figure 4

8. Related Open Issues

There are a number of open issues that are related to the asset transfer protocol between gateway nodes. Some of the issues are due to the fact that blockchain technology is relatively new, and that technical constructs designed for interoperability have yet to be addressed. Some of the issues are due to the nascency of the virtual asset industry and lack of conventions, and therefore require industry collaboration to determine these.

8.1. Global identification of blockchain systems and public-keys

There is currently no standard nomenclature to identify blockchain systems in a globally unique manner. The analog to this is the AS-numbers associated with IP routing autonomous systems.

Furthermore, an address (public-key) may not be unique to one blockchain system. An entity (e.g. user) may in fact employ the same public-key at multiple distinct blockchain systems simultaneously.

However, in order to perform an asset transfer from one blockchain system to another, there needs to be mechanism that resolves the beneficiary identifier (as known to the originator) to the correct public-key and blockchain system as intended by the originator.

8.2. Selection of gateways nodes within a blockchain system

A given blockchain system must possess the capability to select or designate gateway nodes that will perform an asset transfer across blockchain systems.

A number of blockchain systems already employ consensus mechanisms that elect a node to perform the transaction processing (e.g. proof of stake in Ethereum). The same consensus mechanisms may be used to elect the gateway node.

However, there are some blockchain systems that do not elect a single node and which employ a race-to-process strategy (e.g. proof of work in Bitcoin). Since the winner of the proof of work can be any node in the blockchain system, this implies that all the nodes in these types of blockchains must be gateway-capable.

8.3. Commitment protocols and forms of commitment evidence

Within Phase 2, the gateway nodes must implement one (or more) transactional commitment protocols that permit the coordination between two gateways, and the final commitment of the asset transfer.

The choice of the commitment protocol (type/version) and the corresponding commitment evidence must be negotiated between the gateways during Phase 1.

For example, in Phase 2 and Phase 3 discussed above the gateways G1 and G2 may implement the classic 2 Phase Commit (2PC) protocol [Gray81] as a means to ensure efficient and non-disputable commitments to the asset transfer.

Historically, transactional commitment protocols employ locking mechanisms to prevent update conflicts on the data item in question. When used within the context of virtual asset transfers across blockchain systems, the fact that an asset has been locked by G1 (as the 2PC coordinator) must be communicated to G2 (as the 2PC participant) in an indisputable manner.

The exact form of this evidence of asset-locking must be standardized (for the given transactional commitment protocol) to eliminate any ambiguity.

9. Security Considerations

Although the current interoperability architecture for blockchain gateways assumes the externalization of the value of assets, as a blockchain system holds an increasing number of virtual assets it becomes attractive to attackers seeking to obtain cryptographic keys of its nodes and its end-users.

Gateway nodes are of particular interest to attackers because they enable the transferal of virtual assets to external blockchain systems, which may or may not be regulated. As such, hardening technologies and tamper-resistant crypto-processors (e.g. TPM, SGX) should be used for implementations of gateways [HS19].

10. Policy Considerations

Virtual asset transfers must be policy-driven in the sense that it must observe and enforce the policies defined for the blockchain domain. Resources that make-up a blockchain systems are owned and operated by entities (e.g. legal persons or organizations), and these entities typically operate within regulatory jurisdictions [FATF]. It is the responsibility of these entities to translate regulatory policies into functions on blockchain systems that comply to the relevant regulatory policies.

At the application layer, asset transfers must take into consideration the legal status of assets and incorporate relevant asset-related policies into their business logic. These policies

must permeate down to the nodes that implement the functions of asset transaction processing.

The smart contract abstraction, based on replicated shared code/state on the ledger [Herl19], must additionally incorporate the notion of policy into the abstraction.

11. References

11.1. Normative References

[FATF] FATF, "International Standards on Combating Money Laundering and the Financing of Terrorism and Proliferation - FATF Revision of Recommendation 15", October 2018, <http://www.fatf-gafi.org/publications/fatfrecommendations/documents/fatf-recommendations.html>.

[ISO] ISO, "Blockchain and distributed ledger technologies- Vocabulary (ISO:22739:2020)", July 2020, <https://www.iso.org>.

[NIST] Yaga, D., Mell, P., Roby, N., and K. Scarfone, "NIST Blockchain Technology Overview (NISTR-8202)", October 2018, <https://doi.org/10.6028/NIST.IR.8202>.

[ODAP] Hargreaves, M. and T. Hardjono, "Open Digital Asset Protocol, October 2020, IETF, draft-hargreaves-odap-00.", October 2020, <https://datatracker.ietf.org/doc/draft-hargreaves-odap/>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

11.2. Informative References

[ABCH20] Ankenbrand, T., Bieri, D., Cortivo, R., Hoehener, J., and T. Hardjono, "Proposal for a Comprehensive Crypto Asset Taxonomy", May 2020, <https://arxiv.org/abs/2007.11877>.

[BVGC20] Belchior, R., Vasconcelos, A., Guerreiro, S., and M. Correia, "A Survey on Blockchain Interoperability: Past, Present, and Future Trends", May 2020, <https://arxiv.org/abs/2005.14282v2>.

[Clar88] Clark, D., "The Design Philosophy of the DARPA Internet Protocols, ACM Computer Communication Review, Proc SIGCOMM 88, vol. 18, no. 4, pp. 106-114", August 1988.

[Gray81] Gray, J., "The Transaction Concept: Virtues and Limitations, in VLDB Proceedings of the 7th International Conference, Cannes, France, September 1981, pp. 144-154", September 1981.

[Herl19] Herlihy, M., "Blockchains From a Distributed Computing Perspective, Communications of the ACM, vol. 62, no. 2, pp. 78-85", February 2019, <https://doi.org/10.1145/3209623>.

[HLP19] Hardjono, T., Lipton, A., and A. Pentland, "Towards and Interoperability Architecture for Blockchain Autonomous Systems, IEEE Transactions on Engineering Management", June 2019, <https://doi:10.1109/TEM.2019.2920154>.

[HS2019] Hardjono, T. and N. Smith, "Decentralized Trusted Computing Base for Blockchain Infrastructure Security, Frontiers Journal, Special Issue on Blockchain Technology, Vol. 2, No. 24", December 2019, <https://doi.org/10.3389/fbloc.2019.00024>.

[IDevID] Richardson, M. and J. Yang, "A Taxonomy of operational security of manufacturer installed keys and anchors. IETF draft-richardson-t2trg-idevid-considerations-01", August 2020, <https://tools.ietf.org/html/draft-richardson-t2trg-idevid-considerations-01>.

[SRC84] Saltzer, J., Reed, D., and D. Clark, "End-to-End Arguments in System Design, ACM Transactions on Computer Systems, vol. 2, no. 4, pp. 277-288", November 1984.

Authors' Addresses

Thomas Hardjono
MIT

Email: hardjono@mit.edu

Martin Hargreaves
Quant Network

Email: martin.hargreaves@quant.network

Ned Smith
Intel

Email: ned.smith@intel.com

Internet Engineering Task Force
Internet-Draft
Updates: 6698, 7671 (if approved)
Intended status: Standards Track
Expires: 3 May 2021

S. Huque
Salesforce
V. Dukhovni
Two Sigma
A. Wilson
Valimail

30 October 2020

TLS Client Authentication via DANE TLSA records
draft-huque-dane-client-cert-05

Abstract

The DANE TLSA protocol [RFC6698] [RFC7671] describes how to publish Transport Layer Security (TLS) server certificates or public keys in the DNS. This document updates RFC 6698 and RFC 7671. It describes how to additionally use the TLSA record to publish client certificates or public keys, and also the rules and considerations for using them with TLS.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 3 May 2021.

Copyright Notice

Table of Contents

1. Introduction and Motivation

The Transport Layer Security (TLS) protocol [RFC5246] [RFC8446] optionally supports the authentication of clients using X.509 certificates [RFC5280] or raw public keys [RFC7250]. TLS applications that perform DANE authentication of servers using TLSA records may also desire to authenticate clients using the same mechanism, especially if the client identity is in the form of or can be represented by a DNS domain name. Some design patterns from the Internet of Things (IoT) plan to make use of this form of authentication, where large networks of physical objects identified by DNS names may authenticate themselves using TLS to centralized device management and control platforms.

