Saturn Core Flow

The following slides describe such a solution coined “Saturn”, which borrows heavily from the currently only established payment authorization standard, EMV (used in 10 billon+ bank cards, as well as by Apple and Google Pay). Although EMV is defined by the international card network giants and is pretty dated with respect to technology, the actual *concept* has a lot of untapped potential.

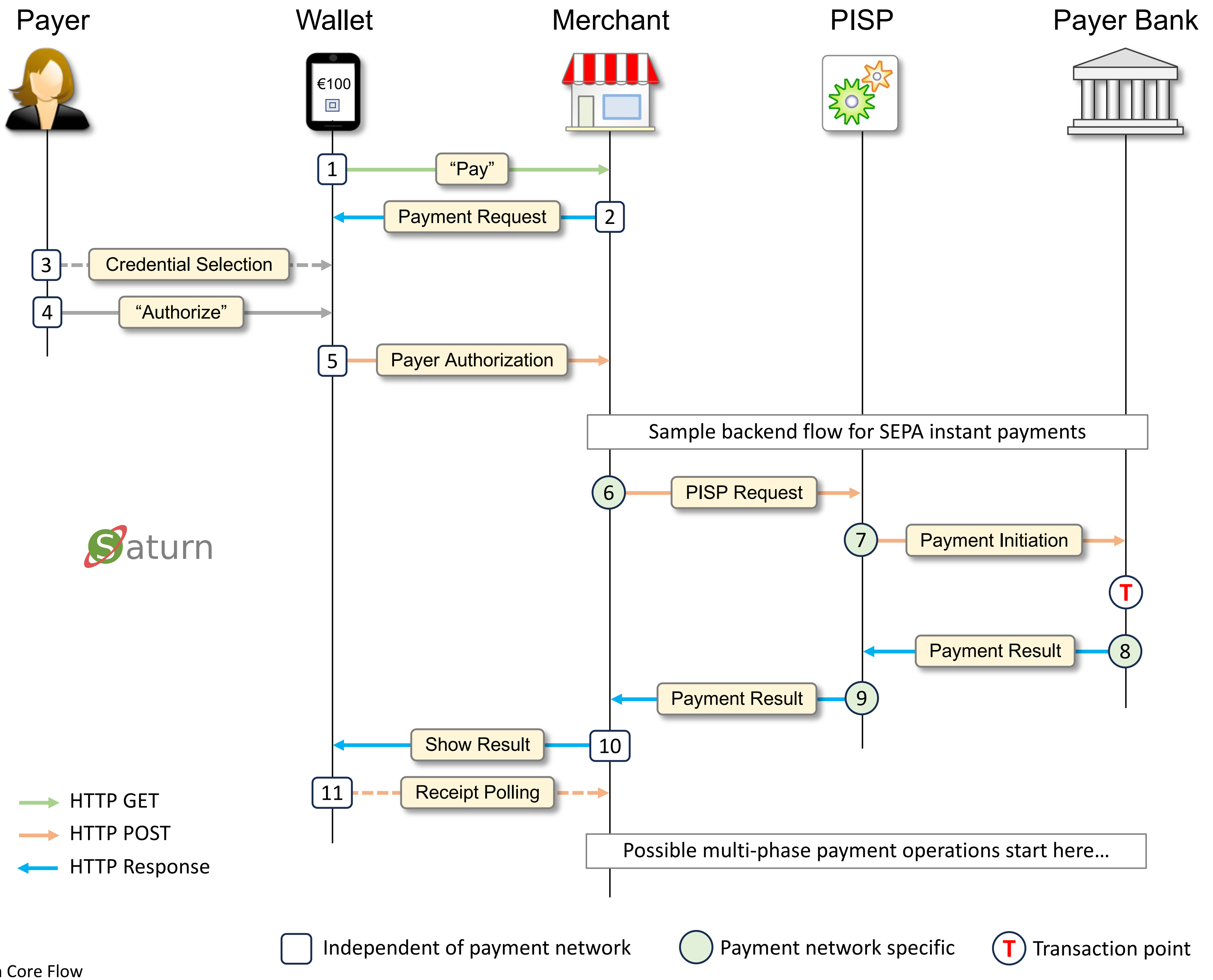
The primary extensions to EMV include:

