Phish and Chips
(Traditional and New Recipes for Attacking EMV)

Ben Adida, Mike Bond, Jolyon Clulow, Amerson Lin, Steven Murdoch, Ross Anderson, and Ron Rivest

No Institute Given

Abstract. This paper surveys existing and new security issues affecting the EMV electronic payments protocol. We first introduce a new price/effort point for the cost of deploying eavesdropping and relay attacks – a microcontroller-based interceptor costing less than $100. We look next at EMV protocol failures in the back-end security API, where we describe two new attacks based on chosen-plaintext CBC weaknesses, and on key separation failues. We then consider future modes of attack, specifically looking at combining the phenomenon of *phishing* (sending unsolicited messages by email, post or phone to trick users into divulging their account details) with chip card sabotage. Our proposed attacks exploit covert channels through the payments network to allow sabotaged cards to signal back their PINS. We hope these new recipes will enliven the debate about the pros and cons of Chip and PIN at both technical and commercial levels.

1 Introduction

The EMV[1] protocol suite – the technology underlying “Chip and PIN” – has now existed in one form or another for over ten years, though it has only been deployed in Europe for less than two years. Over this period there have been plenty of hypothetical attacks and fixes to the protocol in turn, yet it is only since deployment that there has been enough clarity to fully explore possible weaknesses both at a design and implementation level.

In this paper, we look at the big picture of EMV, exploring the feasability of attacks exploiting the fundamental shortcoming of smartcard-based systems – lack of a trusted user interface. We then look at technical errors in the detail, specifically at how the bank’s security API to receive and process messages from chip cards implements the required functionality; we show that several vendors have tripped up here. Finally we consider brand new attack modes that may become important in the future (once the easier vulnerabilities have been counteracted), specifically looking at combining technical sabotage attacks with the ever-problematic phising phenomenon.

We hope that this whistle-stop tour shows that whilst EMV is undeniably a robust and secure payment protocol at heart, there is so much matter and

[1] EMV is named after the original contributing corporations: Europay, Mastercard and Visa

complexity around the edges to get wrong that there will be plenty to keep the criminals fed and watered in the future; we look forward in particular to *phish and chips*!

Section 2 of this paper describes eavesdropping and relay attacks, section 3 describes API attacks at the back end, and section 4 considers phishing attacks. We conclude in section 5.

2 Eavesdropping & Relay Attacks

While a smartcard is a very convenient form-factor to carry, it lacks a trusted user interface, unlike for instance a mobile phone. This means the PIN cannot be provided directly to the card, so there is the possibility of eavesdropping en-route. Lack of trusted display means there is no way to confirm who you are doing business with, and what amount is being transacted, so it becomes possible to relay the entire data stream to another location. Let's look at these two well-known drawbacks in more detail.

2.1 Eavesdropping POS Terminals

If account and PIN data can be eavesdropped from an EMV transaction at a Point-Of-Sale (POS) terminal, it is easy to make a magnetic stripe card containing that data, for fraudulent use in a foreign country where EMV is not supported. Eavesdropping equipment is already commonplace for unmanned ATMs, usually consisting of a overlay for the card slot and a concealed camera. However there are multiple approaches to eavesdropping POS equipment, each with advantages and drawbacks:

- *Camera and Double-swipe.* The most basic approach requires a collusive merchant. The merchant positions a camera with view of the PIN pad, and secretly swipes the card through his own equipment before inserting it into the genuine device.
- *Hacked Terminal.* A real POS terminal is opened up and additional circuitry/probes are added to monitor the keypad, and record the data from the smartcard.
- *Counterfeit Terminal.* The shell of a genuine POS terminal is used to make a counterfeit, which appears to accept card and PIN, but performs no transaction. Alternatively the counterfeit may pass on the data stream from the smartcard to a hidden genuine terminal, but a physical actuator system to enter the PIN on the real terminal may be required.
- *Terminal Skimmer.* A miniaturised interceptor device can be overlayed on top of the smartcard slot on the POS terminal. From this position it can intercept the smartcard stream (capturing account details), and also the PIN. The PIN can be captured here because most European systems are using the cheaper *Static Data Authentication* EMV cards, which possess no private key, thus the PIN can only be sent in the clear.

