Payment for Services in Session Initiation Protocol (SIP)
draft-jennings-sipping-pay-03.txt

Status of this Memo

By submitting this Internet-Draft, each author represents that any applicable patent or other IPR claims of which he or she is aware have been or will be disclosed, and any of which he or she becomes aware will be disclosed, in accordance with Section 6 of BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt.

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

This Internet-Draft will expire on April 26, 2006.

Copyright Notice

Copyright (C) The Internet Society (2005).

Abstract

Service usage might require some form of compensation and this is also true for many communication systems where an entity receiving a call should be able to charge the caller. This is necessary for
allowing fair communication between two communicating parties and is a major strategy for reducing the viability of SPAM. This draft proposes an approach for doing this in SIP using the Security Assertion Markup Language (SAML). It relies on a third party to act as a payment provider and is designed for low value transactions. It does not aim to provide the same capability as other authentication, authorization and accounting systems.

This draft is in a fairly early state and has many details that are missing. Earlier versions of this document did not use SAML. This version offers a sketch of what the SAML based solution would look like but still lacks many details that would be needed for an actual implementation.
Table of Contents

1. Introduction .............................................. 4
2. Terminology ............................................. 6
3. Requirements, Assumptions and Goals .......................... 7
4. Non-Goals ............................................... 7
5. Protocol ................................................ 8
   5.1. UAC Behavior ........................................ 8
   5.2. UAS Behavior ....................................... 9
   5.3. Computing costs .................................... 10
   5.4. Transition Scenarios ............................... 10
       5.4.1. Merchant Proxy .................................. 11
       5.4.2. Customer Proxy .................................. 13
   5.5. Payment Provider Behavior ......................... 14
   5.6. Account Names ...................................... 15
   5.7. Merchant Fetching Public Key ....................... 15
6. SIP Extensions ........................................... 15
   6.1. Update response code 402 ........................... 16
7. Syntax .................................................. 16
   7.1. Charge ............................................. 16
   7.2. Request for Payment ................................ 17
   7.3. Receipt ........................................... 19
   7.4. Refunds ........................................... 21
   7.5. Verifying the Receipt ............................. 21
8. Usage Scenarios ......................................... 21
   8.1. SPAM ............................................... 21
   8.2. Micro Billing ...................................... 22
9. XML Schemas ............................................. 22
   9.1. Payment Request and Payment Receipt ............... 22
   9.2. Charge ............................................ 24
10. Security Considerations ................................. 26
   10.1. Stolen Assertion ................................... 27
   10.2. MitM Attack ....................................... 27
   10.3. Forged Assertion ................................... 27
   10.4. Replay Attack ..................................... 28
11. IANA Considerations ................................... 28
   11.1. Payment-Receipt Header ............................. 28
   11.2. 402 Response ....................................... 29
   11.3. charge+xml ........................................ 29
12. Open Issues ............................................. 30
13. References .............................................. 30
   13.1. Normative References .............................. 30
   13.2. Informational References ......................... 31
Authors’ Addresses ......................................... 32
Intellectual Property and Copyright Statements ............... 33
1. Introduction

