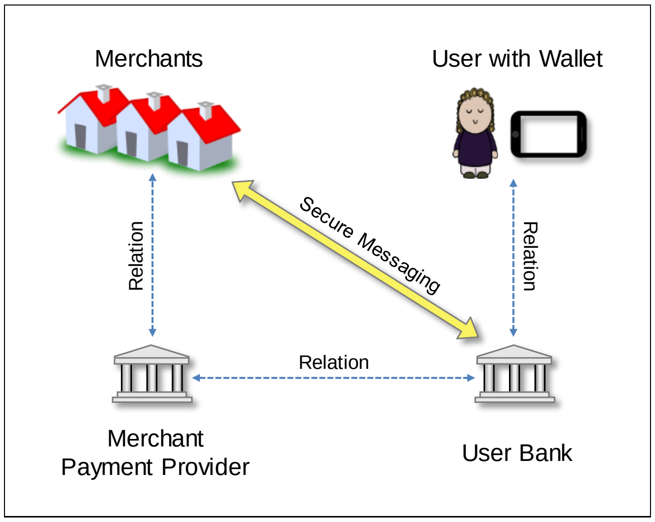
Authority Objects

This document outlines a scheme where Internet-based “Authority” objects enable a dynamic but still secure way of building large-scale trust networks and associated protocols. Although the document describes a Merchant and Bank scenario, there may be other constellations benefitting from a similar architecture.

Problem Area

Before going into the details it might be a good idea describing the problem area on a higher level which the figure below is supposed to do:



This looks a four-corner model with an alien element in the middle which is exactly what it is (in the original four-corner model all Merchant messaging would go through the Merchants’ Payment Providers).

The four-corner model is a scheme where clients usually only interact through trusted third parties

Is there a problem with the four-corner model? Well, not really except that it requires quite a bit of networking and integration to work. The purpose of this document is showing an alternative model which is claimed to achieve similar goals with respect to security while building on a more nimble framework.

In addition, the alternative model introduces new elements which can provide information on a peer-to-peer basis allowing trust networks to grow and add new functionality in an organic fashion in contrast to one-size-fits-all approaches.

This document borrows heavily from a proof-of-concept implementation of a payment authorization system known as Saturn.

There are also multiple references to this document in another core document: PAS (Payment Authorization Scheme).