In this document, the term TLS is used generically to describe both the TLS and DTLS (Datagram Transport Layer Security) [RFC6347] protocols.

2. Associating Client Identities in TLSA Records

Different applications may have quite different conventions for naming clients via domain names. This document thus does not proscribe a single format, but mentions a few that may have wide applicability.

2.1. Format 1: Service specific client identity

In this format, the owner name of the client TLSA record has the following structure:

_service.[client-domain-name]

The first label identifies the application service name. The remaining labels are composed of the client domain name.

Encoding the application service name into the owner name allows the same client domain name to have different authentication credentials for different application services. There is no need to encode the transport label - the same name form is usable with both TLS and DTLS.

The _service label could be a custom string for an application, but more commonly is expected to be a service name registered in the IANA Service Name Registry [SRVREG].

The RDATA or data field portion of the TLSA record is formed exactly as specified in RFC 6698 and RFC 7671, and carries the same meaning.

2.2. Format 2: DevId: IOT Device Identity

The DevID form of the TLSA record has the following structure:

[devicename]._device.[org-domain-name]

It is loosely based on the proposed PKI Certificate Identifier Format for Devices [CERTDEVID], but is simpler in form. It makes no distinction between manufacturer issued and locally issued certificates, and does away with the "serial" and "type" labels. The "_device" label that precedes the organization domain name allows all the device identities to be delegated to a subzone or to another party.

3. Authentication Model

The authentication model assumed in this document is the following:

The client is assigned an identity corresponding to a DNS domain name. This domain name doesn't necessarily have any relation to its network layer addresses. Clients often have dynamic or unpredictable addresses, and may move around the network, so tying their identity to network addresses is not feasible or wise in the general case.

The client generates (or has generated for it) a private and public key pair. Where client certificates are being used, the client also has a certificate binding the name to its public key. The certificate or public key has a corresponding TLSA record published in the DNS, which allows it to be authenticated directly via the DNS (using the DANE-TA or DANE-EE certificate usage modes) or via a PKIX public CA system constraint if the client's certificate was issued by a public CA (using the PKIX-TA or PKIX-EE DANE usage modes).

4. Client Identifiers in X.509 certificates

If the TLS DANE Client Identity extension (see Section 5) is not being used, the client certificate MUST have have the client's DNS name specified in the Subject Alternative Name extension's dNSName type.

If the TLS DANE Client Identity extension is in use, then with DANE-EE(3), the subject name need not be present in the certificate.

5. Signaling the Client's DANE Identity in TLS

The client SHOULD explicitly signal that it has a DANE identity. The most important reason is that the server may want an explicit indication from the client that it has a DANE record, so as to avoid unnecessary DNS queries in-band with the TLS handshake.

The DANE Client Identity TLS extension [TLSCLIENTID] is used for this purpose. This extension can also be used to convey the actual DANE client identity (i.e. domain name) that the TLS server should attempt to authenticate. This is required when using TLS raw public key authentication, since there is no client certificate from which to extract the client's DNS identity. It is also helpful when the client certificate contains multiple identities, and only a specific one has a DANE record.

An additional case where such client signaling is helpful, is one where DANE client authentication is optional, and there is a population of buggy client software that does not react gracefully to receipt of a Certificate Request message from the TLS server. This extension allows TLS servers to deal with this situation by selectively sending a Certificate Request message only to clients that have sent this extension.

6. Example TLSA records for clients

The following examples are provided in the textual presentation format of the TLSA record.

6.1. Format 1: Service Specific Client Identity

An example TLSA record for the client "device1.example.com." and the application "smtp-client". This record specifies the SHA-256 hash of the subject public key component of the end-entity certificate corresponding to the client. The certificate usage for this record is 3 (DANE-EE) and thus is validated in accordance with section 5.1 of RFC 7671.

```
_smtp-client.device1.example.com. IN TLSA (
   3 1 1 d2abde240d7cd3ee6b4b28c54df034b9
         7983a1d16e8a410e4561cb106618e971 )
```

6.2. Format 2: DevID

An example TLSA record for the device named "sensor7" managed by the organization "example.com" This record specifies the SHA-512 hash of the subject public key component of an EE certificate corresponding to the client.

```
sensor7._device.example.com. IN TLSA (
   3 1 2 0f8b48ff5fd94117f21b6550aaee89c8
         d8adbc3f433c8e587a85a14e54667b25
         f4dcd8c4ae6162121ea9166984831b57
         b408534451fd1b9702f8de0532ecd03c )
```

7. Changes to Client and Server behavior

A TLS Client conforming to this specification MUST have a signed DNS TLSA record published corresponding to its DNS name and X.509 certificate or public key. The client presents this certificate or public key in the TLS handshake with the server. The client should not offer ciphersuites that are incompatible with its certificate or public key. If the client’s certificate has a DANE record with a certificate usage other than DANE-EE, then the presented client certificate MUST have have the client’s DNS name specified in the Subject Alternative Name extension’s dNSName type.

Additionally, when using raw public key authentication, the client MUST send the TLS DANE Client Identity extension [TLSCLIENTID] in its Client Hello message. When using X.509 certificate authentication, it SHOULD send this extension.

A TLS Server implementing this specification performs the following steps:

* Request a client certificate in the TLS handshake (the "Client Certificate Request" message). This could be done unconditionally, or only when it receives the TLS DANE Client Identity extension from the client.

* If the client has sent a non-empty DANE Client Identity extension, then extract the client's domain name from the extension. Otherwise, extract the client identity from the Subject Alternative Name extension's dNSName type.

* Construct the DNS query name for the corresponding TLSA record. If the TLS DANE client identity extension was present, then this name should be used. Otherwise, identities from the client certificate are used.

* Look up the TLSA record set in the DNS. The response MUST be cryptographically validated using DNSSEC. The server could perform the DNSSEC validation itself. It could also be configured to trust responses obtained via a validating resolver to which it has a secure connection.

* Extract the RDATA of the TLSA records and match them to the presented client certificate according to the rules specified in the DANE TLS protocol [RFC6698] [RFC7671]. If successfully matched, the client is authenticated and the TLS session proceeds. If unsuccessful, the server MUST treat the client as unauthenticated (e.g. it could terminate the session, or proceed with the session giving the client access to resources as a generic unauthenticated user).

* If there are multiple records in the TLSA record set, then the client is authenticated as long as at least one of the TLSA records matches, subject to RFC7671 digest agility, which SHOULD be implemented.

If the DANE Client Identity extension is not present, and the presented client certificate has multiple distinct reference identifier types (e.g. a dNSName, and an rfc822Name) then TLS servers configured to perform DANE authentication according to this specification should only examine and authenticate the dNSName.

If the presented client certificate has multiple dNSName identities, then the client MUST use the TLS DANE client identity extension to unambiguously indicate its intended name to the server.

Specific applications may be designed to require additional validation steps. For example, a server might want to verify the client's IP address is associated with the certificate in some manner, e.g. by confirming that a secure reverse DNS lookup of that address ties it back to the same domain name, or by requiring an iPAddress component to be included in the certificate. Such details are outside the scope of this document, and should be outlined in other documents specific to the applications that require this behavior.

Servers may have their own whitelisting and authorization rules for which certificates they accept. For example a TLS server may be configured to only allow TLS sessions from clients with certificate identities within a specific domain or set of domains.

8. Raw Public Keys

When using raw public keys in TLS [RFC7250], this specification requires the use of the TLS DANE Client Identity extension. The associated DANE TLSA records employ only certificate usage 3 (DANE-EE) and a selector value of 1 (SPKI), as described in [RFC7671].

9. Acknowledgements

TBD.

10. IANA Considerations

This document includes no request to IANA.

11. Security Considerations

This document updates RFC 6698 by defining the use of the TLSA record for client TLS certificates. There are no security considerations for this document beyond those described in RFC 6698 and RFC 7671 and in the specifications for TLS and DTLS [RFC8446], [RFC5246], [RFC6347].