The camera and double-swipe approach is definitely workable, but a significant disadvantage is the level of collusion from the merchant required, in order to set up the camera and conceal the second magstripe reader. In addition, a human must go through the video footage study the PINs entered, then correlate with the captured data, which is time consuming and error-prone. Modifying a working terminal requires bypassing of the tamper-resistance, and though this is unlikely to be of a high standard, the attack is still technically very complicated, and requires considerable manual effort for each terminal sabotaged.

The counterfeit terminal approach is appealing, and scales better than a hacked terminal. However the effort required to program a brand-new terminal counterfeit (particularly to drive the receipt printer and LCD display) is substantial. It works well within the business model of giving the customer a free lunch in return for his card and PIN data, but ideally the corrupt merchant would like to mix genuine and counterfeit business over the course of the day. Setting up solenoid actuators so that the PIN can be forwarded is a further complication.

In all, our preferred approach in terms of cost, development time and convenience is to create a skimming device which sits on the smartcard slot, and captures card and PIN data. We created a prototype device using an EZ-USB microcontroller and laptop computer, costing in total comfortably under $100, in development time of approximately one man-month. Our prototype is shown in figure 1.

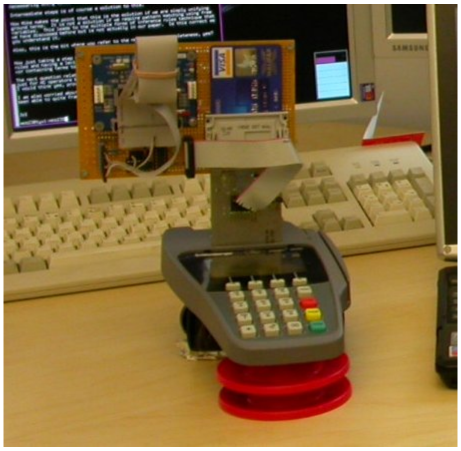

Fig. 1. Prototype POS skimmer.

It can trivially capture and store account details and PINs for SDA cards in large quantities. Such devices exist for smartcard and POS development, but cost more like $2500 per device. A finished device could be slotted onto the terminal in seconds, and removed equally quickly should there be a problem. It is thus easier to deploy in environments with only limited collusion from the merchant, and more deniable than installing complex counterfeit terminal equipment or hidden cameras.

2.2 Smartcard Relay Attacks

Eavesdropping attacks collect account and PIN data for use at a later date, but rely on the magnetic stripe fallback mode of operation, something which might one day be discountinued (though it is unlikely). However if the attackers are well-prepared, they can use the access to the customer's card and PIN in real-time: this is called a *relay attack*.

For example, as you pay for a meal in a dodgy restaurant using your chip card, the waiter's sabotaged reader could simply forward all the traffic from your card wirelessly to an accomplice at a jewellers on the other side of town. The smartcard data stream would go maybe via GPRS to a PDA in the crooks pocket, then to his fake card, and the captured PIN read out via a headphone in his ear. You think you're paying for lunch, but in fact you're buying the crooks a diamond!

This sort of relay attack is a variation on the counterfeit terminal eavesdropping attack, and we imagine the equipment reqired to deploy it would cost less than $2000, though substantial development and debugging time would be required. We discuss possible countermeasures to the relay attack problem in a forthcoming paper [1].

3 Back-end API Attacks

Back at the bank data centre, a rack of Hardware Security Modules (HSMs) are tasked with providing the back-end support for EMV cards in the field. There are two major roles: *processing authorisation requests and responses*, and *sending secure messages*. An authorisation request or response is simply a MAC over specific transaction data fields, constructed using a specially derived 3DES key shared between HSM and smartcard. A secure message can be thought of as an authenticated script command sent to a card, which usually acts to update some internal variable in the smartcard's non-volatile memory. Secure messages can have encrypted fields, for instance so that a new PIN can be securely sent to the card.

3.1 EMV Secure Messaging in the IBM CCA

IBM's Common Cryptographic Architecture is a popular security API implemented by IBM mainframes and in the 4758. As part of our study of EMV, we looked at the recently-added support for EMV transactions in both the CCA API and the Thales RG7000 series API. We found several vulnerabilities in the support for secure messaging, which are described in detail in a forthcoming paper [2]. These attacks are significant because they show that the EMV protocol has not mitigated the risks of abuse by bank programmers at operations centres, and insider attack there can rapidly undermine the system.

