Disposition of Public Review Comments on eDelivery AS4 1.15

22 September 2020

Introduction

In November 2018, the eDelivery team requested public review comments on its draft v1.15 of its eDelivery AS4 profile. This draft made two changes:

- A small change on the way certificates are carried in the WS-Security header
- A more substantive addition of a new feature for exchanging large messages using the ebMS3 Part 2 “Split & Join” protocol. This Split & Join protocol is not required in the Common Profile but instead is a new Profile Enhancement. This means users that do not need it (most likely the majority of current users) can ignore it. Implementations of eDelivery AS4 are also not required to implement it.

There were no comments on the first change, but several comments to the second one, which can be grouped in five categories. This document reviews these comments.

Two additional changes were made subsequently as explained further in separate sections below.

- In June 2020, a third minor change was made related to also allow TLS 1.3 (in addition to just TLS 1.2).
- In September 2020 some editorial errors were found and fixed.

Compression algorithm used

The initial draft v1.15 proposed to use Brotli, a superior recent compression algorithm, as part of the Split & Join protocol. While Brotli is well supported and easy to use in programming languages like C, Python and .NET, feedback from the community and experience in an internal proof-of-concept indicates that it is difficult to use this algorithm in Java.

Conclusion

Since many known AS4 implementations are written in Java, the CEF team updated the draft to use the GZIP algorithm. It is expected to not be significantly inferior in practice than Brotli.

Application-level alternative to Split & Join

A comment was made that the concept of splitting a large message into fragments, sending the fragments separately and then re-assembling them on the receiving side could be done at an application level (e.g. in a “Connector”) rather than the messaging software. An advantage would be that existing AS4 messaging software could be used without changes. The functionality would need to be implemented at the application layer instead.

In response to this suggestion, the following comments can be made:

No specification

There is no specification other than ebMS3 Part 2 Split and Join that describes a similar feature at application layer (connector). It is not possible to discuss pros and cons of two approaches if only one of them is properly specified.

It can be speculated what an application-level protocol could look like. When the OASIS TC designed Split & Join, the obvious first idea was to use only the ebMS3 header (i.e. not introducing a separate *MessageFragment* header), but when thinking this over, it became clear that this had a number of drawbacks and this was the reason that Split & Join as specified took the different approach that it did.

At first thought, the elements *GroupId*, *FragmentCount* and *FragmentNum* that in Split & Join are in a separate *MessageFragment* header container could become message properties. Using these properties, one can identify a set of AS4 messages as actually being related fragments that together are to be processed as a group. The more or less exact same administration is needed on the receiving MSH side to iteratively collect (and possibly re-order) the inbound fragments until they are all there. This would suggest the complexity could be similar.

However, there are some complications when we consider payloads and payload information. Assume a source message that has a sequence of six payloads. The size of the first is 1.6 GB, then there are three payloads of a few megabytes, then there is a 1 GB payload and then a final 2 GB payload. Now assume the maximum fragment size is 1 GB.

With Split & Join, this is simply a large source message with a *PayloadInfo* that has six *PartInfo* elements, before splitting and after joining. The relative order of the six payloads is encoded in the relative order of the *PartInfo* elements in *PayloadInfo*. The fragments don't need to bother about *PayloadInfo* at all, the source message is just a (very long) string that can be cut in an arbitrary number of substrings of arbitrary length. So the first fragment could have (approximately) 1 GB of the 1.6 GB of the first payload, the second fragment could have the remaining 0.6 GB of the first payload, the three small payloads and about 0.4 GB of the fifth payload, the third fragment could have the remaining 0.6 GB of the fifth payload and the first 0.4 of the sixth payload, the fourth fragment could be a 1 GB middle part of the sixth payload, and then the fifth fragment could have the last 0.6 GB of that last payload.

Now imagine attempting to do this with just regular AS4 messages. The three smaller payloads are contained fully in the second fragment, but that fragment also has the second half of the first payload and the first half of the fifth payload. So a first additional part property (e.g. the payload MIME content ID) is needed to correlate an underlying (source message level) “real” payload to one or more real or “pseudo” payloads in fragment messages. In case there are multiple such (pseudo) payloads, a second additional part property (e.g. payloadfragmentnumber) is needed to encode the order of the various payload content substrings. In effect, this approach becomes a kind of application-level split-and-join, not at the top-level of SOAP-with-attachment messages, but at the level of individual ebMS3 message parts.