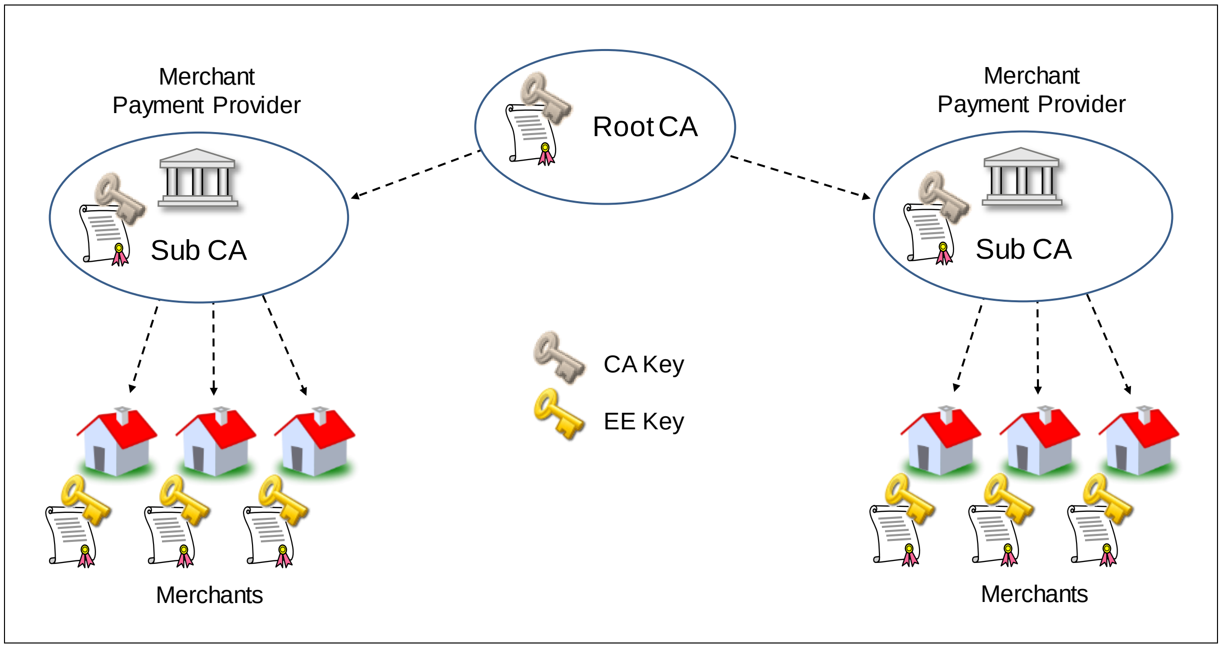
Continued on the next page…

Authenticating Merchants

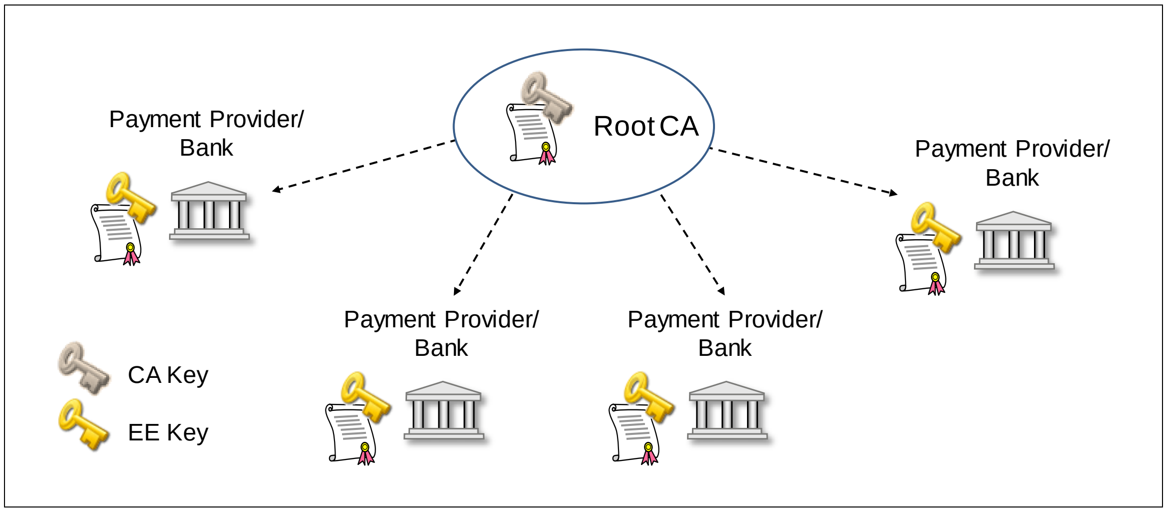
If Merchants in some way call User Banks directly as in the figure in the previous page it is not entirely obvious how they are supposed to be authenticated except that messages would preferably be digitally signed. The question is rather signed by what?

Initial Approach: Create a Merchant PKI

A straightforward solution would be using PKI where Merchants belong to a specific Merchant trust network as illustrated by the following figure:



Although obviously working, this scheme introduces an additional central authority and related trust anchor (to distribute) which seems redundant when the Bank/Payment Provider parties *presumably* already are “certified” like this:

