

Saturn™

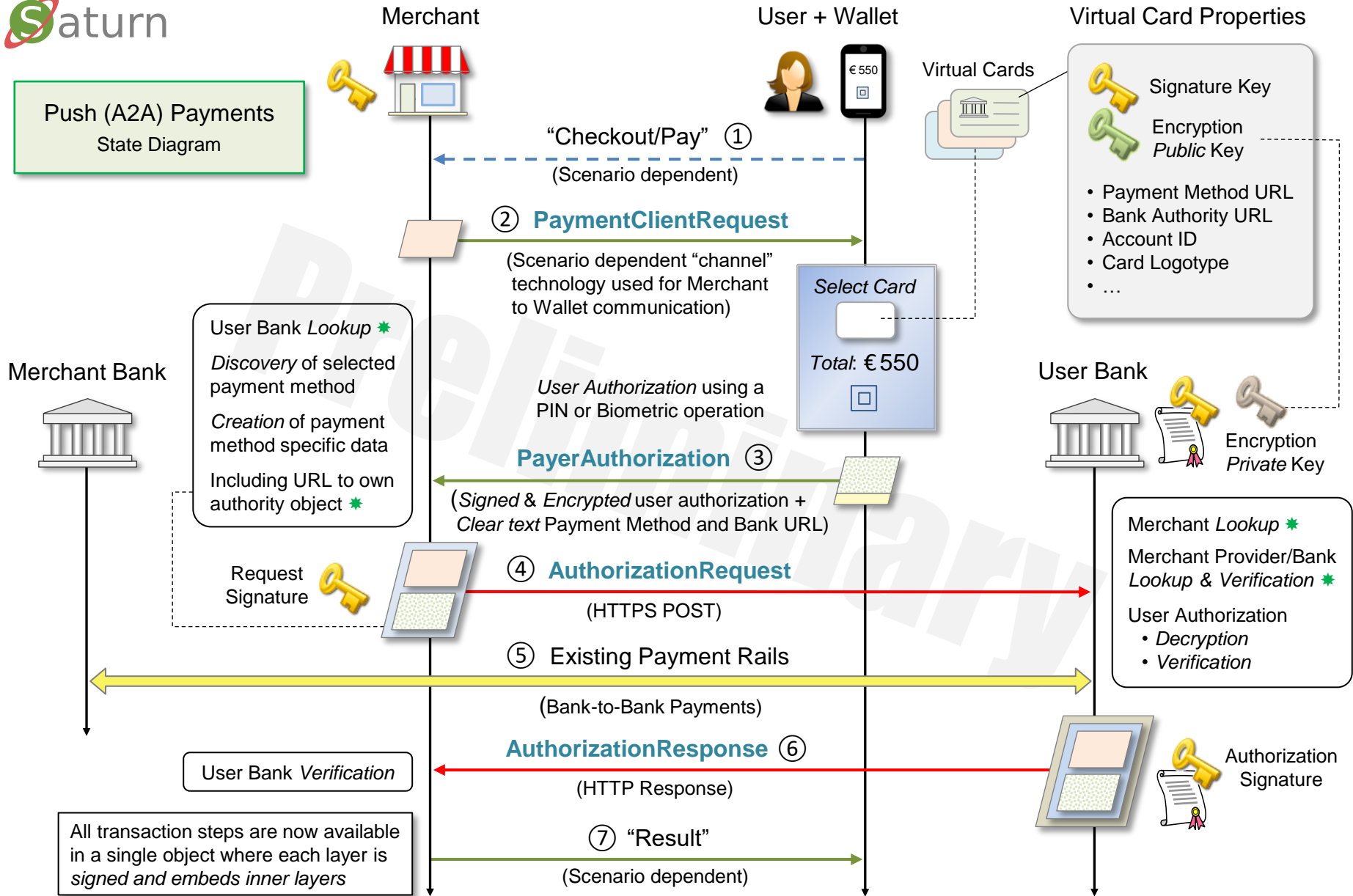
End-to-End Secured Payment Authorization System

- *Decentralized operation* accomplishes similar goals as 3D Secure and “Tokenization” but *without registries or additional services*
- Facilitates the design of brand/bank independent, “rich UI” wallets, supporting both card- and account-to-account payments
- Equally applicable on the mobile Web, locally in a shop, at an automated gas station, or as a “PC companion” on the Web
- *Eliminates* the traditional payment terminal and reduces merchant PCI requirements to a minimum
- Requires a *single* “active” method on the issuer side to function*

* *Reservations and recurring payments will in non-card-based scenarios need a second method as well*

Disclaimer: This is a system in development and specifications are subject to change without notice

Push (A2A) Payments State Diagram

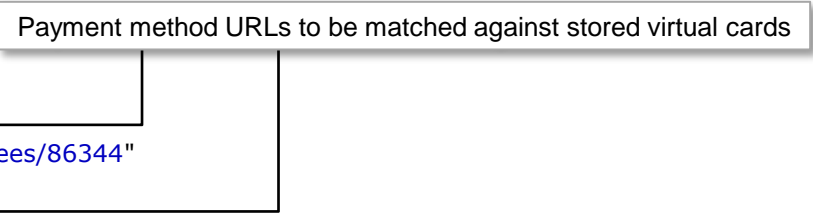


* See [Authority Objects](#). The rationale for encrypting user authorizations is for enabling such data to pass through Merchants which simplifies the Wallet as described in the [Saturn FAQ](#). Step #5 does not apply when running under the conditions outlined in [Hybrid Mode](#).

