Card Brand Mixup Attack: Bypassing the PIN in non-Visa Cards by Using Them for Visa Transactions*

David Basin, Ralf Sasse, and Jorge Toro-Pozo
Department of Computer Science
ETH Zurich

Abstract

Most EMV transactions require online authorization by the card issuer. Namely, the merchant's payment terminal sends an authorization request to the card issuer over a payment network, typically operated by the company that brands the card such as Visa or Mastercard. In this paper we show that it is possible to induce a mismatch between the card brand and the payment network, from the terminal's perspective. The resulting card brand mixup attack has serious security consequences. In particular, it enables criminals to use a victim's Mastercard contactless card to pay for expensive goods without knowing the card's PIN. Concretely, the attacker fools the terminal into believing that the card being used is a Visa card and then applies the recent PIN bypass attack that we reported on Visa. We have built an Android application and successfully used it to carry out this attack for transactions with both Mastercard debit and credit cards, including a transaction for over 400 USD with a Maestro debit card. Finally, we extend our formal model of the EMV contactless protocol to machine-check fixes to the issues found.

1 Introduction

There are more than 3.3 billion Visa credit and debit cards in circulation worldwide [23]. Under the Mastercard brand (excluding Maestro and Cirrus products) there are over 2 billion cards [22]. These two companies, together with Europay, are the founders of EMV, the *de facto* protocol standard for in-store smartcard payments. Other companies like American Express, JCB, Discover, and UnionPay have also joined the EMV consortium.

EMV transactions for high amounts require online authorization from the card issuer. For this, the payment terminal sends an authorization request to the card issuer, carrying transaction details and a cryptographic Message Authentication Code (MAC) computed by the card over these details. Upon reception, the card issuer performs various checks, including that the associated account has sufficient funds and that the MAC is correct. While these checks offer cryptographically verifiable guarantees to cardholders and merchants, one must understand the properties of the payment system as a whole, including the process by which terminals and issuers exchange requests and responses.

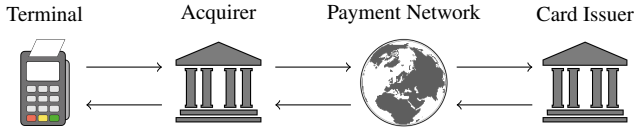

Figure 1: Communication flow for online transaction authorization. Upper and lower arrows represent the authorization request and response, respectively.

Figure 1 displays the communication flow of the online authorization process, involving four parties: (1) the payment terminal; (2) the merchant's acquirer, which is a bank or financial institution that processes card payments on behalf of the merchant; (3) the payment network, which connects the acquirer and the card issuer; and (4) the issuer itself. There are several payment networks, such as the Visa or Mastercard networks, and the mechanism by which the acquirer chooses the one which the authorization request is sent to is called *routing*. Typically, routing is based on the payment card's brand. For example, if the card is Visa branded, then the authorization request is routed to the Visa payment network.

The payment terminal can determine the card brand from different data objects supplied by the card during the transaction. These objects include the Primary Account Number (PAN) and the Application Identifiers (AID). From the PAN, more commonly known as the card number, the card brand can be inferred from the leading digits. For example, if the PAN starts with 4 then it is a Visa card. From the AIDs, which indicate the EMV applications that the card supports (e.g., Visa Electron or V Pay), the card brand can be inferred from the shared prefix, called the Registered Application Provider Identifier, which is usually a 10-digit value (5 bytes).

In this paper we show that it is possible to deceive a terminal, and by extension the acquirer, into accepting contactless

transactions with a PAN and an AID that indicate different card brands. Concretely, we have identified a man-in-the-middle attack that tricks the terminal into completing a Visa transaction with a Mastercard card.

Our attack, which we call a *card brand mixup*, has catastrophic consequences. In particular, it allows criminals to use a victim's Mastercard card to pay for expensive goods without entering a PIN. The attack effectively turns the card into a Visa card and then applies our recent PIN bypass attack [6]. In other words, *the PIN can be bypassed for Mastercard cards* too, which so far had been considered protected against unauthorized purchases for amounts that require the entry of the card owner's secret PIN.

