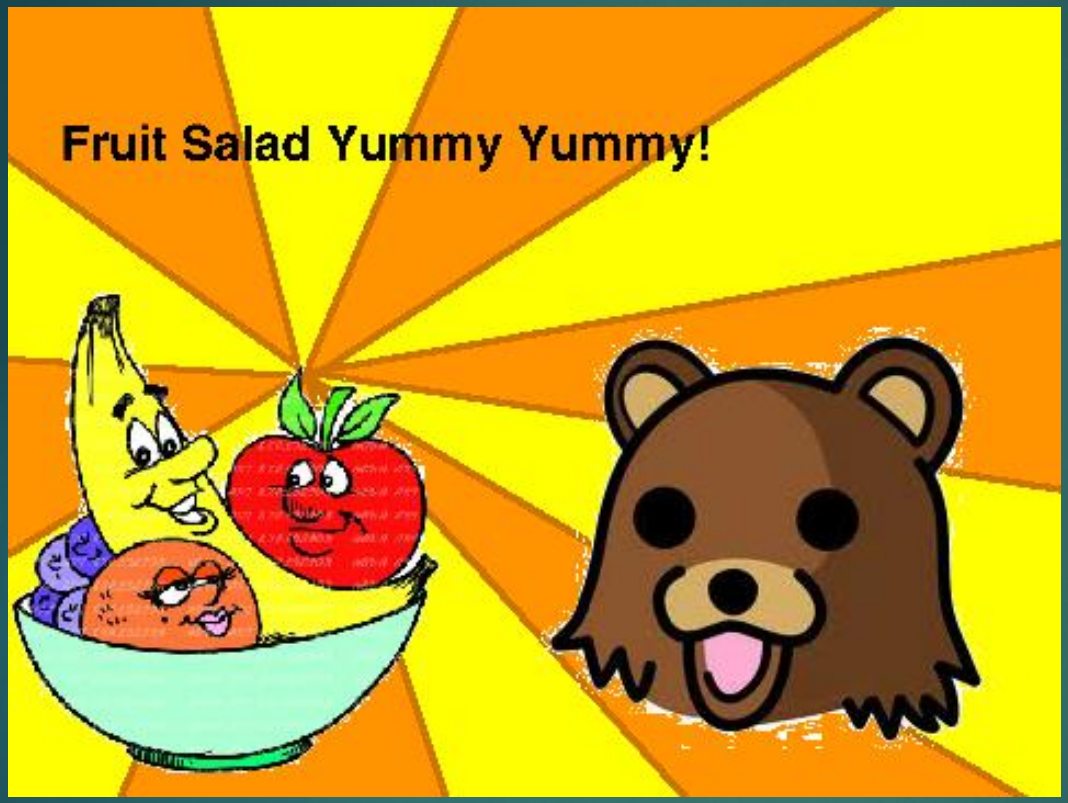


YUMMY YUMMY FRUIT SALAD:
AN ANALYSIS OF APPLE PAY

Image stolen from: https://scratch.mit.edu/projects/10813101/

$whoami

- Principle Consultant @ Payment Security Consulting
- Usually do PCI based work (hey, it's a living)
- Enjoy hardware stuff
- Also enjoy poking at iOS applications
- Did a silly badge this year for the Hardware Hacking Village – hope you got one!
- Trying to encourage more people to break stuff.
- See https://www.github.com/peterfillmore for some (badly) written code.

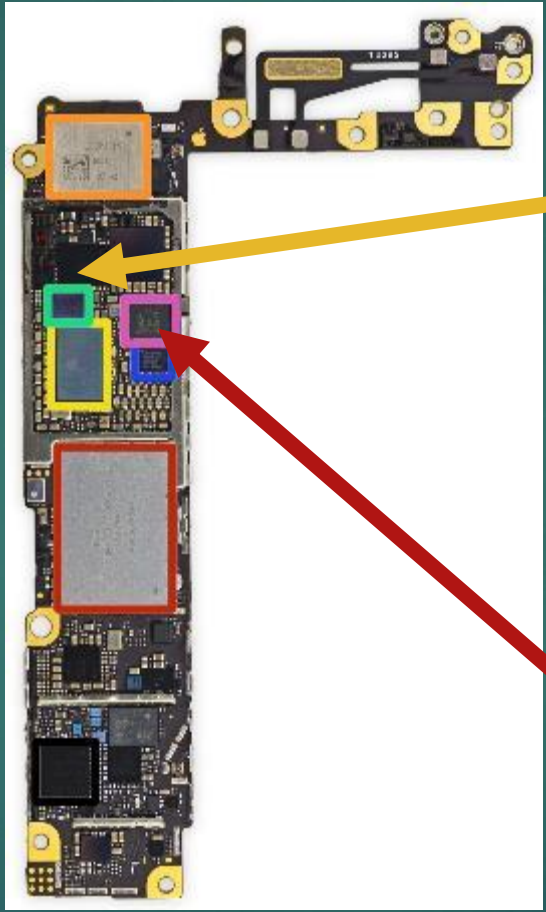
Agenda

- What is ApplePay exactly
- Apple Pay Architecture
- XPC – How the components communicate
- Registering a card
- Using a card
- Remote wiping of cards
- Issues
- Tools developed and used
- References

ApplePay

- Another in the long running examples of Apple “inventing” something
- Nothing but a lot of existing technologies bolted together with a fancy façade!
- Publicly available information is scant – people who work with the stuff are NDA’d heavily ☹
- Good thing I blew a grand on an iPhone 6

ApplePay consists of (iPhone 6):



