TRUSTED GAT RESULTS FILE FORMAT V1.0

Gaming Standards Association
S2S Technical Committee

Released: 2016/03/14

GAMINGSTANDARDS.COM

Copyright Page

Document Title: Trusted GAT Results File Format V1.0

Release Date: 2016/03/14, by Gaming Standards Association (GSA).

Patents and Intellectual Property

NOTE: The user's attention is called to the possibility that compliance with this [standard/specification] may require use of an invention covered by patent rights. By publication of this [standard/specification], GSA takes no position with respect to the validity of any such patent rights or their impact on this [standard/specification]. Similarly, GSA takes no position with respect to the terms or conditions under which such rights may be made available from the holder of any such rights. Contact GSA for further information.

Trademarks and Copyright

GSA Contact Information

E-mail: sec@gamingstandards.com

WWW: http://www.gamingstandards.com

Table of Contents

I About This Document

This document describes the Trusted GAT Results File Format, which is designed to allow expected GAT results to be distributed in a common and consistent way.

I.I Acknowledgements

The Gaming Standards Association would like to express its appreciation to all members of the S2S committee, past and present, for their significant contribution and dedication to the creation of this standard.

I.II Document Conventions

I.II.I Indicating Requirements, Recommendations, and Options

Terms and phrases in this document that indicate requirements, recommendations, and options are used as defined in the IETF RFC 2119.

In summary:

Requirements: To indicate requirements, this document uses "MUST", "MUST NOT", "REQUIRED".

Recommendations: To indicate recommendations, this document uses "SHOULD", "SHOULD NOT", "RECOMMENDED".

Options: To indicate options, this document uses "MAY" or "OPTIONAL".

I.II.II Other Formatting Conventions

- Blue text indicates an internal link or external hyperlink to a URL.
- **Bold** (other than in headings) or underlined text is used for emphasis, unless specifically indicated otherwise.
- `Courier New font` is used to indicate XML components and their values, and other types of code or pseudo code.

I.II.III Abbreviations and Labels Used in Tables

Element, attribute, and data type descriptions in this document appear in three-column tables. The following abbreviations and labels appear in the Restrictions columns.

Table I.1 Abbreviations and Labels

Abbreviation/Label	Description
default: <empty>	A documentation convention used to specify a string value with a length of 0 (zero) characters.
default: <null>	A documentation convention used to specify that no default value exists for an optional attribute. This implies that no default value will exist in the schema and, therefore, the optional attribute has no default value assigned to it by an end-point processing a message. For example, this convention is used throughout the documentation when a date/time attribute is optional.
length	Total number of characters required.

Table I.1 Abbreviations and Labels

Abbreviation/Label	Description
minIncl/maxIncl	Minimum and maximum inclusive values, for example, 1 through 99.
minLen/maxLen	Minimum and maximum number of characters allowed.
minOcc/maxOcc	Minimum and maximum number of instances allowed.
type	Data type of the element or attribute being described.
∞	Unbounded.

I.III Categorization of Standards

To help provide guidance to implementers regarding the maturity and stability of its standards, GSA categorizes its standards as Candidate Standards, Proposed Standards, or Recommended Standards. This specification is categorized as a Candidate Standard.

- Standards identified as Candidate Standards are the least mature; changes to these standards should be expected in future releases.
- Standards identified as Proposed Standards have been reduced to practice and deployed; very few changes to these standards should be expected.
- Standards identified as Recommended are the most mature and have been widely deployed; no changes to these standards should be expected.

Further details about the categorization of standards and extensions can be found in the GSA Policy Handbook.