② Merchant Invokes the Wallet with a `PaymentClientRequest` Message

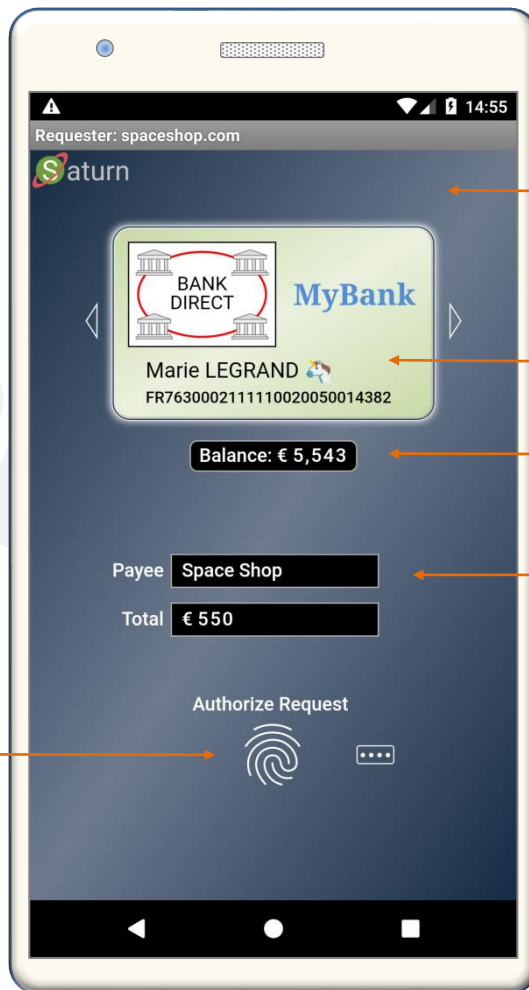
```
{
  "@context": "https://webpki.github.io/saturn/v3",
  "@qualifier": "PaymentClientRequest",
  "supportedPaymentMethods": [{
    "paymentMethod": "https://bankdirect.net",
    "payeeAuthorityUrl": "https://payments.bigbank.com/payees/86344"
  }, {
    "paymentMethod": "https://supercard.com",
    "payeeAuthorityUrl": "https://secure.cardprocessor.com/payees/1077342"
  }],
  "paymentRequest": {
    "commonName": "Space Shop",
    "amount": "550.00",
    "currency": "EUR",
    "referenceId": "2020100700000013",
    "timeStamp": "2020-10-07T08:31:44Z",
    "expires": "2020-10-07T09:02:00Z"
  },
  "receiptUrl": "https://spaceshop.com/receipts/2020100700000013j5IOEL2w9cWBFUwkbrFgjQ"
}
```

Payment method URLs to be matched against stored virtual cards



The Merchant (Payee) invokes the Wallet (after a user action) with a list of supported payment methods. Those who are matching the user's virtual cards will be shown in the Wallet UI to select from. The `payeeAuthorityUrl` properties are used to securely bind the Merchant's [request signature](#) to a particular payment method / network. The `paymentRequest` object contains the actual request data to be reflected in the wallet UI. The *optional* `receiptUrl` points to a Web address where an associated digital receipt will be published. To limit misuse, the `receiptUrl` is supposed to be randomized as well. See [Receipt Processing](#).

② Wallet Receives the `PaymentClientRequest`



Adapted to:

- Your Language
- Your Disability

Virtual Card Logotype & Account Selector (⇐ Swipe ⇒)

Optional: [Real-Time Account Balance](#)

UI Showing:

- Direct Payment
- Booking
- [Gas Station](#)
- Subscription
- Etc.