But that is not all: there is no source message, so no *PayloadInfo* for the source message. So the receiving MSH needs to infer how many payloads there are, and what their relative order is, as it processes the fragments. (In XML, order of elements is significant and nowhere in ebMS3 or AS4 is it stated that the order of *PayloadPart* in *PayloadInfo* is not significant). This may require an additional administration, an additional (Manifest-like) payload and/or maybe yet another payload property.

In summary, trying to send large ebMS3 messages, that may include multiple ordered payloads of different types and sizes, as a series of ebMS3 messages using just the ebMS3 mechanisms and application-level splitting and joining is likely to be messy and in no way less complex than Split & Join in the MSH as specified in ebMS3 Part 2. It only avoids the two-stage processing of Split & Join (where the source message is built before it is fragmented), but CEF internal prototyping indicates that that is the easy part of a Split & Join implementation. Split & Join uses the serialized ebMS3 message format to encode the source message, including all headers and all parts and their headers and content. Every ebMS3 MSH knows how to serialize and de-serialize the ebMS3 SOAP-with-attachments message format.

Split & Join in the MSH allows reuse of existing ebMS3 signals

The Split & Join protocol uses ebMS3 Error signals to report errors in the re-assembly of the source message from its fragments. Similarly, it uses the ebMS3 Receipt to report successful re-assembly to the message.

If Split & Join functionality is not part of the MSH, but done in some application, then new proprietary application level user message types need to be defined, generated on one side and processed on the other to report on errors and on successful reassembly.

Reusability

When Split & Join was defined, no other Web Services specification provided a messaging layer protocol for sending and receiving large messages as separate messages. The way Split & Join is specified in ebMS3 Part 2 decouples Split & Join from ebMS3/AS4, thus potentially allowing the feature to be used by other Web Services users. This is the way Web Services (in which ebMS3 was positioned) are expected to be architected: multiple, independent specifications than can be used in combination.

Composability with ebMS3 Part 2 Bundling Features

Another reason to use Split & Join as an MSH feature, rather than as an application feature, is to be able to use it in conjunction with ebMS3 Bundling, yet another advanced feature. The Bundling feature allows an MSH to send multiple (separately submitted) user messages as a single ebMS3 Message. Section 4.4.2 of ebMS3 Part 2 explains why it can make sense to use bundling and splitting/joining in combination. Since the Bundling feature is an MSH feature and since Bundling, in the ebMS3 pipeline, is applied before splitting and joining, splitting and joining has to be an MSH feature (rather than an application feature) to use them in combination. Note: eDelivery AS4 does not currently use ebMS3 Bundling. But maybe a future version will, and then it is best to be well-prepared for it.

Full message compression

Assuming an alternative specification for application-level splitting and joining using AS4 payload compression would exist, it would be suboptimal compared to the Split & Join compression feature. Split and Join compression of a complete message in general results in superior compression ratio compared to compression of individual payloads only. If there is no concept of a source message that can be compressed separately from payload parts, this benefit is lost.

Compression works by have recognizing common patterns, replacing occurrences of those patterns by (shorter) references and creating an index of references to patterns. The data can be decompressed by replacing the shorter references by the longer strings. So if there are common patterns among different payloads (for example, an AS4 message that has multiple invoices in an XML format based on the same schema), then compressing all of them as defined in Split/Join results in a much better overall compression ratio than if they are compressed individually. This was tested with UBL, GS1 and OAG sample XML payloads and the benefits are very clear.

Furthermore, compression of the entire message also compresses the AS4 SOAP/MIME headers. In eDelivery we never have more than on *UserMessage* but ebMS3 Part 2 also defines a bundling feature (where you can send multiple *UserMessage*s, each with their separate payloads) that can be used in conjunction with Split & Join, and then the SOAP header can be compressed a lot.

Of course, there is not much, if any, benefit if all payloads are already compressed, are random data, don't have any shared patterns, or if there is only one payload, but that is also true for separate compression. More generally, compression of individual parts in practice does not have a better compression ratio than compression of the full message.