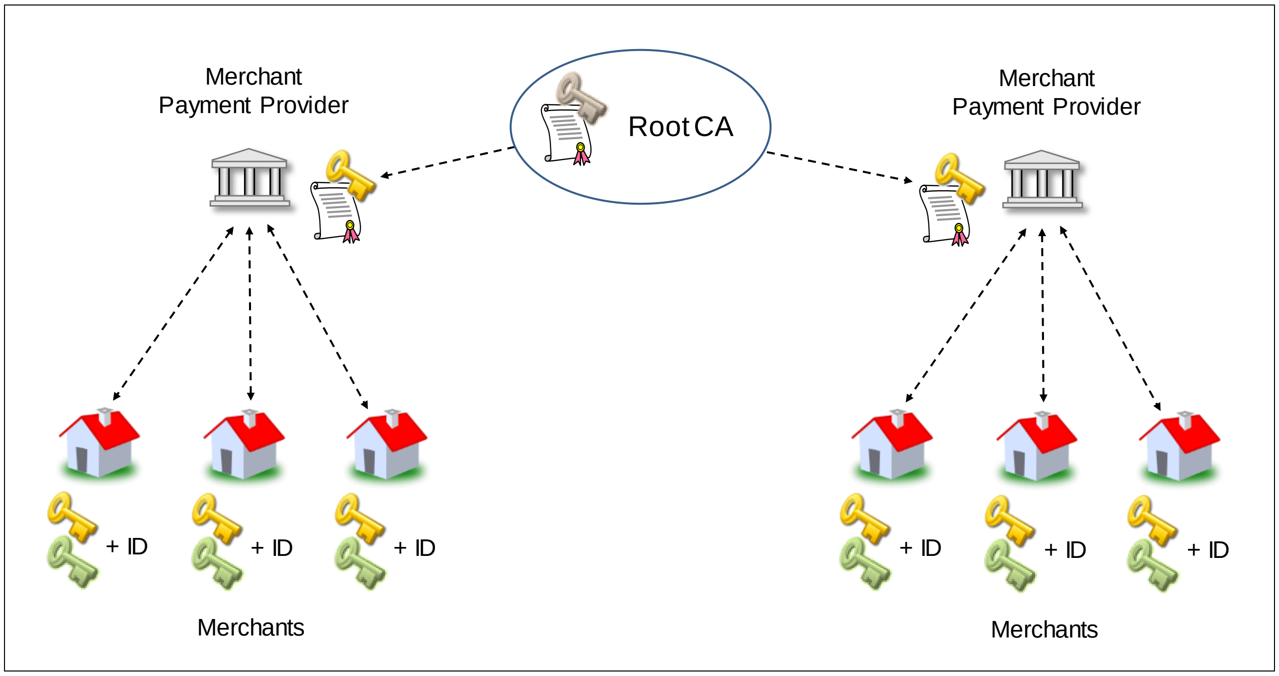


Since EE (End Entity) certificates are not permitted to sign certificates, other Merchant solutions than X.509 certificates were evaluated while still building on the four-corner trust model.

Continued on the next page…

Revised Approach: Reusing the Bank/Payment Provider PKI

Instead of providing a full-fledged Merchant PKI, Merchant Bank and Payment Providers could *vouch* for Merchants through the anticipated already existing Bank/Payment Provider PKI. To do that Merchants still need some kind of signature key and/or identifier, otherwise it effectively becomes the Bank/Payment Providers who request payments which probably is not what you want. The following picture shows the trust architecture (with respect to *keys*), exploited in this document:

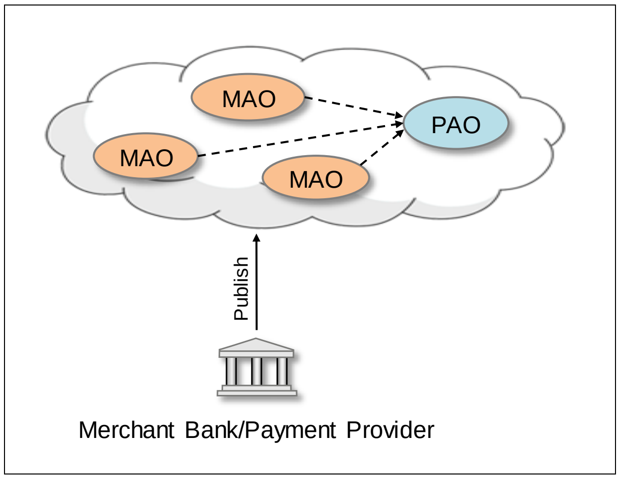


The key icons under the Merchant images represent *asymmetric key-pairs*, while the `ID`s hold Merchant identity information which minimally is just a number.