Figure 1 shows protocol exchange that implements payment capabilities using SIP and SAML. It involves three parties: a Customer who is the caller, a Merchant who is the party being called, and a third party to broker the transaction, which is called the Payment Provider. In SAML terminology this entity is called identity provider.

```
<table>
<thead>
<tr>
<th>Payment Provider (P)</th>
<th>Customer (C)</th>
<th>Merchant (M)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) Call</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3) Request for a Payment</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;---------------------</td>
<td>--------------</td>
<td>--------------</td>
</tr>
<tr>
<td>4) SAML Assertion (=Receipt)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------------------</td>
<td>--------------</td>
<td>--------------</td>
</tr>
<tr>
<td>5) Call + Receipt</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6) 200 OK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Figure 1: Communication Overview

The Customer and the Merchant interact with each other using SIP and the Customer uses HTTP to exchange messages with the Payment Provider.

1. Initially, Customer makes a call to Merchant
2. Merchant determines that a payment is required and includes information about the payment in an Charge body of a 402 (Payment Required) response to Customer
3. Customer looks at this Charge and decides to make a payment. The Customer therefore instructs its Payment Provider to make a transfer from the Customer’s account to the Merchant’s account using a request for a SAML assertion with the extensions defined in this document.
4. The Payment Provider returns a receipt for this transfer. This receipt is a SAML Assertion (or a SAML Artifact, if the exchange is triggered by a proxy or if desired by the Customer).
5. Customer resubmits the call to Merchant but this time provides the Receipt for the transaction. Merchant determines whether the Receipt is valid (by checking the digital signature and the content of the assertion) and continues with the call processing, if the authorization was successful.

The Charge contains information about the three parties:
- Payment Providers that are acceptable to the Merchant
- cost(s) of the transaction
- account identifier for the Merchant at each supplied Payment Provider
- cookie to make it easier for the Merchant to avoid replay attacks.

The Customer includes this information when making the Request for Payment to a Payment Provider; adds its own account information and authorization password; and sends this to Payment Provider, which produces a Receipt for the transaction if it is successful. This transfer from Customer to Payment Provider is made across an encrypted, integrity protected channel. The Receipt includes a timestamp when the Payment Provider made the transaction and protects the Receipt with a digital signature.

The Customer resubmits the call to the Merchant with the Receipt from Payment Provider. The Merchant can check for replay attacks using the timestamp and the cookie initially provided with the Charge. The Merchant can then check the signature is valid using Payment Provider’s public key.

Figure 1 does not show the interaction between the Merchant and the Customer if refunding is necessary. Additionally, the interaction between the Merchant and the Payment Provider to reconcile is not shown.

The proposal described in this document does not aim to provide functionality equivalent to AAA protocols. For example, there is no guarantee or recourse if Merchant does not provide the service after Customer transfers money into Merchant’s account. The system is designed for low value transactions in which, if Merchant cheats, Customer can choose to never deal with Merchant again but the value of the transaction is lost. This scheme is designed for systems whereby the communication between Merchant and Customer and the communication between Customer and Payment Provider can be executed in real time. While it is possible to develop schemes that deal with some of these problems, Payment Providers deploying them have not been willing to provide services for transaction fees on the order of one US cent. The authors believe that the simplified scheme presented here will make it easier to reach these low value transactions.
2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [1].

This work adopts the terminology from the framework in RFC 2801 [12].

Additionally, the following terms are used:

Customer:
The entity that is paying for the call, typically the SIP UAC or a proxy in the same administrative domain as the SIP UAC.

Merchant:
The entity wishing to be paid for the call, typically the SIP UAS or a proxy in the same administrative domain as the UAS.

Payment Provider:
The third party that handles the transfer of currency from Customer to Merchant. The Internet Open Trading Protocol (IOTP) [12] refers to this as the Brand.

Charge:
The information sent from the Merchant to the Customer describing what payment is needed.

Request for Payment:
The information sent from the Customer to the Payment Provider describing the transfer of currency needed. This request is implemented as a request for a SAML assertion.

Receipt:
The information sent from the Payment Service Provider to the Customer and passed on to the Merchant. The Receipt is either a SAML assertion or a SAML artifact. The Receipt tells the Customer that the payment transaction was successfully completed (if either a SAML assertion or a SAML artifact is returned). The Merchant receives an assurance that the transaction was successful after receiving the Receipt and verifying it successfully.

Currency:
Could be a classical currency such as the Euro or US Dollar or could be a pseudo currency such as airline mileage points.

Assertion:
An assertion is an XML document that contains authentication statements, attribute statements and authorization decision statements. In an authentication statement, an issuing authority asserts that a certain subject was authenticated by certain means at a certain time. In an attribute statement, an issuing authority asserts that a certain subject is associated with certain attributes which has certain values. In an authorization decision statement, a certain subject with a certain access type to a certain resource has given certain evidence that the identity
is correct. The assertion is digitally signed to protect it against unwanted modifications by third parties.

Artifact:
An artifact is a base-64 encoded string that is 40 bytes long that serves as a pointer to a server that temporarily stores an assertion. The first 20 bytes provide an indication from which server the assertion needs to be retrieved. The remaining 20 bytes identify a unique assertion.