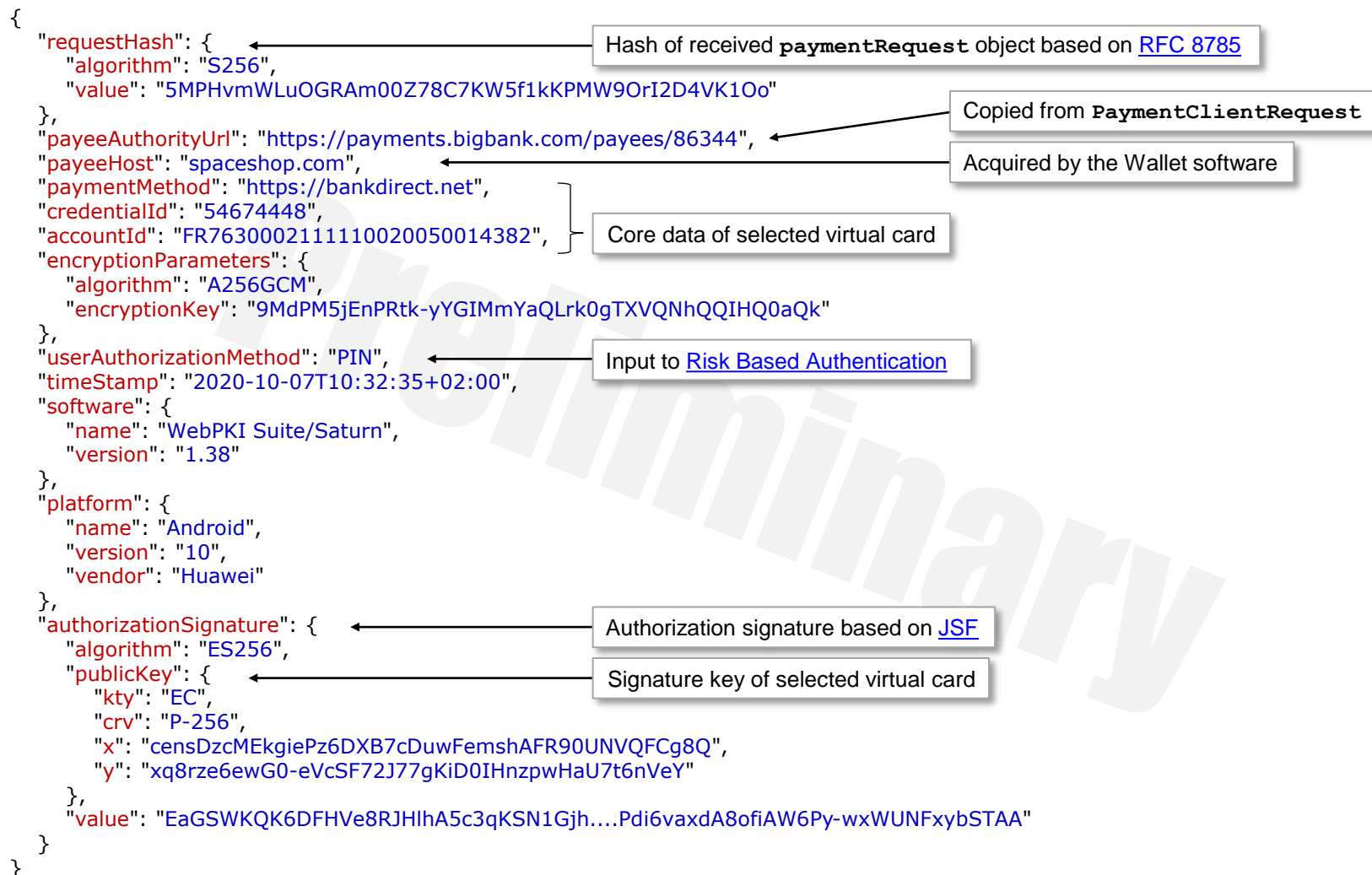


TEE Protected Keys

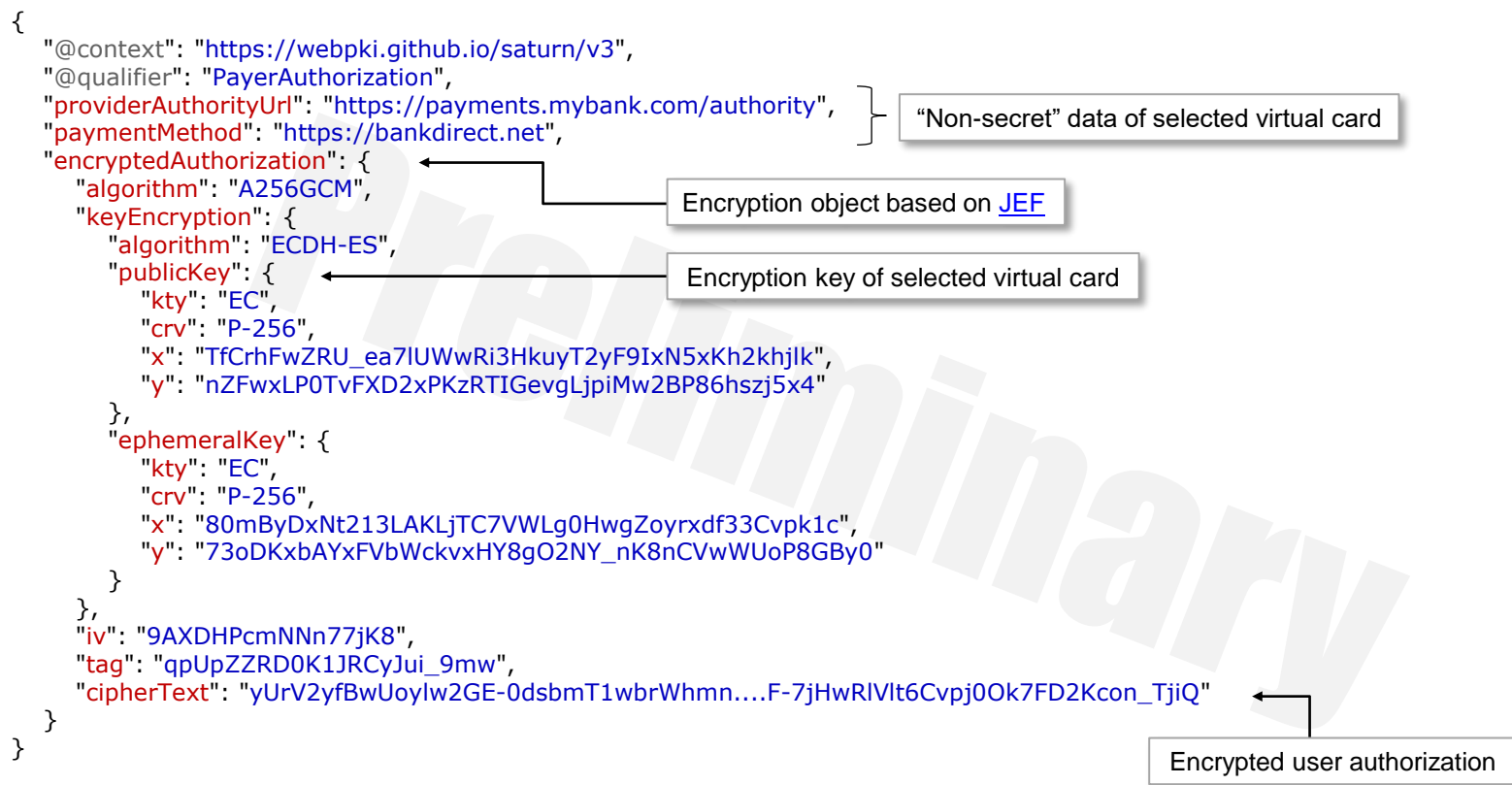
PIN or Biometric for User Authorization
(as defined by the *virtual card issuer*, not the Wallet)

When `PaymentClientRequest` has been received by the client, the Wallet user interface is launched. The authorization method may consist of a PIN but could also be a biometric option such as touching a fingerprint reader. The authorization is only used to unlock the Signature Key as described in next slide. Note that the Saturn authorization concept not only emulates payment cards, *but payment terminals as well*.

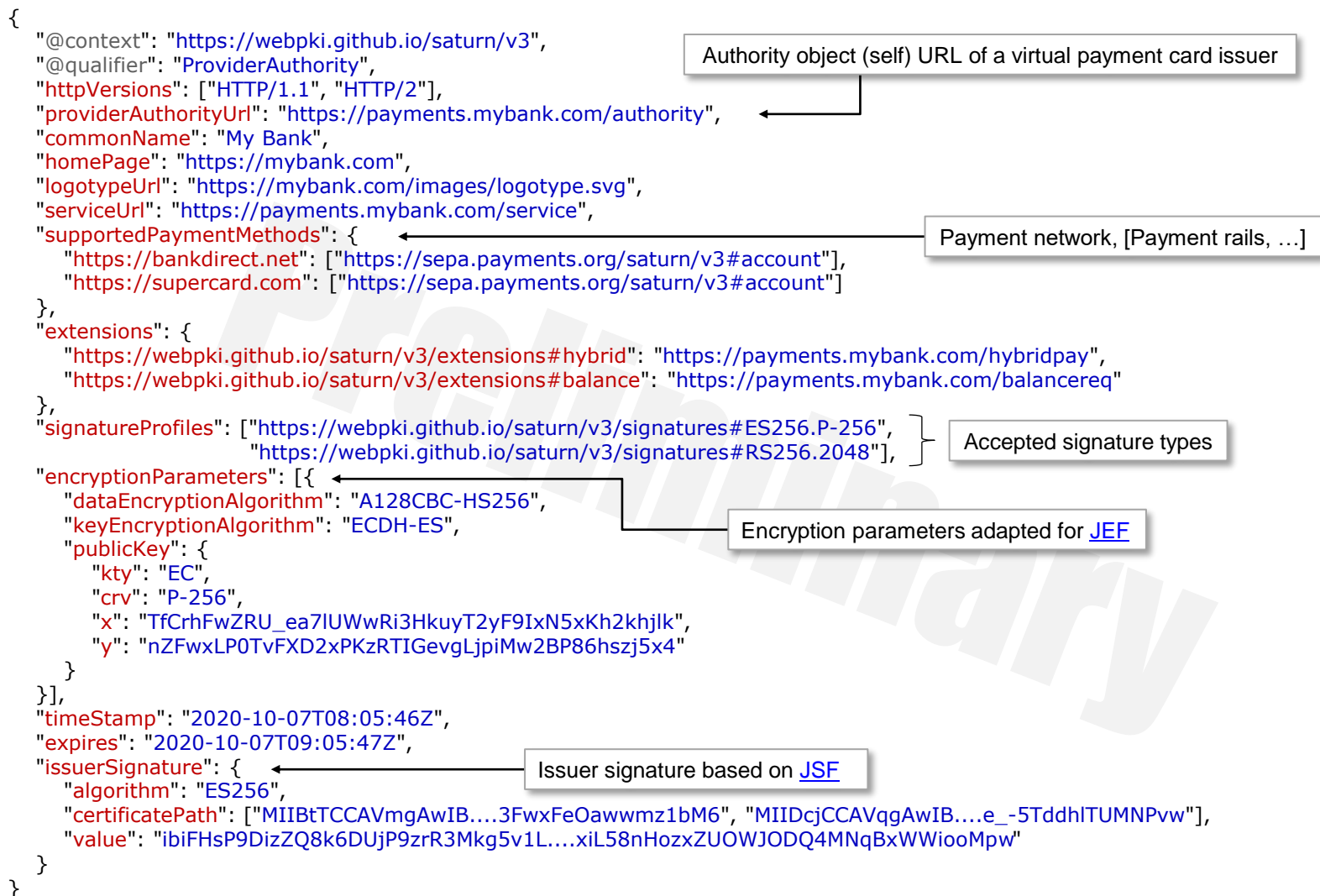
③ Internal Wallet Processing – Creation of Signed Authorization Data