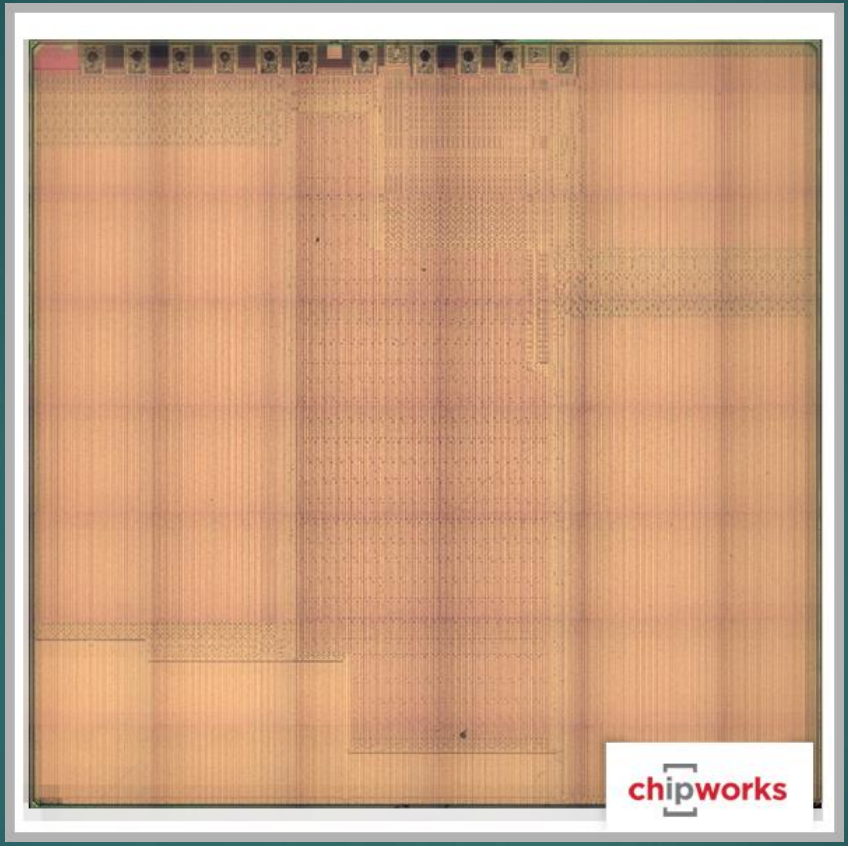
AMS AS3923
Power Booster

NXP 65v10

PN548

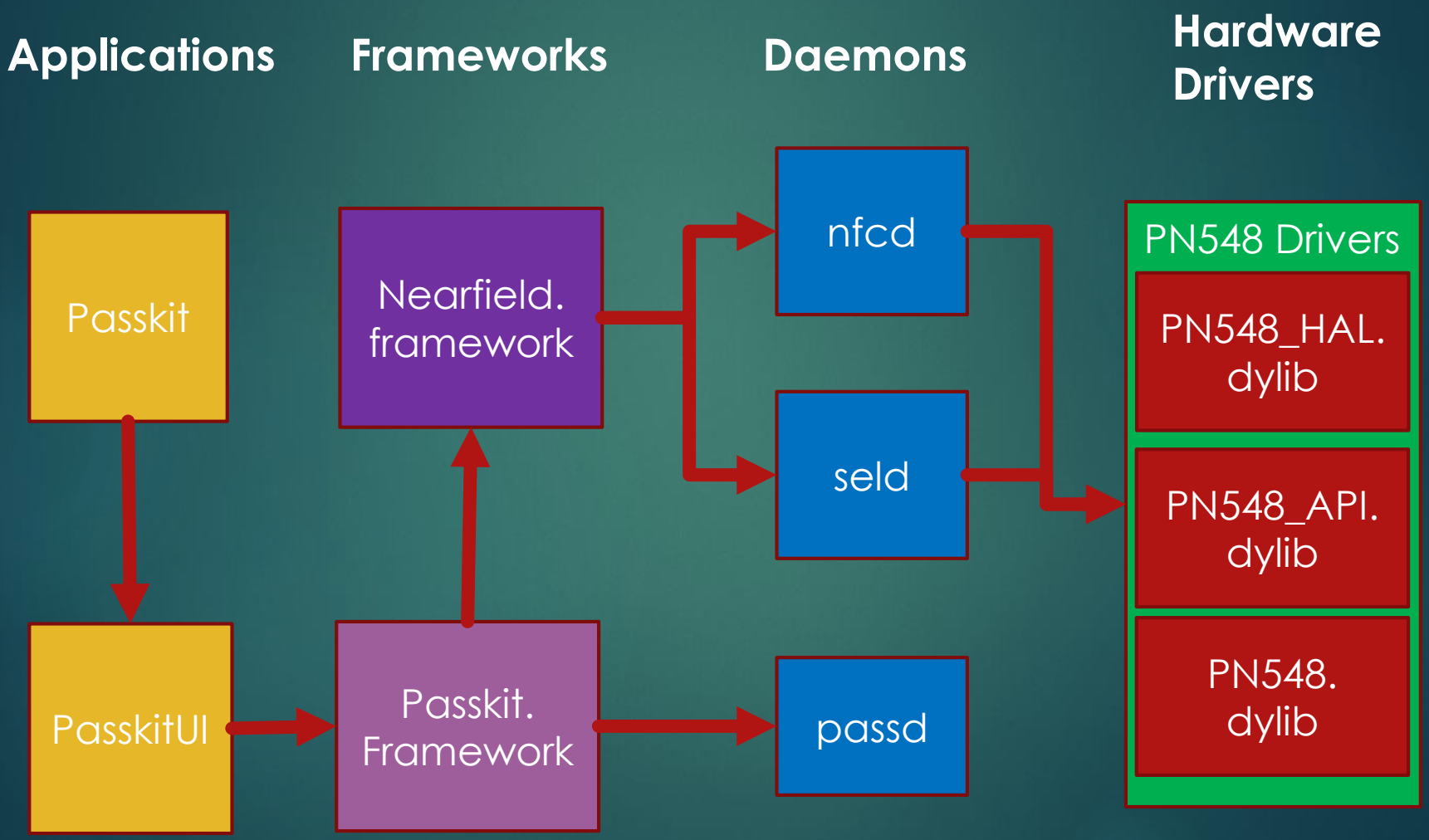
Secure Element

Secure Element



http://www.chipworks.com/about-chipworks/overview/blog/inside-the-iphone-6-and-iphone-6-plus

Software


Applications
Frameworks
Daemons
Hardware Drivers
Passkit
PasskitUI
Nearfield. framework
Passkit. Framework
nfcd
seld
passd
PN548 Drivers
PN548_HAL. dylib
PN548_API. dylib
PN548. dylib

XPC

- The primary method of inter-process communication in iOS
- Client/Server Model
- Designed to provide stability and privilege separation
- Passes serialized messages via a MACH message call

XPC Services used by ApplePay

nfcd

```
{ … Label = "com.apple.nfcd";
MachServices =
{ "com.apple.nfcd" = 1;};
ProcessType = Interactive;
Program = "/usr/libexec/nfcd";
UserName = mobile;}
```

seld

```
{…Label = "com.apple.seld";
MachServices =     {
"com.apple.seld" = 1;
"com.apple.seld.aps" = 1; };
Program = "/usr/libexec/seld";
RunAtLoad = 1;
UserName = mobile;}
```

passd

```
{ …
EnableTransactions = 1;
Label = "com.apple.passd";
….
MachServices =     {
"com.apple.passd.aps" = 1;
"com.apple.passd.bulletins" = 1;
"com.apple.passd.in-app-payment" = 1;
"com.apple.passd.library" = 1;
"com.apple.passd.payment" = 1;    };
POSIXSpawnType = Adaptive;
ProgramArguments =     (
"/System/Library/Frameworks/PassKit.framework/passd"    );
ThrottleInterval = 0;
UserName = mobile;}
```

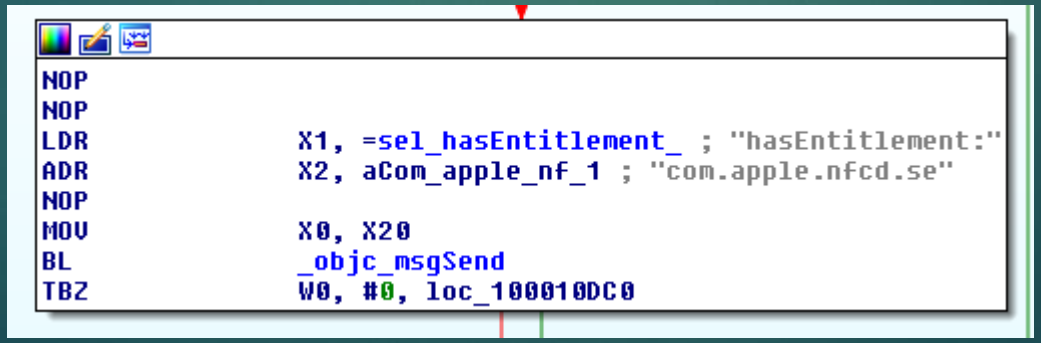
Client Must have the correct entitlements to use an XPC Service

NFCD

- com.apple.nfcd.se
- com.apple.nfcd.debug
- com.apple.nfcd.info

SELD

- com.apple.seld.debug
- com.apple.seld.cm



Example calling an XPC Service

Create Connection

```
xpc_connection_t connection =
    xpc_connection_create_mach_service("com.apple.nfcd", NULL, 0);
```

Set Handler

```
xpc_connection_set_event_handler(connection, ^(xpc_object_t object) {
```

Create XPC Object

```
xpc_object_t msgobject = xpc_dictionary_create(NULL,NULL,0);
xpc_object_t object = xpc_dictionary_create(NULL, NULL, 0);
xpc_dictionary_set_int64(msgobject, "Controller", 6);
```

Send object and get result

```
xpc_object_t reply =
    xpc_connection_send_message_with_reply_sync(connection, object);
```

XPC Sum-up

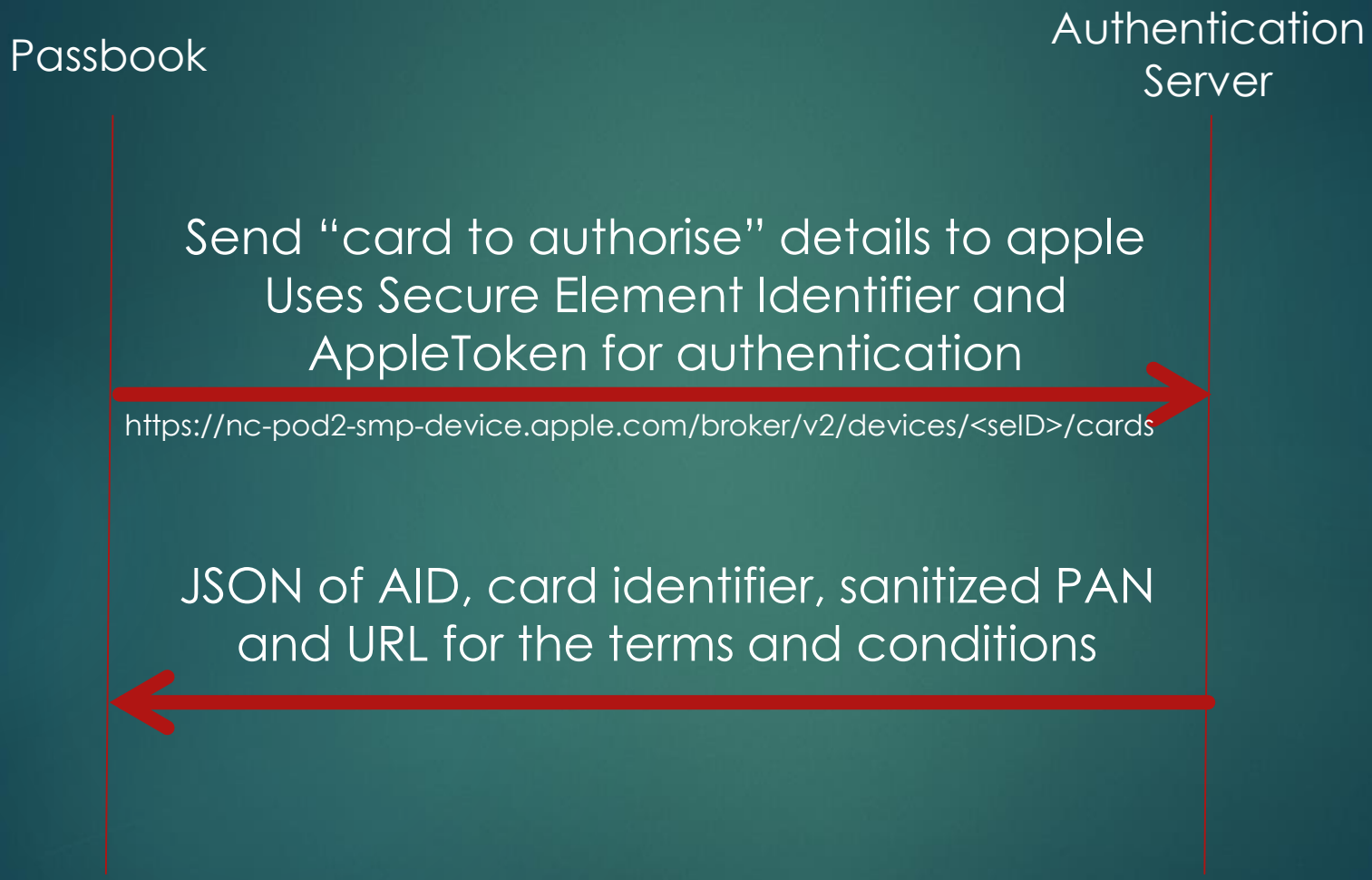
- Calling applications must have appropriate entitlements to use an XPC service.
- XPC services run under a nominated account ("mobile" in the case of NFC components)
- Harder to exploit from userland.