Chapter 1

Trusted GAT Results

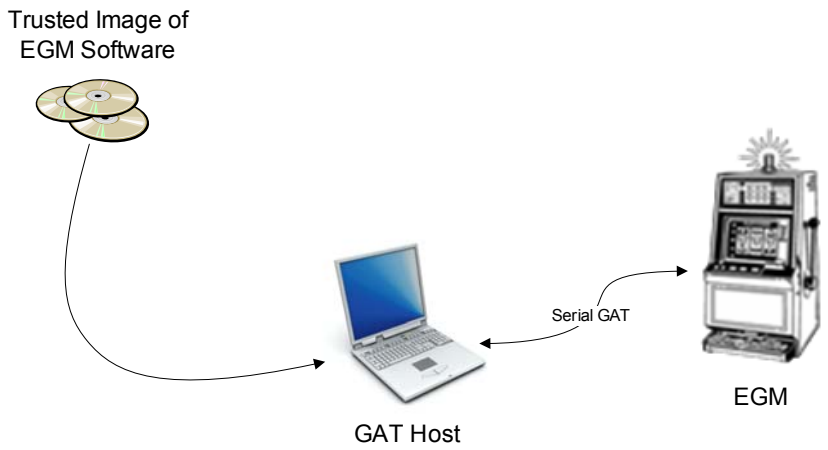
Overview

1.1 Introduction

This document contains the specifications for a standard file format that can be used to convey a list of trusted GAT (Game Authentication Terminal) results to a GAT host. The GAT results correspond to specific components that are exposed by an EGM, or other type of end-point, to a GAT host. The schema for the file format accompanies this document.

A problem that both regulators and systems manufacturers face today is how to trust GAT results returned from an EGM, or other end-point. Without an alternative mechanism, results returned over GAT can only be validated by comparing the value returned by the end-point to a value calculated by the GAT host against a trusted version of the software being challenged. This forces the software performing GAT, which may be run on a download system or regulator's laptop, to understand how all manufacturer software is built so that it can calculate a trusted hash value for the software. This can be difficult due to technology diversity in the industry and changes to software packaging over time. One example is illustrated in the following diagram:

Figure 1.1 GAT Example



This specification helps to resolve this issue by allowing the software manufacturer, regulator, or test lab to publish a file containing a set of trusted GAT values for the regulated software in a secure manner. This allows a GAT host to simply compare the value returned by the end-point being challenged to the values stored in the trusted file. If the value returned by the device being challenged is not in the trusted file, the GAT host can consider the GAT result invalid and take appropriate action. On the other hand, if the result returned matches a value in the trusted file, the GAT host can inform the operator running the system that the GAT comparison was successful.

1.1.1 Product

This specification relies upon the concept of components and products. The product concept was first introduced in the GSA Manifest Package File Format specification. It describes a typical "product" that an operator purchases from a manufacturer. Typically, products are downloaded using G2S or S2S and installed on an EGM or other end-point. Examples of products include game content, such as Win Like a Wild Man, peripheral firmware updates, and even operating system updates. See the GSA Package Manifest File Format for the requirements for constructing product identifiers.

1.1.2 Components

Products contain "components" which can be directly challenged using GAT. A component can be firmware for a physical device or a software module on an end-point – for example, firmware for a ticket printer or bill validator; or, software such as server-side executables, DLLs, or downloadable content for an EGM. See the G2S Message Protocol and the S2S Message Protocol for the requirements for constructing component identifiers.

1.1.3 Security

Trusted GAT files MUST be digitally signed by their author so that recipients of the files can verify that they have not been manipulated. Trusted GAT files MUST be signed as follows:

- Cryptographic Message Syntax (CMS) MUST be used - RFC 5652
- A SignedData ContentType MUST be used, with the Trusted GAT XML as the encapsulated content
- All certificates required to re-create the certificate chain from the signer to the trusted root certificate MUST be included in the CMS message
- At a minimum, SHA-1 with RSA 2048 bit keys MUST be used.
    - An implementation MAY support SHA-256 or SHA-512 with RSA 2048 bit keys or greater. Please see RFC 5754 for more information about use of these algorithms with CMS.
    - An implementation MAY support SHA-256 or SHA-512 with ECDSA. Please see RFC-5753 for more information about use of these algorithms with CMS.