For further terminology related to SAML the reader is referred to [2].

3. Requirements, Assumptions and Goals

This section lists some basic requirements and goals for the protocol presented in this document:

- Provide a system for callers to pay the person they are calling using a 3rd party clearing house. A trust relationship between the Merchant and the Payment Provider and between the Customer and the Payment Provider must exist. The Customer must be able to authenticate itself to the Payment Provider and to instruct the Payment Provider to transfer money from its account to another account (i.e., the account of the Merchant). It is not necessary to for the Customer and the Merchant to have an account at the same Payment Provider.
- The protocol must support multiple currencies and must offer the ability for the Customer to learn the price before initiating a transaction.
- Support various billing models including: flat rate, per unit time, per unit data
- The solution must be cost effective for low cost transactions.
- The solution must allow Customers to remain anonymous towards the Merchants.
- Support for a simple refund mechanism must be provided.

4. Non-Goals

[Editor’s Note: A future version of this document will provide a list of non-goals in addition to goals.]

There needs to be a way for Customers and Merchants to enroll and transfer money in and out from their accounts. The mechanisms to accomplish this functionality are outside the scope of this document. They can be provided by using web forms to enroll, get an account number, and provide the typical credit card mechanism to transfer
money into the account. Transfers out of the account could be done with the typical mechanism for transfers to bank accounts.

5. Protocol

5.1. UAC Behavior

The UAC SHOULD indicate that it can accept the application/charge MIME type in SIP requests it sends.

In the case where the UAC receives a 402 response containing an application/charge body, it MUST check that this Charge is acceptable to the user of the UAC. This could be done using a policy that was previously entered by the user. If the Charge contains a Payment Provider with which the user has an account and the Charge is acceptable, then the UAC sends a Request for Payment to the Payment Provider.

The UAC needs to look at the available Payment Providers, cost, and currency and select an appropriate one. The UAC MUST copy the PaymentServiceProviderData fields from the Charge into the Request for Payment parameters. The UAC must look at the cost elements in the Charge to decide how large a payment the user wishes to make and set the amount and currency parameters appropriately. Finally, it needs to fill the CustomerData parameters. It is critical that the UAC check the certificate of the HTTPS TLS connection as specified in RFC 2818 [3] and RFC 2246 [4].

After selecting and constructing the SAML request message it is sent over the HTTPS secured connection. The detailed format of the request is shown in Section 9.

The response will be an error or a SAML artifact/assertion based on the Request for Payment. The user needs to be informed if an error is received and the transaction SHOULD NOT be retried unchanged. When a valid response is received, the UAC SHOULD resubmit the SIP request that caused the 402 but this time by including the Receipt (i.e., the artifact or the assertion).

The UAC needs to compute the amount of payment it wishes to make by looking at the cost information provided as part of the Charge. The UAC is also responsible for determine when a new payment has to be made and refreshing the call with additional payments before this happens. For example, the UAC could initially decide to provide enough payment for 3 minutes. After 2.5 minutes the UAC might decide to pay for an additional 3 minutes. It would do this by issuing a new Payment Request to the Payment Provider for an additional 3
minutes’ worth of resources and then sending the new receipt with a SIP Re-INVITE or UPDATE transaction to the UAS.

5.2. UAS Behavior

When the UAS receives a request it wishes to charge for, it should check whether the UAC has set the Accept header to include application/charge. If it has, it MAY reject the request with a 402 and attach an application/charge to the response. Note that the application/charge document can be attached on any failure response. For example, this could allow a UAS to combine the Charge with a request for authorization in a 401 response. It needs to include lists of all the Payment Providers that are acceptable to the UAS and include the identity of the Merchant. It also needs to form a list of currencies that are acceptable and what the cost in each one is. The merchant may also specify a minimum cost and/or a maximum cost in the Charge. The costs are described in Section 5.3.

When the UAS receives a request that contains a Receipt (as a SAML assertion) as defined in [5], it MUST verify that the assertion/artifact is valid using the following steps.

1. The UAS needs to make sure that the amount of the payment is appropriate and if this receipt matches a previous Charge to prevent replay attacks.
2. Check that the digital signature of the Receipt is valid. This includes path validation and to check that the Payment Provider’s certificate is still valid.
3. Check that the time between the payment and now is acceptably low. This MUST be a configurable parameter and should default to 30 seconds. The UAS SHOULD support NTP RFC 1305 [13]. [Editor’s Note: I think that this mechanism is not needed.]
4. The UAS MUST check that this receipt has not been previous used. The limited time window limits the amount of state the UAS must keep to make this check. If several UASs are using the same merchant-id at the, this replay detection needs to be done across all the UASs. The ChargeData can be used with opaque encrypted data to help do this. [Editor’s Note: This aspect has to do with the first item.]

If the payment is accepted, it is the Merchant’s responsibility to end the call after the amount paid becomes inadequate to cover the session. The UAS SHOULD use a mechanism such as SIP session timer [14]. The UAC MAY send a Re-INVITE or an UPDATE message with a new receipt for a payment to prolong the session.

There are certain cases where the Merchant may wish to offer a refund. This could occur if the Customer has prepaid for a 10 minute session and the call terminates after 1 minute. In this case, the
Merchant may wish to provide a refund for the 9 minutes that were not used. Alternatively, the Customer could provide a receipt to place a call but the destination is busy. In this case, the Merchant would likely want to provide a full refund.

