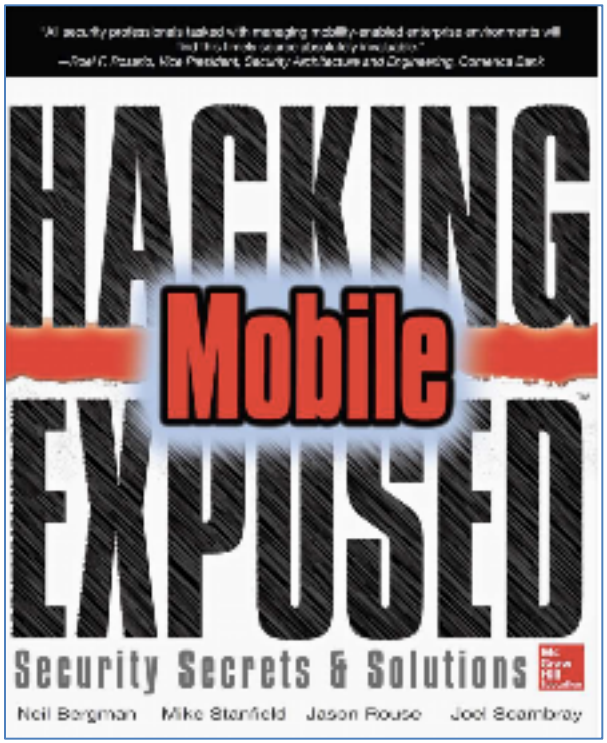
Ch 9: Mobile Payments



CNIT 128: Hacking Mobile Devices

Updated 4-24-17

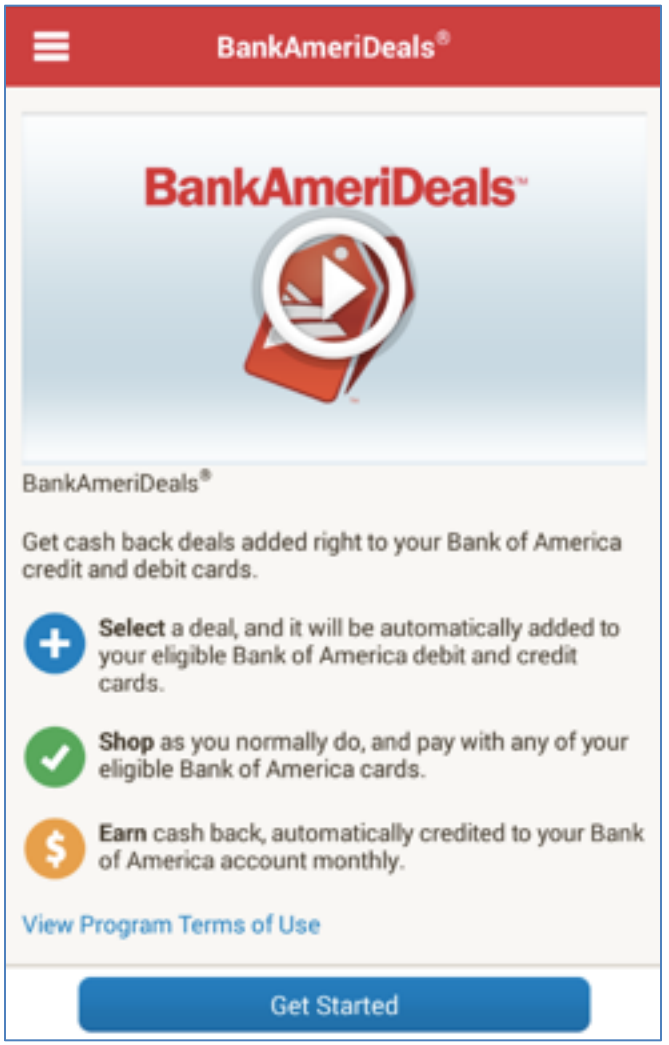
Current Generation

Scenarios

- Mobile banking apps
- NFC-based or barcode-based payment apps used by consumers to purchase goods
- Premium-rated SMS messages to purchase virtual goods within games, or music
  - Users are billed later via their telephone bill

Mobile Banking Apps

- Banking transactions using a phone
  - View account balances
  - Transfer money
- Web applications designed to be viewed within the mobile browser
  - Or a WebView inside a native mobile app

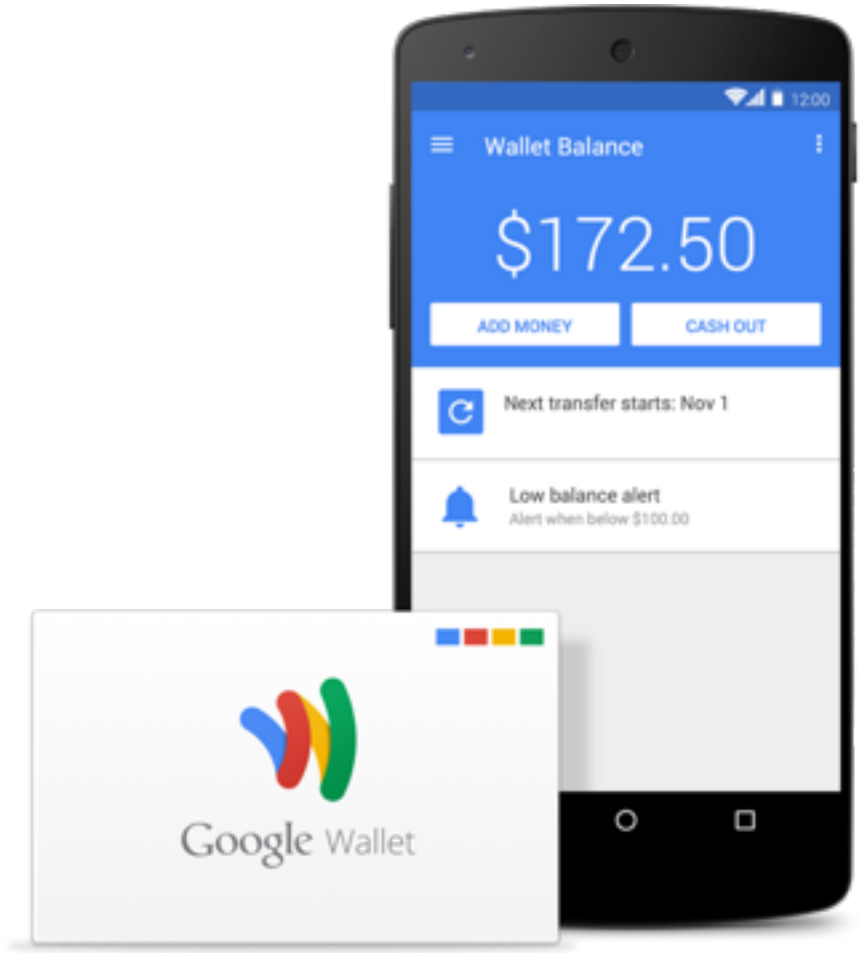


Mobile Banking Apps

- Back-end components are the same as for desktop online banking
- Similar vulnerabilities
- But with mobile, must also consider device theft
  - Sensitive information may be stored on the device improperly