12. References

12.1. Normative References

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <https://www.rfc-editor.org/info/rfc5246>.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <https://www.rfc-editor.org/info/rfc5280>.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <https://www.rfc-editor.org/info/rfc6347>.

[RFC6698] Hoffman, P. and J. Schlyter, "The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA", RFC 6698, DOI 10.17487/RFC6698, August 2012, <https://www.rfc-editor.org/info/rfc6698>.

[RFC7250] Wouters, P., Ed., Tschofenig, H., Ed., Gilmore, J., Weiler, S., and T. Kivinen, "Using Raw Public Keys in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", RFC 7250, DOI 10.17487/RFC7250, June 2014, <https://www.rfc-editor.org/info/rfc7250>.

[RFC7671] Dukhovni, V. and W. Hardaker, "The DNS-Based Authentication of Named Entities (DANE) Protocol: Updates and Operational Guidance", RFC 7671, DOI 10.17487/RFC7671, October 2015, <https://www.rfc-editor.org/info/rfc7671>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

[TLSCLIENTID] Huque, S. and V. Dukhovni, "TLS Extension for DANE Client Identity", <https://tools.ietf.org/html/draft-huque-tls-dane-clientid>.

12.2. Informative References

[CERTDEVID] Friel, O. and R. Barnes, "PKI Certificate Identifier Format for Devices", <https://tools.ietf.org/id/draft-friel-pki-for-devices-00.html>.

[SRVREG] IANA, "Service Name and Transport Protocol Port Number Registry", <https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.txt>.

Authors' Addresses

Shumon Huque
Salesforce

Email: shuque@gmail.com

Viktor Dukhovni
Two Sigma

Email: ietf-dane@dukhovni.org

Ash Wilson
Valimail

Email: ash.wilson@valimail.com

Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: 3 May 2021

S. Huque
Salesforce
V. Dukhovni
Two Sigma
A. Wilson
Valimail

30 October 2020

TLS Extension for DANE Client Identity
draft-huque-tls-dane-clientid-03

Abstract

This document specifies a TLS and DTLS extension to convey a DNS-Based Authentication of Named Entities (DANE) Client Identity to a TLS or DTLS server. This is useful for applications that perform TLS client authentication via DANE TLSA records.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 3 May 2021.

Copyright Notice

Table of Contents

1. Introduction

This document specifies a Transport Layer Security (TLS) extension [RFC6066] to convey a DANE [RFC6698] Client Identity to the TLS server. This is useful for applications that perform TLS client authentication via DANE TLSA records, as described in [DANECLIENT]. The extension could be empty to indicate to the server that the client has a DANE record and that the server can perform DANE authentication of the client with the identity extracted from the client certificate. Or the extension can contain the full client identity, in the form of the DNS domain name that is expected to have a DANE TLSA record published for it.

This extension supports both TLS [RFC5246] [RFC8446] and DTLS [RFC6347], and the term TLS in this document is used generically to describe both protocols.

2. Overview

When TLS clients use X.509 client certificates or raw public keys that are authenticated via DANE TLSA records, it is useful for them to convey their intent to be authenticated via DANE, or even to convey their complete DANE identity to the server. The TLS extension defined in this document is used to accomplish this.

In the case of X.509 client certificates, a TLS server can learn the client's identity by examining subject alternative names included in the certificate itself. However, without a mechanism such as the one defined in this extension, the TLS server cannot know apriori that

the client has a published TLSA record, and thus may unnecessarily issue DNS queries for DANE TLSA records in-band with the TLS handshake even in cases where the client has no TLSA record associated with it. When multiple identities are present in the certificate, a client can use this extension to specify exactly which one the server should use. An additional situation in which this extension helps is where some TLS servers may need to selectively prompt for client certificate credentials only for clients that are equipped to provide certificates.

When TLS raw public keys [RFC7250] are being used to authenticate the client, the client uses this extension to explicitly indicate to the server what its domain name identity is (since there is no X.509 certificate from which the identity can be extracted).

Detailed protocol behavior of TLS clients and servers is described in [DANECLIENT].

3. DANE Client Identity Extension

The DANE Client Identity Extension type, "dane_clientid", will have a value assigned and registered in the IANA TLS Extensions registry. Its format is similar to the TLS Server Name Indication extension.

A TLS client implementing this specification SHOULD send an extension of type "dane_clientid" in the ClientHello handshake message to TLS servers it intends to perform DANE client authentication with.

If the client only needs to indicate that it has a DANE record and that the client's domain name identity can be obtained from its certificate, then the extension sent can be empty.

If the client additionally needs to send its domain name identity, then the "extension_data" field of the extension MUST contain a "ClientNameList" data structure defined in the following format:

```
struct {
    NameType name_type;
    select (name_type) {
        case host_name: HostName;
    } name;
} ClientName;

enum {
    host_name(0), (255)
} NameType;

opaque HostName<1..2^16-1>;

struct {
    ClientName client_name_list<1..2^16-1>
} ClientNameList;
```

The opaque HostName field contains the domain name of the client in textual presentation format, as described in RFC 1035 [RFC1035]. The ClientNameList MUST NOT contain more than one name of the same name_type. Currently only one "host_name" type is defined.

A TLS server implementing this specification MAY send back an empty extension of type "dane_clientid" in its SeverHello handshake message to indicate that it understands the extension and intends to perform DANE client authentication. (Is there a compelling reason to do this?)

4. Security Considerations

This extension is sent in the first flight of the TLS client's network data (Client Hello), which is in clear text.

To prevent unnecessary privacy leakage of the client's name in cleartext, a TLS client implementing this specification should be configured to only send this extension to TLS servers it intends to perform client authentication with.

Ideally, this extension should be used with the proposed TLS Encrypted Client Hello extension [ECH], which encrypts the entire Client Hello message. This will prevent leakage of the hostname, if included in the extension, in clear text.

5. IANA Considerations

This extension requires the registration of a new value in the TLS ExtensionsType registry.

6. Normative References

[DANECLIENT]
Huque, S. and V. Dukhovni, "TLS Client Authentication via DANE TLSA Records", <https://tools.ietf.org/html/draft-huque-dane-client-cert>.

[ECH] Rescorla, E., Oku, K., Sullivan, N., and C.A. Wood, "TLS Encrypted Client Hello", <https://tools.ietf.org/html/draft-ietf-tls-esni>.

[RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, DOI 10.17487/RFC1035, November 1987, <https://www.rfc-editor.org/info/rfc1035>.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <https://www.rfc-editor.org/info/rfc5246>.

[RFC6066] Eastlake 3rd, D., "Transport Layer Security (TLS) Extensions: Extension Definitions", RFC 6066, DOI 10.17487/RFC6066, January 2011, <https://www.rfc-editor.org/info/rfc6066>.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <https://www.rfc-editor.org/info/rfc6347>.

[RFC6698] Hoffman, P. and J. Schlyter, "The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA", RFC 6698, DOI 10.17487/RFC6698, August 2012, <https://www.rfc-editor.org/info/rfc6698>.

[RFC7250] Wouters, P., Ed., Tschofenig, H., Ed., Gilmore, J., Weiler, S., and T. Kivinen, "Using Raw Public Keys in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", RFC 7250, DOI 10.17487/RFC7250, June 2014, <https://www.rfc-editor.org/info/rfc7250>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

Authors' Addresses

Shumon Huque
Salesforce

Email: shuque@gmail.com

Viktor Dukhovni
Two Sigma

Email: ietf-dane@dukhovni.org

Ash Wilson
Valimail

Email: ash.wilson@valimail.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: 24 April 2021

S. Santesson
3xA Security
R. Housley
Vigil Security

21 October 2020

Signature Validation Token
draft-santesson-svt-00

Abstract

Electronic signatures have a limited lifespan with respect to the time period that they can be validated and determined to be authentic. The Signature Validation Token (SVT) defined in this specification provides evidence that asserts the validity of an electronic signature. The SVT is provided by a trusted authority, which asserts that a particular signature was successfully validated according to defined procedures at a certain time. Any future validation of that electronic signature can be satisfied by validating the SVT without any need to also validate the original electronic signature or the associated digital certificates. SVT supports electronic signatures in CMS, XML, and PDF documents.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 24 April 2021.