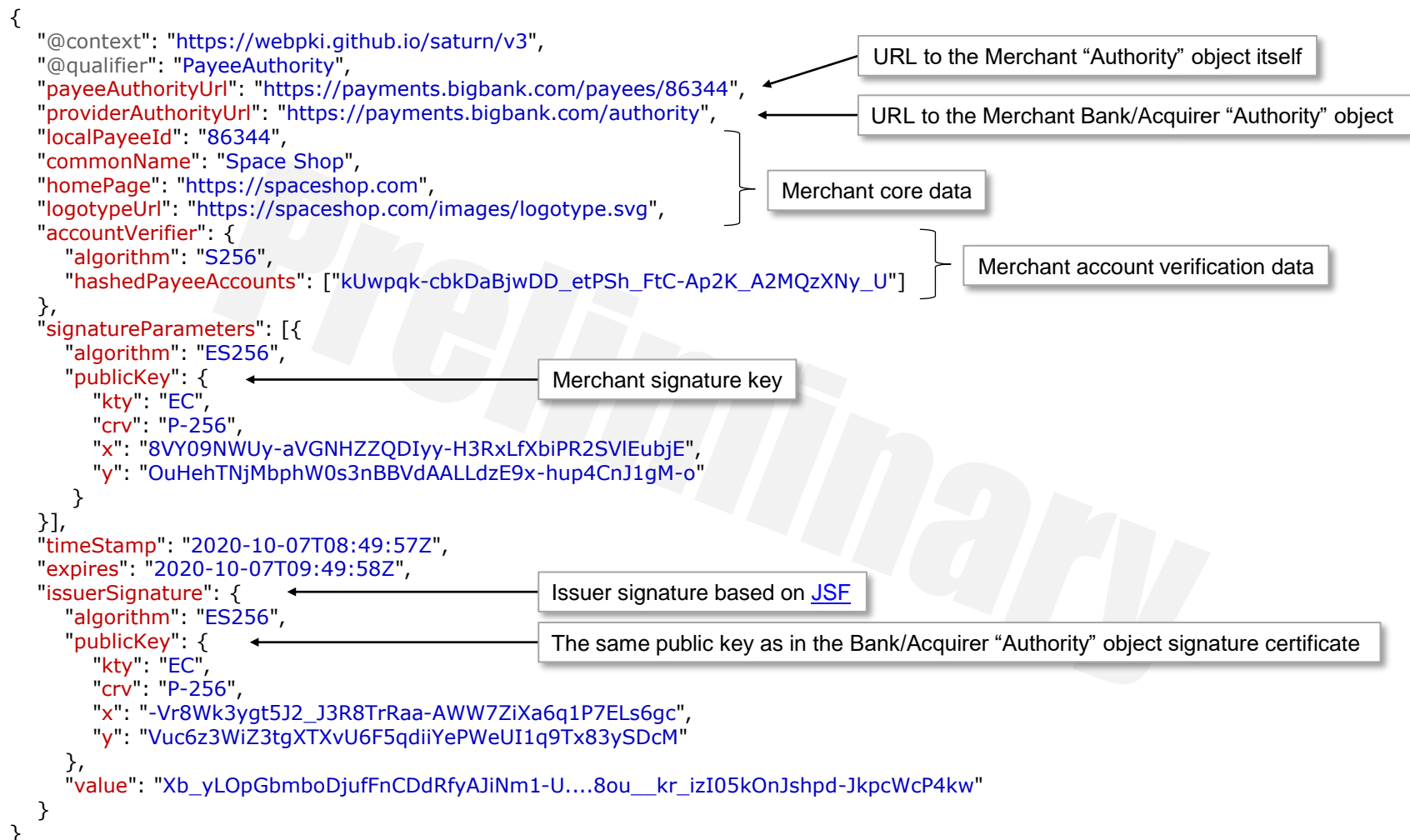
When the user has authorized the transaction the Signature (private) key associated with the selected card is used to sign a JSON object holding authorization data as follows : `requestHash` holds the *hash* of the `paymentRequest` object (see [PaymentClientRequest](#) slide), while `accountId` holds the actual Account ID of the selected card. For more information about `encryptionParameters`, turn to the slide [Risk Based Authentication](#).



PayerAuthorization messages provide the URL to the issuing bank's "Authority" object and the selected payment method which both are featured in virtual cards. This data is used by Merchants (Payees) for routing payment authorization requests to the applicable Bank. The actual authorization data (see previous slides) is *encrypted* by the Wallet using an *Issuer (not User) specific Encryption key* (with a *matching private key only known by the issuing Bank*), which also is stored in the virtual card. That is, Merchants do not get any information concerning Users (Payers) except their Bank and associated payment method.



“Authority” objects hold Keys, Payment methods, and URLs which are used by Merchants, Banks, and Acquirers as *Secure Distributed Entity Databases* creating the foundation for scalability including [Delegated Trust](#). “Authority” objects are *published on the Internet* and accessed by HTTP GET operations. A Bank/Acquirer “Authority” object is signed by the Bank/Acquirer itself. The **supportedPaymentMethods** object declares the payment methods understood by the Bank. The **encryptionParameters** are used by Issuers for encrypting user account data.



A Merchant (Payee) "Authority" object is like a *short-lived, automatically updated, X.509 certificate not requiring a CA*. Such an object is published on the address `payeeAuthorityUrl` hosted by the party (Bank or Acquirer) which vouches for the Merchant. If a Merchant is to be "revoked", the object is simply removed. To automate revocation checks, there is an `expires` attribute which also is used to clear caching of Merchant "Authority" objects. The `signatureParameters` list enable key renewals as well as validation of signatures using old keys.