The refund mechanism is simple and identical to the Payment Request procedure described in Section 5.1. In this case, the Merchant posts a Payment Request to the Payment Provider specified in the Payment Receipt.

If the call ends early or cannot be completed, it may still be possible that the Customer has provided a receipt of payment where no service has been delivered. This may have occurred due to an upstream proxy error or a network connectivity problem between the UAC and the UAS. Since the receipt of payment was never delivered to the UAS, there is no immediate mechanism of delivering a refund to the Customer.

5.3. Computing costs

There are three types of costs:
- initial setup costs,
- costs per second, and
- cost per unit data.

All three costs are added together to form the total cost and are assumed to be zero if not specified. The cost of the first time unit block size worth of time and the first data unit block size of data are considered to be included in the initial connection or setup costs.

If a call costs 30 cents to connect and then 12 cents per minute and is billed in 15 second increments (rounded down), the cost would be set so that the currency was USD and the currency divisor (power of 10) was 1000 making the initial cost 300, the cost per unit time is 40, and the time unit size is 15000. If the time is to be rounded up, then some extra to cover the price of the first increment would be added to the connect cost.

Note that the additional specification of a currency divisor allows all currency amounts to be specified in fixed point. In the above example, a currency divisor of 1000 means that all currency amounts are in tenths of a cent (USD).

5.4. Transition Scenarios

The deployment of the mechanisms described in this document might take place incrementally. This section provides some information
about two likely transition scenarios: Merchant Proxy and Customer Proxy.

5.4.1. Merchant Proxy

In this scenario, the Customer places a call to a Merchant where the Merchant’s UAS does not implement this mechanism. The Merchant’s Proxy acts on behalf of the Merchant. The key difference here is that the Merchant’s proxy must use SAML Artifacts instead of SAML Assertions and an extra step must be taken by the Proxy to validate the Artifact with the Payment Provider.

Note: Many of the normal steps in a SIP call flow have been left out of this example to focus on the pertinent items.

<table>
<thead>
<tr>
<th>Customer</th>
<th>Merchant Proxy</th>
<th>Payment Provider</th>
<th>Merchant</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>INVITE F1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>----------------&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>402+Charge F2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;---------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Request for Payment F3</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>----------------&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Receipt (Artifact) F4</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;---------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>INVITE F5</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>+ Receipt</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>----------------&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>180 F6</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;---------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Artifact F7</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>----------------&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Assertion F8</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;---------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>INVITE F9</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>----------------&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>200 F10</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;---------------</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

F1 INVITE: Customer -> Merchant Proxy

The Customer sends a SIP INVITE which arrives at the Merchant’s
Proxy.

F2 402 + Charge: Merchant Proxy -> Customer

The Merchant’s Proxy, acting on behalf of the Merchant, sends a 402 response with a Charge Request back to the Customer. The Charge Request must indicate that the Payment Provider must provide a SAML Artifact rather than a SAML Assertion as a Receipt.

F3 Request for Payment: Customer -> Payment Provider

Based on the contents of the Charge Request, the Customer makes a Request for Payment to one of the specified Payment Providers. The Request for Payment specifies that the Payment Provider should return an Artifact.

F4 Receipt (Artifact): Payment Provider -> Customer

The Payment Provider transfers the appropriate funds to the Merchant’s account and returns a SAML Artifact to the Customer.

F5 Invite + Receipt: Customer -> Merchant Proxy

The Customer inserts a new header containing the SAML Artifact into the INVITE request.

F6 180: Merchant Proxy -> Customer

The Merchant Proxy immediately sends a 180 response to the Customer indicating that the Receipt is being validated with the Payment Provider.

F7 Artifact: Merchant Proxy -> Payment Provider

The Merchant Proxy, acting on behalf of the Merchant, requests the SAML Assertion from the Payment Provider by providing the SAML Artifact.

F8 Assertion: Payment Provider -> Merchant Proxy

The SAML Assertion is returned to the Merchant Proxy.

F9 INVITE: Merchant Proxy -> Merchant

Since the SAML Assertion was valid, and was not used before by this Merchant, the Merchant Proxy forwards the INVITE request to the Merchant.
5.4.2. Customer Proxy

In this example, the Customer endpoint does not implement this payment mechanism so the Customer’s Edge Proxy acts on behalf of the Customer. Note that the Merchant’s Proxy has been left out of the example.

F1 INVITE: Customer -> Customer Proxy

The Customer sends an INVITE to the Customer’s Edge Proxy (in the same Administrative domain as the Customer).

F2 100: Customer Proxy -> Customer
The Customer’s Proxy sends a 100 response to the Customer in order to quench retransmissions.

F3 INVITE: Customer Proxy -> Merchant

The Customer’s Proxy forwards the request to the Merchant. In most cases, the Merchant will also have a Proxy but this has not been indicated in this example.

F4 402 + charge: Merchant -> Customer Proxy

The Merchant sends back a 402 response with a Charge Request to the Customer’s Proxy.

F5 Request Payment: Customer Proxy -> Payment Provider

The Customer Proxy, acting on behalf of the Customer, makes a Request for Payment to the selected Payment Provider specifying that a SAML Artifact is to be returned. It is assumed that the Customer’s Proxy would have the payment credentials and Payment Provider preferences for the Customer.