1.1.4 Trusted GAT File Extension

Trusted GAT files MUST be written to storage media with a file extension set to ".gsaTrusted".

1.1.5 Trusted GAT Import

Trusted GAT files are expected to be distributed with the software packages that they verify using the approach specified in Section 1.8 Product/Package Import of the GSA Package Manifest File Format specification.

When distributed with a GSA package manifest file, the trusted GAT file MUST be located in the same folder as the GSA package manifest file.

It is also possible for trusted GAT files to be distributed separately from the software packages that they verify, especially if they are created by a regulator or testing laboratory.

When needed to meet regulatory or other requirements, trusted GAT files may be accessible over a network from a trusted source.

Chapter 2

Trusted GAT Results Schema

2.1 Schema

The highest level element of the trusted GAT file is the `trustedGatResults` element.

The manufacturer, regulator, or test lab is responsible for assigning a unique ID to each trusted GAT file. The unique ID is reported in the `resultSetId` attribute.

The file may also contain version information that can be used by a GAT host to determine which version of the Trusted GAT Results File Format is to be used when parsing the trusted GAT file. The version is reported in the `tgrVersion` attribute.

The `resultSetDateTime` attribute indicates when the trusted GAT file was created. If a trusted GAT file is updated for some reason and delivered to the field with the same `resultSetId`, the GAT host SHOULD only use the trusted GAT file with the most recent `resultSetDateTime`. If trusted GAT files are loaded from multiple sources, such as a manufacturer and test lab, for a given product, the GAT host SHOULD only use the trusted GAT file with the latest `resultSetDateTime`.

The `trustedGatResults` element may contain one or more product sub-elements, each containing a set of trusted verification results for the components that make up the product.

Figure 2.1 trustedGatResults Element

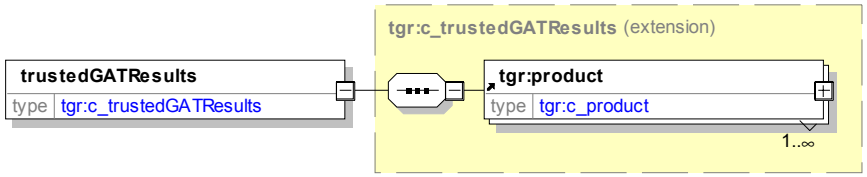


Table 2.1 trustedGatResults Attributes

Attribute	Restrictions	Description
`tgrVersion`	`type: t_tgrVersion` `use:required`	Identifies the version of the Trusted GAT Results File Format. For version 1.0, this value is "1.0".
`resultSetId`	`type: t_uniqueIdentifier64` `use: required`	Identifier that uniquely identifies a particular trusted GAT result file. This string is a manufacturer-specific unique identifier that is controlled by the release process of the manufacturer. Two copies of a trusted GAT results file MUST NOT contain the same `resultSetId` and `resultSetDateTime` unless the information contained in both files is EXACTLY the same. If a new trusted GAT result file is generated by a regulator or testing laboratory to complement or supersede a file produced by the manufacturer, the same `resultSetId` SHOULD be used.

Table 2.1 trustedGatResults Attributes

Attribute	Restrictions	Description
`resultSetDateTime`	`type: t_dateTime` `use: required`	Date/time that the trusted GAT result file was created. When multiple trusted GAT files contain the same `resultSetId`, this attribute can be used to identify the most recent version.

Table 2.2 trustedGatResults Elements

Element	Restrictions	Description
`product`	`minOcc: 1` `maxOcc: ∞`	Contains a set of verification results for the components of a specific product.

2.2 Product

The `product` element identifies a specific product and contains one or more sub-elements that identify components of the product during it the product's lifecycle, such as when the product is downloaded, after it is installed and becomes a module, etc.