Assuming a Merchant wants to interact with another Payment Provider than the one which it is associated with it would not be sufficient merely signing a request message with its private key only because neither the corresponding public key nor the `ID` is necessarily known by the called provider.

In a “true” four-corner model this would be addressed by first letting the Merchant create and sign a message and then have its trusted Payment Provider party *counter-sign* that message.

However, this document outlines a “modified” four-corner model where *public*, *Internet-based*, “Authority” objects eliminate the need for *active* participation of Merchants’ trust partner in every transaction. The figure below illustrates the core:



Each MAO (Merchant Authority Object) contains information about a single Merchant, whereas a PAO (Provider Authority Object) holds information about the provider itself. Both types of objects are supposed to be hosted by the associated trust provider and be accessible on static URLs by HTTP GET requests over TLS (aka HTTPS).

MAO - Merchant Authority Object

The following shows possible contents of a Merchant Authority Object:

```
{
   "@context": "https://example.com/paymentstd",
   "@qualifier": "PayeeAuthority",
   "authorityUrl": "https://payments.bigbank.com/payees/86344",
   "providerAuthorityUrl": "https://payments.bigbank.com/authority",
   "commonName": "Demo Merchant",
   "id": "86344",
   "publicKey": {
      "type": "EC",
      "curve": "P-256",
      "x": "rZ344aiTaOATmLBOdfYThvnQu_zyB1aJZrbbbks2P9I",
      "y": "lKOvfJdgN8WqEbXMDYPRSMsPicm0Tk10pmer9LxvxLg"
   },
   "timeStamp": "2016-10-20T16:26:57Z",
   "expires": "2016-10-20T17:26:58Z",
   "signature": {
      "algorithm": "ES256",
      "signerCertificate": {
         "issuer": "CN=Payment Network Sub CA3,C=EU",
         "serialNumber": "1461174554959",
         "subject": "CN=Big Bank,2.5.4.5=#1306383936363430,C=DE"
      },
      "certificatePath": [
         "MIIBtTCCAVmgAwIBAgIGAVQ0 … 5CHYX3i2r67iG_MsApiD3jFnqaJhxCZ",
         "MIIDcjCCAVqgAwIBAgIBAzANB … gkqhkiG9w0BAQ0FADAwMQswCQYD"
      ],
      "value": "CKAfEg7wzCoxGiCvRAMVIug0RKU … 4AOtKZK_RPNoshOGVnxry7vQZeIuIw"
   }
}
```

Note: Payee = Merchant. Although the MAO shown here is expressed in JSON, the *concept* is virtually independent of format.

The **`@context`** and **`@qualifier`** properties are used throughout this specification as a way to identify the actual message (object) type.

The **`authorityUrl`** property holds a URL to the object itself.

The **`providerAuthorityUrl`** property holds the URL to the associated PAO.

The **`commonName`** property holds the common name of the Merchant.

The **`id`** property holds a static Merchant-identifier which usually is assigned by the associated trust provider.

The **`publicKey`** object holds the Merchant's public key in JCS format.

The **`timeStamp`** property holds the creation time of the MAO.

The **`expires`** property holds the expiration time of the MAO. See MAO Revocation and Renewals.

The **`signature`** object holds the associated trust provider's attest signature in JCS format. Note that this is an X.509 based signature.

Continued on the next page…

Using MAOs

The message object below shows the intended use of MAOs when calling trust providers:

```
{
  "@context": "https://example.com/paymentstd",
  "@qualifier": "SampleRequest",
  "authorityUrl": "https://payments.bigbank.com/payees/86344",
        .
        .
    Other elements
        .
        .
  "signature": {
    "algorithm": "ES256",
    "publicKey": {
      "type": "EC",
      "curve": "P-256",
      "x": "rZ344aiTaOATmLBOdfYThvnQu_zyB1aJZrbbbks2P9I",
      "y": "lKOvfJdgN8WqEbXMDYPRSMsPicm0Tk10pmer9LxvxLg"
    },
    "value": "G4ct46eTx-GgF2qrSnHKRR9f9Ajd … ju85d56gSON2M3I20-u6sfcejw"
  }
}
```

The **`authorityUrl`** property holds the URL to the requesting Merchant's MAO.