④ Merchant Creates and Sends an **AuthorizationRequest** Message

```
{
  "@context": "https://webpki.github.io/saturn/v3",
  "@qualifier": "AuthorizationRequest",
  "recipientUrl": "https://payments.mybank.com/service",
  "payeeAuthorityUrl": "https://payments.bigbank.com/payees/86344",
  "paymentMethod": "https://bankdirect.net",
  "paymentRequest": {

    Copy of the paymentRequest object
  },
  "encryptedAuthorization": {

    Copy of the encryptedAuthorization object
  },
  "payeeReceiveAccount": {
    ←
  },
  "clientIpAddress": "220.13.198.144",
  "timeStamp": "2020-10-07T08:32:38Z",
  "software": {
    "name": "WebPKI.org - Payee",
    "version": "1.00"
  },
  "requestSignature": {
    ← Request signature based on JSF
    "algorithm": "ES256",
    "publicKey": {
      ← Merchant signature key
      "kty": "EC",
      "crv": "P-256",
      "x": "8VY09NWUy-aVGNHZZQDIyy-H3RxLfXbiPR2SVIEubjE",
      "y": "OuHehTNjMbphW0s3nBBVdAALLdzE9x-hup4CnJ1gM-o"
    },
    "value": "91wNxmoZt-TKUGD1R7prluueL2DSv9iZ....TqYipTRDXSewSlfWgnoxsTkjkw07pJog"
  }
}
```

Where the message is actually sent

URL to Merchant "Authority" object

Payment method (must match user authorization)

Sample data for a SEPA payment method:

```
"@context": "https://sepa.payments.org/saturn/v3#account",
"iban": "FR7630004003200001019471656",
"nonce": "nZFwxLP0TvFXD2xPKzRTIGevgLjpiMw2BP86hszj5x4"
```

The **nonce** is adding privacy protection to the account verification hashes published in the **accountVerifier** object of [PayeeAuthority](#).

The **AuthorizationRequest** is sent by a Merchant (Payee) to the **serviceUrl** of the "Authority" object given by the user's choice of payment card (method). See [providerAuthorityUrl](#). The inclusion of **payeeAuthorityUrl** enables the targeted User Bank to verify that the Merchant belongs to a known payment network.

⑤ User Bank Responds with an **AuthorizationResponse** Message

```
{
  "@context": "https://webpki.github.io/saturn/v3",
  "@qualifier": "AuthorizationResponse",
  "accountReference": "FR*0504",
  "encryptedAccountData": {
    Parameters removed for brevity...
    "cipherText": "okjRig8y97oHa0kw7buu17XcTZOZAtS1....XG4BoMqDwY0e2fxIGPSHzko5Hs_0UHXz"
  },
  "referenceId": "#0100345648",
  "logData": "CT100006",
  "timeStamp": "2020-10-07T08:32:38Z",
  "software": {
    "name": "WebPKI.org - Bank",
    "version": "1.00"
  },
  "authorizationRequest": {
    Copy of the entire AuthorizationRequest message
  },
  "authorizationSignature": {
    "algorithm": "ES256",
    "certificatePath": ["MIIBtTCCAVmgAwIB...3FwxFeOawwmz1bM6", "MIIDcjCCAVqgAwIB....e_-5TddhITUMNPvw"],
    "value": "b03W5RPCmoA2ARILtbdvCrlrAj5i0Cr4....hib3XUqun9KxpbL6Ig7i4pA_ko7Gf4yA"
  }
}
```

Optional short form of the user account to be featured in receipts etc.

Encryption object based on [JEF](#)

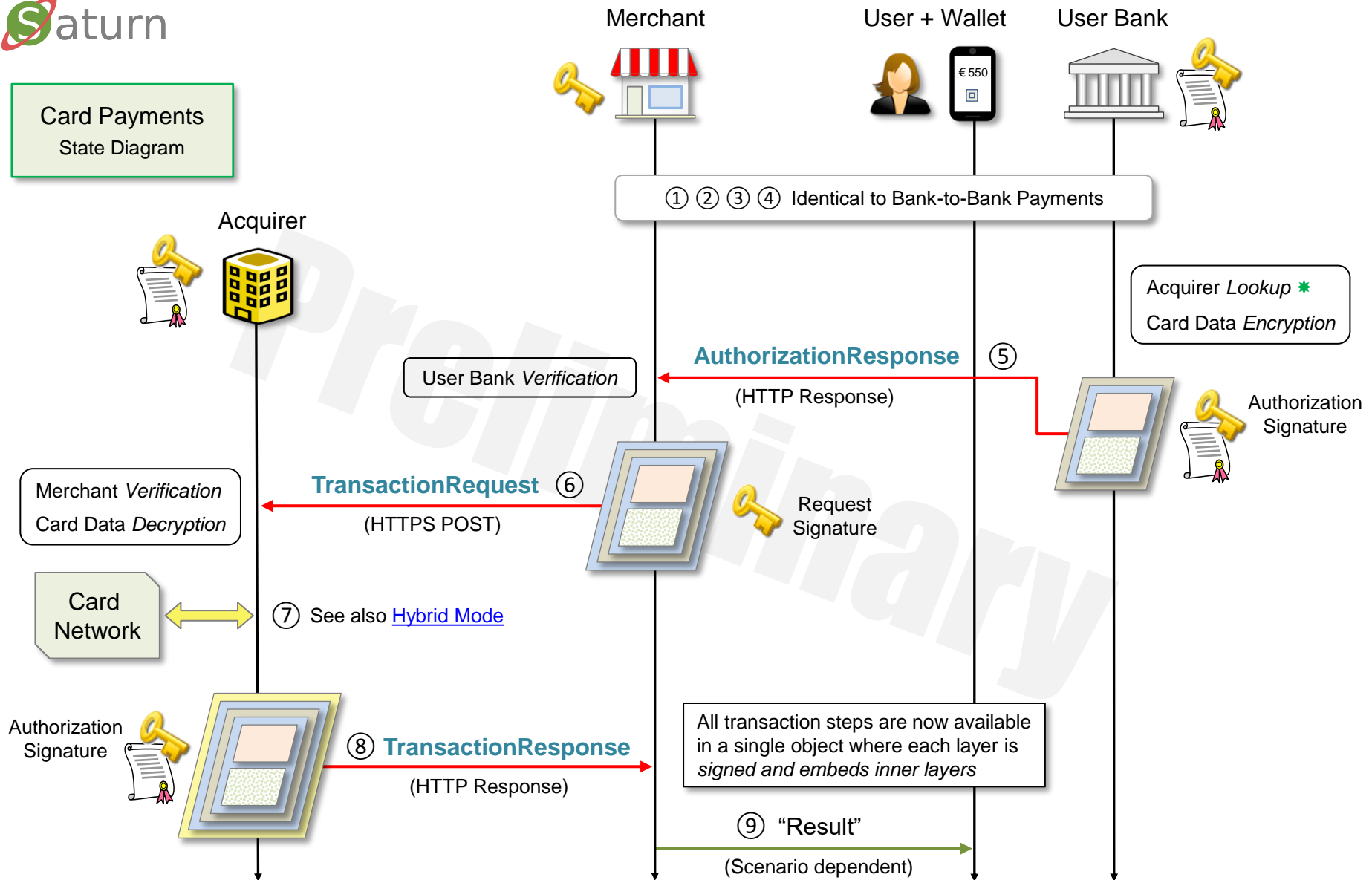
Encrypted user account data

Authorization signature based on [JSF](#)