Copyright Notice

Table of Contents

1. Introduction

Electronic signatures have a limited lifespan regarding when they can be validated and determined to be authentic. Many factors make it more difficult to validate electronic signatures over time. For example:

* Trusted information about the validity of the certificate containing the signer's public key is not available.

* Trusted information about the date and time when the signature was actually created is not available.

* Algorithms used to create the electronic signature are no longer considered secure.

* Services necessary to validate the signature are no longer available.

* Supporting evidence such as CA certificates, OCSP responses, CRLs, or timestamps.

The challenges to validation of an electronic signature increases over time, and eventually it is simply impossible to verify the signature with a sufficient level of assurance.

Existing standards, such as the ETSI XAdES [XADES] profile for XML signatures [XMLDSIG11], ETSI PAdES [PADES] profile for PDF signatures [ISOPDF2], and ETSI CAdES [CADES] profile for CMS signatures [RFC5652] can be used to prolong the lifetime of a signature by storing data that supports validation of the electronic signature beyond the lifetime of the certificate containing the signer's public key, which is often referred to as the signing certificate. The problem with this approach is that the amount of information that must be stored along with the electronic signature constantly grows over time. The increasing amount of information and signed objects that need to be validated in order to verify the original electronic signature grows in complexity to the point where validation of the electronic signature may become infeasible.

The Signature Validation Token (SVT) defined in this specification takes a fundamentally different approach to the problem by providing evidence by a trusted authority that asserts the validity of an electronic signature. The SVT asserts that a particular electronic signature was successfully validated by a trusted authority according to defined procedures at a certain date and time. Once the SVT is issued by a trusted authority, any future validation of that electronic signature is satisfied by validating the SVT, without any need to also validate the original electronic signature.

This approach drastically reduces the complexity of validation of older electronic signatures for the simple reason that validating the SVT eliminates the need to validate the many signed objects that would otherwise been needed to provide the same level of assurance. The SVT can be signed with private keys and algorithms that provide confidence for a considerable time period. In fact, multiple SVTs can be used to offer greater assurance. For example, one SVT could be produced with a large RSA private key, a second one with a strong elliptic curve, and a third one with a quantum safe digital signature algorithm to protect against advances in computing power and cryptanalytic capabilities. Further, the trusted authority can add additional SVTs in the future using fresh private keys and signatures to extend the lifetime of the, if necessary.

2. Definitions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

This document use the following terms:

* Signed Data - The data covered by a particular electronic signature. This is typically equivalent to the signed content of a document, and it represents the data that the signer intended to sign. In some cases, such as in some XML signatures, the signed data can be the collection of several data fragments each referenced by the signature. In the case of PDF, this is the data covered by the "ByteRange" parameter in the signature dictionary.

* Signed Bytes - These are the actual bytes of data that were hashed and signed by the digital signature algorithm. In most cases, this is not the actual Signed Data, but a collection of signature metadata that includes references (hash) of the Signed Data as well as information about algorithms and other data bound to a signature. In XML, this is the canonicalized SignedInfo element. In CMS and PDF signatures, this is the DER-encoded SignedAttributes structure.

When these terms are used as defined in this section, they appear with a capitalized first letter.

3. Signature Validation Token

The Signature Validation Token (SVT) is created by a trusted service to capture evidence of successful electronic signature verification, and then relying parties can depend on the checking that has already taken place by the trusted service.

3.1. Signature Validation Token Function

The function of the SVT is to capture evidence of electronic signature validity at one instance of secure signature validation process and to use that evidence to eliminate the need to perform any repeated cryptographic validation of the original electronic signature value, as well as reliance on any hash values bound to that signature. The SVT achieves this by binding the following information to a specific electronic signature:

* A unique identification of the electronic signature.

* The data and metadata signed by the electronic signature.

* The signer's certificate that was validated as part of electronic signature verification.

* The certification path that was used to validate the signer's certificate.

* An assertion providing evidence of that the signature was verified, the date and time the verification was performed, the procedures used to verify the electronic signature, and the outcome of the verification.

* An assertion providing evidence of the date and time at which the signature is known to have existed, the procedures used to validate the date and time of existence, and the outcome of the validation.

Using an SVT is equivalent to validating a signed document in a system once, and then using that document multiple times without subsequent revalidating the electronic signature for each usage. Such procedures are common in systems where the document is residing in a safe and trusted environment where it is protected against modification. The SVT allows the safe and trusted environment to expand beyond a locally controlled environment, and the SVT allows a greater period between original electronic signature verification and subsequent usage.

Using the SVT, the electronic signature verification of a document can be take place once using a reliable trusted service, and then any relying party that is able to depend on the verification process already performed by the trusted service. The SVT is therefore not only a valuable tool to extend the lifetime of a signed document, but also avoids the need for careful integration between electronic signature verification and document usage.

3.2. Signature Validation Token Syntax

The SVT is carried in a JSON Web Token (JWT) as defined in [RFC7519].

3.2.1. Data Types

The contents of claims in an SVT are specified using the following data types:

* String - JSON Data Type of string that contains an arbitrary case sensitive string value.

* Base64Binary - JSON Data Type of string that contains of Base64 encoded byte array of binary data.

* StringOrURI - JSON Data Type of string that contains an arbitrary string or an URI as defined in [RFC7519], which REQUIRES a value containing the colon character (":") to be a URI.

* URI - JSON Data Type of string that contains an URI as defined in [RFC7519].

* Integer - JSON Data Type of number that contains a 32-bit signed integer value (from -2^31 to 2^31-1).

* Long - JSON Data Type of number that contains a 64-bit signed integer value (from -2^63 to 2^63-1).

* NumericDate - JSON Data Type of number that contains a data as defined in [RFC7519], which is the number of seconds from 1970-01-01T00:00:00Z UTC until the specified UTC date/time, ignoring leap seconds.

* Boolean - JSON Data Type of boolean that contains explicit value of true or false.

* Object<Class> - A JSON object holding a claims object of a class defined in this specification (see Section 3.2.2).

* Map<Type> - A JSON object with name-value pairs where the value is an object of the specified Type in the notation. For example, Map<String> is a JSON object with name value pairs where all values are of type String.

* Array - A JSON array of a specific data type as defined in this section. An array is expressed in this specification by square brackets. For example, [String] indicates an array of String values, and [Object<DocHash>] indicates an array of DocHash objects.

* Null - A JSON null that represents an absent value. A claim with a null value is equivalent with an absent claim.

3.2.2. Signature Validation Token JWT Claims

The SVT MUST contain only JWT claims in the following list:

* jti - A String data type that is a "JWT ID" registered claim according to [RFC7519]. It is RECOMMENDED that the identifier holds a hexadecimal string representation of a 128-bit unsigned integer. A SVT MUST contain one "JWT ID" claim.

* iss - A StringOrURI data type that is an "Issuer" registered claim according to [RFC7519], which is an arbitrary unique identifier of the SVT issuer. This value SHOULD have the value of an URI based on a domain owned by the issuer. A SVT MUST contain one "Issuer" claim.

* iat - A NumericDate data type that is an "Issued At" registered claim according to [RFC7519], which expresses the date and time when this SVT was issued. A SVT MUST contain one "Issued At" claim.

* aud - A [StringOrURI] data type or a StringOrURI data type that is an "Audience" registered claim according to [RFC7519]. The audience claim is an array of one or more identifiers, identifying intended recipients of the SVT. Each identifier MAY identify a single entity, a group of entities or a common policy adopted by a group of entities. If only one value is provided it MAY be provided as a single StringOrURI data type value instead of as an array of values. Inclusion of the "Audience" claim in a SVT is OPTIONAL.

* exp - A NumericDate data type that is an "Expiration Time" registered claim according to [RFC7519], which expresses the date and time when services and responsibilities related to this SVT is no longer provided by the SVT issuer. The precise meaning of the

expiration time claim is defined by local policies. See implementation note below. Inclusion of the "Expiration Time" claim in a SVT is OPTIONAL.