The **`signature`** object holds a JCS signature created by the Merchant's private (signature) key.

Merchant Authentication

After verifying that the message type is known and correctly formatted (including the signature), the receiver fetches the claimed MAO by performing an HTTP GET using the address given by **`authorityUrl`**. Although indirection adds latency, *MAOs can be cached*.

Next the MAO is checked for correctness.

Next the **`certificatePath`** in the MAO is verified belonging to a (for the provider) known and trusted party.

Next the **`publicKey`** vouched for in the MAO is compared with the **`publicKey`** in the received message (they must be identical).

Note: The authentication scheme above may not fit every purpose. In the PAS specification a request message must also contain an embedded user authorization to be valid.

MAO Revocation and Renewals

Since Merchants are not using PKI a question arises: How do you "revoke" a Merchant? Answer: This is a part of the MAO renewal process. That is, MAOs are supposed to be *renewed* with an interval like 1 hour to 1 day. The lifetime of a MAO is expressed by the **`timeStamp`** and **`expires`** properties.

A proper implementation should preferably renew MAOs at every half of the declared lifetime in order to maintain robust operation.

If a MAO needs to be revoked, the renew process is simply halted, eventually leading to an expired (=invalid) MAO. A faster (and maybe safer) alternative with respect to revocation is removing revoked MAOs so that requesters get HTTP 401 as responses.

Continued on the next page…

PAO - Provider Authority Object

The following shows possible contents of a Provider Authority Object:

```
{
  "@context": "https://example.com/paymentstd",
  "@qualifier": "ProviderAuthority",
  "httpVersion": "HTTP/1.1",
  "authorityUrl": "https://payments.bigbank.com/authority",
  "authorizationUrl": "https://payments.bigbank.com/authorize",
  "providerAccountTypes": ["https://swift.com","https://ultragiro.se"],
  "encryptionParameters": {
    "dataEncryptionAlgorithm": "A128CBC-HS256",
    "keyEncryptionAlgorithm": "ECDH-ES",
    "publicKey": {
      "type": "EC",
      "curve": "P-256",
      "x": "TfCrhFwZRU_ea7lUWwRi3HkuyT2yF9IxN5xKh2khjlk",
      "y": "nZFwxLP0TvFXD2xPKzRTIGevgLjpiMw2BP86hszj5x4"
    }
  },
  "extensions": {
    "https://somehost/clearingNumber": 45678,
    "https://somehost/addSEPALog": true
  },
  "timeStamp": "2016-09-20T04:34:29Z",
  "expires": "2016-09-20T05:34:29Z",
  "signature": {
    "algorithm": "ES256",
    "signerCertificate": {
      "issuer": "CN=Payment Network Sub CA3,C=EU",
      "serialNumber": "1461174554959",
      "subject": "CN=Big Bank,2.5.4.5=#1306383936363430,C=DE"
    },
    "certificatePath": [
      "MIIBtTCCAVmgAwIBAgIGAVQ0 … 5CHYX3i2r67iG_MsApiD3jFnqaJhxCZ",
      "MIIDcjCCAVqgAwIBAgIBAzANB … gkqhkiG9w0BAQ0FADAwMQswCQYD"
    ],
    "value": "cdRqFlzVEou5Zj-EqWGCCLtxY … JkEBD4fFOqVnU9dstv_P2BoHQ"
  }
}
```

The **`httpVersion`** property holds the supported HTTP version for provider communication.

The **`authorityUrl`** property holds a URL to the object itself

The **`authorizationUrl`** property holds a URL to an exposed service, in this case authorization.

The **`providerAccountTypes`** property holds an *optional* array of supported account types (payment methods). See Payment Method Discovery.