Figure 2.2 product Element

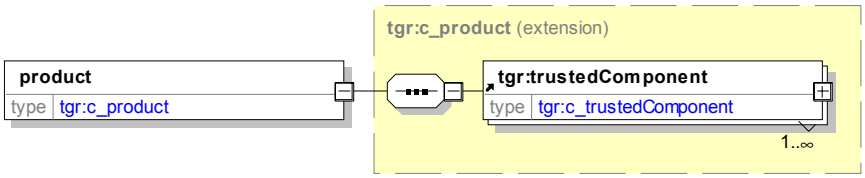


Table 2.3 product Attributes

Attribute	Restrictions	Description
`productType`	`type: t_productTypes` `use: required`	Type of product.
`productId`	`type: t_uniqueIdentifier64` `use: required`	Unique identifier for the product. This value is assigned by the manufacturer.
`mfgCode`	`type: t_mfgCode` `use: required`	GSA -assigned manufacturer code for the manufacturer of the product.
`releaseNum`	`type: t_descriptorText` `use: required`	Release number of the product.
`releaseDateTime`	`type: t_dateTime` `use: required`	Date/time that the product was released by the manufacturer.

Table 2.4 product Sub-Elements

Sub-Element	Restrictions	Description
`trustedComponent`	`minOcc: 1` `maxOcc: ∞`	Identifies a component of a product that can be verified using GAT at some stage in the product's lifecycle.

2.3 trustedComponent Element

The `trustedComponent` element identifies a component of a product that can be verified using GAT at some stage in the product's lifecycle. Each `trustedComponent` element is identified by the `componentId` attribute. There MUST only be one `trustedComponent` element with a given `componentId` value for a specific product. The same `componentId` value may appear in multiple products.

Figure 2.3 trustedComponent Element

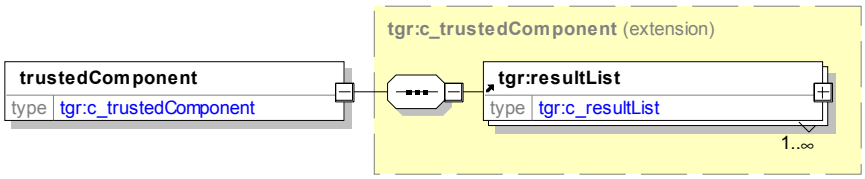


Table 2.5 trustedComponent Attributes

Attribute	Restrictions	Description
`componentId`	`type: t_componentId` `use: required`	Unique identifier for the component. Assigned by the manufacturer.
`componentType`	`type: t_componentTypes` `use: required`	Type of component.

Table 2.6 trustedComponent Sub-Elements

Sub-Element	Restrictions	Description
`resultList`	`minOcc: 1` `maxOcc: ∞`	Contains a list of verification results for the component using the specified algorithm.

2.4 resultList Element

The `resultList` element contains a list of verification results for a component using the algorithm identified in the `algoirthmType` attribute. For a given `algorithmType`, there MUST only be one `resultList` sub-element under the `trustedComponent` element.

Figure 2.4 resultList Element

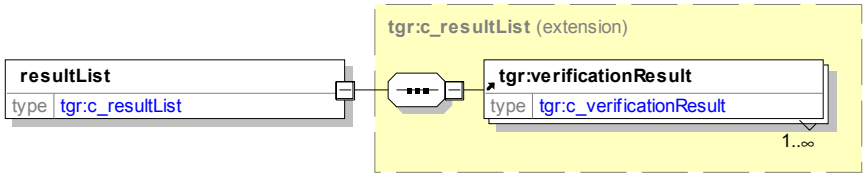


Table 2.7 resultList Attributes

Attribute	Restrictions	Description
`algorithmType`	`type: t_algorithmTypes` `use: required`	GAT verification algorithm.

Table 2.8 resultList Sub-Elements

Sub-Element	Restrictions	Description
`verificationResult`	`minOcc: 1` `maxOcc: ∞`	Contains a verification result for the component using the specified algorithm.