* sig_val_claims - A Object<SigValidation> data type that contains signature validation claims for this SVT extending the standard registered JTW claims above. A SVT MUST contain one sig_val_claims claim.

Note: An SVT asserts that a particular validation process was undertaken at a stated date and time. This fact never changes and never expires. However, some other aspects of the SVT such as liability for false claims or service provision related to a specific SVT may expire after a certain period of time, such as a service where an old SVT can be upgraded to a new SVT signed with fresh keys and algorithms.

3.2.3. SigValidation Object Class

The sig_val_claims JWT claim uses the SigValidation object class. A SigValidation object holds all custom claims, and a SigValidation object contains the following parameters:

* ver - A String data type representing the version. This parameter MUST be present, and the version in this specification indicated by the value "1.0".

* profile - A StringOrURI data type representing the name of a profile that defines conventions followed for specific claims and any extension points used by the SVT issuer. Inclusion of this parameter is OPTIONAL.

* hash_algo - A URI data type that identifies the hash algorithm used to compute the hash values within the SVT. The URI identifier MUST be one defined in [RFC6931] or in the IANA registry defined by this specification. This parameter MUST be present.

* sig - A [Object<Signature>] data type that gives information about validated electronic signatures as an array of Signature objects. If the SVT contains signature validation evidence for more than one signature, then each signature is represented by a separate Signature object. At least one Signature object MUST be present.

* ext - A Map<String> data type that provides additional claims related to the SVT. Extension claims are added at the discretion of the SVT issuer; however, extension claims MUST follow any conventions defined in a profile of this specification (see Section 4). Inclusion of this parameter is OPTIONAL.

3.2.4. Signature Claims Object Class

The sig parameter in the SigValidation object class uses the Signature object class. The Signature object contains claims related to signature validation evidence for one signature, and it contains the following parameters:

* sig_ref - A Object<SigReference> data type that contains reference information identifying the target signature. This parameter MUST be present.

* sig_data - A [Object<SignedData>] data type that contains an array of references to Signed Data that was signed by the target electronic signature. This parameter MUST be present.

* signer_cert_ref - A Object<CertReference> data type that references the signer's certificate and optionally reference to a supporting certification path that was used to verify the target electronic signature. This parameter MUST be present.

* sig_val - A [Object<PolicyValidation>] data type that contains an array of results of signature verification according to defined procedures. This parameter MUST be present.

* time_val - A [Object<TimeValidation>] data type that contains an array of time verification results that the target signature has existed at a specific date and time in the past. Inclusion of this parameter is OPTIONAL.

* ext - A MAP<String> data type that provides additional claims related to the target signature. Extension claims are added at the discretion of the SVT Issuer; however, extension claims MUST follow any conventions defined in a profile of this specification (see Section 4). Inclusion of this parameter is OPTIONAL.

3.2.5. SigReference Claims Object Class

The sig_ref parameter in the Signature object class uses the SigReference object class. The SigReference object provides information used to match the Signature claims object to a specific target electronic signature and to verify the integrity of the target signature value and Signed Bytes, and it contains the following parameters:

* id - A String data type that contains an identifier assigned to the target signature. Inclusion of this parameter is OPTIONAL.

* sig_hash - A Base64Binary data type that contains a hash value of the target electronic signature value. This parameter MUST be present.

* sb_hash - A Base64Binary data type that contains a hash value of the Signed Bytes of the target electronic signature. This parameter MUST be present.

3.2.6. SignedData Claims Object Class

The sig_data parameter in the Signature object class uses the SignedData object class. The SignedData object provides information used to verify the target electronic signature references to Signed Data as well as to verify the integrity of all data that is signed by the target signature, and it contains the following parameters:

* ref - A String data type that contains a reference identifier for the data or data fragment covered by the target electronic signature. This parameter MUST be present.

* hash - A Base64Binary data type that contains the hash value for the data covered by the target electronic signature. This parameter MUST be present.

3.2.7. PolicyValidation Claims Object Class

The sig_val parameter in the Signature object class uses the PolicyValidation object class. The PolicyValidation object provides information about the result of a validation process according to a spefific policy, and it contains the following parameters:

* pol - A StringOrURI data type that contains the identifier of the policy governing the electronic signature verification process. This parameter MUST be present.

* res - A String data type that contains the result of the electronic signature verification process. The value MUST be one of "PASSED", "FAILED" or "INDETERMINATE" as defined by [ETSI319102-1]. This parameter MUST be present.

* msg - A String data type that contains a message describing the result. Inclusion of this parameter is OPTIONAL.

* ext - A MAP<String> data type that provides additional claims related to the target signature. Extension claims are added at the discretion of the SVT Issuer; however, extension claims MUST follow any conventions defined in a profile of this specification (see Section 4). Inclusion of this parameter is OPTIONAL.

3.2.8. TimeValidation Claims Object Class

The time_val parameter in the Signature object class uses the TimeValidation object class. The TimeValidation claims object provides information about the result of validating time evidence asserting that the target signature existed at a particular date and time in the past, and it contains the following parameters:

* time - A NumericDate data type that contains the verified time. This parameter MUST be present.

* type - A StringOrURI data type that contains an identifier of the type of evidence of time. This parameter MUST be present.

* iss - A StringOrURI data type that contains an identifier of the entity that issued the evidence of time. This parameter MUST be present.

* id - A String data type that contains an unique identifier assigned to the evidence of time. Inclusion of this parameter is OPTIONAL.

* val - A [Object<PolicyValidation>] data type that contains an array of results of the time evidence validation according to defined validation procedures. Inclusion of this parameter is OPTIONAL.

* ext - A MAP<String> data type that provides additional claims related to the target signature. Extension claims are added at the discretion of the SVT Issuer; however, extension claims MUST follow any conventions defined in a profile of this specification (see Section 4). Inclusion of this parameter is OPTIONAL.

3.2.9. CertReference Claims Object Class

The signer_cert_ref parameter in the Signature object class uses the CertReference object class. The CertReference object references a single X.509 certificate or a X.509 certification path, either by providing the certificate data or by providing hash references for certificates that can be located in the target electronic signature, and it contains the following parameters:

* type - A StringOrURI data type that contains an identifier of the type of reference. The type identifier MUST be one of the identifiers defined below, an identifier specified by the selected profile, or a URI identifier. This parameter MUST be present.

* ref - A [String] data type that contains an array of string parameters according to conventions defined by the type identifier. This parameter MUST be present.

The following type identifiers are defined:

* "chain" - The ref contains an array of Base64 encoded X.509 certificates [RFC5280]. The certificates MUST be provided in the order starting with the end entity certificate. Any following certificate must be able to validate the signature on the previous certificate in the array.

* chain_hash - The ref contains an array of one or more Base64 encoded hash values where each hash value is a hash over a X.509 certificate [RFC5280] used to validate the signature. The certificates MUST be provided in the order starting with the end entity certificate. Any following certificate must be able to validate the signature on the previous certificate in the array. This option MUST NOT be used unless all hashed certificates are present in the target electronic signature.

Note: All certificates referenced using the identifiers above are X.509 certificates. Profiles of this specification MAY define alternative types of public key containers; however, a major function of these referenced certificates is not just to reference the public key, but also to provide the subject name of the signer. It is therefore important for the full function of an SVT that the referenced public key container also provides the means to identify of the signer.

3.2.10. SVT JOSE Header

The SVT JWT MUST contain the following JOSE header parameters in accordance with Section 5 of [RFC7519]:

* typ - This parameter MUST have the string value "JWT" (upper case).

* alg - This parameter identifies the algorithm used to sign the SVT JWT. The algorithm identifier MUST be specified in [RFC7518] or the IANA JSON Web Signature and Encryption Algorithms Registry [add a ref]. The specified signature hash algorithm MUST be identical to the hash algorithm specified in the hash_algo parameter of the SigValidation object within the sig_val_claims claim.

The SVT header MUST contain a public key or a reference to a public key used to verify the signature on the SVT in accordance with [RFC7515]. Each profile, as discussed in Section 4, MUST define the requirements for how the key or key reference is included in the header.

4. Profiles

Each signed document and signature type will have to define the precise content and use of several claims in the SVT.