The **`encryptionParameters`** property holds an *optional* object containing parameters needed to send encrypted data to the provider. See Replacing Tokenization and Supporting Refunding for possible applications.

The **`extensions`** property holds an *optional* object containing properties that may be defined on a peer-to-peer basis or by communities. See Extending Protocols.

The **`timeStamp`** property holds the PAO creation time.

The **`expires`** property holds the PAO expiration time. The purpose of **`timeStamp`** and **`expires`** is to enable efficient caching of PAOs while still permitting updates in a timely manner.

The **`signature`** object holds a JCS signature created by the trust provider's private (signature) key.

Continued on the next page…

Extending Protocols

In theory you could define a protocol once and declare it as “final”. In reality you cannot because requirements are moving targets. A popular way of dealing with this problem is leaving some “free space” in protocols allowing parties to add things later. However, this introduces several new problems because it requires agreements between parties to work while handling unrecognized extensions by software is not completely straightforward either. After a while you typically end up with incompatible implementations.

Although PAOs are not “silver bullets” they can simply the process of adding and using extensions by *publishing* extension support, making it possible for a party in *real-time* discovering what extensions other parties understand.

Extensions have the following (recommended) syntax:

"*URI identifying the extension*": *arbitrary JSON compatible data*

By using URIs, communities can add their own extensions without running into name-clashes with other communities’ or standards bodies’ extensions. That is, *unrecognized extensions are assumed to simply be ignored*.

There are two kinds of extensions:

- *Value.* This kind of extension provides a value for direct consumption like `"https://somehost/clearingNumber": 45678`
- *Protocol.* This kind of extension signals that the provider supports a protocol extension but exactly where and how is still a matter for the designers of the extension

Note that there are no syntactic differences between these two forms of extensions! In the Saturn scheme specific extension support has been added to core messages.

Payment Method Discovery

If you take a peek in section 2.3 of the PAS document you find that the **`providerUrl`** property of virtual cards, points to the actual service end point.

By rather pointing to the account-holding provider’s PAO, the service end point can be found indirectly in the associated PAO’s **`authorizeUrl`** property. Although indirection adds latency, *PAOs can be cached*.

Now assume that the underlying payment systems may differ between payment providers (some may implement many schemes while some may only support one or two schemes), the problem for an external party is verifying that it is compatible and select one method/system which is. Here the PAO property **`providerAccountTypes`** can provide a list of supported methods. If a mutually compatible payment method is found, a request like specified in section 3 could be extended with an object like

```
"payeeAccount": {
  "type": "https://swift.com",
  "id": "IBAN:FR7630004003200001019471656"
}
```

holding the requester’s receive account.