Standardized

Split & Join was developed as part of ebMS3 Part 2, Advanced Features specification in the OASIS ebXML Messaging Services TC. This specification was approved by the members of the TC as a Committee Specification, which in OASIS is a final material level.

There is no specification for splitting and joining at application layer, let alone a standardized one. Also, it is difficult for CEF to argue for a proprietary application-level feature if a standardized equivalent exists, as CEF policy is to base its specifications on open standards where possible.

Generic functionality

Splitting and Joining is generic messaging functionality. There is no generic “connector”, as the various domains have their own domain-specific requirements (e.g. to process certain document types, obtain configuration parameters etc.).

Furthermore, as a connector is a bridge between the outside (B2B) world and the inside (EAI/ESB) world, different organizations (even in the same domain) need different implementations.

If the Split & Join feature were to be implemented at application/connector layer, then many more organizations would need to implement it than there are eDelivery AS4 implementations.

For example, in the e-Justice domain, most known deployments use only a very small number of AS4 implementations, so if those vendors support Split & Join in their ebMS3 product, all users benefit. They can use it more or less as a black-box feature. By contrast, there are many forks and versions of the e-

CODEX connector in use among e-Justice users, so requiring support for Split & Join at connector layer means a big delay in deployment and a massive increase in costs.

Conclusion

In the opinion of the eDelivery team, an alternative Split & Join-like approach at application level has more drawbacks than advantages.

Can eDelivery AS4 still be called AS4 if it includes additional headers?

A related comment was that eDelivery AS4 is called AS4 because it profiles AS4. The comment argues that by adding support for Split & Join, which defines its own additional headers, the specification cannot be called eDelivery AS4 but would probably have to be renamed eDelivery ebMS3. To this the following comments can be made:

Using a separate header is not incompatible with AS4

Ideally, functional enhancements would use only the ebMS3 *eb:Messaging* header defined in ebMS3 Core and would not require additional headers. However, we think that, to support an important new feature, this is acceptable and not incompatible with practice in the OASIS TC.

First of all, ebMS3 Part 2 already introduces additional headers for its multihop feature (the *wsa:To* and *ebint:RoutingInput*) headers, which is already used in AS4. The *MessageFragment* is not an eDelivery specific header, but is defined in the same ebMS3 Part 2 specification. Just as multihop support was added to AS4 because users indicated at the time it was finalized it was functionality they would want to use, Split & Join supports functionality that we now see requests from users for.

After AS4 was published, the OASIS ebXML Messaging Services TC published a SAML Conformance Clause for AS4, which specifies the use of AS4 with SAML tokens. This means that the OASIS TC has itself set a precedent case of an AS4 extension that uses additional headers not specified in ebMS3/AS4 themselves to provide additional features and that is still called AS4. So there is no rule or principle preventing using headers other than the ebMS3 *Messaging* header defined in other OASIS specifications in AS4 and the proposed use of Split & Join just follows that.

Other users of AS4 also reference Split & Join and call the result an AS4 profile

Other users of AS4 also reference Split & Join, and don't seem to have any problem calling the combined use “AS4”.

https://softwaredevelopers.ato.gov.au/sites/default/files/resource-attachments/Schedule_5_Message_Orchestration_and_Profiles_v2.0_Apr2017.pdf

Conclusion

In the opinion of the eDelivery team, the eDelivery AS4 profile can retain its name even after Split & Join is added. In any case, it would be an optional Profile Enhancement, not part of the Common Profile.

Support for Non-Repudiation in Split & Join

In the draft eDelivery 1.15, in the Split & Join protocol only fragments are signed and NRR-acknowledged. There is no separate signature on the source message and no non-repudiation receipt (a receipt that provides content-commitment) for the source message. It was questioned whether Non-Repudiation holds in this set-up.

The CEF eDelivery team contacted a third-party security consultant to review this issue. In his opinion, non-repudiation of signing is obtained by transitivity (or by "assembling") by considering the sum of the non-repudiation of signing on each piece (fragment). In his opinion, there is no security risk.