We now briefly describe the attack on the CCA's `Secure_Messaging_For_Keys`, which allows us to extract secret keys (and PIN updates) being sent to a smartcard, and inject our own keys and messages without authorisation. The CCA command `Secure_Messaging_For_Keys` is basically a special kind of key export. It takes a key stored locally on an HSM, decrypts it, then formats it up as part of a *secure message*. This secure message format is specified by template input arguments to the command – consisting of a template and and offset at which to insert the encrypted data. The command then re-encrypts the message under a specially derived key shared between the HSM and the destination smartcard. Finally, a separate command `MAC_Generate` is used to create an authentication code over the whole message. Here is the `Secure_Messaging_For_Keys` call in detail:

$$\mathit{template},\ \ \mathit{offset}\ \ , \{K_1\}_{KM/T}, \{K_2\}_{KM/SMSG} \longrightarrow \{\mathit{template}[K_1 :\ \ \mathit{offset}\ \ ]\}_{K_2}$$

– *template*: the message template, a byte-string to be used in preparing the plaintext.
– *offset* : the offset within *template* where the key material should be placed.
– $\{K_1\}_{KM/T}$: K_1 is the payload – a key to export to the smartcard. KM/T represents an encryption key used to store the payload key locally.
– $\{K_2\}_{KM/SMSG}$: K_2 is the key shared between HSM and EMV smartcard. This is used to encrypt the confidential data within the secure message.
– $\mathit{template}[K_1 :\ \ \mathit{offset}\ \ ]$: represents the template plaintext *template* interpolated with key material K_1 at offset *offset* .
– $\{\mathit{template}[K_1 :\ \ \mathit{offset}\ \ ]\}_{K_2}$: the finished result – an encrypted secure message consisting of template with $K1$ interpolated, all encrypted under $K2$.

3.2 Construction of an Encryption Oracle

Our injection and extraction attacks work by gaining access to an encryption oracle. We first note that the CBC mode used in `Secure_Messaging_For_Keys` has an unfortunate malleability property: a ciphertext can be truncated to create a ciphertext of an identically truncated plaintext – so long as the truncation is block-aligned. Thus, we can thus construct an encryption oracle for an arbitrary input message m as follows:

$$\mathsf{EncryptionOracle}\Big(\mathit{plaintext}, \{K_2\}_{KM/SMSG}\Big):$$

1. create a template *template* by extending *plaintext* by a single block, e.g. the 0-block.
2. set the *offset* to $|\mathit{plaintext}|$, which is effectively the beginning of the 0-block just added.

3. perform the call to `Secure_Messaging_For_Keys` using any available exportable key $\{K_1\}_{KM/T}$:

   $$plaintext || \text{“00000000”}, |plaintext|, \{K_1\}_{KM/T}, \{K_2\}_{KM/SMSG} \longrightarrow c$$

   the HSM will fill in the last block *template* (as indicated by *offset*) with K_1, leaving the entire *plaintext* component of *template* untouched.
4. consider the first $|plaintext|$ blocks of c, effectively discarding the last block. This truncated value is simply $\{plaintext\}_{K_2}$, our desired result.

This very straightforward observation undermines any security merits of the template-fill-in operation of the HSM – the programmer might as well be able to use the special wrapping key shared between HSM and card in a conventional `Data_Encrypt` command.

3.3 Extracting Keys

Such message injection can compromise the operation of particular cards actively, for instance by constructing a message containing a known PIN for the card. However active attacks at a bank data centre carry a significant risk of revealing the attacker's location, so retrieval of communications keys or PINs without affecting card state is far more dangerous.