F6 Receipt (Artifact): Payment Provider -> Customer Proxy

The Receipt is returned as a SAML Artifact.

F7 INVITE + Receipt: Customer Proxy -> Merchant

The Customer Proxy now inserts the SAML Artifact as a SIP header in the INVITE request to be sent to the Merchant.

F10 Artifact: Merchant -> Payment Provider

The Merchant pulls the SAML Artifact from the INVITE and sends a request to the specified Payment Provider to return the SAML Assertion.

F11 Assertion: Payment Provider -> Merchant

When the SAML Assertion is verified, the Merchant completes the call and sends a 200 OK response.

5.5. Payment Provider Behavior

The primary function of the Payment Provider is to receive Requests for Payments over HTTPS, to transfer the currency from one account to another one, and to return a Receipt over HTTPS.
A Payment Provider MUST support TLS to offer channel security (with integrity, replay and confidentiality protection).

When a Payment Provider receives a Request for Payment, it performs the following steps:
1. Verify that the customer-id corresponds to a valid account and that the presented credentials are correct for the given account. If either the account is invalid or the authentication of the account holder was not successful then an error message MUST be returned. The procedure for responding to error responses is described in the respective SAML specifications and the Payment Provider MUST respect the SAML specific message processing handling.
2. MUST validate that the currency is acceptable by the Merchant
3. MUST ensure that the Customer has sufficient funds, and
4. MUST verify that the account identifier of the Merchant corresponds to a valid account.
5. MUST create a Receipt as a SAML assertion (or an artifact) by setting the corresponding fields in the Assertion.
6. MUST set the Date to the current time.
7. MUST digitally sign the Assertion.
8. MUST transfer the money from the customer’s account to the merchant’s account.
9. MUST return either the Assertion or the Artifact back to the payment requesting entity (depending on the request).

5.6. Account Names

Note: Need to define they syntax for valid account id. Email style account names (where the host part is not the same as the PSP domain) need to be allowed.

5.7. Merchant Fetching Public Key

The Merchant needs to be able to fetch the Payment Provider’s public key. This MAY be done by an HTTPS request to a URI provided by the Payment Service Provider. The Merchant and should use some standard online certificate revocation mechanism to protect against the private key being compromised. Note that the s may wish to use signing certificates that are only valid for a short period of time. The duration should be determined by the PSP based on the amount of money at risk.

6. SIP Extensions
6.1. Update response code 402

This document updates the 402 response code in RFC 3261 [6] to mean that "A Payment is Required". Other mechanisms are used to indicate what type of payment is required. In the case of this draft, a particular MIME body type indicates the type of payment required. A single 402 may indicate that more than one type of payment is required. Both proxies and UASs can issue a 402 response code.

7. Syntax

7.1. Charge

The Charge contains a list of costs and a list of Payment Providers. The Customer can choose to pay any one of the provided costs and can choose any one of the Payment Providers to use, as long as the Payment Provider supports the currency for the chosen cost. A Merchant can also specify a currency namespace with a particular cost. This allows the Merchant to create non-standard currencies, e.g., airline mileage/points. A cost can also specify an optional minimum and/or maximum cost.

The currency is specified as 3 uppercase letters using the ISO currency code from ISO.4217. All cost attributes (initialCost, costPerUnitTime, costPerUnitData, minCost, maxCost) are specified in currency units where the actual cost is to be divided by currencyDivisor. All amounts are base 10 integers with no leading zeros and zero is not a valid entry. The timeUnitSize is in milliseconds. The dataUnitSize is in octets. merchantBits and pspBits are base 64 encoded data.

Each PaymentServiceProvider provided in a Charge has a serviceUrl attribute which is where the PaymentRequest will be sent to. There is also an optional url attribute which is a general HTTP URL that can provide information about the specific Payment Provider.

A simple example is provided with a single charge for a single Payment Provider where the initial cost is $0.25 and the charge is 0.06/minute charged in 6 second increments.
The body is defined with XML Schema. The syntax is specified in Section 9.

7.2. Request for Payment

In order to make the generation of the Request for Payment as simple as possible for both the Customer and the Payment Provider, the RequestForPayment is constructed as a SAML Request. The Request for Payment consists of three main components:

- The ChargeData is copied from the Charge from the Merchant.
- The PaymentServiceProviderData is selected from the possible Payment Providers by the Customer.
- The amount that needs to be payed.

Additionally, the Customer might provide authentication credentials as part of the SAML request. Note that it is up to the Customer to select a currency.

The ChargeData consists of the following fields from the Payment Request: ChargeExpiry and merchantBits.

The PaymentServiceProviderData is copied from the chosen Payment Provider/Cost in the Payment Request consisting of the following fields: merchantId, serviceUrl, pspBits, currencyNamespace, currencyDivisor, and currency.

The CustomerData is provided by the Customer and consists of the following fields: customerId, customerAuth, customerBillingCode and
an amount. The customerBillingCode is optional and could be stored by the PaymentServiceProvider and included in the transaction history for the convenience of the Customer. The amount divided by currencyDivisor indicates the amount of funds being requested to be transferred from the Customer to the Merchant in currency units. The customerId is a token identifying the Customer’s account with the and the customerAuth is the authentication credentials.

The SAML Request MUST be TLS protected. Authentication of the client MUST be provided, either as part of the TLS handshake or using SAML specific mechanisms.

The XML schema of the Payment Request is shown in Section 9.

Below is an example ‘Payment Request’ based on the previous ‘Charge’ example with a payment of $0.30.

The example below should give the reader a first impression about the layout of the Payment request. A future version of this document will provide examples that can be validated.