Contactless Payment
Google Wallet

- Started in 2011
- Supports all major credit cards
- Card # stored in the cloud
- A virtual account number is sent to the contactless POS terminal via NFC



Contactless Payment
ISIS (now Softcard)

- Joint venture of Verizon, AT&T, and T-Mobile
  - Began in 2012
  - Changed its name in 2014 because of the "Islamic State" (link Ch 9a)
- Purchased by Google in Feb., 2015
- Google Wallet will be prominently preinstalled on U.S. Android phones that run KitKat (4.4) or later (link Ch 9e)

Android Pay

- Google's new payment system
- Replaced Google Wallet for most purposes in 2015
  - Google Wallet can no longer use NFC
- No app needed on most phones
- Android Pay is integrated into the OS
  - Link Ch 9w

Security of Mobile Payments

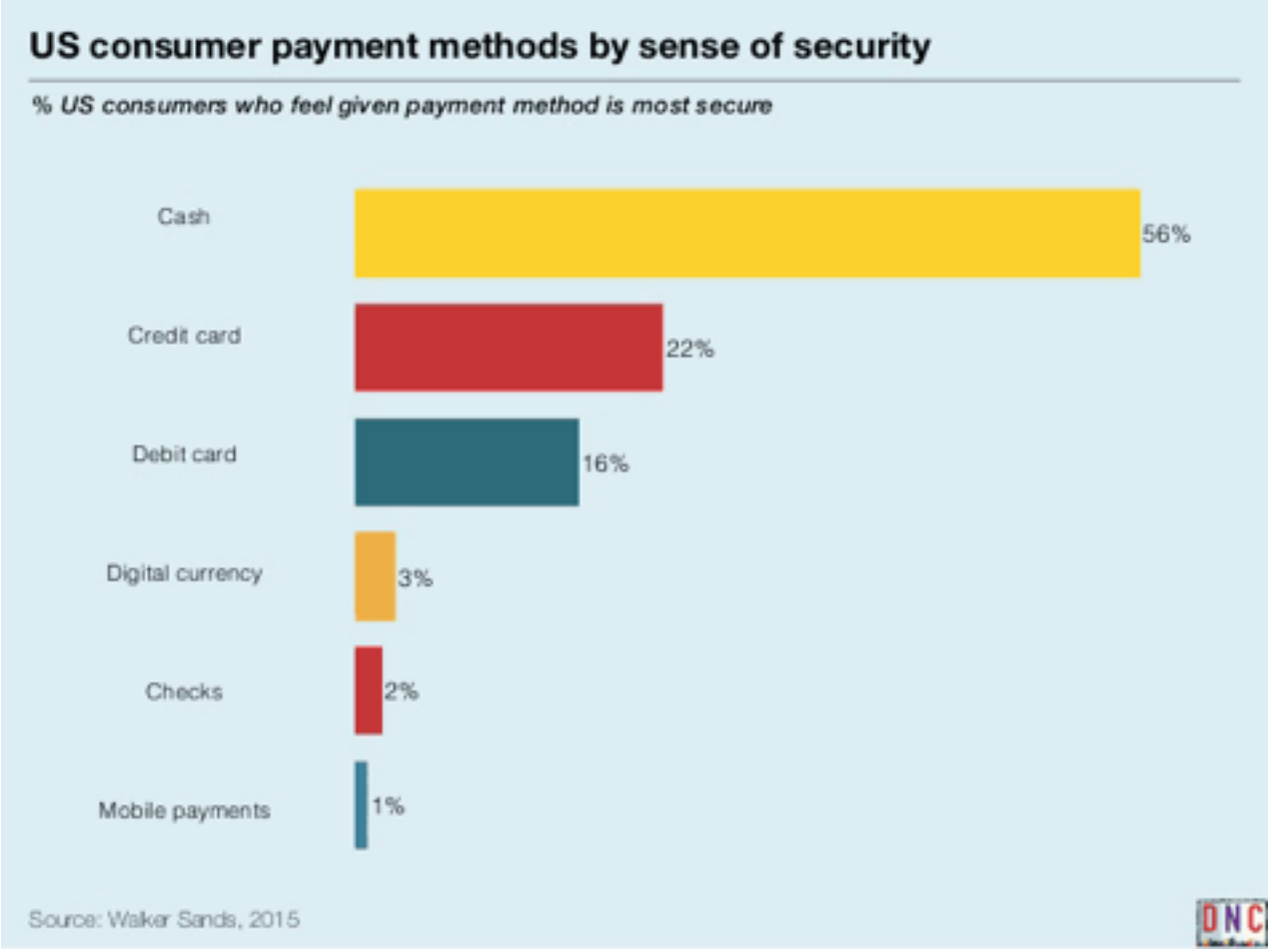
With Apple Pay, tokens are generated in a chip called the Secure Element. With Android Pay, they're generated in the cloud, which is what Host Card Emulation is. If you're without Internet and need to use Android Pay, the app will tap into a limited number of stored tokens on the device. (Where and how those tokens are stored isn't clear.)

An Android Pay user adds a credit or debit card to their Android Pay app. Android Pay requests a token to represent the card they're trying to add from the bank that issued that card. Once the token is issued, this card is now "tokenized," meaning it has a unique identification number associated with it. Android Pay encrypts the newly tokenized card and it is ready to be used for payments.