- CBOR/JSON format rather than ISO 7816
- State-of-the-art cryptographic algorithms
- End-to-end encryption for GDPR and PCI compliance
- Universal account identifiers (PAN, IBAN, etc.)
- Payment network neutral (SEPA, VISA, etc.)
- Financial institution URLs rather than databases + BINs
- Support for account balance lookups
- Support for e-receipts
- Distinguishing between direct payments, deposits, subscriptions, gas station payments, etc.

However, the main feature of this design is in the relative simplicity of the wallet which is accomplished by limiting the wallet functionality to payer authorizations. This also increases the overall flexibility of the design since most of the functionality is offloaded to the “backend”.

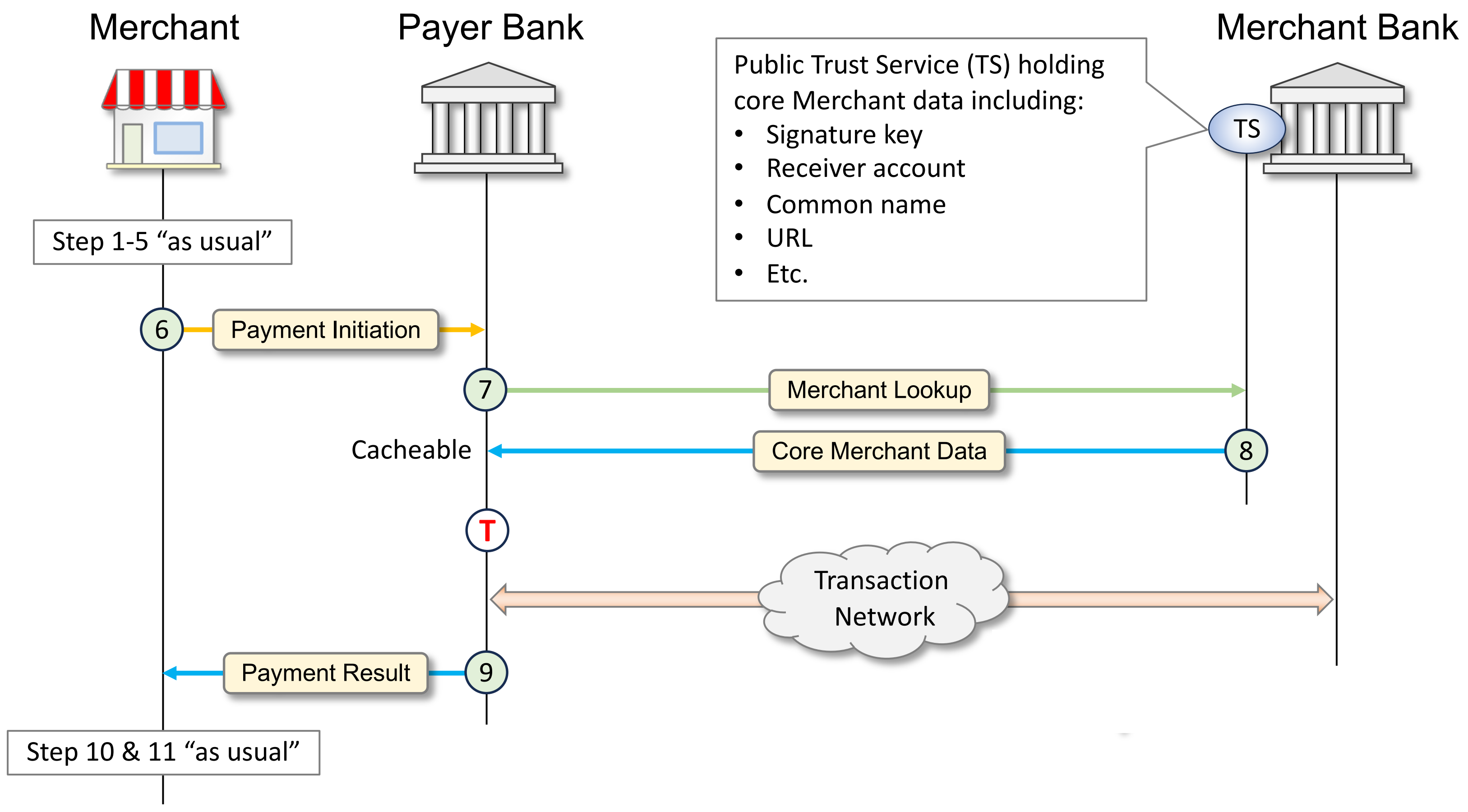
This system has been in the workings since 2015 and is currently testable at https://test.webpki.org/saturn
46 second UI demo: https://www.youtube.com/shorts/ntqQzuWGW14