Each profile MUST as a minimum define:

* How to reference the Signed Data content of the signed document.

* How to reference to the target electronic signature and the Signed Bytes of the signature.

* How to reference certificates supporting each electronic siganture.

* How to include public keys or references to public keys in the SVT.

* Whether each electronic signature is supported by a single SVT, or whether one SVT may support multiple electronic signatures of the same document.

A profile MAY also define:

* Explicit information on how to perform signature validation based on an SVT.

* How to attach an SVT to an electronic signature or signed document.

5. Signature Verification with a SVT

Signature verification based on an a SVT MUST follow these steps:

1. Locate all available SVTs available for the signed document that are relevant for the target electronic signature.

2. Select the most recent SVT that can be successfully validated and meets the requirement of the relying party.

3. Verify the integrity of the signature and the Signed Bytes of the target electronic signature using the sig_ref claim.

4. Verify that the Signed Data reference in the original electronic signature matches the reference values in the sig_data_ref claim.

5. Verify the integrity of referenced Signed Data using provided hash values in the sig_data_ref claim.

6. Obtain the verified certificates supporting the asserted electronic signature verification through the signer_cert_ref claim.

7. Verify that signature validation policy results satisfy the requirements of the relying party.

8. Verify that verified time results satisfy the context for the use of the signed document.

After successfully performing these steps, signature validity is established as well as the trusted signer certificate binding the identity of the signer to the electronic signature.

6. IANA Considerations

{ To be written }

7. Security Considerations

{ To be written }

8. Normative References

[CADES] ETSI, "Electronic Signatures and Infrastructures (ESI); CAdES digital signatures; Part 1: Building blocks and CAdES baseline signatures", ETSI EN 319 122-1 v1.1.1, April 2016.

[ETSI319102-1]
ETSI, "ETSI - Electronic Signatures and Infrastructures (ESI); Procedures for Creation and Validation of AdES Digital Signatures; Part 1: Creation and Validation", ETSI EN 319 102-1 v1.1.1, May 2016.

[ISOPDF2] ISO, "Document management -- Portable document format -- Part 2: PDF 2.0", ISO 32000-2, July 2017.

[PADES] ETSI, "Electronic Signatures and Infrastructures (ESI); PAdES digital signatures; Part 1: Building blocks and PAdES baseline signatures", ETSI EN 319 142-1 v1.1.1, April 2016.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <https://www.rfc-editor.org/info/rfc5280>.

[RFC5652] Housley, R., "Cryptographic Message Syntax (CMS)", STD 70, RFC 5652, DOI 10.17487/RFC5652, September 2009, <https://www.rfc-editor.org/info/rfc5652>.

[RFC6931] Eastlake 3rd, D., "Additional XML Security Uniform Resource Identifiers (URIs)", RFC 6931, DOI 10.17487/RFC6931, April 2013, <https://www.rfc-editor.org/info/rfc6931>.

[RFC7515] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", RFC 7515, DOI 10.17487/RFC7515, May 2015, <https://www.rfc-editor.org/info/rfc7515>.

[RFC7518] Jones, M., "JSON Web Algorithms (JWA)", RFC 7518, DOI 10.17487/RFC7518, May 2015, <https://www.rfc-editor.org/info/rfc7518>.

[RFC7519] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015, <https://www.rfc-editor.org/info/rfc7519>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[XADES] ETSI, "Electronic Signatures and Infrastructures (ESI); XAdES digital signatures; Part 1: Building blocks and XAdES baseline signatures", ETSI EN 319 132-1 v1.1.1, April 2016.

[XMLDSIG11] Eastlake, D., Reagle, J., Solo, D., Hirsch, F., Nystrom, M., Roessler, T., and K. Yiu, "XML Signature Syntax and Processing Version 1.1", W3C Proposed Recommendation, 11 April 2013.

Appendix A. Appendix: Examples

The following example illustrates a basic SVT according to this specification issued for a signed PDF document.

Note: Line breaks in the decoded example are inserted for readablilty. Line breaks are not allowed in valid JSON data.

Signature validation token JWT:

```
eyJraWQiOiJPZW5JKzQzNEpoYnZmRG50ZlZcLzhyT3hHN0Zrdnlq
QlhvaFZoQWU1Zks4YW5vdjFTNjg4cjdLYmFsK2Z2cGFIMWo4aWJnNTJRQnkxUFE9
PSIsInR5cCI6IkpXVCIsImFsZyI6IlJTNTEyIn0.eyJhdWQiOiJodHRwOlwvXC9l
eGFtcGxlLmNvbVwvYXVkaWVuY2UxIiwiaXNzIjoiaHR0cHM6XC9cL3N3ZWRlbmNv
bm5lY3Quc2VcL3ZhbGlkYXRvciIsImlhdCI6MTU4MjczMDY0NSwianRpIjoiZTIy
YzViZTZkZDZjYzZkYjgzNGJjY2QwNjZmNWUyZTMiLCJzaWdfdmFsX2NsYWltcyI6
eyJzaWciOlt7ImV4dCI6bnVsbCwic2lnX3ZhbCI6W3sibXNnIjoiSW52YWxpZCBz
aWduYXR1cmUiLCJleHQiOm51bGwsInJlcyI6IkZBSUxFRCIsInBvbCI6Imh0dHA6
XC9cL2lkLnN3ZWRlbmNvbm5lY3Quc2VcL3N2dFwvc2lndmFsLXBvbGljeVwvY2hh
aW5cLzAxIn1dLCJzaWdfcmVmIjp7InNpZ19oYXNoIjoiQmh1RTlCQ1lkcUxneW93
bDJQYm1uSzlkSkFtaVZ0VDF1OVZnaUY5OWgyaFZQekU0WExXdmJDUGU0YUNKM0l6
RmZvTDlrM3RXcjBXK3d5OUJlcWlyY1E9PSIsImlkIjpudWxsLCJzYl9oYXNoIjoi
YnVlcTVIVE8xYnRwQ3JYUlg3VHpFS1VyTkpRaEdHOHFCaDR3eEVTcVJMM0J6bjRj
bHZLMzdqWXUwS2pNTWtnSlFFTWZBMWIzaW1peTc5dDdoK1loOHc9PSJ9LCJzaWdu
ZXJfY2VydF9yZWYiOnsicmVmIjpbIk5TdUZNXC92SitiZUJsUXRRVHptY1loNXg3
TDhXQzlFMUtQSFJBMWlvTk9sS1ZHYmxhOVVSelljc2lzQXgyYmNzcU9oa3ZWVGMz
bUs5RTZhZzA3aGZhdz09Il0sInR5cGUiOiJjaGFpbl9oYXNoIn0sInNpZ19kYXRh
X3JlZiI6W3sicmVmIjoiMCAxMjI5MzUgMTI3OTM3IDI3NDMwIiwiaGFzaCI6Imt1
VWI4NkZzTU5tSmwzdjRiUUswOUZrUWd2bzlReDAxbk5SeVFLVVppaEdFdW1kVnF0
dUJLTlBxWkkxVHpDUWV3Nm44b0ZNak5oQjhDMFhNSmxrRE9RPT0ifV0sInRpbWVf
dmFsIjpbXX1dLCJleHQiOnsibmFtZTIiOiJ2YWwyIiwibmFtZTEiOiJ2YWwxIn0s
InZlciI6IjEuMCIsInByb2ZpbGUiOiJQREYiLCJoYXNoX2FsZ28iOiJodHRwOlwv
XC93d3cudzMub3JnXC8yMDAxXC8wNFwveG1sZW5jI3NoYTUxMiJ9fQ.DhrCRxT_U
8LeqK1BU9-5Bqui2cs5n21PrSqPnDtVa7mxUtqTnouOXjVfuWR0lfNAjEkc1y2QS
X5x2dmMdCpNLWX127UHYiAm8NzeYuoWqdnxKiy61hZ1l0Jldnk52ngG_2UNDnrCG
Bo9OgC90kG2bFQimZB3WgVtE7ad_HAwIXwd-vEHt6Sf2yWXlUTYqQ1Dxgq6pTKQn
uf5ahsHVyeDihgNeix8-cGx1MEvvHNUpCcIXBx67BEcZ-SrqRoIZkVqEcW83KFMg
qKWmWDgp4z_CKM5ix2dVzwp1GvYOM6M3QUKYgmiNA6dMWJvXeJZ-KKi5A-6gEqfg
OsixuZechcDon_3nMzEeNBSJFXU7ohkvxIJN9LXNFAxzAT2UmASxrL9wvaQMmJHY
Meet-vUsOPWcsq07eKO5bnsYwrs9igYeotgcT_Nl74Rmf9uMye_IgyzlS_NLL4xq
9Aaf6LPXWZ0S_plugvfzv7HuzXNOY994voq8sOpO9xKYhqYnzbdDFKU
```