Better people then I have looked at this stuff:

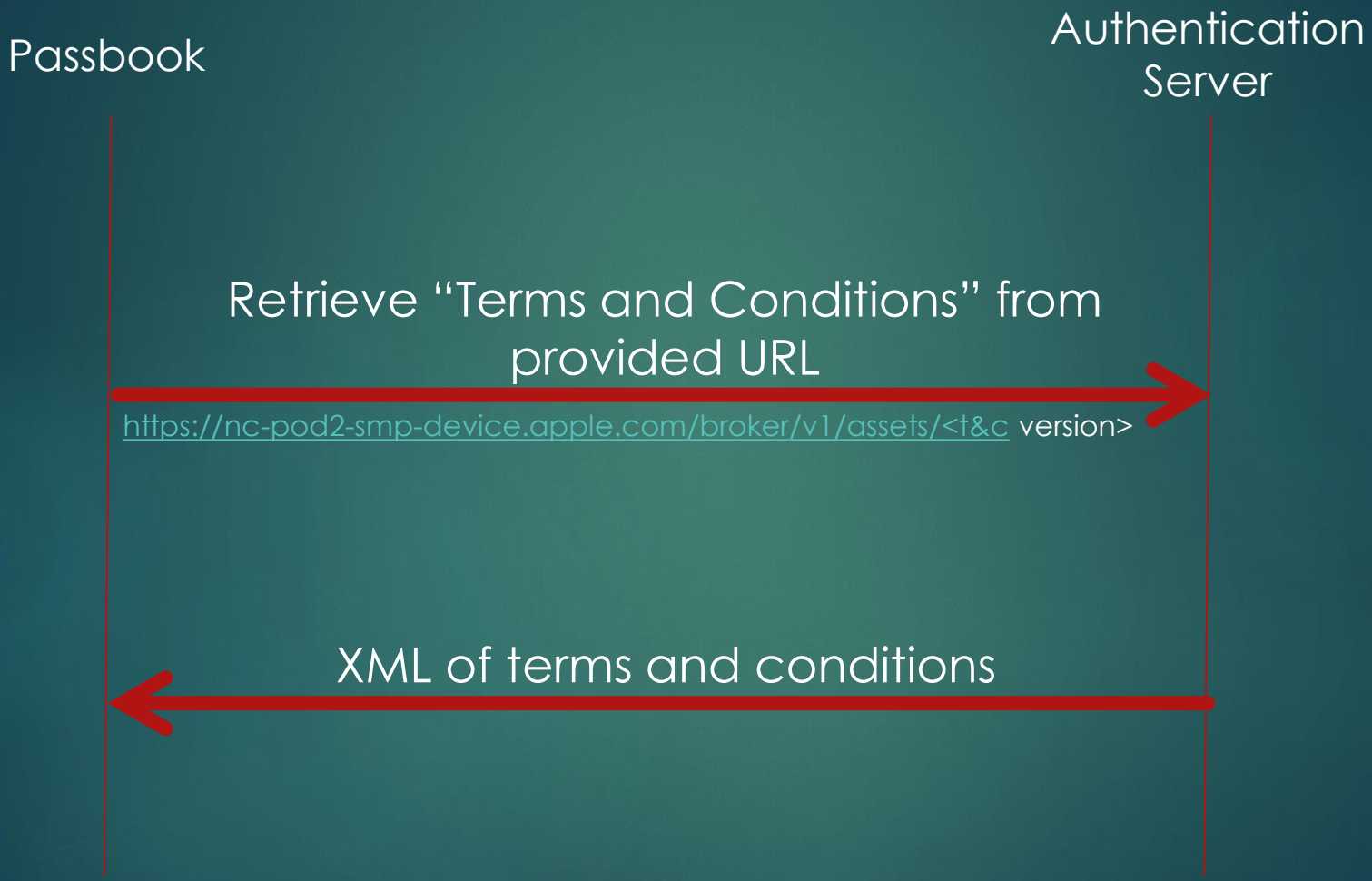
References:

- Ian Beer - http://googleprojectzero.blogspot.com.au/2015/09/revisiting-apple-ipc-1-distributed_28.html