- Links Ch 9y, 9z

Security of Mobile Payments

- Should be safer than magstripe cards, which are very insecure
- But customers are wary
  - Link Ch 9x



Apple Pay

- Released on Oct 20, 2014
- With iPhone 6 and Apple Watch
- Customer payment information is kept from retailer
- creates a "dynamic security code [...] generated for each transaction"
  - Link Ch 9d

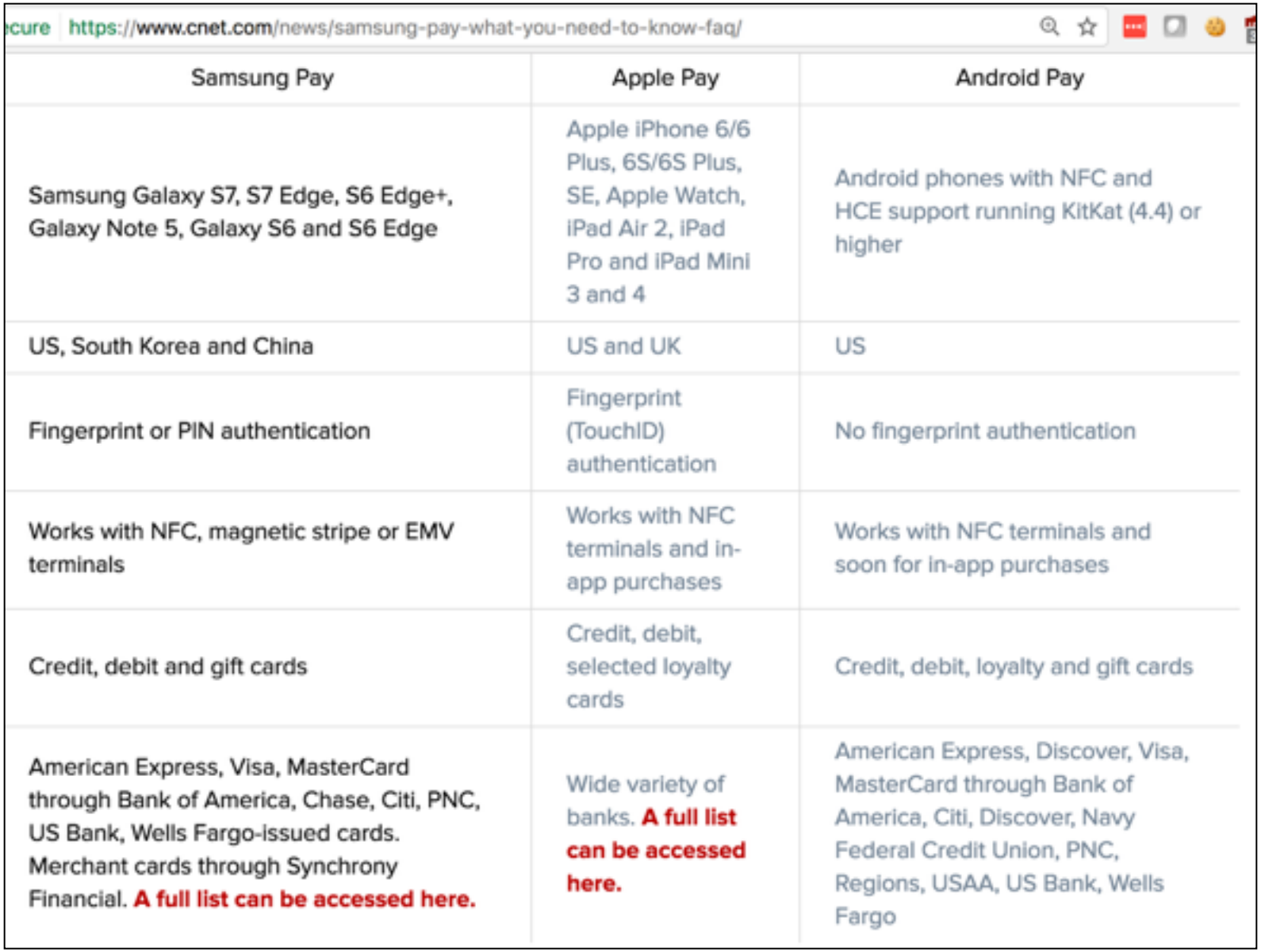
Market History

- Google was first, but retailers didn't play along
  - Only 2.4% of retailers had NFC in Oct, 2014
- Chip-and-PIN deadline was Oct. 2015
  - Retailers must update POS systems or accept liability for credit card fraud (link Ch 9c)
  - But the USA actually uses Chip-without-PIN
  - Security done on the server side

Samsung Pay

- Only available on Samsung devices
- Works with NFC or magstripe readers
  - 90% of merchants

Samsung Pay



Samsung Pay	Apple Pay	Android Pay
Samsung Galaxy S7, S7 Edge, S6 Edge+, Galaxy Note 5, Galaxy S6 and S6 Edge	Apple iPhone 6/6 Plus, 6S/6S Plus, SE, Apple Watch, iPad Air 2, iPad Pro and iPad Mini 3 and 4	Android phones with NFC and HCE support running KitKat (4.4) or higher
US, South Korea and China	US and UK	US
Fingerprint or PIN authentication	Fingerprint (TouchID) authentication	No fingerprint authentication
Works with NFC, magnetic stripe or EMV terminals	Works with NFC terminals and in-app purchases	Works with NFC terminals and soon for in-app purchases
Credit, debit and gift cards	Credit, debit, selected loyalty cards	Credit, debit, loyalty and gift cards
American Express, Visa, MasterCard through Bank of America, Chase, Citi, PNC, US Bank, Wells Fargo-issued cards. Merchant cards through Synchrony Financial. **A full list can be accessed here.**	Wide variety of banks. **A full list can be accessed here.**	American Express, Discover, Visa, MasterCard through Bank of America, Citi, Discover, Navy Federal Credit Union, PNC, Regions, USAA, US Bank, Wells Fargo