Decoded JWT Header:


```
{
  "kid" : "OenI+434JhbvfDntfV\/8rOxG7FkvyjaKVJaVqIFBXohVhAe5fK8an
           ov1S688r7Kbal+fvpaH1j8ibg52QBy1PQ==",
  "typ" : "JWT",
  "alg" : "RS512"
}
```


Decoded JWT Claims:

```
{
  "aud" : "http://example.com/audience1",
  "iss" : "https://swedenconnect.se/validator",
  "iat" : 1582730645,
  "jti" : "e22c5be6dd6cc6db834bccd066f5e2e3",
  "sig_val_claims" : {
    "sig" : [ {
      "ext" : null,
      "sig_val" : [ {
        "msg" : "Invalid signature",
        "ext" : null,
        "res" : "FAILED",
        "pol" : "http://id.swedenconnect.se/svt/sigval-policy/
                 chain/01"
      } ],
      "sig_ref" : {
        "sig_hash" : "BhuE9BCYdqLgyowl2PbmnK9dJAmiVtT1u9VgiF99h2h
                      VPzE4XLWvbCPe4aCJ3IzFfoL9k3tWr0W+wy9BeqircQ
                      ==",
        "id" : null,
        "sb_hash" : "bueq5HTO1btpCrXRX7TzEKUrNJQhGG8qBh4wxESqRL3B
                     Bzn4clvK37jYu0KjMMkgJQEMfA1b3imiy79t7h+Yh8w=
                     ="
      },
      "signer_cert_ref" : {
        "ref" : [ "NSuFM/vJ+beBlQtQTzmcYh5x7L8WC9E1KPHRA1ioNOlKVG
                   bla9URzYcsisAx2bcsqOhkvVTc3mK9E6ag07hfaw==" ],
        "type" : "chain_hash"
      },
      "sig_data_ref" : [ {
        "ref" : "0 122935 127937 27430",
        "hash" : "kuUb86FsMNmJl3v4bQK09FkQgvo9Qx01nNRyQKUZihGEumd
                  VqtuBKNPqZI1TzCQew6n8oFMjNhB8C0XMJlkDOQ=="
      } ],
      "time_val" : [ ]
    } ],
    "ext" : {
      "name2" : "val2",
      "name1" : "val1"
    },
    "ver" : "1.0",
    "profile" : "PDF",
    "hash_algo" : "http://www.w3.org/2001/04/xmlenc#sha512"
  }
}
```


Authors' Addresses

Stefan Santesson
3xA Security AB
Forskningsbyn Ideon
SE-223 70 Lund
Sweden

Email: sts@aaa-sec.com

Russ Housley
Vigil Security, LLC
516 Dranesville Road
Herndon, VA, 20170
United States of America

Email: housley@vigilsec.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: 24 April 2021

S. Santesson
IDsec Solutions
R. Housley
Vigil Security

21 October 2020

Signature Validation Token
draft-santesson-svt-pdf-00

Abstract

This document defines a PDF profile for the Signature Validation Token defined in [SVT].

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 24 April 2021.

Copyright Notice

Table of Contents

1. Introduction

The "Signature Validation Token" specification [SVT] defines a basic token to support signature validation in a way that can significantly extend the lifetime of a signature.

This specification defines a profile for implementing SVT with a signed PDF document, and defines the following aspects of SVT usage:

* How to include reference data related to PDF signatures and PDF documents in an SVT.

* How to add an SVT token to a PDF document.

PDF document signatures are added as incremental updates to the signed PDF document and signs all data of the PDF document up until the current signature. When more than one signature is added to a PDF document the previous signature is signed by the next signature and can not be updated with additional data after this event.

To minimize the impact on PDF documents with multiple signatures and to stay backwards compatible with PDF software that do not understand SVT, PDF documents add one SVT token for all signatures of the PDF as an extension to a document timestamp added to the signed PDF as an incremental update. This SVT covers all signatures of the signed SVT.

2. Definitions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

The definitions in [SVT] apply also to this document.

3. SVT in PDF Documents

An SVT added to a signed PDF document SHALL be added to a document timestamp accordance with ISO 32000-2:2017 [ISOPDF2].

The document timestamp contains an [RFC3161] timestamp token (TSTInfo) in EncapsulatedContentInfo of the CMS signature. The SVT SHALL be added to the timestamp token (TSTInfo) as an Extension object as defined in Section 3.1.

3.1. SVT Extension to Timestamp Tokens

The SVT extension is an Extension suitable to be included in TSTInfo as defined by [RFC3161].

The SVT extension is identified by the Object Identifier (OID) 1.2.752.201.5.2

Editors note: This is the current used OID. Consider assigning an IETF extension OID.

This extension data (OCTET STRING) holds the bytes of SVT JWT, represented as a UTF-8 encoded string.

This extension SHALL NOT be marked critical.

Note: Extensions in timestamp tokens according to [RFC3161] are imported from the definition of the X.509 certificate extensions defined in [RFC5280].

4. SVT Claims

4.1. Signature Reference Data

The SVT SHALL contain a SigReference claims object that SHALL contain the following data:

* id - Absent or a Null value.

* sig_hash - The hash over the signature value bytes.

* sb_hash - The hash over the DER encoded SignedAttributes in SignerInfo.

4.2. Signed Data Reference Data

An SVT according to this profile SHALL contain exactly one instance of the *SignedData* claims object. The *SignedData* claims object shall contain the following data:

* ref - The string representation of the ByteRange value of the PDF signature dictionary of the target signature. This is a sequence of integers separated by space where each integer pair specifies the start index and length of a byte range.

* hash - The hash of all bytes identified by the ByteRange value. This is the concatenation of all byte ranges identified by the ByteRange value.

4.3. Signer Certificate References

The SVT SHALL contain a CertReference claims object. The type claim of the CertReference claims object SHALL be either chain or chain_hash`.

* The chain type SHALL be used when signature validation was performed using one or more certificates where some or all of the certificates in the chain are not present in the target signature.

* The chain_hash type SHALL be used when signature validation was performed using one or more certificates where all of the certificates are present in the target signature.

Note: The referenced signer certificate MUST match any certificates referenced using ESSCertID or ESSCertIDv2 from [RFC5035].

5. JOSE Header

5.1. SVT Signing Key Reference

The SVT JOSE header must contain one of the following header parameters in accordance with [RFC7515], for storing a reference to the public key used to verify the signature on the SVT:

* x5c - Holds an X.509 certificate [RFC5280] or a chain of certificates. The certificate holding the public key that verifies the signature on the SVT MUST be the first certificate in the chain.

* kid - A key identifier holding the Base64 encoded hash value of the certificate that can verify the signature on the SVT. The hash algorithm MUST be the same hash algorithm used when signing the SVT as specified by the "alg" header parameter. The referenced certificate SHOULD be the same certificate that was used to sign the document timestamp that contains the SVT.

6. Normative References

[ISOPDF2] ISO, "Document management -- Portable document format -- Part 2: PDF 2.0", ISO 32000-2, July 2017.

[PADES] ETSI, "Electronic Signatures and Infrastructures (ESI); PAdES digital signatures; Part 1: Building blocks and PAdES baseline signatures", ETSI EN 319 142-1 v1.1.1, April 2016.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC3161] Adams, C., Cain, P., Pinkas, D., and R. Zuccherato, "Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP)", RFC 3161, DOI 10.17487/RFC3161, August 2001, <https://www.rfc-editor.org/info/rfc3161>.