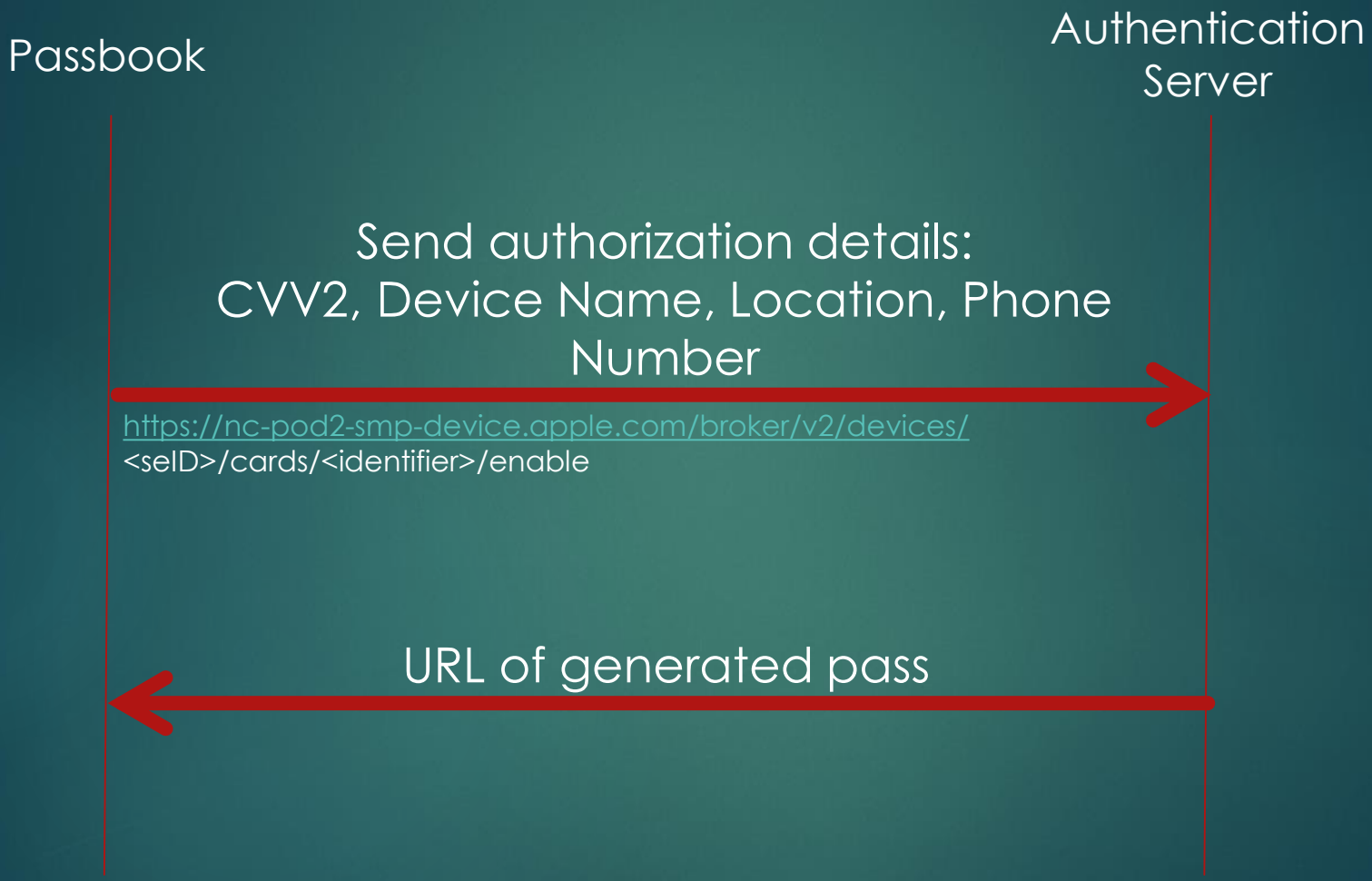
Enrolling a card – Step 1



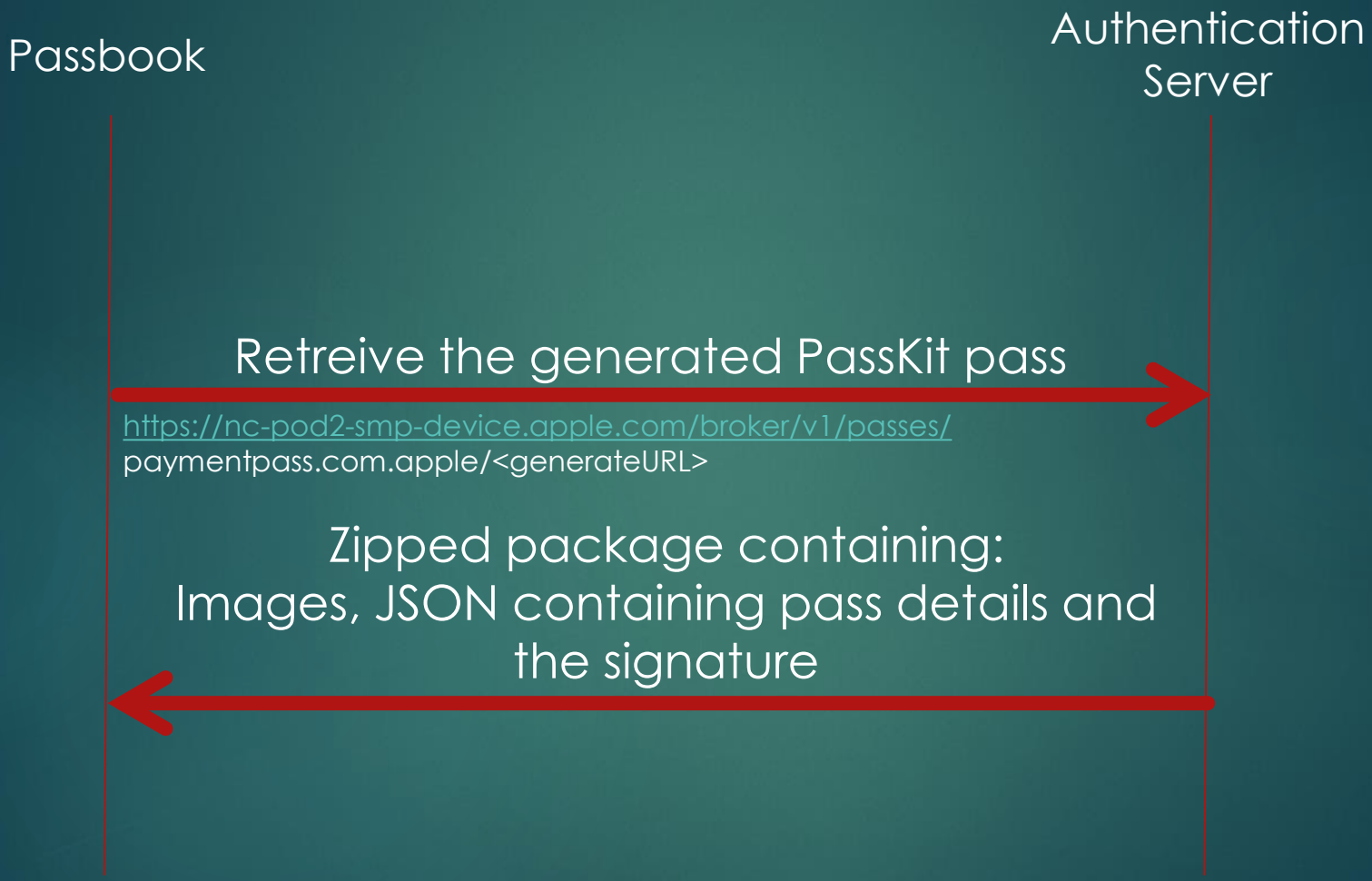
Enrolling a card – Step 2



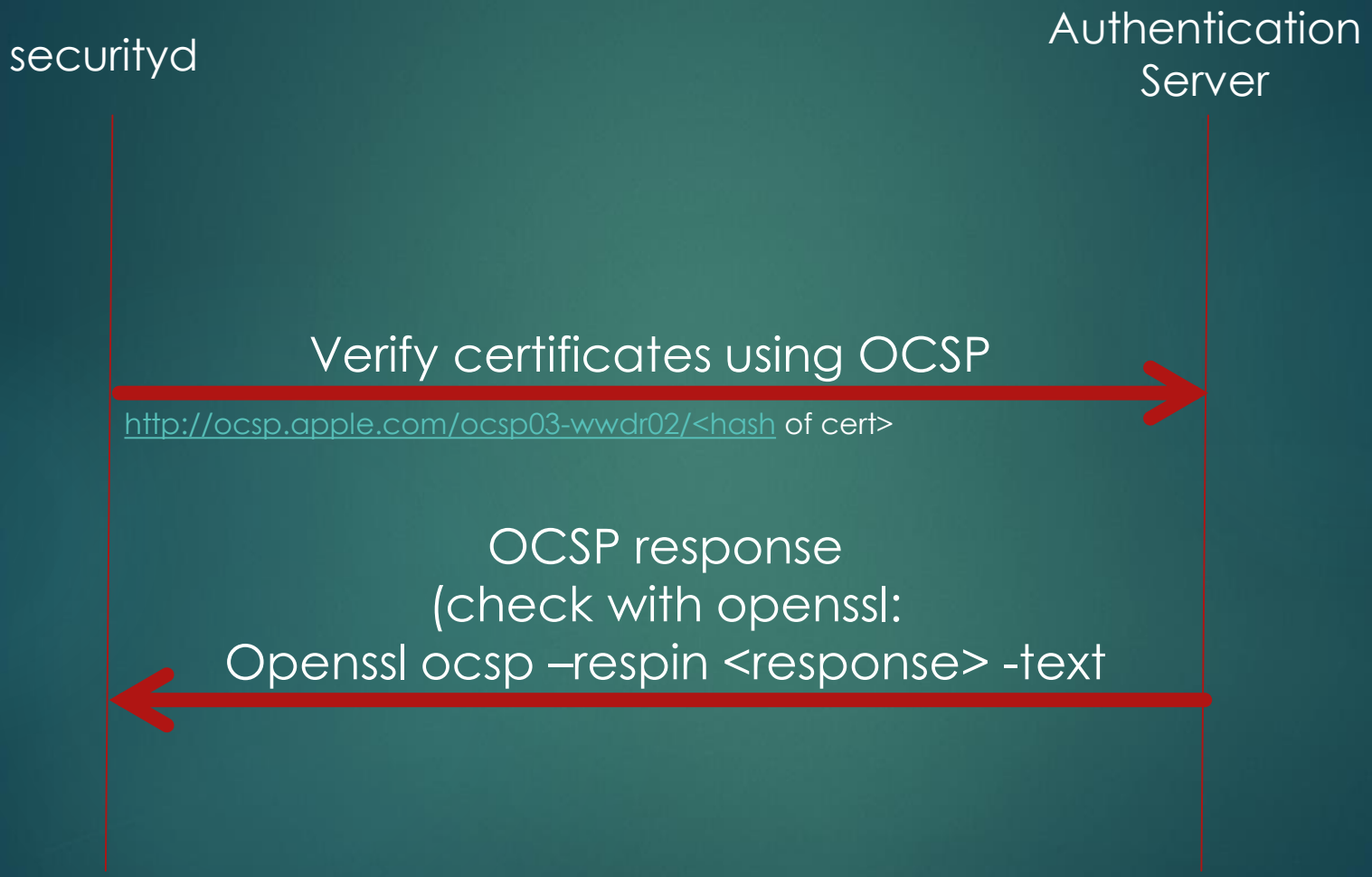
Enrolling a card – Step 3



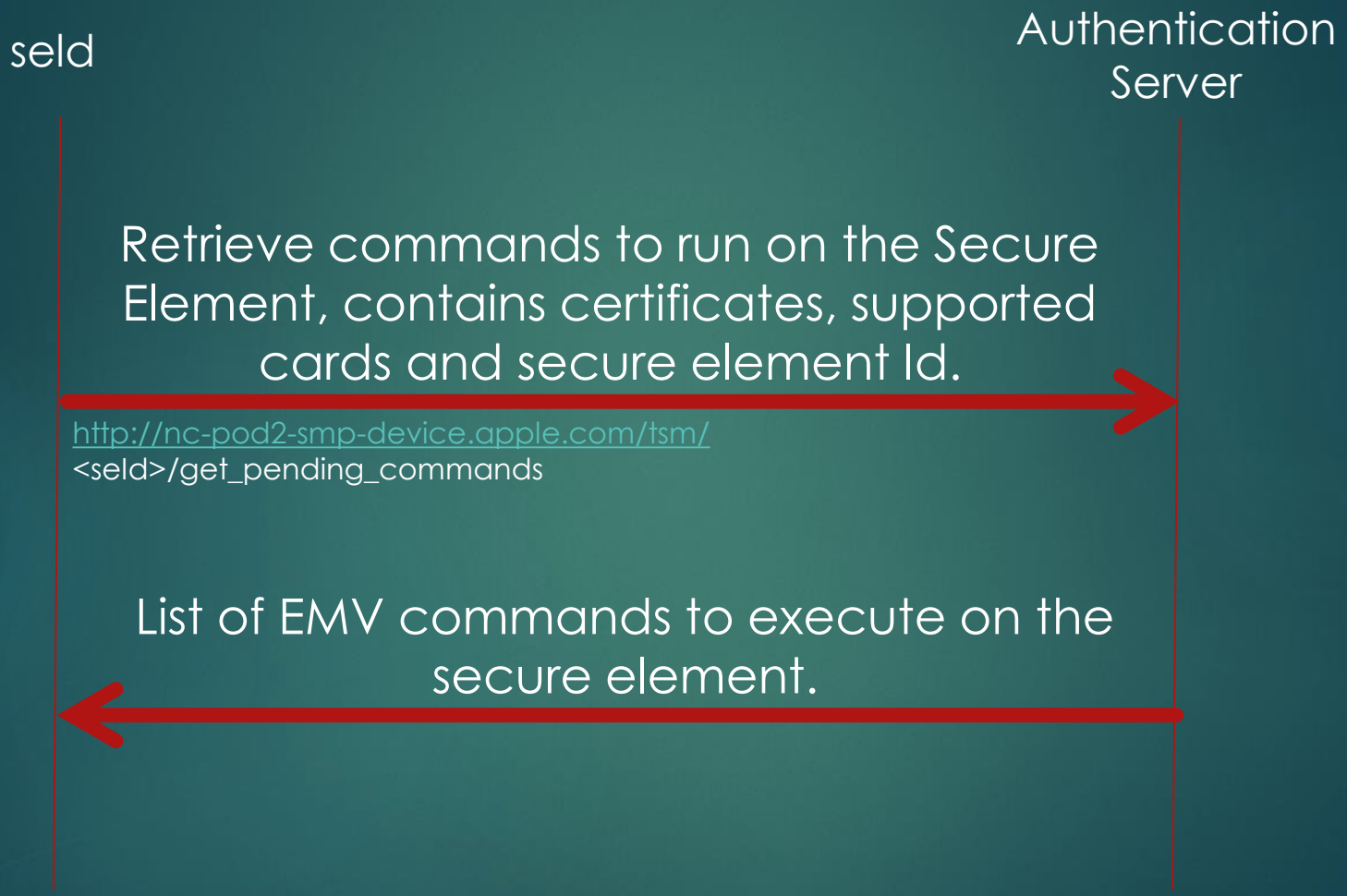
Enrolling a card – Step 4



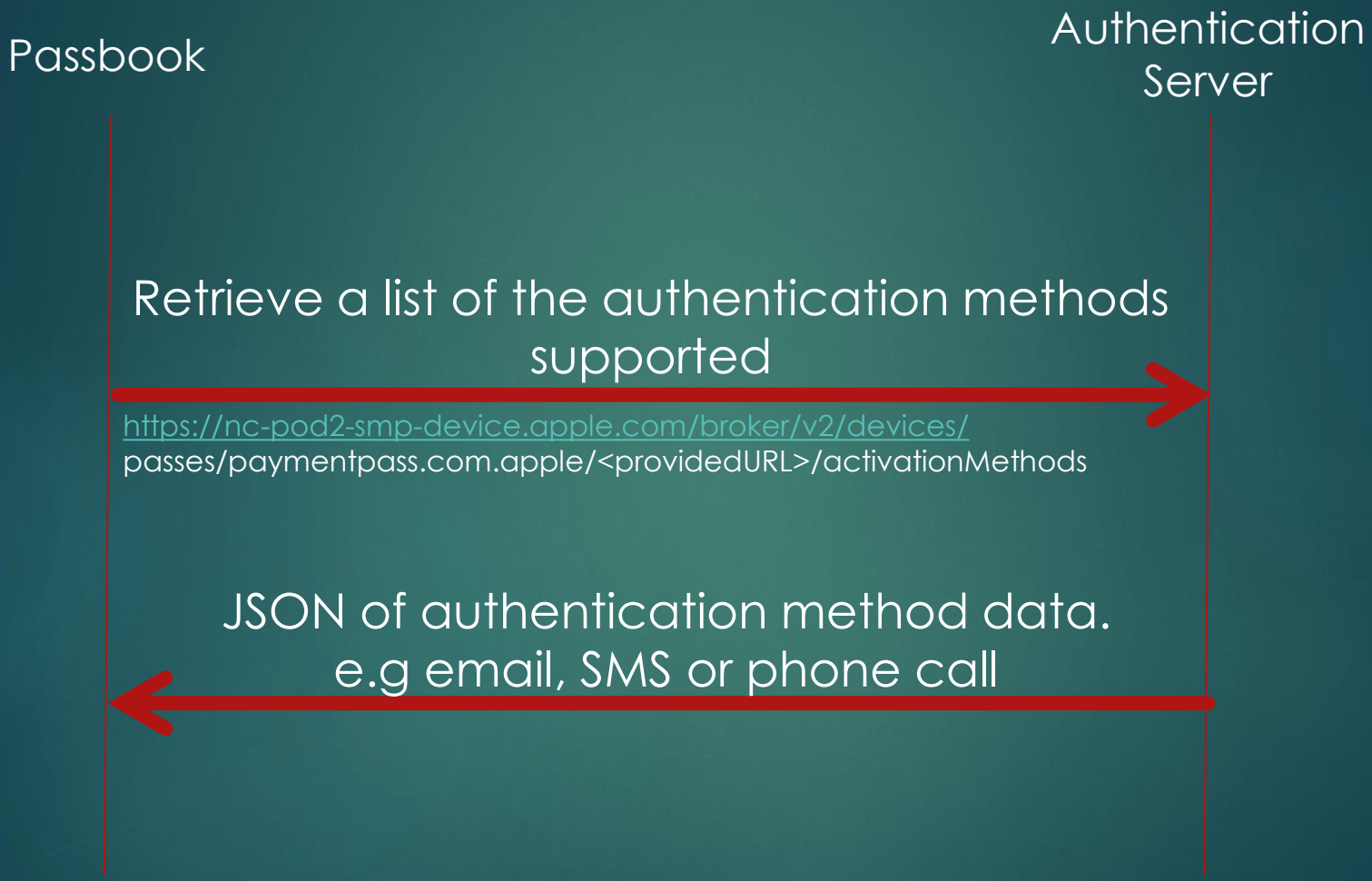
Enrolling a card – Step 5



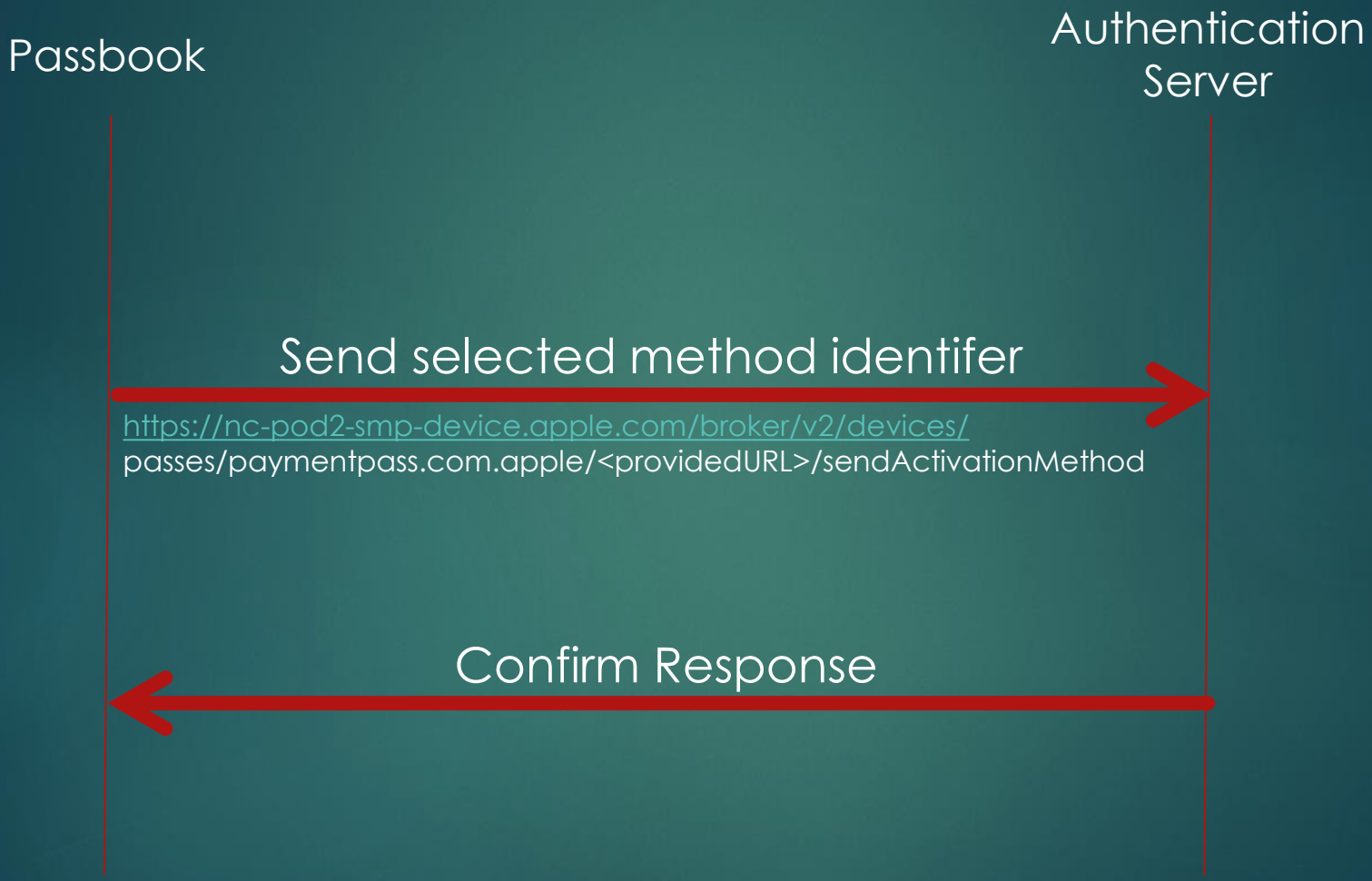
Enrolling a card – Step 6



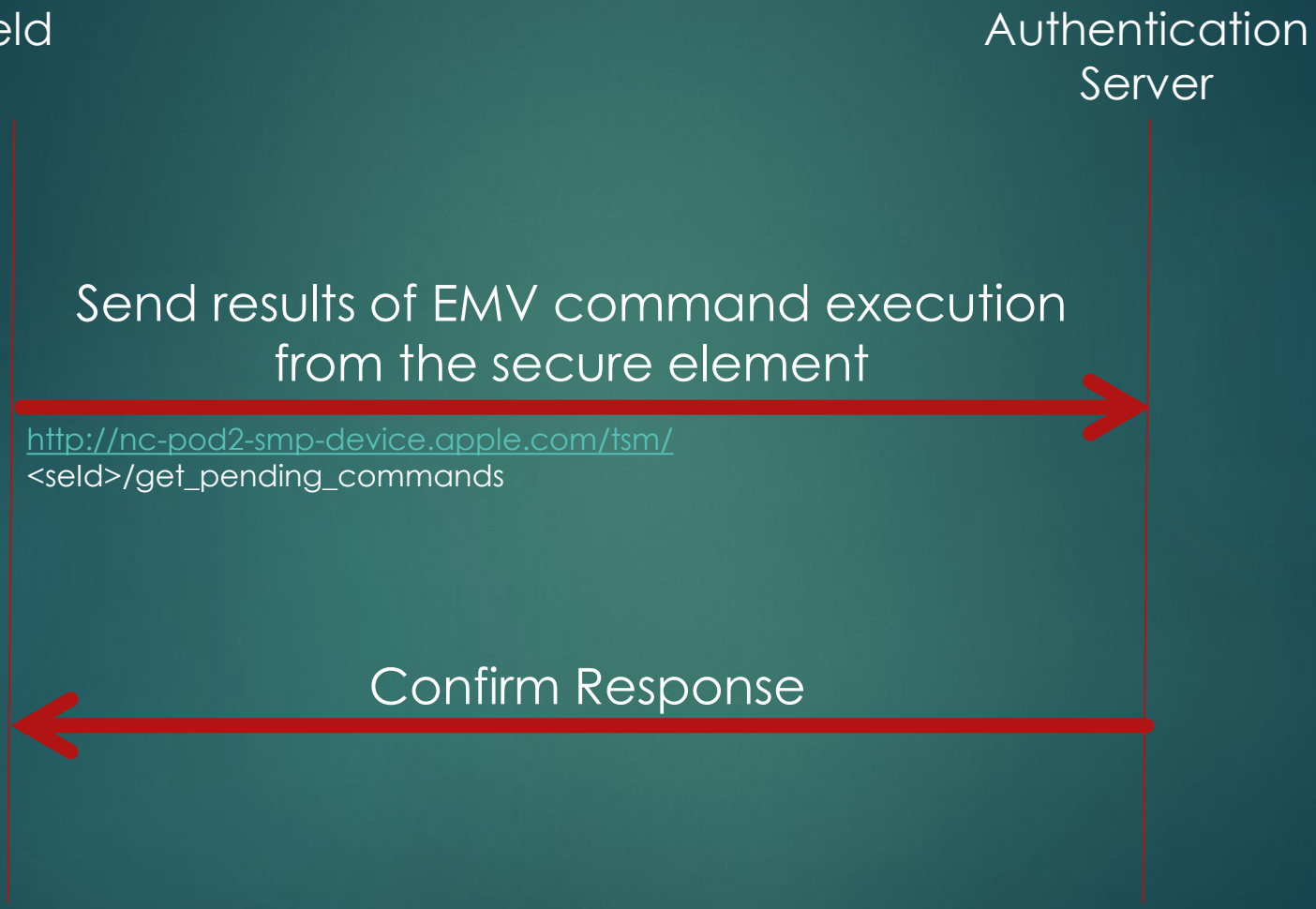
Enrolling a card – Step 7



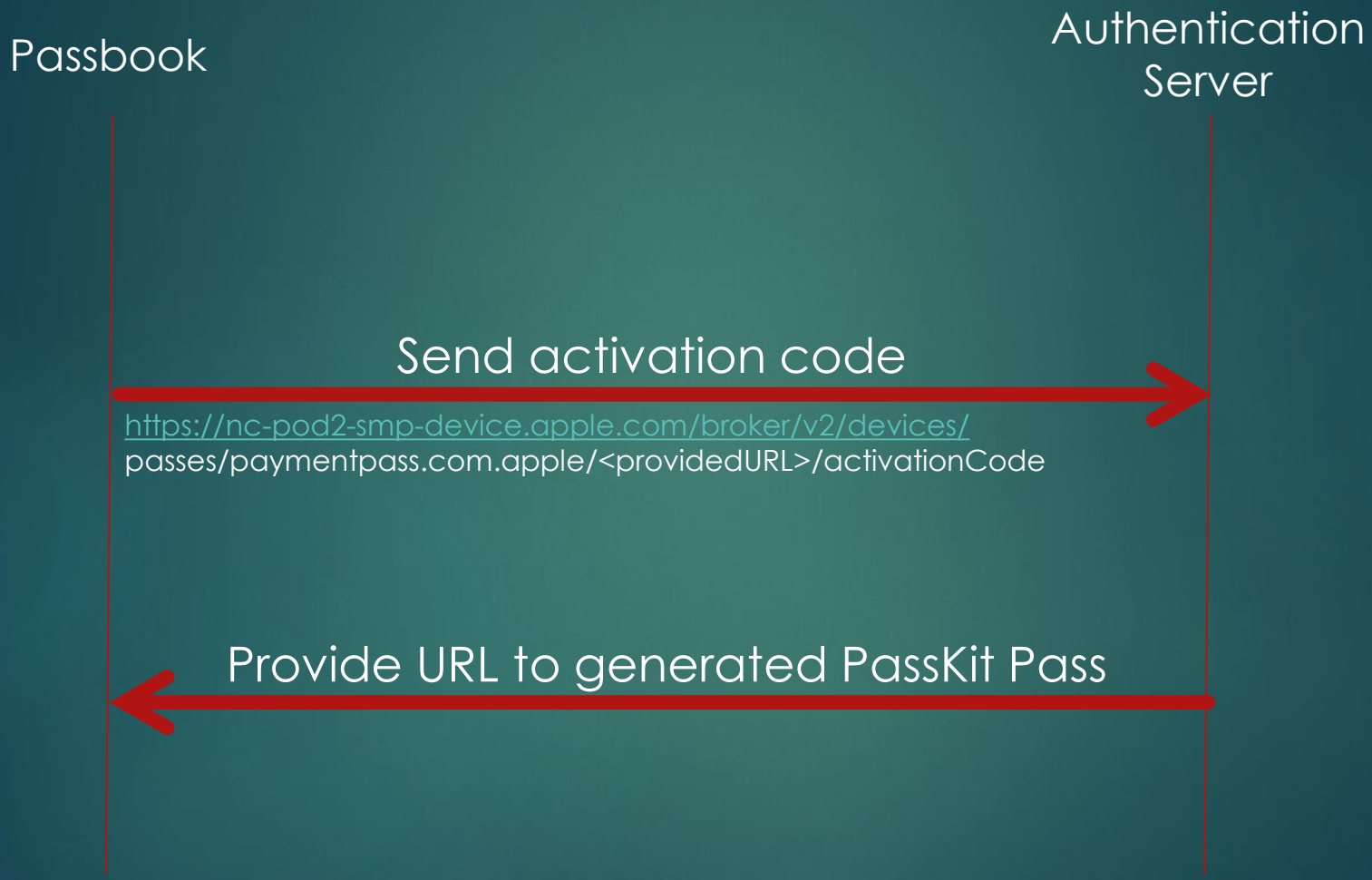
Enrolling a card – Step 8



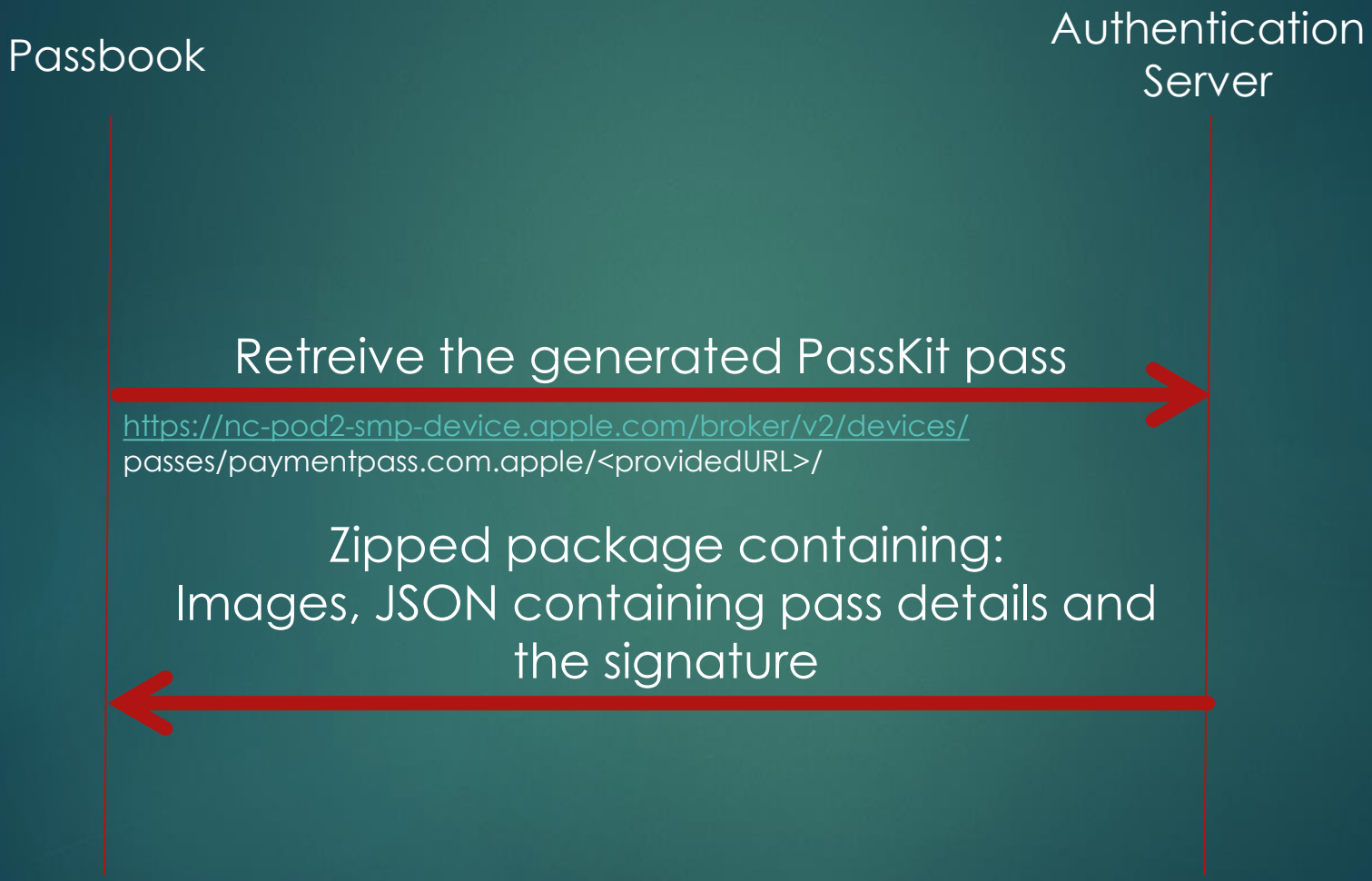
Enrolling a card – Step 9



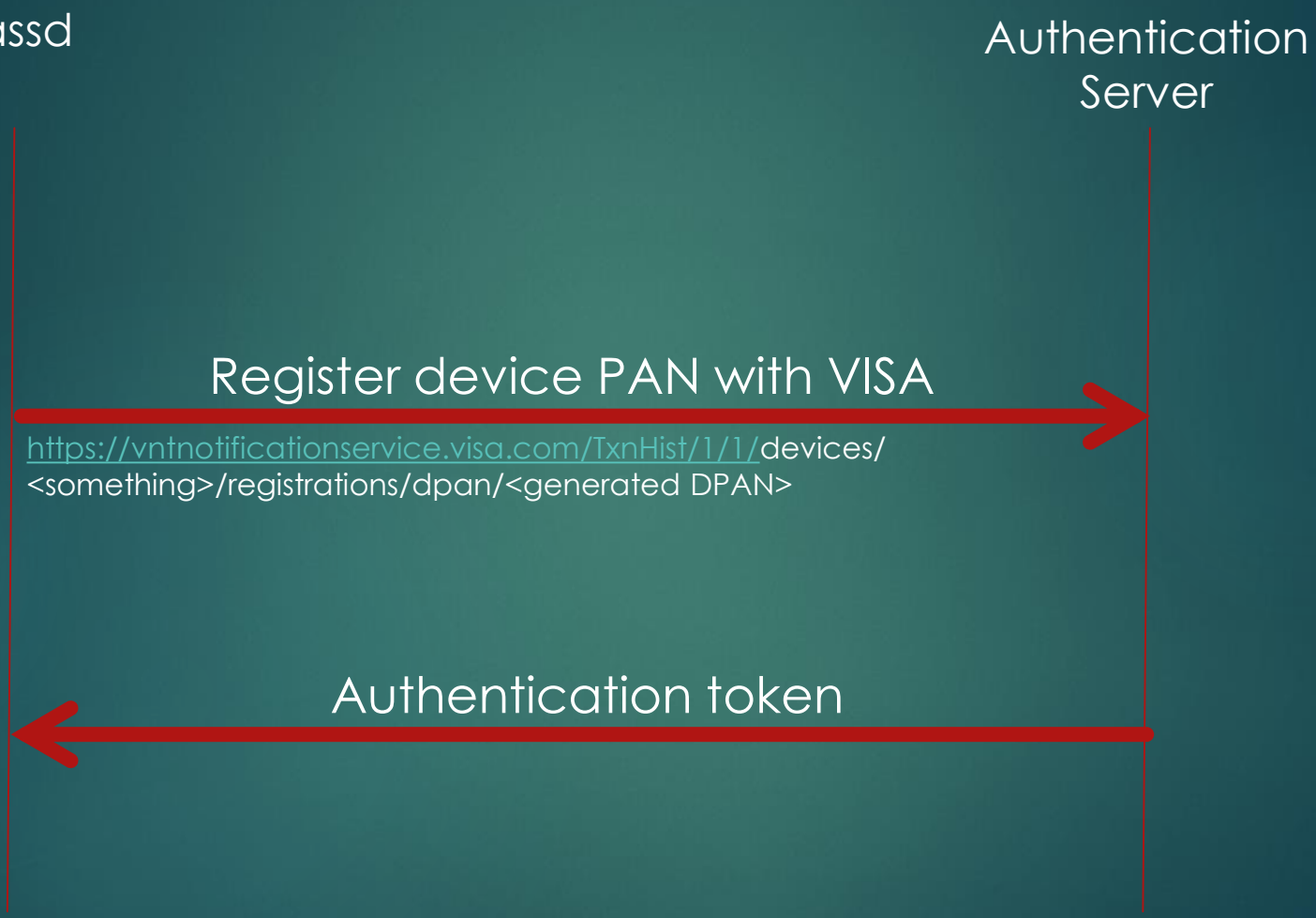
Enrolling a card – Step 10



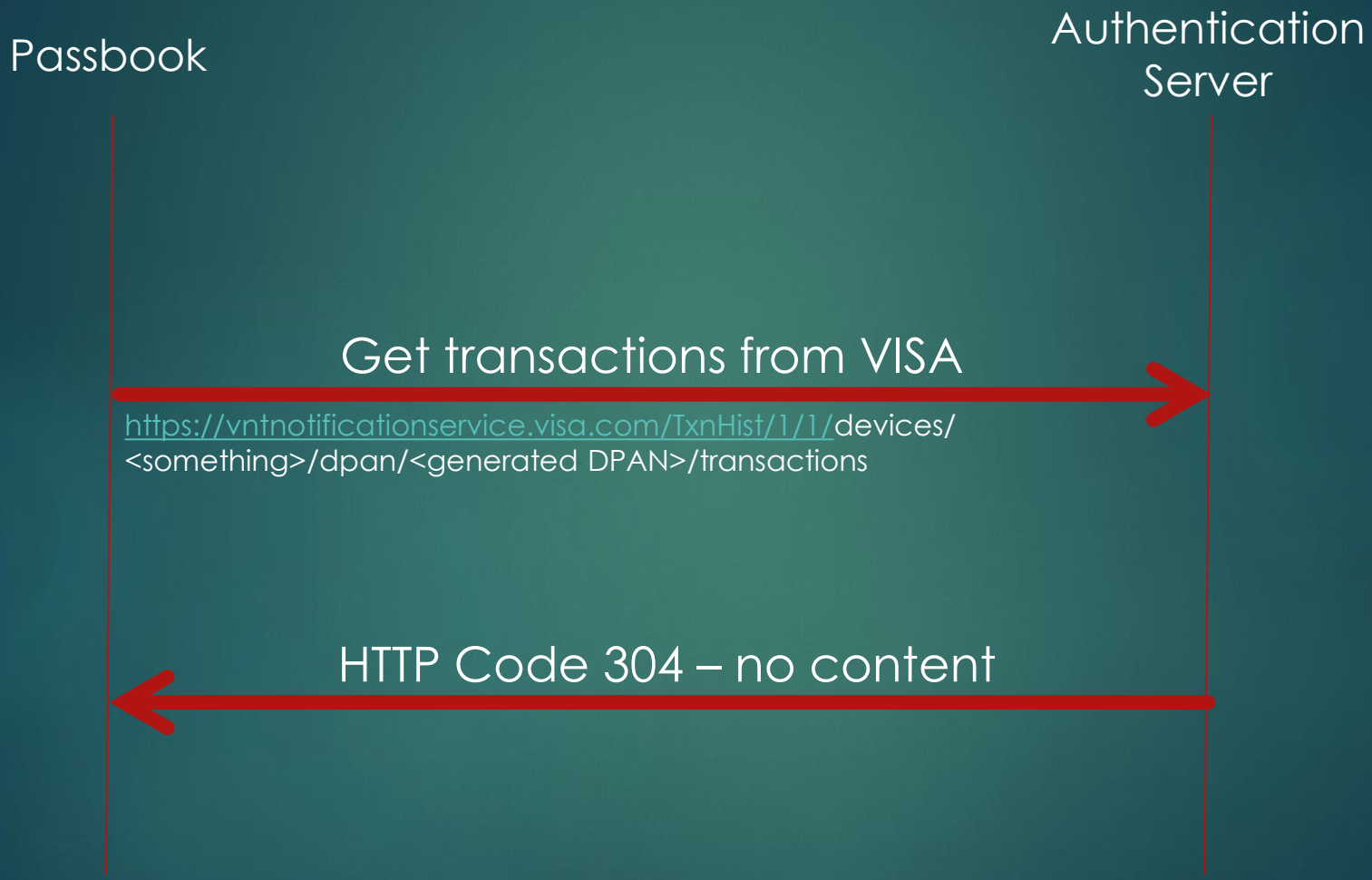
Enrolling a card – Step 11



Enrolling a card – Step 12



Enrolling a card – Step 13



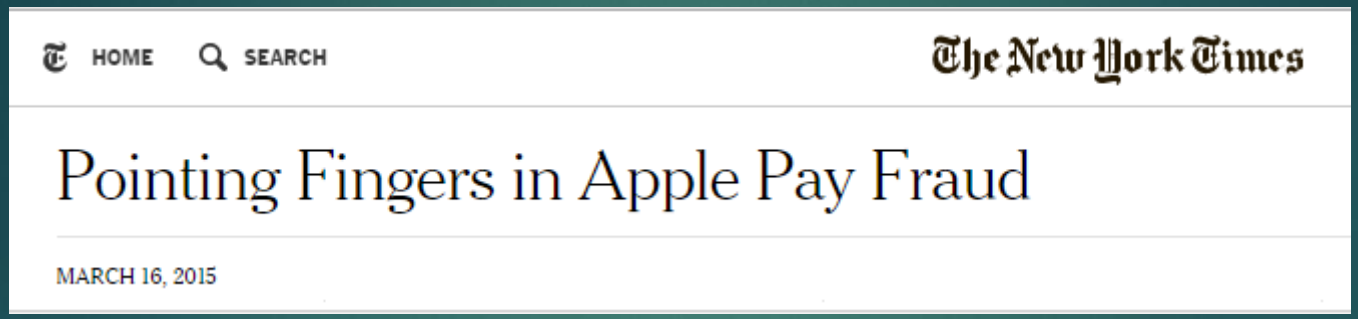
Using a card

Remote Wiping of a Card

What is right

- Secure element provides a highly limited attack surface. – all sensitive information is loaded encrypted
- Tokenization means that your personal account number is not stored or used by the device (I haven't found anything but a sanitized version)
- Issuer of the card can deactivate the token remotely (has happened to me twice so far)