Samsung Pay

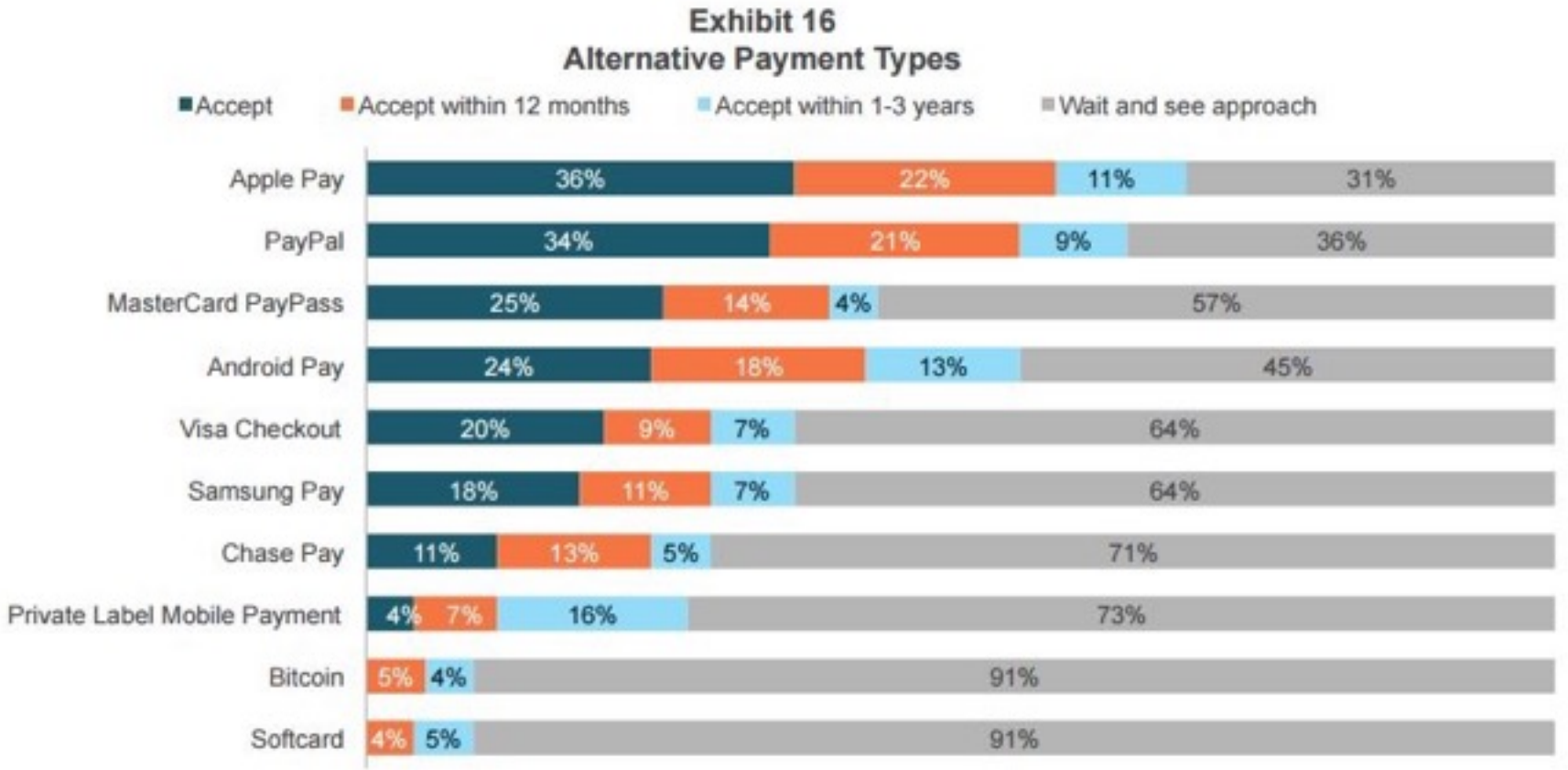
Is it secure?
Samsung Pay does not store the account or credit card numbers of cards on the device, instead using tokenization for transactions. Each time a purchase is made, the Samsung Pay handset sends two pieces of data to the payment terminal. The first is a 16-digit token that represents the credit or debit card number, while the second piece is a one-time code or cryptogram that's generated by the phone's encryption key.

What if I lose my phone?
Payments can't be made from your phone without being authorized via fingerprint or the PIN chosen during the setup process. If you register with Samsung's **Find My Mobile** service you can remotely erase information on the phone, including any cards stored in Samsung Pay.

Can I use Samsung Pay even without a Wi-Fi/cellular connection?
Yes, although you will only be able to make 10 payments without the device being on Wi-Fi or cellular data. You will also need to have an active internet connection to add a card and to access transaction history.

- Link Ch 9z1

US Retailers Prefer Apple Pay



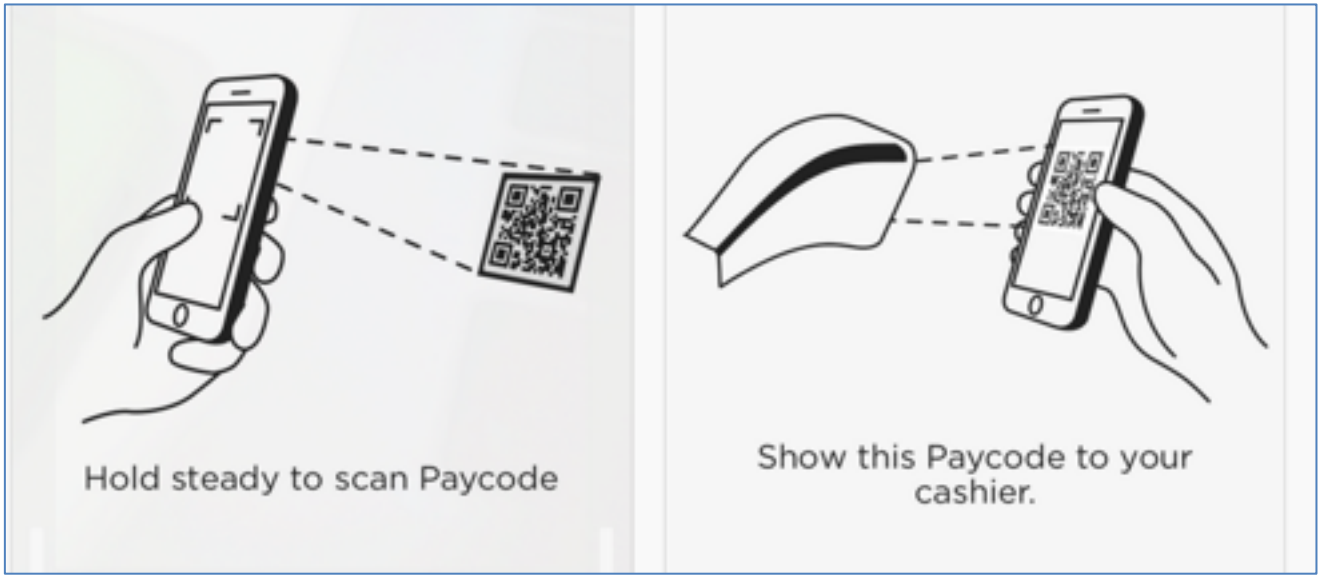
- Link Ch 9z2, from Feb., 2017

CurrentC

- A group of merchants (MCX)
  - Rite-Aid, CVS, Walmart, Target, etc.
- Saves merchants credit card processing fees
- Gives stores access to consumer data
  - Unlike Apple Pay
  - Link Ch 9b, 9h
- Designed for merchants, not end-users