[RFC5035] Schaad, J., "Enhanced Security Services (ESS) Update: Adding CertID Algorithm Agility", RFC 5035, DOI 10.17487/RFC5035, August 2007, <https://www.rfc-editor.org/info/rfc5035>.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <https://www.rfc-editor.org/info/rfc5280>.

[RFC7515] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", RFC 7515, DOI 10.17487/RFC7515, May 2015, <https://www.rfc-editor.org/info/rfc7515>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[SVT] Santesson, S. and R. Housley, "Signature Validation Token", IETF draft-santesson-svt-00, October 2020.

Authors' Addresses

Stefan Santesson
IDsec Solutions AB
Forskningsbyn Ideon
SE-223 70 Lund
Sweden

Email: sts@aaa-sec.com

Russ Housley
Vigil Security, LLC
516 Dranesville Road
Herndon, VA, 20170
United States of America

Email: housley@vigilsec.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: 24 April 2021

S. Santesson
IDsec Solutions
R. Housley
Vigil Security

21 October 2020

Signature Validation Token
draft-santesson-svt-xml-00

Abstract

This document defines a XML profile for the Signature Validation Token defined in [SVT].

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 24 April 2021.

Copyright Notice

Table of Contents

1. Introduction

The "Signature Validation Token" specification [SVT] defines the basic token to support signature validation in a way that can significantly extend the lifetime of a signature.

This specification defines a profile for implementing SVT with a signed XML document, and defines the following aspects of SVT usage:

* How to include reference data related to XML signatures and XML documents in an SVT.

* How to add an SVT token to a XML signature.

XML documents can have any number of signature elements, signing an arbitrary number of fragments of XML documents. The actual signature element may be included in the signed XML document (enveloped), include the signed data (enveloping) or may be separate from the signed content (detached).

To provide a generic solution for any type of XML signature an SVT is added to each XML signature element within the XML signature <ds:Object> element.

2. Definitions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

The definitions in [SVT] apply also to this document.

2.1. Notation

2.1.1. References to XML Elements from XML Schemas

When referring to elements from the W3C XML Signature namespace (http://www.w3.org/2000/09/xmldsig#) the following syntax is used:

* <ds:Signature>

When referring to elements from the ETSI XAdES XML Signature namespace (http://uri.etsi.org/01903/v1.3.2#) the following syntax is used:

* <xades:CertDigest>

When referring to elements defined in this specification (http://id.swedenconnect.se/svt/1.0/sig-prop/ns) the following syntax is used:

* <svt:Element>

3. SVT in XML Documents

When SVT is provided for XML signatures then one SVT SHALL be provided for each XML signature.

An SVT embedded within the XML signature element SHALL be placed in a <svt:SignatureValidationToken> element as defined in Section 3.1.

3.1. SignatureValidationToken Signature Property

The <svt:SignatureValidationToken> element SHALL be placed in a <ds:SignatureProperty> element in accordance with [XMLDSIG11]. The <ds:SignatureProperty> element SHALL be placed inside a <ds:SignatureProperties> element inside a <ds:Object> element inside a <ds:Signature> element.

Note: [XMLDSIG11] requires the Target attribute to be present in <ds:SignatureProperty>, referencing the signature targeted by this signature property. If an SVT is added to a signature that do not have an Id attribute, implementations SHOULD add an Id attribute to the <ds:Signature> element and reference that Id in the Target attribute. This Id attribute and Target attribute value matching is required by the [XMLDSIG11] standard, but it is redundant in the context of SVT validation as the SVT already contains information that uniquely identifies the target signature. Validation applications SHOULD not reject an SVT token because of Id and Target attribute mismatch, and MUST rely on matching against signature using signed information in the SVT itself.

The <svt:SignatureValidationToken> element is defined by the following XML Schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified"
    targetNamespace="http://id.swedenconnect.se/svt/1.0/sig-prop/ns"
    xmlns:svt="http://id.swedenconnect.se/svt/1.0/sig-prop/ns">

  <xs:element name="SignatureValidationToken"
      type="svt:SignatureValidationTokenType" />

  <xs:complexType name="SignatureValidationTokenType">
    <xs:simpleContent>
      <xs:extension base="xs:string">
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>

</xs:schema>
```

The SVT token SHALL be included as a string representation of the SVT JWT. Note that this is the string representation of the JWT without further encoding. The SVT MUST NOT be represented by the Base64 encoded bytes of the JWT string.

Example:

```
<ds:Signature Id="MySignatureId">
  ...
  <ds:Object>
    <ds:SignatureProperties>
      <ds:SignatureProperty Target="#MySignatureId">
        <svt:SignatureValidationToken>
              eyJ0eXAiOiJKV1QiLCJhb...2aNZ
        </svt:SignatureValidationToken>
      </ds:SignatureProperty>
    </ds:SignatureProperties>
  </ds:Object>
</ds:Signature>
```

3.2. Multiple SVT in a signature

If a new SVT is stored in a signature which already contains a previously issued SVT, implementations can choose to either replace the existing SVT or to store the new SVT in addition to the existing SVT.

If the new SVT is stored in addition to the old SVT, it SHOULD be stored in a new <ds:SignatureProperty> element inside the existing <ds:SignatureProperties> element where the old SVT is located.

For interoperability robustness, signature validation applications MUST be able to handle signatures where the new SVT is located in a new <ds:Object> element.

4. SVT Claims

4.1. Signature Reference Data

The SVT SHALL contain a SigReference claims object that SHALL contain the following data:

* id - The Id-attribute of the XML signature, if present.

* sig_hash - The hash over the signature value bytes.

* sb_hash - The hash over the canonicalized <ds:SignedInfo> element (the bytes the XML signature algorithm has signed to generated the signature value).

4.2. Signed Data Reference Data

An SVT according to this profile SHALL contain one instance of the SignedData claims object for each <ds:Reference> element in the <ds:SignedInfo> element. The SignedData claims object shall contain the following data:

* ref - The value of the URI attribute of the corresponding <ds:Reference> element.

* hash - The hash of all bytes identified corresponding <ds:Reference> element after applying all identified canonicalization and transformation algorithms. These are the same bytes that is hashed by the hash value in the <ds:DigestValue> element inside the <ds:Reference> element.

4.3. Signer Certificate References

The SVT SHALL contain a CertReference claims object. The type claim of the CertReference claims object SHALL be either chain or chain_hash‘.

* The chain type SHALL be used when signature validation was performed using one or more certificates where some or all of the certificates in the chain are not present in the target signature.

* The chain_hash type SHALL be used when signature validation was performed using one or more certificates where all of the certificates are present in the target signature.

5. JOSE Header

5.1. SVT Signing Key Reference

The SVT JOSE header must contain one of the following header parameters in accordance with [RFC7515], for storing a reference to the public key used to verify the signature on the SVT:

* x5c - Holds an X.509 certificate [RFC5280] or a chain of certificates. The certificate holding the public key that verifies the signature on the SVT MUST be the first certificate in the chain.

* kid - A key identifier holding the Base64 encoded hash value of the certificate that can verify the signature on the SVT. The hash algorithm MUST be the same hash algorithm used when signing the SVT as specified by the "alg" header parameter.

6. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <https://www.rfc-editor.org/info/rfc5280>.

[RFC7515] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", RFC 7515, DOI 10.17487/RFC7515, May 2015, <https://www.rfc-editor.org/info/rfc7515>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[SVT] Santesson, S. and R. Housley, "Signature Validation Token", IETF draft-santesson-svt-00, October 2020.

[XMLDSIG11] Eastlake, D., Reagle, J., Solo, D., Hirsch, F., Nystrom, M., Roessler, T., and K. Yiu, "XML Signature Syntax and Processing Version 1.1", W3C Proposed Recommendation, 11 April 2013.

Authors' Addresses

Stefan Santesson
IDsec Solutions AB
Forskningsbyn Ideon
SE-223 70 Lund
Sweden

Email: sts@aaa-sec.com

Russ Housley
Vigil Security, LLC
516 Dranesville Road
Herndon, VA, 20170
United States of America

Email: housley@vigilsec.com