- Applications which use ApplePay must have the correct entitlements to use it.
- You have to authenticate to use ApplePay

Bad verification of the cardholders


HOME SEARCH

The New York Times

Pointing Fingers in Apple Pay Fraud

MARCH 16, 2015

http://www.nytimes.com/2015/03/17/business/banks-find-fraud-abounds-in-apple-pay.html?_r=0

- Verification methods and back end are chosen by the card issuers/bank
- In the case of Wells Fargo the verification code can be sent through email, text or call to them.
- Banks were skipping verification checks – allowing for the ability to load stolen cards onto devices with minimal verification

Depends on existing contactless standards

- Same attacks are possible on ApplePay
- This is because it is the issuers that control the transaction – not Apple
- And because reasons it has to support these broken modes
- US market is still heavily invested in MagStripe – and this means broken contactless modes as
- Proper EMV rollout in the US is not going as quick as thought (In a month there in 2015 I they my EMV chip two times…)
- See my talk from last year (Crash & Pay)

Cloning Demo!

Transactions are logged and stored unencrypted

adm...	transaction_date ▾	location_date	location_latitude	location_longitude	location_altitude
VIC	466730751.274887	466730751.742518	-3.7922945431982598	145.00696355123301	22.002027511596701
VIC	466730429.932347	466730440.330548	-3.7922949382049303	145.00687642631399	22.287864685058601
VIC	463396260.956939	463396272.946306	-3.7922943253348997	145.00688488673401	23.0098686218262
VIC	463385957.748216	463385969.878485	-3.7922948910434698	145.00696156844299	23
VIC	463383730.269593	463383655.927718	-3.7922967957155102	145.00688969828201	22.006217956543001
null	463370164.361298	null	0.0	0.0	0.0
VIC	463302464.723265	463302438.044367	-3.7922745840659303	145.006889663772	19.1510009765625
VIC	463302409.874257	463302409.916261	-3.7922936950834	145.00686534785899	23.2420978546143

- Located in /var/mobile/Library/passes/passes23.sqlite
- Contains amounts, (accurate) locations, merchant location etc
- Make sure you remote wipe your ApplePay device if lost!

SQL Dump Demo

Android Pay

- Similar to ApplePay with the implementation
- Intended to replace "Google Wallet"
- Standalone application rather then imbedded into the OS
- Can't use it on a 'rooted' device
- http://nelenkov.blogspot.com.au/2012/08/exploring-google-wallet-using-secure.html