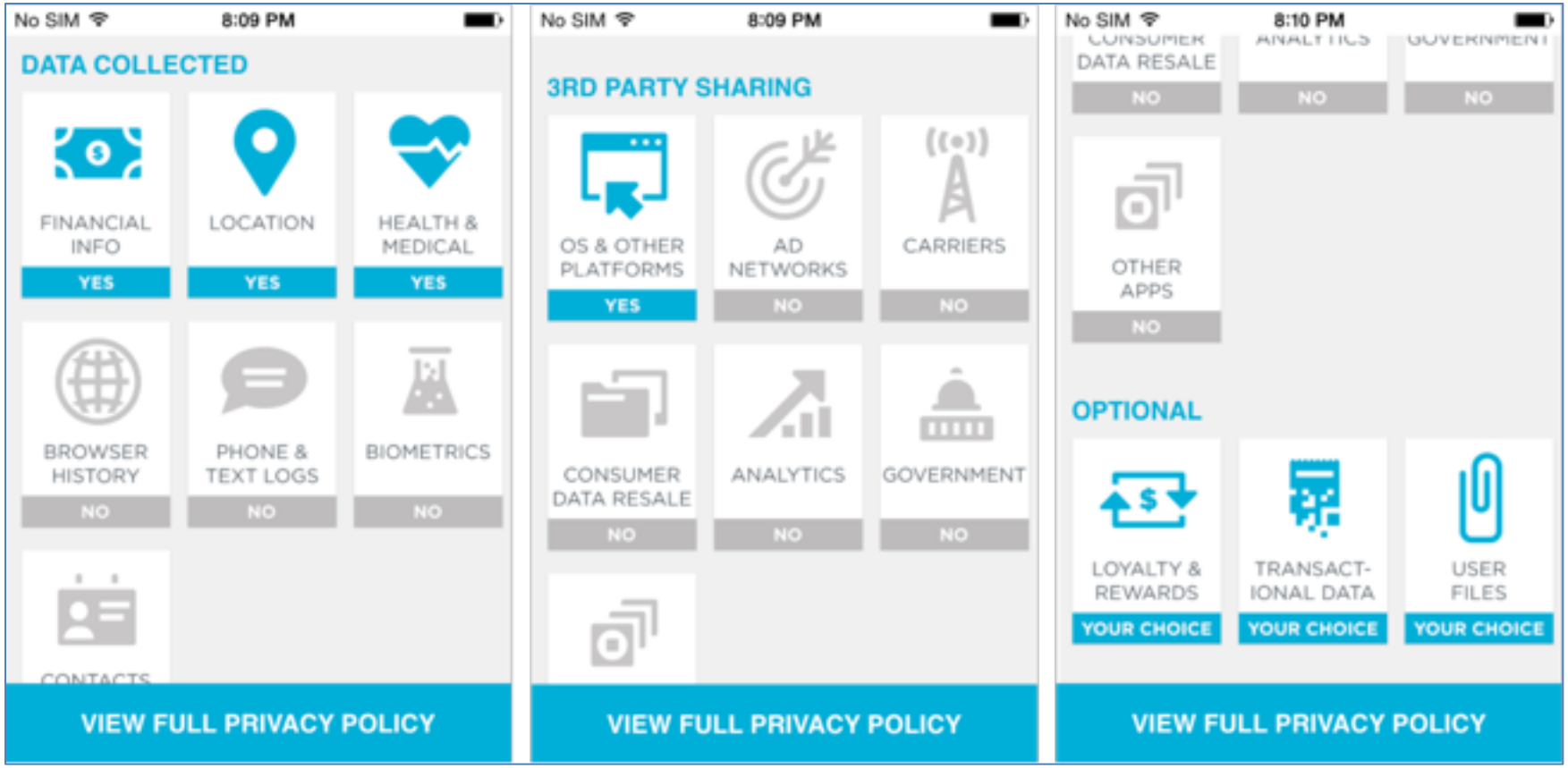
How CurrentC Works

- Tied directly to your bank account
- Pay with QC code



Retailers Supporting CurrentC

CurrentC Collects Health Data



CurrentC Hacked in Oct. 2014

- Email addresses of early testers exposed
  - Link Ch 9j

Startup hacked before they could even launch

CSO | Oct 29, 2014 12:31 PM PT

Merchant Customer Exchange (MCX) says that CurrentC, a mobile payment offering backed by giant retailers like Wal-Mart, Best Buy, Old Navy, Target, CVS, and more, has been compromised.

Current-C Died in 2016

NOT SO CURRENT C —

CurrentC—retailers' defiant answer to Apple Pay—will deactivate its user accounts

The mobile payments scheme was distrusted before it even hit the big time.

MEGAN GEUSS - 6/8/2016, 10:05 AM

- Lin Ch 9z3

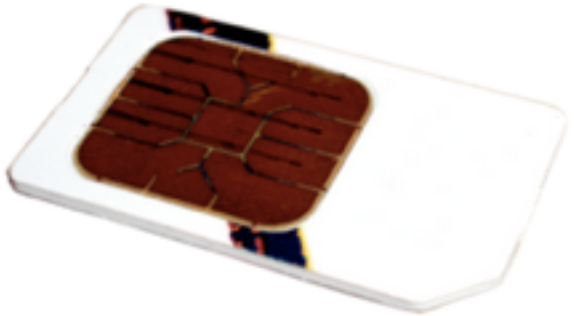
Square

- Free card reader or stand
- Plugs into audio jack on iOS or Android phone
- Takes credit card payments by reading the magstripe
- Used by Starbucks and Whole Foods
- Began taking Bitcoin in 2014
- Will take Apple Pay in 2015

Contactless Smartcard Payments

Secure Element (SE)

- Core of the mobile payment platform
- Secure storage of sensitive information
- **Embedded SE** contained within the mobile device
  - Galaxy Nexus
- **UICC** aka **SIM card**
  - Universal Integrated Circuit Card
  - Another SE form factor
  - Link Ch 9m



microSD Cards with NFC

- Allowed early iPhones without NFC to use NFC
- NFC radio included in the microSD card
- Pioneered by DeviceFidelity
- Purchased by Kili in 2014
- Kili purchased by Square in 2015
  - Links Ch 9o, p, q

Java Card Runtime Environment (JCRE)