User Bank certificate path

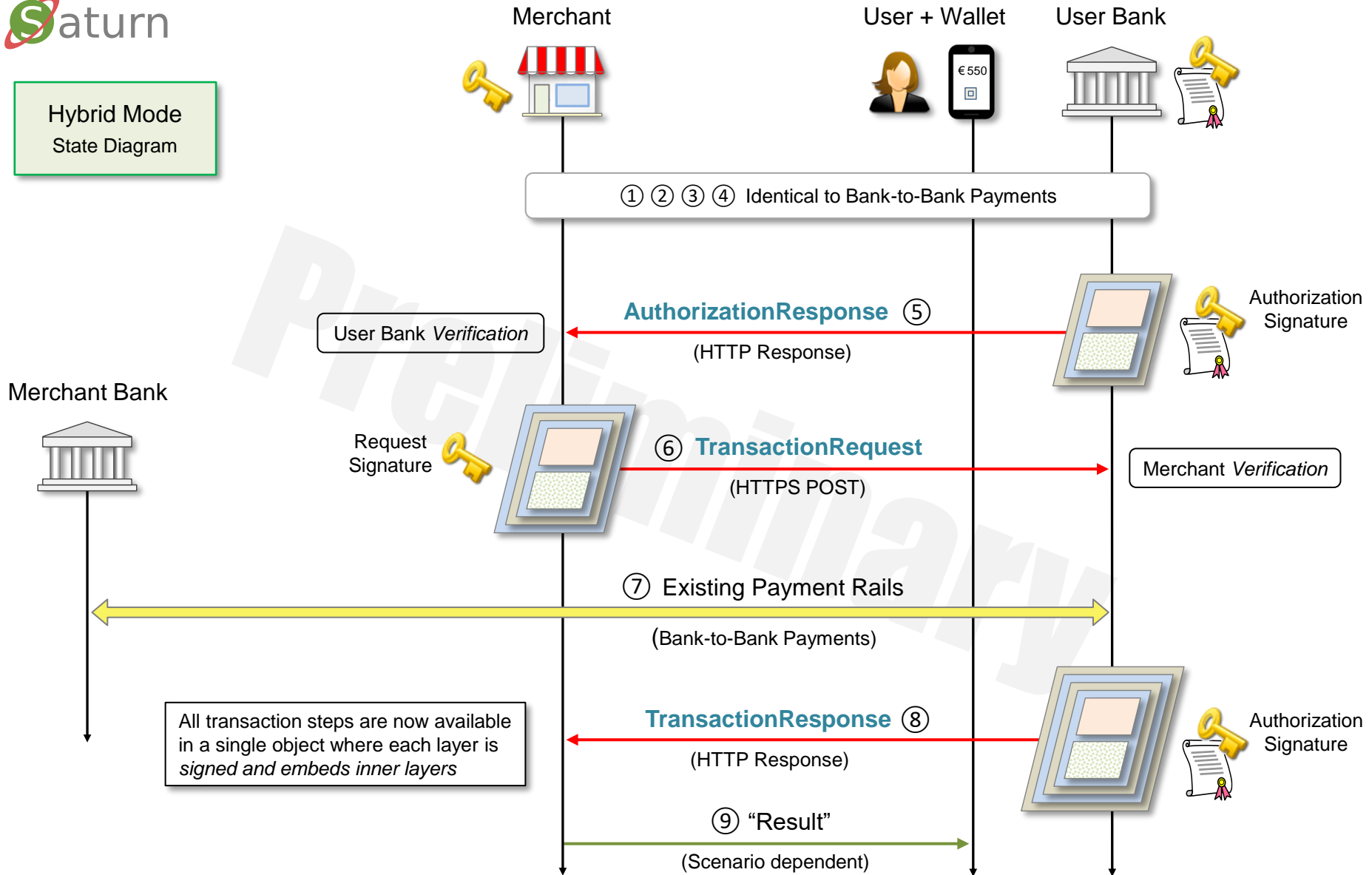
After received the **AuthorizationRequest**, User Bank performs an extensive list of operations to verify the validity of the request, including fetching the Merchant's (Payee) [PayeeAuthority](#) and [ProviderAuthority](#) objects. If the verification succeeds, User Bank responds with an **AuthorizationResponse** message which in addition to the original **AuthorizationRequest**, also holds the user's account data (ID) encrypted by the Merchant provider's encryption key. This information is used for [Card Payments](#) and [Refunds](#).

Card Payments State Diagram



* See [Authority Objects](#). The flow may stop after step #5 resulting in a *Secure Authorization Object* which *only* can be activated by another *Request*. This scheme supports hotel bookings, upfront reservations for automated gas stations, as well as recurring payments. The card data *Encryption* and *Decryption* processes enable standard card data to securely pass through Merchants from Issuers to Acquirers.

Hybrid Mode State Diagram



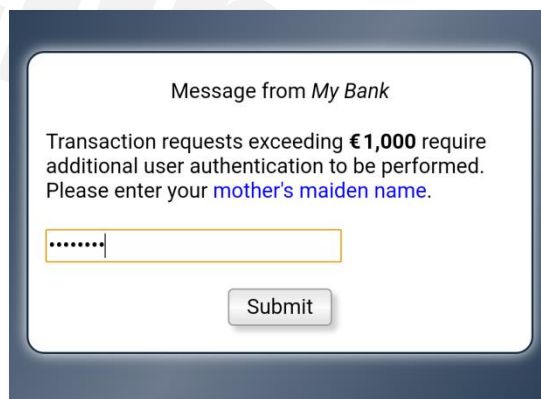
In the Hybrid mode traditional card payment methods are "emulated" including support for hotel bookings, upfront reservations for automated gas stations, as well as reoccurring payments. For three-corner payment schemes like PayPal and AliPay as well as for payments where the User and Merchant have the same bank, *step #7 is not applicable*.

```
{
  "@context": "https://webpki.github.io/saturn/v3",
  "@qualifier": "ProviderUserResponse",
  "encryptedMessage": {
    "algorithm": "A256GCM",
    "iv": "_K4Sgt5y1uKhwiSi",
    "tag": "Xmqyx5XZWmxSFfypag-y_A",
    "cipherText": "qXIsLsZ-zIxVIIv920dpxPmTOWGRghU_....fsxbw1LX61Tu6GbsSw1gXEcwkW8S4fOQ"
  }
}
```

Encryption object based on [JEF](#)

Encrypted message from User Bank

Decrypted and rendered by the Wallet
(non-normative sample)



Message from My Bank

Transaction requests exceeding €1,000 require additional user authentication to be performed. Please enter your [mother's maiden name](#).