- http://nelenkov.blogspot.com.au/2012/08/android-secure-element-execution.html
- http://forum.xda-developers.com/google-nexus-5/help/android-pay-custom-rom-t3199843

SamsungPay

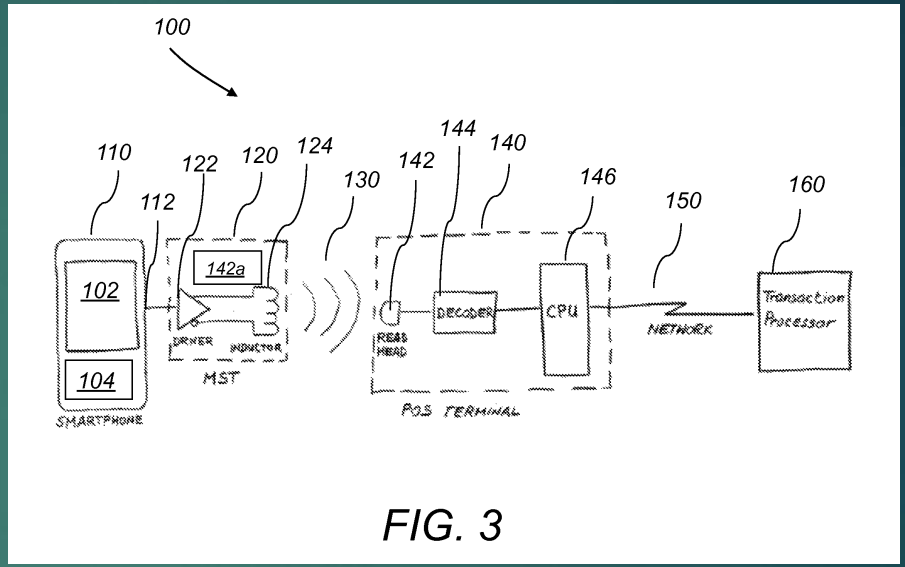


FIG. 3

- Based off "LoopPay" which uses traditional magstripe over NFC. (you heard me right)
- Idea is that you can use it with old terminals.
- Utilises tokenization again.
- Doesn't work "rooted"
- Further research needed!

Logging function calls using tweaks

1. Dump all the headers from the device

   ```
   $classdump-dyld -o <dump dir> -r / -c
   ```

2. Generate the "Tweak.xm" file to log the service you want

   ```
   $logify.pl *.h > Tweak.xm
   ```

3. Create the tweak project

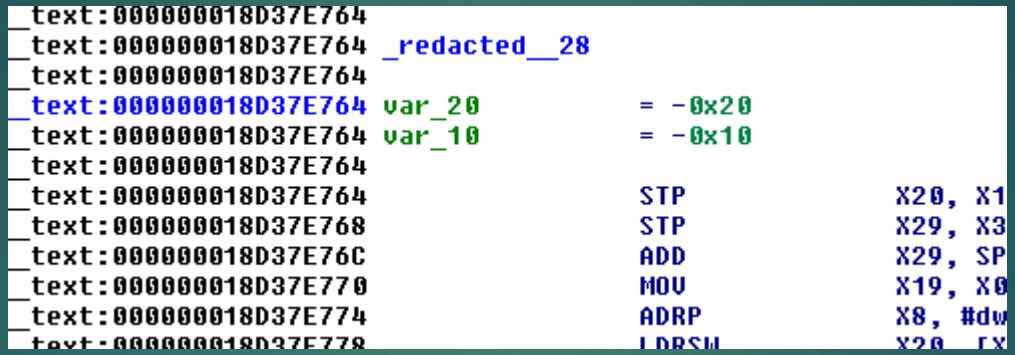
   ```
   $nic.pl

   NIC 2.0 - New Instance Creator------------
------------

   [9.] iphone/tweak
   ```

4. Copy the Tweak.xm to the project and compile

Logging function calls using tweaks

- Generated a whole heap of iOS tweaks to log calls to work out what was happening.
- Updated my RFIDiot scripts to do ApplePay Transations
- Quick IDA scripts to rename “redacted” functions in some binaries.
- Test XPC programs to test reversed functions.