We now show how to expand the above oracle into a partial-key dictionary attack mechanism: using this approach, we can rapidly extract the key from any encrypted data field in a secure message, one byte at a time. In our explanation, we use `[ ... ]` to denote hex notation of a single 8 byte block. Here is the algorithm:

$\mathsf{ExtractKey}\left(\{\mathsf{K}_1\}_{KM/\mathsf{T}}\right)$:

1. prepare 256 plaintext blocks of the form `[0000 0000 0000 00yy]` where `00` $\leq$ `yy` $\leq$ `ff`.
2. use EncryptionOracle on all of 256 plaintext blocks to generate a dictionary of 256 ciphertexts indexed by the ciphertext: $\{\forall yy, \texttt{00} \leq yy \leq \texttt{ff} : (c, yy)\}$.
3. given any secure messaging key $\{K_2\}_{KM/SMSG}$, make an API call as follows:

   $$\texttt{[0000 0000 0000 0000]},\ \ offset = 7, \{K_1\}_{KM/T}, \{K_2\}_{KM/SMSG} \longrightarrow c$$

4. compare c against the dictionary of ciphertext-indexed bytes. The match yields the first byte of the key, call it `aa`.
5. in order to discover the next byte of the key, repeat the process with a dictionary built from 256 plaintext blocks of the form `0x000000000000aayy`, with an *offset* of 6. This will yield the 2nd byte `bb` of K_1. By continually shifting the key over by one block, we can extract the entire key, one byte at a time.

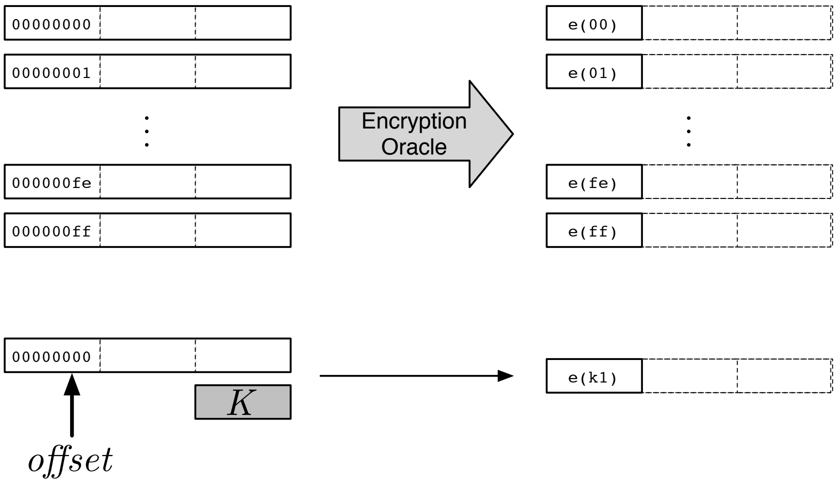

Fig. 2. In the key-shifting attack, A 256-element dictionary is built up for each byte of the key that we want to check.

For a k-byte key, it takes $257k$ queries to extract the whole key: 256 to build up each dictionary, and one more query to identify the specific key byte. Thus a DES key can be extracted in 2056 queries, while a two-key 3DES key can be extracted in 5112 queries. With such an attack, a key update message between bank and card could be eavesdropped, and then a cloned chip card produced, or PINs could be discovered at will. It is interesting to see that while there is nothing wrong with the concept of a secure message in the EMV standard, the flexibility and extensibility requirements of the protocl have made it difficult to implement in an API. It seems IBM chose to make a general-purpose API command, which supported arbitrary secure messages, but unfortunately was also open to abuse.

4 Phishing with Chips

Sections 2 and 3 show that EMV users must brace themselves for live deployment of upgraded skimming attacks, as well as inevitable implementation issues with the protocols coming out of the woodwork. But what of blue-sky future attacks, assuming practical measures are found to ameliorate the problems of eavesdropping and back-end attack?

The term *phishing* may have arisen in an internet context, particularly to capture online banking details, but conventional payment systems have been subject to exactly the same social-engineering tricks for some time – by both post and telephone. These include postal redirection scams, unauthorised re-sale (and re-pricing) of another's service, brazenly phoning up and asking for the customer PIN or CVV, and extension of fraud opportunity by faking card cancellation or fraud prevention calls to a customer. But now consider the introduction of chip cards, which has primed the customer to be receptive to new technology and banking requirements with little explanation. Here is an example scam:

1. Open a dummy bank acount in the name of Bob, stick 100 quid in it. Set the PIN to 0000.
2. Set up a skimming scam and get some account numbers, or buy simply some account numbers wholesale. Assume you now have the card number of the victim, Alice.
3. Prepare a counterfeit card from a fictional credit card company bearing Alice's name, but containing Bob's card details.
4. Send the card to Alice, purporting to be a free offer "GBP2000 of credit, and 100 quid pre-charged. Your initial PIN is 0000. Be sure to change it to a memorable number as soon as possible."
5. Wait for Alice to use her free card. Now phone up and request PIN re-advice.
6. When PIN re-advice arrives, it will come to an address under your control, you now have both Alice's PIN (step 5) and her card number (step 2).