Continued on the next page…

Replacing Tokenization

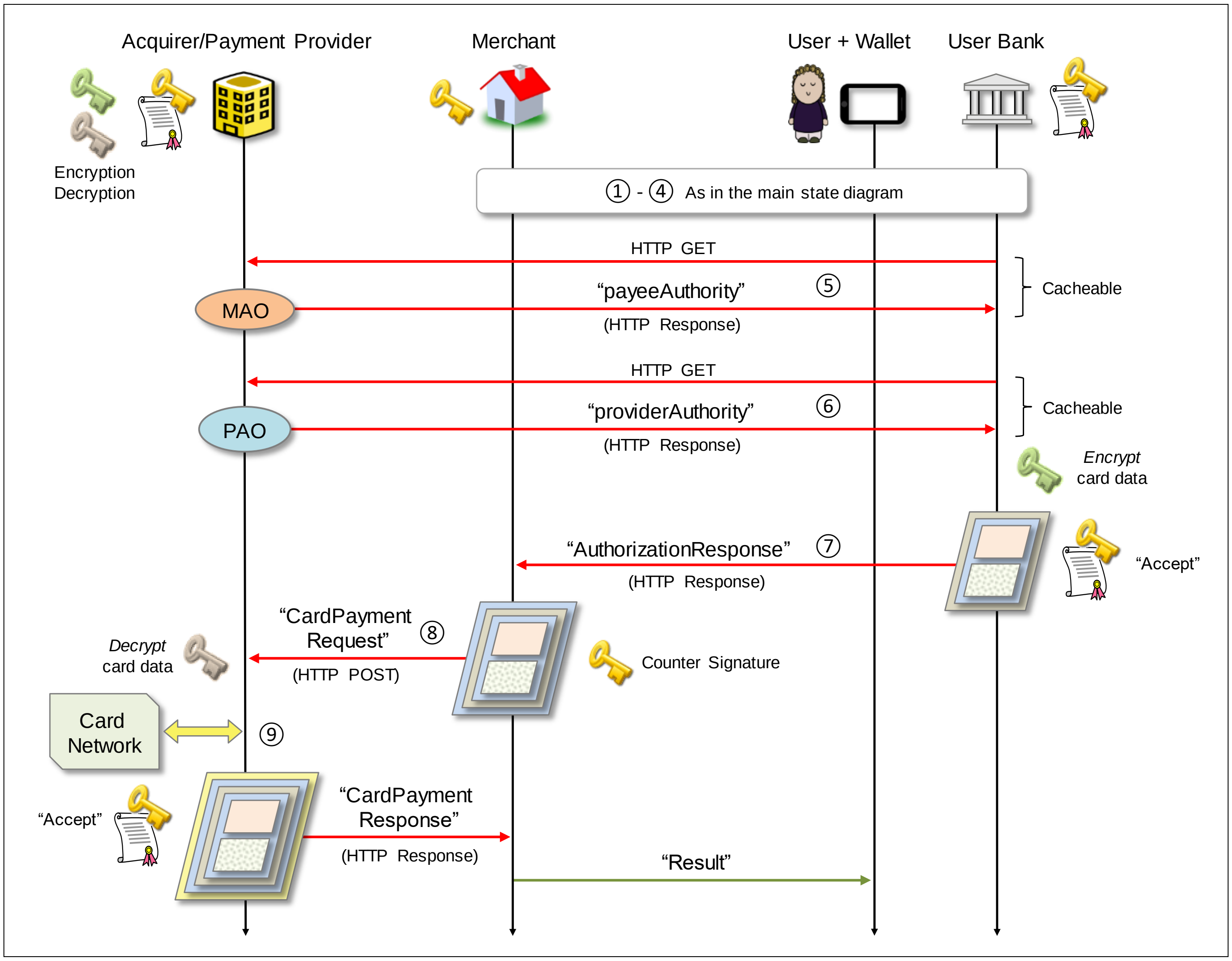
For improving the security of traditional credit card transactions, the payment networks have introduced a concept called "tokenization" which through cryptography and one-time keys makes card data less vulnerable to theft. However, the scalability of this scheme is limited, creating an *implicit* dependence on highly centralized operations.

This document outlines another solution achieving similar goals as tokenization: Aided by authority objects, a fully decentralized replacement for tokenization appears to be feasible.

Preconditions:

- The Merchant's payment provider is an acquirer (aka "Card Processor").
- The **`AuthorizationRequest`** message described in section 3 of the PAS document is extended by a property holding the **`authorityUrl`** of the Merchant's MAO.

Below there is state diagram enhancing the PAS main state diagram to suit card based transactions:



Step by step description:

⑤ *User Bank:* If not already in the cache fetch the MAO using the **`authorityUrl`** of the received **`AuthorizationRequest`**.

⑥ *User Bank:* if not already in the cache, fetch the associated PAO using the **`providerUrl`** of the MAO retrieved in ⑤.