.....

Submit

Occasionally a User Bank needs to inform the user of something related to an [AuthorizationRequest](#) like an account overdraft. Another situation requiring an action from the user's side is when the amount requested is unusually high or when "suspicious" user patterns have been identified. In both cases the request is *ignored* and the normal response is replaced by a message which only the (Wallet) user can read while still being delivered through the Merchant's "channel" to the Wallet. Privacy is accomplished by the User Bank encrypting the message contents with the symmetric key supplied in **encryptionParameters** (see [Creation of Signed Authorization Data](#)). This key is a random value generated for each Wallet invocation.

A private message like above (requiring an *action*), forces the Wallet adding the response to the user authorization data and then performing a full signed and encrypted User Authorization request again. This process may be repeated if necessary.

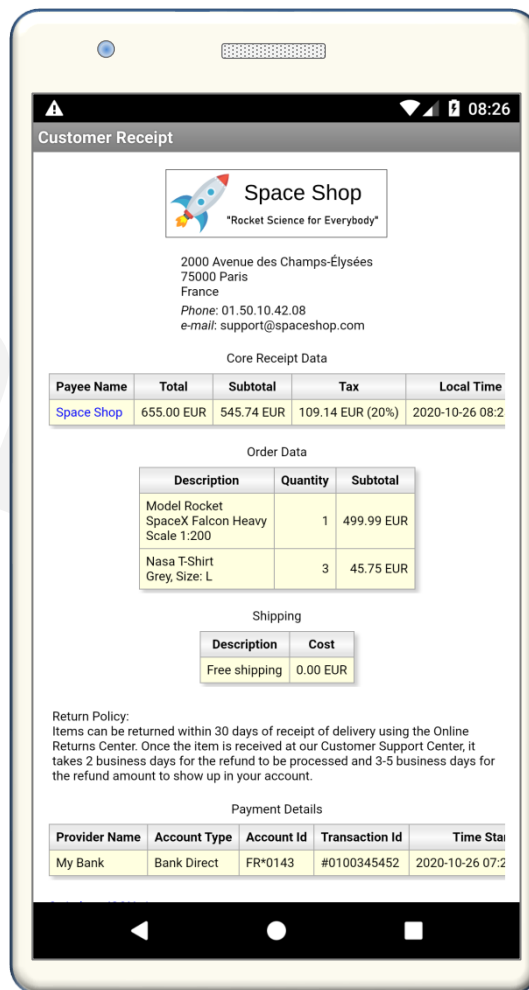
```
{
  "@context": "https://webpki.github.io/saturn/v3",
  "@qualifier": "RefundRequest",
  "recipientUrl": "https://payments.bigbank.com/refund",
  "amount": "550.00",
  "payeeSourceAccount": {
    "@context": "https://sepa.payments.org/saturn/v3#account",
    "iban": "FR7630004003200001019471656"
  },
  "referenceId": "#1000004",
  "timeStamp": "2020-10-07T22:07:50Z",
  "software": {
    "name": "WebPKI.org - Payee",
    "version": "1.00"
  },
  "authorizationResponse": {
    Copy of the entire AuthorizationResponse message
  },
  "requestSignature": {
    "algorithm": "ES256",
    "publicKey": {
      "kty": "EC",
      "crv": "P-256",
      "x": "8VY09NWUy-aVGNHZZQDIyy-H3RxLfXbiPR2SVIEubjE",
      "y": "OuHehTNjMbphW0s3nBBVdAALLdzE9x-hup4CnJ1gM-o"
    },
    "value": "rrqbEkm7ZM6uGjnIWg-3c2YHPXsDhzVz....FsMSNotc7QvAsvn2sTFJ-GGdN5Fx6EfQ"
  }
}
```

Request signature based on [JSE](#)

Merchant signature key

By including the account ID of the user (but *encrypted* with the Merchant's *payment provider* key), in the [AuthorizationResponse](#) object the Merchant can (aided by their payment provider), transfer money in the opposite direction. A [RefundRequest](#) message consists of an embedded [AuthorizationResponse](#) and an *amount*, signed by the Merchant. Note that the Merchant must send the refund request to *its own bank*. The Merchant's Bank is supposed to respond with (a here not shown) [RefundResponse](#) object.

Receipt Processing



If a **receiptUrl** is included in the [PaymentClientRequest](#) the Saturn wallet begins polling the URL after a successful payment operation. If succeeding, the wallet stores the [JSON-formatted receipt object](#) locally. Retrieved receipts can be viewed with a built-in receipt rendering application like shown above. Receipts may also be "synched" to a cloud service of the user's choice. Since receipts are digitally signed they can also be securely verified as being created by the merchant in question. Receipts objects may also be transferred to other parties like employers.

```
{
  "@context": "https://webpki.github.io/saturn/v3",
  "@qualifier": "Receipt",
  "status": "AVAILABLE",
  "referenceId": "2020102600000002",
  "timeStamp": "2020-10-26T07:25:33+00:00",
  "commonName": "Space Shop",
  "physicalAddress": ["2000 Avenue des Champs-Élysées",
    "75000 Paris",
    "France"],
  "phoneNumber": "01.50.10.42.08",
  "emailAddress": "support@spaceshop.com",
  "amount": "655.00",
  "currency": "EUR",
  "subtotal": "545.74",
  "tax": {
    "amount": "109.14",
    "percentage": "20"
  },
  "shipping": {
    "description": ["Free shipping"],
    "amount": "0.00"
  },
  "barcode": {
    "type": "CODE_128",
    "value": "2020102600000002"
  },
  "freeText": [
    "Return Policy:",
    "Items can be returned within 30 days of receipt of deli",
    "very using the Online Returns Center. Once the item is recei",
    "ved at our Customer Support Center, it takes 2 business day",
    "s for the refund to be processed and 3-5 business days for t",
    "he refund amount to show up in your account."
  ],
  "lineItems": [{
    "description": ["Model Rocket",
      "SpaceX Falcon Heavy",
      "Scale 1:200"],
    "quantity": "1",
    "subtotal": "499.99"
  }
}
```

```
{
  {
    "description": ["Nasa T-Shirt",
      "Grey, Size: L"],
    "quantity": "3",
    "subtotal": "45.75"
  },
  "paymentMethodName": "Bank Direct",
  "accountReference": "FR*0143",
  "payeeAuthorityUrl": "https://payments.bigbank.com/payees/86344",
  "payerProviderData": {
    "commonName": "My Bank",
    "providerAuthorityUrl": "https://payments.mybank.com/authority",
    "referenceId": "#0100345452",
    "payeeRequestId": "2020102600000002",
    "timeStamp": "2020-10-26T07:25:34+00:00"
  },
  "receiptSignature": {
    "algorithm": "ES256",
    "publicKey": {
      "kty": "EC",
      "crv": "P-256",
      "x": "8VY09NWUy-aVGNHZZQDIyy-H3RxLfxbiPR2SVIEubjE",
      "y": "OuHehTNjMbphW0s3nBBVdAALLdzE9x-hup4CnJ1gM-o"
    },
    "value": "_17N4_ZhdDgSBtV-Q9QzVqu....BrQijgVNJGR3c0-gTeB3NdV18zckw"
  }
}
```