- All SE's use this system
- Payment applet stored on the card
- Applet firewall keeps applets from accessing each others' information
- Robust cryptography including AES and RSA
- SE's are GlobalPlatform compliant

- Security and interoperability standards for SE devices
- Only the owner of an SE can directly read or write to it
  - Mutual identification uses shared keys
  - SE will lock after a number of failed attempts

Proximity Payment System Environment (PPSE)

- Registry of all payment apps in the SE
- App names and standard Application Identifier
- Tells the payment terminal what apps are available
- Allows terminal to select which app it wants to use

Payment Apps

- Responsible for making the actual contactless payment
- Contain sensitive information associated with a particular payment account
- Java Card applets that are stored and run inside the SE

Payment Apps

- Cryptographic capabilities of the JCRE allow banks to securely verify transactions
  - One method is to generate a one-time Card Verification Value for each transaction, called a dynamic CVV (dCVV)
- Application Protocol Data Unit (APDU)
  - Used to send instructions to applets on the SE

Command Application Protocol Data Unit (C-APDU)

Name	Length (in bytes)	Description
CLA	1	Class byte. Specifies what type of command is being issued.
INS	1	Instruction byte. Specifies the specific instruction being carried out, such as read data.
P1 and P2	2 (one byte each)	Parameter bytes. Contain instruction-specific parameters.
L_C	Up to 3 bytes	Contains the length (in bytes) of the following command data buffer. The value is zero if no command data is included.
Command Data	Variable, up to 256 bytes	Contains information being passed to the applet. This information is typically encoded in a tag-length-value (TLV) format.
L_e	Up to 3 bytes	Contains the maximum number of response bytes expected.

Large Commands

- If the amount of data to be transmitted to the applet is greater than 256 bytes
- Multiple C-APDUs can be chained together

Response Application Protocol Data Unit (R-APDU)

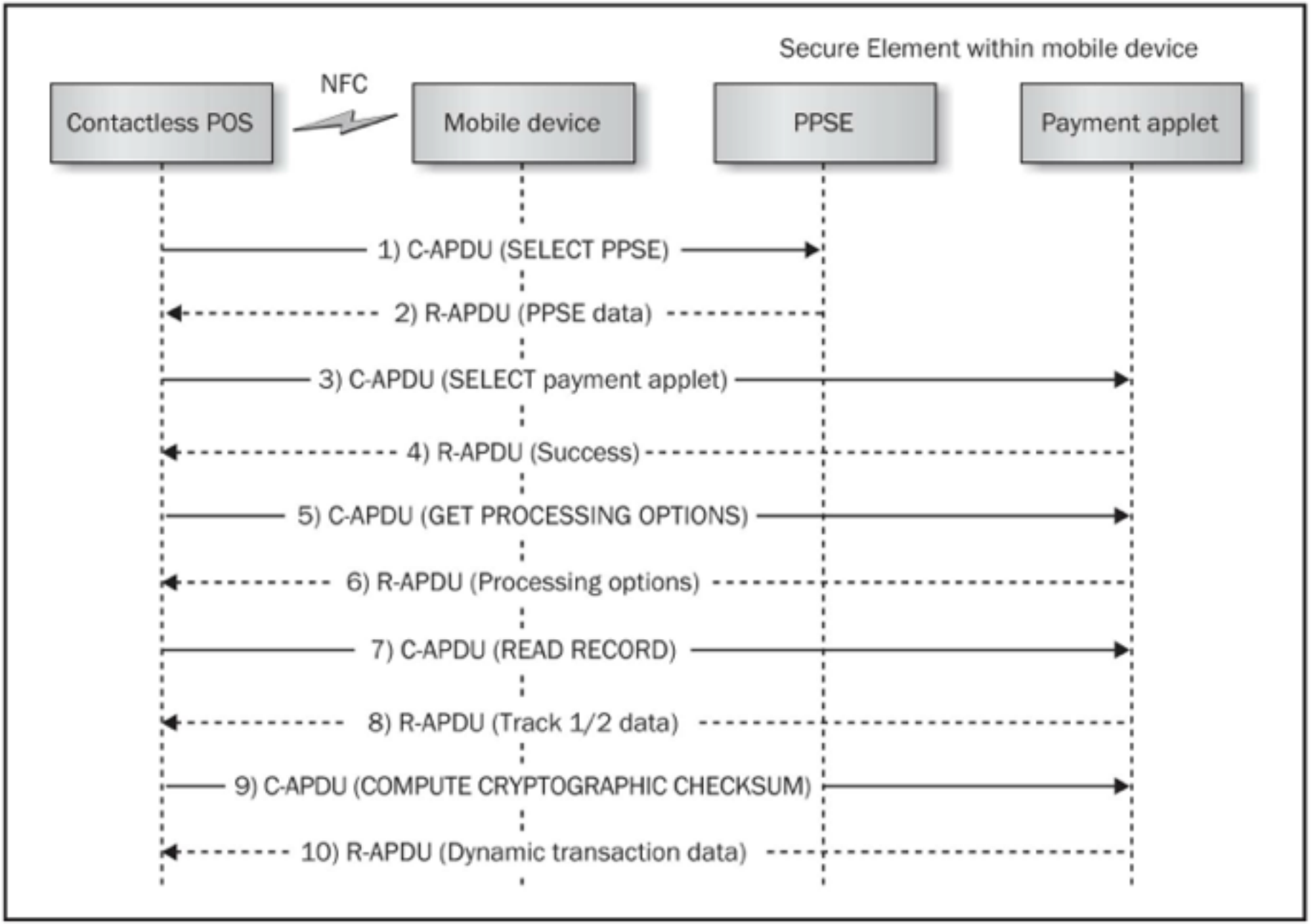
Name	Length (in bytes)	Description
Response Data	Variable, up to 256 bytes	The response data, if any, from the applet. Will typically be tag-length-value (TLV) encoded.
SW1 and SW2	2	These bytes return the status of the command; for example, 0x90, 0x00 indicates the command was successfully executed.