This new attack abuses two fundamental shortcomings of the EMV contactless protocol: (1) the lack of authentication of the card brand to the terminal, and (2) an attacker can build all necessary responses specified by the Visa protocol from the ones obtained from a non-Visa card, including the cryptographic proofs needed for the card issuer to authorize the transaction.

We have built a proof-of-concept Android application and successfully used it to bypass PIN verification for transactions with Mastercard credit and debit cards, including two Maestro debit and two Mastercard credit cards, all issued by different banks. One of these transactions was for over **400 USD**.

We have extended our formal model of the EMV protocol, first presented in [6]. Concretely, we generalize its specification of the issuer and the terminal-issuer channel to model communication between the terminal and the issuer, even when they do not agree on the brand of the payment card used. Our extended model, available at [3] and specified in the Tamarin model checker [26, 28], is precise enough that its analysis uncovers the attack described here. We have also used our extended model to construct security proofs for two sets of fixes. The first set is the one we proposed in [6], which is specific to the Visa kernel. The second set of fixes, first presented in this paper, prevents card brand mixups in general and applies to all EMV kernels.

Contributions. First, by carefully analyzing the EMV protocol with a focus on the terminal-issuer interaction, we discover a novel attack that allows criminals to trick the terminal into believing that the card being used is of a brand that it is not. Surprisingly, this is possible even for transactions authorized online by the card issuer, who clearly does know the right card brand.

Second, we demonstrate that this card brand mixup is not just a mere disagreement between the card issuer and the terminal, but that it has serious consequences. In particular, the PIN does not protect Mastercard cardholders from lost or stolen cards being used in fraudulent purchases for large amounts. Consequently, the consumer should not be liable for fraudulent transactions in which the cardholder was presumably verified. This is known as the *liability shift* in the banking industry.

Finally, we analyze fixes that prevent both the card mixup and the PIN bypass attack. Namely, we extend our previous formal models and provide computer-checked security proofs for these fixes.

Organization. In Section 2 we provide technical background on our previous PIN bypass on Visa cards, which we leverage for our new attack, and the EMV contactless protocol. We then describe our card brand mixup attack and the resulting PIN bypass in Section 3. We also report on our proof-of-concept implementation and the results of our experiments. In Section 4 we analyze and verify countermeasures that secure online-authorized transactions. In Section 5 we elaborate on previous work that exposes and exploits flaws on the EMV standard and we draw conclusions in Section 6.

Ethics and Disclosure. No merchant, bank, or any other entity was defrauded. To test our attack, we setup and used our own SumUp terminal and merchant account. Note that, although the merchant infrastructure we used was our own, it is a fully realistic and functional one. We did not tamper with the hardware or software in any way.

After a successful disclosure process with Mastercard, they confirmed that our attack is effective. Mastercard identified all 9 transactions that were routed to their network when we carried out our Mastercard-Visa mixup attack. Mastercard has since implemented and rolled out defense mechanisms on their network and, in collaboration with Mastercard, we have conducted experiments where our attack failed with their mechanisms in place. Further details are given in Section 4.4.

2 Background

We first provide background on contactless payments and common attacks against them. We briefly recall our previous work [6], which we build upon. Afterwards, we provide technical details on the EMV contactless transaction.

2.1 Relay Attacks and PIN Bypass for Visa

Despite the undeniably smooth experience of a payment with the tap of a card, contactless payment technology has been exposed to numerous security issues. Payment terminals communicate wirelessly with the cards, and so can attackers. In particular, Near Field Communication (NFC), which is the communication technology that contactless payments use, allows any suitable NFC-enabled device to communicate with a contactless card and engage in fraudulent transactions.