The consultant does mention a potential additional safeguard, which would be to send, in addition to the fragments, a concluding message containing a manifest that identifies the source message (group) and lists the fragments and their digests. In his opinion this is not necessary, but the manifest could give an additional value/assurance on that.

In reviewing this solution, the CEF eDelivery team noted the following issues:

- One reason for using Split & Join is that it is not a proprietary CEF specification but is based on an open standard. This additional manifest is not present in the ebMS3 Part 2 specification. The XML schema does not provide a structure for this information, and the protocol definition does not specify a Manifest to be exchanged.
- Successful reassembly (joining) has as precondition that all fragments are successfully transmitted. The Manifest does not include any information that is not already present in these signed fragment messages. Sender and receiver party can already create such a Manifest-like document from the data they already have, it does not need to be re-sent. Furthermore, there is the *PayloadInfo* element in the source message which has Manifest-like information for the source message.

As the consulted expert was a security consultant and not a lawyer, additional advice was obtained from a legal expert in DG CNECT. This expert is a lawyer who has been involved in the eIDAS regulation and therefore knows the area of electronic identification and signature and any relevant legal requirements very well. His conclusions is that the split and join functionality is a service offering fulfilling the essential non-repudiation requirements set out in the eIDAS Regulation.

Conclusion

Based on this third-party advice, the CEF eDelivery team proposes not to add the additional *Manifest* exchange and is of the opinion that the non-repudiation support in the Split & Join feature is legally sound.

External payload alternative

Reviewers noted that large payloads can also be referenced, either from the ebMS3 header or from application payloads, and then retrieved using HTTPS.

The following remarks can be made on these comments:

- There is no detailed specification for this external payload feature with ebMS3 or AS4. It is not used by other communities and there are no known implementations. However, since it is a relatively simple feature it would not be very hard to specify. Depending on some design decisions (e.g. on use of WS-Security) it could be more or less complex to implement.

- An advantage would be that it would use HTTP GET, which (unlike POST) supports resumable downloads.
- One unspecified aspect is if the external payloads would be subject to WS-Security processing.
    - If yes, they would be signed and encrypted. Their signature would be verified. Non-Repudiation receipts could be used, just as for non-external payloads. However, processing external payloads is not common in Web Services and likely to require low-level security library adaptations. Currently, CEF and other implementers of AS4 reuse unmodified production-quality security libraries. Having to use a modified security library has the potential of introducing vulnerabilities and therefore normally avoided.
    - If not, exchange of large payloads is less secure than small/normal size payloads. This is inconsistent and likely unacceptable.
- The external payload feature requires a separate Web server component. Payloads must be secured against unauthorized downloaders. For related reasons the external payload feature does not support client-only MSH implementations (that can only pull and push, but cannot be pulled from and do not have a Web server).
- For very large payloads, the feature still requires that very large payload to be downloaded as a single file (possibly with restarts). This can be problematic e.g. when using firewalls that have download limits.
- The feature is not compatible with AS4 multi-hop. Note: eDelivery AS4 does not currently support multi-hop, but it might in the future. EESSI does use it.

Conclusion

The CEF eDelivery team considers external payloads to be a good alternative candidate to Split & Join, and better than other alternatives. However, our opinion is that overall Split & Join still is the best option.

Support for TLS 1.3

When eDelivery AS4 was originally specified, the most recent version of Transport Layer Security (TLS), which eDelivery AS4 uses to secure AS4 message transport, was the 1.2 version and eDelivery AS4 until and including v1.14 states that this version must be supported.

In August 2018, a newer improved version 1.3 of TLS was approved as RFC 8446 by the Internet Engineering Task Force. At the time of writing, this stable 1.3 version has been available for more than two years and deployment of TLS 1.3 by end users is increasing. While TLS 1.2 is still considered secure and can therefore still be used, it makes sense to change the profile to allow users that support TLS 1.3 in their infrastructure to use it. This change has no impact on existing users, who can continue to use TLS 1.2.

Conclusion

The CEF eDelivery team therefore includes this change in this update.

Editorial

While preparing the 1.15 for publication, some typos were corrected. Also, the Dynamic Receiver and Dynamic Sender sections still referred to the Pull enhancement as a potential future addition, whereas it was added previously. The references were adjusted accordingly.