Version: 0.1, 2024-05-25
Author: Anders Rundgren (anders.rundgren.net@gmail.com, https://www.linkedin.com/in/andersrundgren)


Payer
Wallet
Merchant
PISP
Payer Bank
€100
1
"Pay"
Payment Request
2
3
Credential Selection
4
"Authorize"
5
Payer Authorization
Sample backend flow for SEPA instant payments
6
PISP Request
7
Payment Initiation
T
Payment Result
8
Payment Result
9
Show Result
10
11
Receipt Polling
Possible multi-phase payment operations start here...
Saturn
HTTP GET
HTTP POST
HTTP Response
Independent of payment network
Payment network specific
T
Transaction point

1	The initial operation is to receive a **Payment Request** from the **Merchant**. This is usually accomplished by clicking a "Pay" button on a Web page, or accessing a URL given by a QR-code.
2	An *unsigned* **Payment Request** is received which in turn opens the **Wallet** UI.
3	*Optional*: The **Payment Request** does not only contains an amount, currency, and the name of the **Merchant**, but also a *list of accepted payment networks* (SEPA, VISA, etc.). Only virtual cards in the **Wallet** matching the list will be selectable in this step.
4	The **Payer** authorizes the payment through a biometric operation or by a PIN code.
5	After **Payer** authorization, the wallet **signs** (using a credential-specific key), the **Payment Request** as well as the data needed for clearing the payment including payer account, bank, and payment network. To preserve privacy, PII like account and key identifiers are **encrypted** by a public key shared by multiple clients. Only the **Payer Bank** can decrypt this information. The completed **Payer Authorization** object is returned to the **Merchant**.
6	Since the **Wallet** supports multiple payment networks, the **Merchant** must now select the proper method and route based on the received **Payer Authorization** object. The sample diagram shows the anticipated backend for SEPA instant payments. The **Merchant** creates a **PISP Request** document containing the **Payer Authorization** object, the **Merchant**'s receive account, and a timestamp. Then the **Merchant signs** the request and sends completed message to a **PISP** it has a business relation with.
7	The **PISP** authenticates the **Merchant** and validates the claimed name and account number. Next the **PISP** creates a **Payment Initiation** message including the **PISP Request**, **signs** it and sends it to the **Payer Bank**.
T	Now the **Payer Bank** should have all the data needed for performing the transaction. However, first the **Payer Authorization** object must be **decrypted** and the payer signature be **validated**. The account must also be checked for available funds.
8	Return a newly created **Return Status** for the transaction **signed** by the **Payer Bank**.
9	Propagating **Return Status**.
10	Show the result to the **Payer**.
11	If the **Payment Request** contained a receiptUrl, the **Wallet** should now try to retrieve the receipt by slowly polling the URL. https://cyberphone.github.io/doc/defensive-publications/e-receipts.pdf

Disruptive Approach – Eliminating Payment Intermediaries



The advantages of a *decentralized* authorization scheme include:
- True 4-corner model ⇨ Optimal business model for A2A transactions
- Unlimited scalability
- No single point of failure