Contact and Contactless Interfaces

- These are the two ways to send APDUs to the SE
- Contact Interface
  - Connects the SE to the phone itself
- Contactless Interface
  - Connected to the NFC radio
  - Used to communicate with Point-of-Sale (POS) terminals
  - Not available to applications on the phone

Simplified Contactless Transaction

1. The contactless POS sends a `SELECT` command to the PPSE applet.
2. PPSE responds with a list of available payment applets.
3. The POS chooses a payment applet and then issues a `SELECT` command.
4. The payment applet responds, letting the POS know the `SELECT` command was successfully processed.
5. The POS sends the `GET PROCESSING OPTIONS` command, including information requested by the payment applet about the POS itself.
6. The payment applet responds with the processing options that both it and the POS support.
7. The POS sends a `READ RECORD` command to the payment applet.
8. The payment applet responds with the so-called Track 1 and Track 2 data per ISO/IEC 7813, which includes the Payment Account Number (PAN).
9. The POS sends the `COMPUTE CRYPTOGRAPHIC CHECKSUM` command to the payment applet, including an unpredictable value.
10. The payment applet responds with CVC3s (MasterCard's version of a dCVV), generated using dynamic data (unpredictable value and transaction counter) and a secret key.


Secure Element within mobile device
NFC
Contactless POS
Mobile device
PPSE
Payment applet
1) C-APDU (SELECT PPSE)
2) R-APDU (PPSE data)
3) C-APDU (SELECT payment applet)
4) R-APDU (Success)
5) C-APDU (GET PROCESSING OPTIONS)
6) R-APDU (Processing options)
7) C-APDU (READ RECORD)
8) R-APDU (Track 1/2 data)
9) C-APDU (COMPUTE CRYPTOGRAPHIC CHECKSUM)
10) R-APDU (Dynamic transaction data)

Secure Element API

- Restricted to Google Wallet on Android
  - Introduced in 2.3.4 (Gingerbread)
  - Required system-level permissions through 4.0 (Ice Cream Sandwich)
  - In 4.04, allows apps with a signature in **/etc/nfcee_access.xml**
  - The only signature in that file is Google Wallet
  - Requires root access to update

SE API Limitations

- Very basic—allows application to open a channel to the SE and transmit APDUs
- Works for embedded SE's
  - But not for the UICC or microSD SE's used in some phones
  - For microSD SE's, you need the open-source Secure Element Evaluation Kit (SEEK)
  - UICC SE's is not directly connected to the application processor and must be reached through the proprietary code and the Radio Interface Layer

Access Control for SE's

- Embedded SE's use a whitelist
  - **/etc/nfcee_access.xml**
- SEEK uses GlobalPayment
  - An additional app on the SE with a list of application signatures and applets
- Smartcard API contains *Access Control Enforcer*
  - Compares signature of calling application to signature stored in the SE card to see if application has permission for the chosen applet

Mobile Application

- Consumers see this part
- User selects which card to use for a payment
- Google Wallet requires the user to enter a four-digit PIN to make a payment
  - Protects against device theft
  - Better than contactless credit cards

Google Wallet Vulnerabilities

PIN Storage Vulnerability

- PIN entry required for transactions
- Only six tries permitted
- But an attacker who steals a device and then roots it can extract the PIN from the salted hash
  - Because it's not stored on the SE
  - Storing it on the SE would make banks liable for breaches due to stolen PINs
  - Links Ch 9s, 9t

Rubin said at the time that Google was "extremely responsive to the issue" and recognized that "the only way to properly solve this issue would be to move the PIN verification into the SE itself and to no longer store the PIN hash and salt outside the SE." Unfortunately, it doesn't appear to have released a patch yet for bureaucratic reasons. By storing the PIN hash and salt in the SE, banks would be liable for any breaches related to stolen PINs.

- Link Ch 9t (2012)

PIN Storage

- PIN is salted with a 64-bit random value and hashed with one round of SHA-256

```
protected boolean doSetPin(WalletClient.DeviceInfo.PinInfo.Builder
paramBuilder, UserPin paramUserPin)
  {
    WLog.d(TAG, "doSetPin");
    int i = userPinAsInt(paramUserPin);
    long l = this.mSecureRandom.nextLong();
    String str = hashPin(i, l);
    paramBuilder.setPinTryCounter(6).setPinTryLimit(6).setLocalSalt(l)
    .setLocalPinHash(str);
    return true;
  }
...
String hashPin(int paramInt, long paramLong)
  {
    String str = Integer.toString(paramInt) + Long.toString(paramLong);
    return this.mDigestUtil.sha256(str);
  }
```

Storage of Hash

- Salt and hash stored in a SQLite database in Google Wallet's /data directory
  - **/data/data/ com.google.android.apps.walletnfcrel/ databases/walletDatastore**
- "Wallet Cracker" simply tries all 10,000 four-digit PINs to find PIN from the hash

Google's Response

- Don't run Google Wallet on rooted phones
  - Not very reassuring since the thief can root your phone
- Much better to perform PIN storage and verification on the SE
  - Also store the PIN try counter on the SE

Countermeasures for Google Wallet Cracker

- Don't root your device
- Enable Android lock screen
- Disable ADB debugging
- Keep up-to-date with patches

Relay Attacks (MITM)

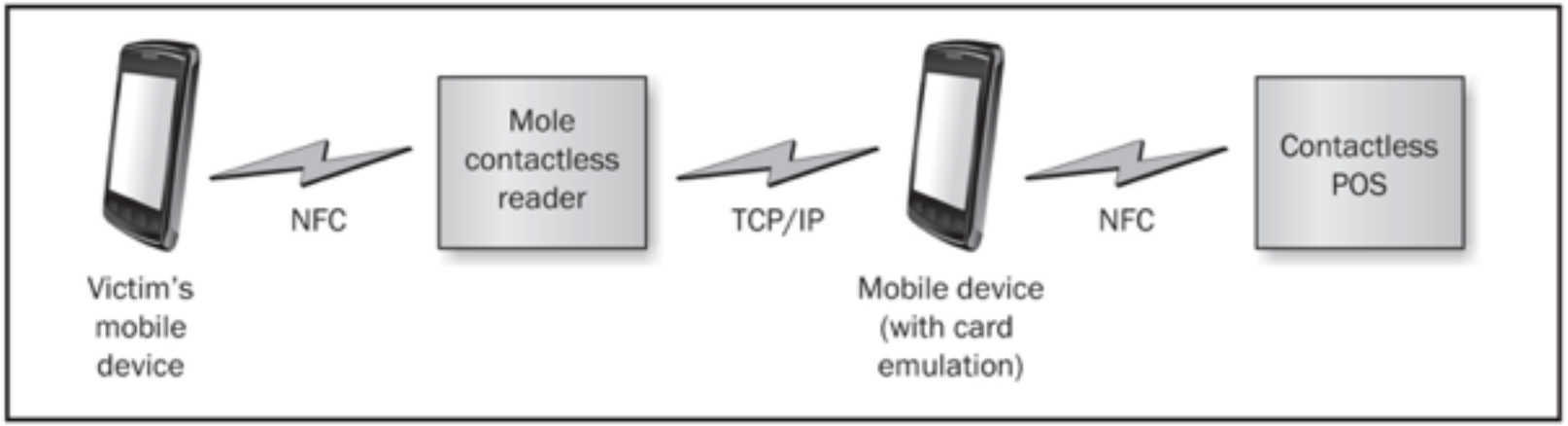


Figure 9-5 A relay attack against a NFC-based mobile payments application

- "Mole" reader gets close to target mobile device
- Attacker's mobile gets near POS terminal
- APDUs are passed via TCP/IP