Receipt signature based on [JSF](#)

Merchant signature key


```
{
  "@context": "https://webpki.github.io/saturn/v3",
  "@qualifier": "PaymentClientRequest",
  "supportedPaymentMethods": [
    Parameters removed for brevity...
  ],
  "paymentRequest": {
    "commonName": "Planet Gas",
    "amount": "200.00",
    "currency": "EUR",
    "nonDirectPayment": {
      "type": "RESERVATION",
      "subType": "GAS_STATION",
      "fixed": true,
      "expires": "2020-09-25T06:22:00Z"
    },
    "referenceId": "#1000017",
    "timeStamp": "2020-10-07T06:22:59Z",
    "expires": "2020-10-07T06:53:00Z"
  }
}
```

Additional object

User Interface Implications
(non-normative sample using rotating text)

Payee	Planet Gas
Total	€ 200 <i>Reserved, actual payment w</i>

Gas Station payments presume [Card Payments](#) or [Hybrid Mode](#) to be carried out.

Virtual Card Properties



Signature Key



Encryption
Public Key



Balance Key

- Payment Method URL
- **Bank Authority URL**
- Account ID
- Card Logotype
- ...

Retrieve the ProviderAuthority extensions object

```
{
  "@context": "https://webpki.github.io/saturn/v3",
  "@qualifier": "ProviderAuthority",
  ....
  "extensions": {
    "https://webpki.github.io/saturn/v3/extensions#balance": "https://payments.mybank.com/balance"
  }
  ....
}
```

The real time account balance option presumes that the virtual card also is equipped with a dedicated *balance key*. This key does unlike the *signature key* not require any user authorization because it can only be used to read the balance of the account associated with the virtual card. The process starts by the Wallet using the virtual card's "Bank Authority URL" to retrieve the [ProviderAuthority](#) object using an HTTP GET. In that object there should be an extensions object as shown above. After that the Wallet creates a **BalanceRequest** (as shown on the next slide), signs it with its balance key, and POSTs it to the URL provided by the specific balance extension.

BalanceRequest Message

```
{
  "@context": "https://webpki.github.io/saturn/v3",
  "@qualifier": "BalanceRequest",
  "recipientUrl": "https://payments.mybank.com/balance",
  "credentialId": "54674448",
  "accountId": "FR7630002111110020050014382",
  "currency": "EUR",
  "timeStamp": "2020-10-07T10:43:05+02:00",
  "requestSignature": {
    "algorithm": "ES256",
    "publicKey": {
      "kty": "EC",
      "crv": "P-256",
      "x": "kiTXwSkkNag5RPjFyPgSNmhPI_97qQPCbPQ2GFmMSP4",
      "y": "g8-4ymBfTg8o14EaJluDE8QmRfkrEy3M0VP61-TsoXg"
    }
  },
  "value": "jstrRDK-2n5FfpiAO896f1TKuc6wTSU.....5zHmAJWMkAIsnA0E679es5KgBiRIH0Ha70XejQUw"
}
```

Core data of selected virtual card

Request signature based on [JSF](#)

Balance key of selected virtual card

BalanceResponse Object

```
{
  "@context": "https://webpki.github.io/saturn/v3",
  "@qualifier": "BalanceResponse",
  "accountId": "FR7630002111110020050014382",
  "amount": "5543.00",
  "currency": "EUR",
  "timeStamp": "2020-10-07T10:43:07+02:00"
}
```

The resulting response object holds the current balance (available funds) of the specified account.

Q: Doesn't Saturn's Merchant-to-User Bank [AuthorizationRequest](#) introduce security risks?

A: Yes, similar risks as on-line bank applications which effectively are open to requests from *anywhere*. Security features include:

- Small and strict message format
- All messages are signed using industry standard cryptographic algorithms
- [AuthorizationRequest](#) is signed by the Merchant and vouched for by the Merchant's Bank/Acquirer through the [PayeeAuthority](#) object which also enables *verifiable* Merchant account data
- User signs a hash of **paymentRequest** with a key *which only the User Bank knows about*
- Only *mutually* signed authorizations are considered valid for processing
- Integral support for RBA (Risk Based Authentication)
- Tokenization of payment authorizations makes attacks on Merchant databases useless

Q: Can you trust the Wallet key storage?

A: Saturn depends on hardware backed keys like the AndroidKeystore.

Q: Doesn't Saturn effectively requires new client-side technology to fly?

A: Yes indeed, exactly like Apple Pay did. W3C's <https://www.w3.org/TR/payment-request/> is instrumental.

Q: Wouldn't it be better sending requests from the Wallet directly to the User Bank and then handing over responses to the Merchant?

A: Not really, see <https://cyberphone.github.io/doc/defensive-publications/payment-authorization-scheme.pdf> for more details on this matter which also is a *prerequisite for Wallet payment method independence*.

Q: Is Saturn a "Push" or "Pull" payment system?

A: *Saturn is not a payment system*, it is rather a scheme where a User *authorizes* Merchant-initiated *requests** which are transported back to the User Bank via the Merchant. That is, the actual payment system is not a part of the depicted scheme.

Q: How does Saturn relate to ISO 20022, ISO 8583, and SEPA?

A: Only the actual payment systems need payment-system specific security, format, names, conventions, and processing. The ability including payment-system specific data in [AuthorizationRequest](#) makes Saturn compatible with just about any payment system.

Q: How are Virtual Cards enrolled?

A: Virtual Cards would typically be enrolled from the User Bank's Web site using a secure enrollment protocol like: <https://cyberphone.github.io/doc/security/keygen2.html>

Q: Is Saturn a REST API?

A: No, because a payment is not a resource but rather an event with transactional behavior. In addition, messages are uniquely defined by their JSON contents making *digitally signing*, *embedding*, *debugging*, and *documenting* straightforward. Wallet communication is based on an *interactive*, *scenario-dependent*, *asynchronous*, *bi-directional* message channel. See: <https://cyberphone.github.io/doc/web/yasmin.html>

* Enabling Saturn supporting not only direct payments, but bookings, recurring payments, and automated gas station payments without modifications to the underlying payment system