The advantage of this scam is that it requires no ability to counterfeit cards, however if opening an account with false name and address is possible the criminal might be better off simply abusing this card straight away. But it highlights the receptiveness of customers to being sent fresh instructions. Consider now how this scam could be improved if the card sent to a customer was sabotaged:

1. Open several bank accounts in false names, F1...Fn. Stick a small float in each (£50–£100 quid). Alternatively apply for several credit cards.
2. Select a target individual Alice, whose address you know.
3. Create a counterfeit card bearing Alice's name, but with an evil smartcard chip on board running your custom firmware. Tell alice that this card replaces her existing one, which she should cut up.
4. When Alice first uses this card, the evil chip will receive her true PIN from the POS terminal in the clear. It now must signal this back to the bad guys using a covert channel through the POS network.
5. Basically, over the next seven days or so, whenever the evil chip detects it is being used for a low value transaction, it switches its identity to assume one of the false identities F1...Fn.
6. The 14 or so bits of information in the PIN are transmitted through this covert chanel of the choice of false identity, and the timing information about which day and hour the identity is switched to. The card has a rough impression of the passage of time from logging the transaction details.
7. The bad guys use internet banking to watch the transaction logs of accounts F1...Fn and from this receive Alice's PIN. They now make up a magstripe of Alice's card and start raiding her account.

Banks have gone to great effort and expense to prevent chip sabotage during the personalisation process, yet a phishing attack provides a perfect solution and bypass of this security – send the customer a sabotaged card and tell her to throw away her old one. Should the fraudsters desire that Alice's card remains fully operational, they need only acquire a legitimate chip through fraudulent means, and embed it in the smartcard in addition to the sabotaged chip. This real chip can then periodically be called upon, for instance if the sabotaged chip wishes for an expensive purchase to succeed.

The key benefit of this scam over brazen solicitation of a PIN (for instance by phone) is that it uses a route through the valid and existing bank network to add the appearance of validity to the scam. The advantage of using choice of account to charge to as a covert channel is that the criminals can collect the resulting PIN data remotely via the internet. A less intricate approach could be to encode the PIN in transaction data which is written onto the paper receipt upon payment (for instance into the *application preferred label* field, or *transaction certificate*). The fraudsters then simply raid the garbage of their target customer, or discarded receipts from a local store in the target area of the scam. Given the transaction data could be routed anywhere, the card could in fact contact a foreign mafia-controlled bank, where an insider simply reads out the PIN chosen.

5 Conclusions

EMV is a massive and complex protocol suite. To get a proper understanding of where the real security threats are, we must consider all the protocols together. Some of these protocols are conventional cryptographic protocols, such as the secure messaging standard; some are Security APIs, such as the interface to the EMV-compliant HSM, and the API to the smartcard itself. Thirdly, some are human protocols, rules and procedures for authenticating, and communicating secrets to humans. Finally there is the economic aspect, where there are not protocols as such, but there are conscious design decisions which manipulate the threat surface by affecting the cost of attack.

This paper has shown that every aspect of EMV needs to have attention paid to it if the system is going to have the desired security properties. It can hopefully encourage other designers and analysts to look at the big picture and the context of a specific protocol before making a crucial decision. The European banks have a very tough goal if they are trying to wipe out banking fraud; if they are willing to accept some failure modes, it is just as important that when these cracks do appear, they appear in the right places. The specific examples detailed here indicate that the system may already have *unexpected* failure modes: maybe controlling breakdown of a complex system is an even harder challenge than making one that is totally secure!

6 Acknowledgements*

The author would like to thank George Danezis for useful ideas and discussion in relation to phish and chip scams; the suggestion to encode PIN data in receipts from transactions is due to him.

References

1. R. Anderson, M. Bond, "The Man-in-the-Middle Defence", to appear at Cambridge Security Protocols Workshop 2006, Mar 27–29th 2006.
2. B. Adida, M.Bond, J.Clulow, A.Lin, R.Anderson, R.Rivest, "On the Security of the EMV Secure Messaging API", to appear.