```xml
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope/">
  <env:Body>
    <samlp:AuthnRequest
      xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
      ForceAuthn="true"
      AssertionConsumerServiceURL="http://www.example.com/
      AttributeConsumingServiceIndex="0"
      ProviderName="string"
      ID="abe567de6"
      Version="2.0"
      IssueInstant="2005-02-28T22:20:51.52Z"
      Destination="http://www.psp.example.com/
      Consent="http://www.example.com/

    <saml:Subject
      xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
      <saml:NameID
        Format="urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress"
        joe@example.com
      </saml:NameID>
    </saml:Subject>
    </samlp:AuthnRequest>
    <samlp:Query xsi:type="PaymentRequest">
      <chargeExpiry>
        2005-02-01T13:40:26.52Z
      <chargeExpiry>
    </samlp:Query>
  </env:Body>
</env:Envelope>
```
<merchantBits>
MDE1Mw==
</merchantBits>
<merchantId>
15
</merchantId>
<serviceUrl>
https://psp.example.com/paymentService
</serviceUrl>
<currencyDivisor>
1000
</currencyDivisor>
<currency>
USD
</currency>
<customerId>
joe
</customerId>
<amount>
300
</amount>
</samlp:Query>
</env:Body>
</env:Envelope>

7.3. Receipt

A SAML assertion or artifact is returned, as a Receipt, in response to the Request for Payment. The assertion or the artifact is included in the SIP message, as described in [5], in the reINVITE from the UAC.

The following example receipt is returned from the Payment Provider and corresponds to a Receipt for a payment of $0.30.

The example below should give the reader a first impression about the layout of the Payment response. A future version of this document will provide examples that can be validated.

<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
<env:Body>
<samlp:Response
xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
ID="abe567de6"
InResponseTo="example-ncname"
Version="2.0" IssueInstant="2005-01-31T12:00:00Z"
7.4. Refunds

Refunds share the same syntax as a Request for Payment. The refund fields populated from the Receipt are: ChargeId, ChargeExpiry, merchantBits, merchantId, serviceUrl, pspBits, currencyNamespace, currencyDivisor, and currency. The customerId and customerAuth refer to the Merchant’s account information with the referenced by the serviceUrl. The amount to refund is determined by the Merchant at the time of the refund.

7.5. Verifying the Receipt

The Merchant MUST verify the signature in the Receipt by applying the following steps:
1. Check ReceiptData.Date. If too old, reject.
2. Check whether receipt-id has been accepted in a previous payment since the TTL used by the UAS. If so, reject. (See section 7.6 on replay prevention)
3. Check whether Charge comes from this UAS. If not, reject.
4. Perform RSA signature verification. UAS chooses the public key based on PaymentServiceProvider-id.

8. Usage Scenarios

This section shows the applicability of the proposed protocol for two example scenarios.

8.1. SPAM

Payment at risk has been suggested as part of a possible solution to SPAM in VoIP systems [15]. The idea is that A would call B. If A was not on B’s white list, B could ask A to pay 5 cents for the privilege of ringing B’s phone. A could pay, and if B wished, B could refund the 5 cents by simply doing a second payment from B to A. The payment service provider would collect two transaction fees in this scenario.
Another possible scenario is that B simply requests that A donate 5 cents to one of B’s favorite charities and show B the receipt for this transaction.

8.2. Micro Billing

In this scenario, a merchant running a PSTN GW may charge a customer 5 cents to connect and operate for the first 90 seconds and then may charge 5 more cents for each additional minute. The customer would initially transfer 5 cents and then, before the 90 seconds ran out, would transfer another 5 cents and keep on doing this until the call ended.

9. XML Schemas

This section defines the XML schema for the protocol introduced in this document.

9.1. Payment Request and Payment Receipt

This schema is a strawman proposal and will change as soon as an agreement for the required attributes can be found.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs='http://www.w3.org/2001/XMLSchema'>


<xs:complexType name="PaymentRequest">  
    <xs:extension base="saml:QueryAbstractType">  
    <xs:sequence>  
        <xs:element name="chargeExpiry" type="xs:dateTime" minOccurs="1" maxOccurs="1"/>  
        <xs:element name="merchantBits" type="xs:base64Binary" minOccurs="1" maxOccurs="1"/>  
    </xs:sequence>  

</xs:complexType>

</xs:schema>
</xs:schema>
```
<xs:element name="merchantId"
  type="xs:token"
  minOccurs="1"
  maxOccurs="1"/>
<xs:element name="serviceUrl"
  type="xs:anyURI"
  minOccurs="1"
  maxOccurs="1"/>
<xs:element name="pspBits"
  type="xs:base64Binary"
  minOccurs="0"
  maxOccurs="1"/>
<xs:element name="currencyNamespace"
  type="xs:token"
  minOccurs="0"
  maxOccurs="1"/>
<xs:element name="currencyDivisor"
  type="xs:unsignedLong"
  minOccurs="1"
  maxOccurs="1"/>
<xs:element name="currency"
  type="xs:token"
  minOccurs="1"
  maxOccurs="1"/>
<xs:element name="customerId"
  type="xs:token"
  minOccurs="0"
  maxOccurs="1"/>
<xs:element name="customerBillingCode"
  type="xs:token"
  minOccurs="0"
  maxOccurs="1"/>
<xs:element name="amount"
  type="xs:unsignedLong"
  minOccurs="1"
  maxOccurs="1"/>
</xs:sequence>
</xs:extension>
</xs:complexContent>
</xs:complexType>