2.5 verificationResult Element

The `verificationResult` element contains one trusted GAT result for the algorithm identified in the `algorithmType` attribute of the `resultList` element using the selected options for that algorithm, such as using seeds, salts, and/or offsets.

If the algorithm supports it, offsets may also be used to help refine the portion of the component verified. The offsets can be used both to identify a subset of the component as well as a starting position from which to wrap around.

Certain algorithms may support a seed value. Each algorithm that supports a non-empty seed will define the data format for the seed. For the G2S_CRC16 and the G2S_CRC32 algorithms, the seed MUST be a positive base-ten integer value in the range supported by the specified algorithm and MUST be represented by a string of decimal characters (U+0030 - U+39). The seed MUST be converted from the UTF-8 string into an unsigned xs:long prior to its use.

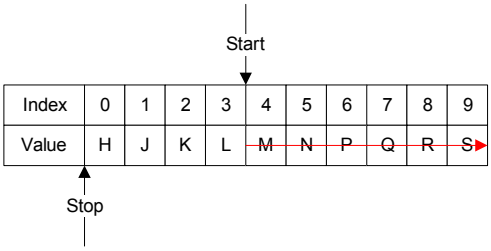
Certain algorithms may support a salt value. The salt value can be used by a host to prepend arbitrary bytes to the component buffer. The salt is hashed before any of the bytes from the component buffer regardless of the offsets.

Using Offsets

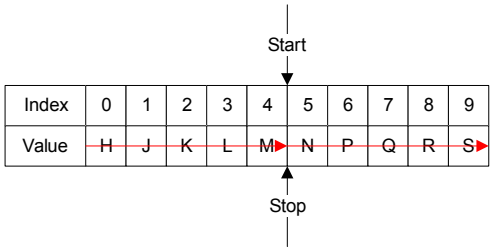
When supported by an algorithm, offsets may be used to refine the portion of the component that should be verified. Offsets can be used to identify a subset of the component that should be verified, as well as a starting position from which the verification algorithm should wrap around.

When zero-based buffer indexing is used by an implementation, the starting offset identifies the first byte to be included in the calculation. The ending offset identifies the byte at which the calculation stops; that byte is not included in the calculation. If the ending offset is less than or equal to the starting offset, the algorithm wraps around. If an offset value is less than zero or greater than the last index value, the value 0 (zero) is used.

For example, to verify the final 6 bytes of the 10-byte buffer containing the ASCII string "HJKLMNPQRS", the starting offset should be set to 4 and the ending offset should be set to 0 (or -1 or 10). The buffer to hash would be "MNPQRS".



To verify the entire 10-byte buffer containing the ASCII string "HJKLMNPQRS" starting in the middle, the starting offset and the ending offset should both be set to 5. The buffer to hash would be "NPQRSHJKLM".



Using Salts

Certain algorithms may support a salt value. When supported by an algorithm, the salt value MUST be prepended to the component buffer. The salt value MUST NOT be padded or otherwise adjusted before it is prepended to the component buffer unless required by the algorithm.

For example, for the 10-byte buffer containing the ASCII string "HJKLMNPQRS", if the client system specifies a 3-byte salt value that is equivalent to the ASCII string "TVW" (no padding or other adjustments required) with a starting offset and an ending offset 4, then the entire 13-byte buffer to hash would be "TVWMNPQRSHJKL".