Solving the “Redacted” problem



Use the Xcode tool “atos” - convert numeric addresses to symbols of binary images or processes.
Requires that the iOS device under test has been connected to the Mac to generate the symbolic files

```
$xcrun atos -arch arm64 -o ~/Library/Developer/Xcode
    /iOS DeviceSupport8.4/(XXX)/Symbols/System/Library
    /PrivateFrameworks/PassKitCore.framework/
    PassKitCore 0x18d3d5298
$__copy_helper_block_162 (in PassKitCore) + 0
```

References

- Iphonedevwiki – http://iphonedevwiki.net/index.php
- "Snakeninny and Hangcom" – iOS App Reverse Engineering– https://github.com/iosre/iOSAppReverseEngineering
- Ian Beers XPC preso at 44 con and Google Zero http://googleprojectzero.blogspot.com.au/2015/09/revisiting-apple-ipc-1-distributed_28.html
- Tielei Wang, Hao Xu, Xiaobo Chen of Team Pangu - https://www.blackhat.com/docs/us-15/materials/us-15-Wang-Review-And-Exploit-Neglected-Attack-Surface-In-iOS-8.pdf
- Sebas Guerro (@0xroot) "Demystifying Apple 'Pie' & TouchID" - http://www.slideshare.net/0xroot/demystifying-apple-pie-touchid

Source Code

- https://github.com/michael-quinlan/swift_basic_apple_pay
- https://github.com/beatty/applepay_crypto_demo
- https://github.com/peterfillmore/ApplePayStuff