Relay Attack Limitations

- Target's mobile payment app must be unlocked
- Google Wallet requires entry of a PIN to unlock

Relay Through a Malicious App

- Works against Google Wallet
- Because it exposes payment credentials to the contact interface
- Requires root privileges to bypass SE API signature authentication

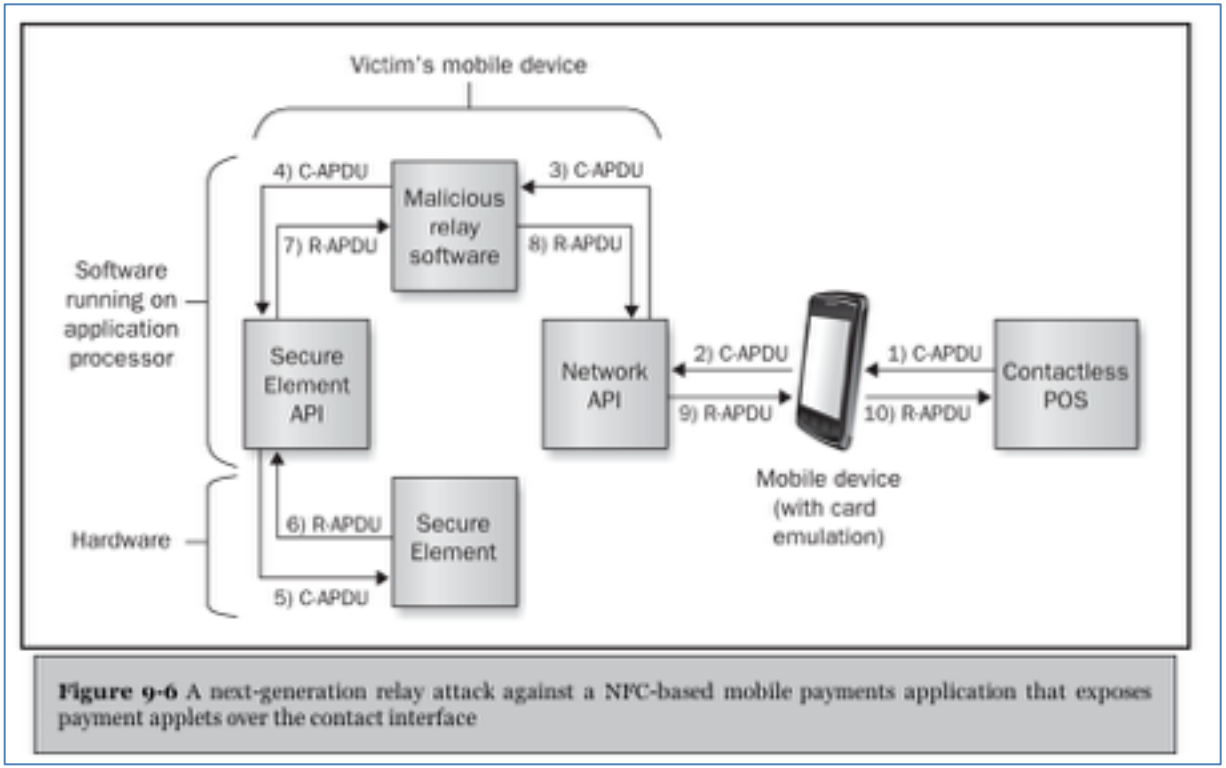


Figure 9-6 A next-generation relay attack against a NFC-based mobile payments application that exposes payment applets over the contact interface

Relay Attack Countermeasures

- Contactless POS terminals should enforce a time-out on all transactions
  - Relay attack requires network communications which slows it down
  - Not very practical because errors can cause delays in legitimate transactions
- Use location information to flag suspicious transactions
  - Target mobile is not really near the POS
  - Requires target GPS to be active and consumer's consent

Relay Attack Countermeasures

- Google Wallet is no longer vulnerable to the second attack
  - It no longer exposes payment applets over the contact interface

Square Vulnerabilities

Square

- Square Register
  - Mobile app
- Magnetic stripe reader
  - Plugs into audio jack
  - Free
- Allows anyone to take credit card transactions
  - Charging 2.75% of each transaction

EMV (Europay, MasterCard and Visa) aka Chip-and-PIN

- Square reader has two slots
- Can use magstripe or chip

Skimming

- Any app that can receive audio data can steal the magnetic data from the Square device
- VeriFone released an app to do this
  - In order to compete with Square

Verifone v. Square

- Link Ch 9u

TWITTER LINKEDIN FACEBOOK GOOGLE+ 5 of 9

Spotlight on Security

The companies' rivalry got ugly in 2011 when VeriFone CEO Douglas Bergeron launched an aggressive attack on Square's credibility, accusing the company of selling a device that could be used for illegal card-skimming. VeriFone went so far as to offer a demo card-skimming app to major card issuers to prove its assertion. *(Image: ThinkStock)*

Skimming Countermeasures

- Manual skimming requires the card
  - Same as skimmers that have been used for years
- A software attack against the reader could do more harm
- In 2012, Square modified their reader to encrypt the audio stream
  - Encrypted data is sent to Square's servers and decrypted there
  - Prevents rogue apps getting the credit card #

Replay Attack

- Malicious app could record audio stream and replay is back to make another purchase
- Demonstrated by Adam Laurie and Zac Franken at Black Hat in 2011
  - Also reverse-engineered the format Square reader uses for data from credit card
  - They could manufacture correct audio streams from magnetic Track 2 data, which can be purchased on the black market

Replay Attack

- They could therefore use Square to perform mass fraud
  - Instead of manufacturing fake credit cards

Replay Attack Countermeasures

- Square's encryption prevents this
- Textbook author verified that replaying an encrypted audio stream is not accepted as a valid Square transaction anymore
- So Square is changing the key, or using a nonce, or something similar