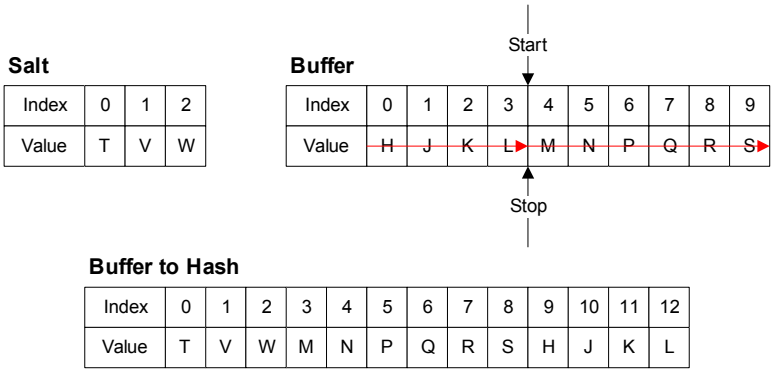


Figure 2.5 verificationResult

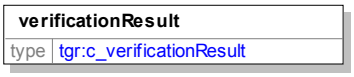


Table 2.9 verificationResult Attributes

Attribute	Restrictions	Description
verifyResult	type: t_verifyResult use: required	The trusted verification result.
seed	type: t_gatSeed use: optional default: <empty>	The seed used when calculating the verification result. Certain algorithms, such as checksums (CRC), use a seed to define the starting value.
salt	type: t_gatSalt use: optional default: <empty>	Arbitrary bytes that were prefixed to the component's byte buffer before the trusted hash result was generated.

Table 2.9 verificationResult Attributes

Attribute	Restrictions	Description
startOffset	type: xs:long use: optional default: 0 minIncl: 0	The starting offset used when calculating the trusted verification result.
endOffset	type: xs:long use: optional default: -1 minIncl: -1	The ending offset used when calculating the trusted verification result.

Chapter 3

Data Types

3.1 Data Types

This section contains the various data types used in this specification.

Table 3.1 Data Types

Data Type	Restrictions	Description
t_algorithmTypes	type: t_extensibleList enumerations: See Section 3.1.1	Extensible list of algorithm types.
t_componentId	type: xs:string minLen: 1 maxLen: 256	Component identifier.
t_componentTypes	type: t_extensibleList enumeration: See Section 3.1.2	Extensible list of component types.
t_dateTime	type: xs:dateTime pattern: \d\d\d\d-\d\d-\d\dT\d\d:\d\d:\d\d(\.\d+)?(([\+\-]\d\d:\d\d)|Z)	Date/time.
t_descriptorText	type: xs:string maxLen: 32	Free-form text.
t_extensibleList	type: t_uniqueIdentifier64	Extensible list.
t_gatSalt	type: xs:base64Binary maxLen: 1024	GAT algorithm salt value.
t_gatSeed	type: xs:string maxLen: 256	GAT algorithm seed value.
t_mfgCode	type: xs:string pattern: [A-Z0-9]{3}	GSA-assigned manufacturer code.
t_productTypes	type: t_extensibleList enumerations: See Section 3.1.3	Extensible list of product types.
t_tgrVersion	type: xs:string minLen: 0 maxLen: 10	String that represents the version of the Trusted GAT Results File Format.
t_uniqueIdentifier64	type: xs:string pattern: [A-Z0-9]{3}_[-~]{1,60}	64-character unique identifier.
t_verifyResult	type: xs:string minLen: 0 maxLen: 256	Verification algorithm result.

3.1.1 t_algorithmTypes Enumerations

Table 3.2 t_algorithmTypes Enumerations*

Enumeration	Description
TGR_CRC16	A 16-bit cyclic redundancy check. Uses a 16-bit unsigned seed.

Table 3.2 t_algorithmTypes Enumerations*

Enumeration	Description
TGR_CRC32	A 32-bit cyclic redundancy check. Uses a 32-bit unsigned seed.
TGR_MD5	MD5 algorithm per RFC 1321 Seed is not supported, use salt.
TGR_SHA1	SHA1 algorithm per FIPS 180-2 Seed is not supported, use salt.
TGR_SHA256	SHA256 algorithm per FIPS 180-2 Seed is not supported, use salt.
TGR_SHA384	SHA384 algorithm per FIPS 180-2 Seed is not supported, use salt.
TGR_SHA512	SHA512 algorithm per FIPS 180-2 Seed is not supported, use salt.
TGR_HMACSHA1	HMAC SHA-1 algorithm per FIPS 180-2 and FIPS 198. The HMAC key MUST be included as if it was a salt; seed and offsets are not supported.

