2D Barcode Payment Protocol

Content

Revisions

Version	Date	Changes	Author
1.0	16.10.2014	-	Alain Diacon
1.1	22.10.2014	AES 128 bit, CBC, IV, zero termination, day example	Alain Diacon
1.2	18.12.2014	Added 2D-Barcode Ticket, new fields, etc.	Alain Diacon

Introduction

This document specifies the ticket printing and communication format used to enable smartphone payments for parking tickets with a 2D-barcode.

2D-Barcode Ticket

Barcode Format

The 2D-barcode printed on the ticket is a QR-Code with the following information:

```
Http Address?id=EncryptedTicketInfo
```

Http and *Address* are texts set in the EAL-Setup of the server.

Http is set to `http://` in the field „Image Replacement Text 1“. Setting this text enables the printing of the 2d barcode. Otherwise the standard image or text is printed on the ticket.

Address is set to the URL of the payment server in the field „Image Replacement Text 2“, for example like `www.subin.cn`

EncryptedTicketInfo is a hex string resulting from the encryption of the ticket information, see next section. The encryption is the same as used by the communication protocol between the payment server and the parking server, see “Communication Protocol”.

Example:

```
http://www.subin.cn?id=7a316df8e29a....
```

Ticket Information

The ticket information included in the *EncryptedTicketInfo* is:

```
c=subin&t=Ticket Number&s=Server IP
```

Ticket Number is the ticket number like it is printed in on the ticket.

Server IP is the IP address of the parking server. It is set in the field „Image Replacement Text 3“ of the EAL-Setup.

Example:

```
c=subin&t=1234.1234.1234&s=10.10.10.10
```

Communication Protocol

When a customer wants to pay a ticket, the payment server and the parking server need to exchange informations about the ticket and the payment. This is done with HTTP GET requests with AES-encrypted parameters and responses. The key length is 128 bits (AES default), if a shorter key is used, the key is padded with 0 bits on the right. Block mode is CBC (cyclic block chaining). The initialization vector (IV) is not used, it is all 0 bits.

Request Format (Payment Server -> Parking Server)

The encrypted strings are transmitted as an URL parameter:

```
http://Server Address/2dbarcode?req=Encrypted Parameters
```

Example:
`http://10.10.10.10/2dbarcode?req=ff72ba813b1a....`

The *Encrypted Parameters* is a string of hex digits that represent the bytes resulting from the encryption of the clear-text parameter string. The clear text is encrypted including a termination '0'-byte, such that the length is known after decryption. The first parameter is always a sequence number. Additional parameters are separated by '`&`':

```
Seq=Sequence Number&OtherParam=OtherValue
```

Sequence Number is calculated as follows:

```
(Days*1000000 + Inc)%(2^32)
```

Days is the number of days since 01.01.2014, so from 01.01.2014 00:00 till 23:59 days is 0, from 02.01.2014 00:00 till 23:59 days is 1, etc…

Inc is a number that is incremented for every new request, and it is reset to 0 at midnight, when *Days* is incremented.

`%(2^32)` means the number is truncated to 32 bits.

Using 32 bit unsigned integers, the received sequence number is validated as follows:

```
0 < Sequence Number – Last Sequence Number  < 2000000
```

If the *Sequence Number* is valid like above, the server carries out the action and produces an answer. If the request is an exact duplicate of the last request, including the *Sequence Number*, it just repeats the answer for the last request without carrying out the action a second time. In all other cases a `HTTP 403 Forbidden` response is sent.

If a server is restarted, the first sequence number sent, and the *Last Sequence Number* are initialized with *Days* like before and with *Inc* set to

```
Minutes since Midnight*1000000/(24*60)
```

This makes the sequence number valid without having to store the last number.

The purpose of this *Sequence Number* is to prevent "Record and playback" attacks. Without it, somebody could for example record a payment message and send it again at a later time, which would enable a ticket to exit much later than it was originally paid for.

Answer Format (Parking Server -> Payment Server)

The answer is encrypted and converted to hex like the *Encrypted Parameters*, and then sent as HTTP Body. The *Sequence Number* parameter from the request is repeated in the answer.

Price Request

The payment server asks the parking server for the price of a ticket. The parameter string is:

```
Seq=Sequence Number&Request=Request Type&Ticket=Ticket Number
```

Request Type is `TicketPrice` in this case.

Ticket Number is the ticket number as printed on the ticket.

Example:
```
Seq=123000678&Request=TicketPrice&Ticket=1234.1234.1234
```

Price Answer

If the parking server could successfully calculate the price it produces this answer:

```
Seq=Sequence Number&Price=Price&Ticket=Ticket Number&Time=Time&Entry=Entry
Time&PaymentNr=Payment Number&Discount=Discount
```

Price is the original price to be paid, not including discounts.

Time is the time to be paid in minutes, it is the time from the entry until now for the first payment (when *Payment Number* = 1), and the time from the last payment until now for following payments (when *Payment Number* > 1).

Entry Time is when the ticket was produced at the entry, format `dd.mm.yyyy hh:mm`

Payment Number is incremented with each payment for this ticket.

Discount is the total discount amount for this ticket. The customer needs to pay the amount *Price* – *Discount*.

Example:
```
Seq=123000678&Price=12.50&Ticket=1234.1234.1234&Time=234&Entry=12.10
.2014 07:55&PaymentNr=1&Discount=3.50
```

If there was an error the parking server produces the answer:

```
Seq=Sequence Number&Error=[Error Number]  Error Text
```

Error Number identifies the error.

Error Text is a description of the error, for example `Ticket not found`

Error Number and *Error Text* are defined in the database table “TCode” (the entries with Type=PriceError) and “TTranslate” (Name = Name from “TCode”).

Example:
```
Seq=123000678&Error=[1] Ticket not found
```

Payment Request

The payment server sends this message to the parking server when a ticket has been successfully paid:

```
Seq=Sequence Number&Request=TicketPayment&Ticket=Ticket
Number&Amount=Amount
```

Amount is the amount paid by the customer.

Example:
```
Seq=123000679&Request=TicketPayment&Ticket=1234.1234.1234&Amount=9.0
0
```

Payment Answer

If the parking server could successfully store the payment it produces this answer:

```
Seq=Sequence Number&PaymentNr=Payment Number
```

Payment Number is incremented with each payment for this ticket.

Example:
```
Seq=123000679&PaymentNr=1
```

If there was an error it produces the answer:

```
Seq=Sequence Number&Error=[Error Number]  Error Text
```