⑦ *User Bank:* Create an **`AuthorizationResponse`** object like in section 4 of PAS while adding an object holding encrypted card data. Expressed in a JEF structure such an object could look like this:

```
"encryptedCardData": {
  "algorithm": "A128CBC-HS256",
  "encryptedKey": {
    "algorithm": "ECDH-ES",
    "publicKey": {
      "type": "EC",
      "curve": "P-256",
      "x": "TfCrhFwZRU_ea7lUWwRi3HkuyT2yF9IxN5xKh2khjlk",
      "y": "nZFwxLP0TvFXD2xPKzRTIGevgLjpiMw2BP86hszj5x4"
    },
    "ephemeralKey": {
      "type": "EC",
      "curve": "P-256",
      "x": "D7zSvy3mbS4WbB2qgKwchLRwQFir5T_p09HpnAi_RqA",
      "y": "gkwNJ2o6BtUASkmp1DO4UvllsQL5zAzvVEHB7t0CqX0"
    }
  },
  "iv": "zyConPq8uA7GFjaTkta-qA",
  "tag": "S8zUQ3tioyYPzbtNBO6Ftw",
  "cipherText": "GLkd4uHnjqL_EX9tssDNLzsZj … s7u2ezBOibNoQN5V3cl2ieB-hLHj4XppJI"
}
```

The **`publicKey`** object holds the **`publicKey`** of the PAO retrieved in ⑥.

The **`algorithm`** properties are derived from the **`encryptionParameters`** of the PAO retrieved in ⑥.

The **`cipherText`** property holds the encrypted card data. The actual contents are not dealt with here but would presumably include *card number*, *expiration date*, *name of card holder*, and *security code*.

Return the **`AuthorizationResponse`** object to the Merchant.

⑧ *Merchant:* After having received the **`AuthorizationResponse`** object and verified its correctness, the Merchant wraps the **`AuthorizationResponse`** object in a newly created **`CardPaymentRequest`** object (not elaborated on in this document), and *counter-signs it with its private key*. The **`CardPaymentRequest`** object is subsequently POSTed to the Merchant's acquirer service.

⑨ *Acquirer:* After having received the **`CardPaymentRequest`** object and verified its correctness, the embedded card data is decrypted using the acquirer's decryption key. Then the card network is called using card network specific methods.

The rest of the state diagram was only included for completeness; it has no impact on the tokenization.

Supporting Refunding

In order to refund a customer, the Merchant must in some way send money back *to* the customer's account. Since giving the Merchant account details is not very cool, the encryption scheme used for Replacing Tokenization can also be used to hold customer refund account data. To effectuate a refund the Merchant would have to put the original transaction into a wrapper document, counter-sign it, and then send it to its own bank for fulfilment.

Continued on the next page…

Invention Summary

This invention disclosure describes the following sub inventions:

1. How objects published on the Internet holding information about parties including (but not limited to) *service end points*, *signature keys*, *encryption keys*, *algorithms*, *certificates*, *payment method support*, and *protocol extensions* can be architected.
2. How URLs to objects as described in claim #1 can be put into messages enabling message receivers to dereference such objects.
3. How objects as described in claim #2 can be used for authenticating a requester who is vouched for by a trusted party.
4. How objects as described in claim #2 can be used for customizing requests with respect to selected payment method.
5. How objects as described in claim #2 can be used for retrieving service end points.
6. How objects as described in claim #2 can be used for customizing requests with respect to extensions.
7. How objects as described in claim #2 can be used for encrypting data which can only be decrypted by the party publishing the object.
8. How an object like described in claim #1 representing a party which is vouched for by a trusted party can contain a URL to a similar object for its trusted party in order to enable discovery of additional information.

References

Name	Description	URL
JCS	JSON Cleartext Signature	https://cyberphone.github.io/doc/security/jcs.html
JEF	JSON Encryption Format	https://cyberphone.github.io/doc/security/jef.html
JSON	JavaScript Object Notation	https://tools.ietf.org/rfc/rfc7159.txt
X.509	Digital Certificates and Support	https://tools.ietf.org/rfc/rfc5280.txt
PAS	Payment Authorization Scheme	https://cyberphone.github.io/doc/defensive-publications/payment-authorization-scheme.pdf
Saturn	Saturn Payment Authorization System	https://cyberphone.github.io/doc/saturn/

Permanent document URL: https://cyberphone.github.io/doc/defensive-publications/authority-objects.pdf

Authored by: anders.rundgren.net@gmail.com