* Note: The algorithms listed in this table are intended to be the same algorithms as specified within the G2S Message Protocol and the S2S Message Protocol for the `t_algorithmTypes` data type. See the G2S Message Protocol and S2S Message Protocol for more details about using the algorithms within those protocols.

3.1.2 t_componentTypes Enumerations

Table 3.3 t_componentTypes Enumerations*

Enumeration	Description
G2S_module	A verification component that maps directly to a G2S download module. When this is defined, then the `componentId` MUST be the same as the `modId` exposed over G2S.
G2S_package	A verification component that maps directly to a G2S download package. When this is defined, then the `componentId` MUST be the same as the `pkgId` exposed over G2S.
G2S_software	A generic software component that is represented by EGMs over G2S.
G2S_OS	A component representing the operating system as exposed by EGMs over G2S.
G2S_hardware	A physical hardware component as exposed by EGMs over G2S.
S2S_software	A generic software component for S2S end-points.
S2S_OS	A component representing the operating system for S2S end-points.
S2S_hardware	A physical hardware component for S2S end-points.
S2S_package	A component that maps directly to a package containing S2S packages (for S2S end-points).
S2S_module	A component that maps directly to an S2S module (for S2S end-points).

* Note: The enumerations listed in this table are intended to be the same enumerations as specified within the G2S Message Protocol and the S2S Message Protocol for the `t_componentTypes` data type. If there are any discrepancies between the lists of enumerations, the enumerations within the G2S Message Protocol MUST take precedence for enumerations that begin with G2S; the enumerations within the S2S Message Protocol MUST take precedence for enumerations that begin with S2S.

3.1.3 t_productTypes Enumerations

Table 3.4 t_productTypes Enumerations*

Enumeration	Description
`G2S_game`	A game product for G2S end-points.
`G2S_OS`	An operating system product for G2S end-points.
`G2S_hardware`	A peripheral firmware product (bill validator), printer, etc.) for G2S endpoints.
`G2S_other`	Other products for G2S end-points.
`S2S_application`	An application product for S2S end-points.
`S2S_OS`	An operating system product for S2S end-points.
`S2S_hardware`	A peripheral firmware product (printer, note acceptor, etc.) for S2S endpoints.
`S2S_other`	Other products for S2S end-points.

* Note: The enumerations listed in this table are intended to be the same enumerations as specified within the GSA Package Manifest File Format for the `t_productType` data type. If there are any discrepancies between the lists of enumerations, the enumerations within the GSA Package Manifest File Format MUST take precedence.

Chapter 4

Example

4.1 Example

```
<tgr:trustedGATResults
    xmlns:tgr="http://www.gamingstandards.com/tgr/schemas/v1.0"
    tgr:tgrVersion="1.0"
    tgr:resultSetId="ABC_123"
    tgr:resultSetDateTime="2015-01-10T12:38:50-05:00"
    >
    <tgr:product
        tgr:productType="G2S_game"
        tgr:productId="ABC_myProduct"
        tgr:mfgCode="ABC"
        tgr:releaseNum="ABC_1.14.3"
        tgr:releaseDateTime="2015-01-01T14:15:29-05:00"
        >
        <tgr:trustedComponent
            tgr:componentId="ABC_myComponent"
            tgr:componentType="G2S_module"
            >
            <tgr:resultList
                tgr:algorithmType="TGR_HMACSHA1"
                >
                <tgr:verificationResult
                    tgr:verifyResult="A1B2C3D4E5F60718293A4B5C6D7E8F90"
                    tgr:salt="FEDCBA0987654321"
                    />
            </tgr:resultList>
        </tgr:trustedComponent>
    </tgr:product>
</tgr:trustedGATResults>
```

END OF DOCUMENT

Released: 2016/03/14