<xs:complexType name="PaymentReceipt">
  <xs:complexContent>
    <xs:extension base="saml:StatementAbstractType">
      <xs:sequence>
        <xs:element name="merchantBits"
          type="xs:base64Binary"
          minOccurs="1"/>
<xs:element name="merchantId"
    type="xs:token"
    minOccurs="1"
    maxOccurs="1"/>
<xs:element name="pspBits"
    type="xs:base64Binary"
    minOccurs="0"
    maxOccurs="1"/>
<xs:element name="serviceUrl"
    type="xs:anyURI"
    minOccurs="1"
    maxOccurs="1"/>
<xs:element name="currencyNamespace"
    type="xs:token"
    minOccurs="0"
    maxOccurs="1"/>
<xs:element name="currencyDivisor"
    type="xs:unsignedLong"
    minOccurs="1"
    maxOccurs="1"/>
<xs:element name="currency"
    type="xs:token"
    minOccurs="1"
    maxOccurs="1"/>
<xs:element name="amount"
    type="xs:unsignedLong"
    minOccurs="1"
    maxOccurs="1"/>
</xs:sequence>
</xs:extension>
</xs:complexContent>
</xs:complexType>
</xs:schema>

9.2. Charge

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified">
    <xs:element name="currency">
        <xs:complexType>
            <xs:attribute name="namespace"
                type="xs:string"
                use="required"/>
            <xs:attribute name="currency"
                type="xs:string"/>
        </xs:complexType>
    </xs:element>
</xs:schema>
<xs:complexType>
  <xs:sequence>
    <xs:element name="currencies">
      <xs:complexType>
        <xs:sequence>
          <xs:element ref="currency"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:attribute name="url" type="xs:anyURI" use="optional"/>
    <xs:attribute name="serviceUrl" type="xs:anyURI" use="required"/>
    <xs:attribute name="merchantId" type="xs:unsignedLong" use="required"/>
    <xs:attribute name="pspBits" type="xs:base64Binary" use="optional"/>
  </xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="charge">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="chargeData"/>
      <xs:element name="costs">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="currencyDivisor" type="xs:unsignedLong" use="required"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="paymentServiceProvider">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="currencies">
        <xs:complexType>
          <xs:sequence>
            <xs:element ref="currency"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:attribute name="url" type="xs:anyURI" use="optional"/>
      <xs:attribute name="serviceUrl" type="xs:anyURI" use="required"/>
      <xs:attribute name="merchantId" type="xs:unsignedLong" use="required"/>
      <xs:attribute name="pspBits" type="xs:base64Binary" use="optional"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="charge">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="chargeData"/>
      <xs:element name="costs">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="currencyDivisor" type="xs:unsignedLong" use="required"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>

10. Security Considerations

The security properties of the proposed protocol depends on the security of the communication between the three parties. The
following threats and countermeasures have been considered:

10.1. Stolen Assertion

Threat:
An adversary can eavesdrop on the communication between the Customer and the Payment Provider and between the Customer and the Merchant and thereby learn the SAML assertion (Receipt). The adversary could use this Assertion to request a service from the Merchant on behalf of the legitimate Customer.

Countermeasures:
Two countermeasures are provided to deal with this threat. The communication between the Customer and the Payment Provider must be confidentiality, integrity and replay protected. The communication between the Customer and the Merchant MAY experience either hop-by-hop (e.g., TLS in a hop-by-hop fashion between the Customer, SIP proxies and the Merchant) or end-to-end by using S/MIME protection. Furthermore, the content of the requested SAML assertion contains statements that prevent reusage of the SAML assertion in other contexts. The Charge, for example, provides a Cookie value that allows the Merchant to match the Charge to the Receipt. The identities of the Customer and the Merchant are included in the Receipt and the lifetime might be limited.

10.2. MitM Attack

Threat:
Since the SAML assertion is carried within a SIP message sent by the Customer towards the Merchant an intermediate SIP proxy could use the assertion in order to impersonate the user towards the Merchant in future protocol sessions.

Countermeasures:
This document does not assume that theAssertion is bound to a symmetric or an asymmetric key. Hence, only the content of the assertion can be used to prevent replay attacks and man-in-the-middle attacks. Similarly to the threat described in Section 10.1 the content of the assertion limits its usage to particular endpoints, potentially for a particular duration, for a particular service and for certain amount of time.

10.3. Forged Assertion

Threat:
A malicious Customer or a malicious intermediate SIP proxy could forge or alter a SAML assertion in order to communicate with the Merchant to lead to unexpected behavior or even to refunding to a preselected account.
Countermeasures:
To avoid this kind of attack, the Payment Provider must assure that proper mechanisms for protecting the SAML assertion is provided. It is recommended to protect the assertion using a digital signature.

10.4. Replay Attack

Threat:
An adversary might eavesdrop an assertion in order to later replay it in order to gain access to resources (e.g., to make a phone call).

Countermeasures:
When the Customer transmits a SIP message that requires payment then the Merchant creates a Charge that contains a Cookie value. The Charge will be used by the Customer to obtain an assertion by the Payment Provider and will again be presented to the Merchant. The Merchant is therefore able to determine whether the Cookie is still valid and that it can be associated with an Charge. The Charge is only valid for a certain time. Timestamp information carried inside the Assertion might also indicate a validity timeframe.

11. IANA Considerations

This specification registers a new header and a new response code. IANA is requested to make the following updates in the registry at: http://www.iana.org/assignments/sip-parameters. It also defines a new mime type that IANA is requested to add to the registry at http://www.iana.org/assignments/media-types/application.

This specification registers a new header and a new response code. IANA is requested to make the following updates in the registry at: http://www.iana.org/assignments/sip-parameters. It also defines a new mime type that IANA is requested to add to the registry at http://www.iana.org/assignments/media-types/application.

11.1. Payment-Receipt Header

Add the following entry to the header sub-registry.

<table>
<thead>
<tr>
<th>Header Name</th>
<th>compact</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Payment-Receipt</td>
<td>[RFCXXXX]</td>
<td></td>
</tr>
</tbody>
</table>
11.2.  402 Response

Add the following entry to the response code sub-registry under the
"Request Failure 4xx" heading.

        402  Payment Required                      [RFCXXXX]

11.3.  charge+xml

To: ietf-types@iana.org
Subject: Registration of MIME media type application/charge+xml

MIME media type name: application

MIME subtype name: charge+xml

Required parameters: None

Optional parameters: charset
Same as charset parameter of application/xml [RFC3023]

Encoding considerations:
Same as for application/xml [RFC3023]

Security considerations: TBD

Interoperability considerations: TBD

Published specification: None

Applications which use this media type: Any MIME-compliant transport

Additional information:
    Magic number(s): None
    File extension(s): None
    Macintosh File Type Code(s): None

Person & email address to contact for further information:
    Cullen Jennings <fluffy@cisco.com>

Intended usage: COMMON

Author/Change controller:
    the IESG

[Editor’s Note: This section also needs to discuss the extensions to
the SAML request and the SAML assertion.]

12. Open Issues

Would like to unify the body syntax so that it can also be used for Advice Of Charge information.

What does a SAML Artifact look like and how does it get associated with the Charge and Dialog ID.

How can a Proxy protect from a SAML Artifact from being used in a Replay Attack?

13. References

13.1. Normative References


13.2. Informational References


Intellectual Property Statement

The IETF takes no position regarding the validity or scope of any Intellectual Property Rights or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; nor does it represent that it has made any independent effort to identify any such rights. Information on the procedures with respect to rights in RFC documents can be found in BCP 78 and BCP 79.

Copies of IPR disclosures made to the IETF Secretariat and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the IETF on-line IPR repository at http://www.ietf.org/ipr.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at ietf-ipr@ietf.org.

Disclaimer of Validity

This document and the information contained herein are provided on an "AS IS" basis and THE CONTRIBUTOR, THE ORGANIZATION HE/SHE REPRESENTS OR IS SPONSORED BY (IF ANY), THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Copyright Statement

Copyright (C) The Internet Society (2005). This document is subject to the rights, licenses and restrictions contained in BCP 78, and except as set forth therein, the authors retain all their rights.

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.