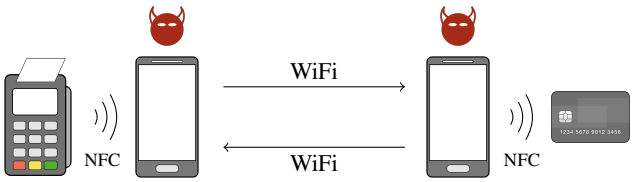

Figure 2: A relay attack on contactless payment. Devices from left to right: payment terminal, attacker's first mobile device, attacker's second mobile device, and victim's card.

While the range of an NFC signal is normally just a few centimeters, it can be extended to a much larger range by *relay attacks* [11, 19, 30, 8, 7]. A relay attacker uses two mobile devices, connected wirelessly, to make the victim's card engage in a transaction with a distant payment terminal. See Figure 2 for a graphical representation.

Relay attacks, however, do not appear lucrative for criminals because they are presumably feasible only for purchases for low amounts (e.g., under 25 EUR in most European countries), due to the need for the card's PIN for transactions with higher amounts. However, in our previous work, we discovered a man-in-the-middle attack that allows criminals not only to perform relay attacks but also to bypass the PIN for contactless transactions with Visa cards.[1]

At a technical level, this attack consists simply in setting the Card Transaction Qualifiers (CTQ) to the value `0x0280`. The CTQ is a data object transmitted from the card to the terminal and instructs the latter which Cardholder Verification Method (CVM) must be used for the transaction. The CTQ value `0x0280` tells the terminal that PIN verification is not required and that the cardholder has been verified on the consumer's device (see [17], pp. 69–70). The flaw in the Visa protocol that leads to this attack is the lack of authentication of the CTQ data object.

This attack does not apply to the Mastercard protocol because, in contrast to the Visa protocol, the card's (lack of) support for cardholder verification on the consumer's device is cryptographically protected against modification. A computer-checked proof of this can be found at [4].

2.2 The EMV Contactless Protocol

EMV's specification for contactless transactions comprises over 1,200 pages of documentation. In this section we summarize this specification. We split our summary into the four overlapping phases of a contactless transaction and briefly indicate, where applicable, the underlying security shortcomings that our attack exploits.

[1] Demo at `https://youtu.be/JyUsMLxCCt8`

2.2.1 *Application Selection*

A transaction is performed using one of the six EMV contactless protocols. Every transaction starts with the application selection process, where the terminal issues a SELECT command and the card submits the Application Identifiers (AIDs) for the supported applications (a.k.a. kernels or protocols). Based on the AIDs received, the terminal activates a kernel for the transaction, which is one of the following:

- Kernel 2 for Mastercard AIDs,
- Kernel 3 for Visa AIDs,
- Kernel 4 for American Express AIDs,
- Kernel 5 for JCB AIDs,
- Kernel 6 for Discover AIDs, and
- Kernel 7 for UnionPay AIDs.

The most relevant kernel for our work is Mastercard's, which we outline in Figure 3 and is specified in the 590-page document [16].

2.2.2 *Offline Data Authentication*

After a kernel has been activated and announced to the card via a second SELECT command, the card requests the Processing Data Object List (PDOL), which indicates some of the transaction-specific data objects needed by the card for the protocol. These data objects include, but are not limited to, the transaction amount, the terminal's country code, and a terminal-generated random number.

Using the GET PROCESSING OPTIONS command, the terminal supplies the requested PDOL data to the card. The latter responds with the Application Interchange Profile (AIP) and the Application File Locator (AFL). The AIP informs the terminal of the card's capabilities and the AFL is a data object that the terminal uses to request the card's static data (also known as records) using the READ RECORD command. These records include:

- *Primary Data* such as the card number (called the Primary Account Number), the card's expiration date, and the list of the supported CVMs;
- *PKI Data* such as the card's Public Key (PK) certificate, the card issuer's PK certificate, and the PK index of the Certificate Authority (CA);
- *Processing and Risk Data* such as the first and second Card Risk Management Data Object Lists (CDOL1 and CDOL2, respectively), which typically include the PDOL and further transaction-specific data.

At this point, the terminal cryptographically authenticates the card. This process is called Offline Data Authentication (ODA) and uses one of the three methods:

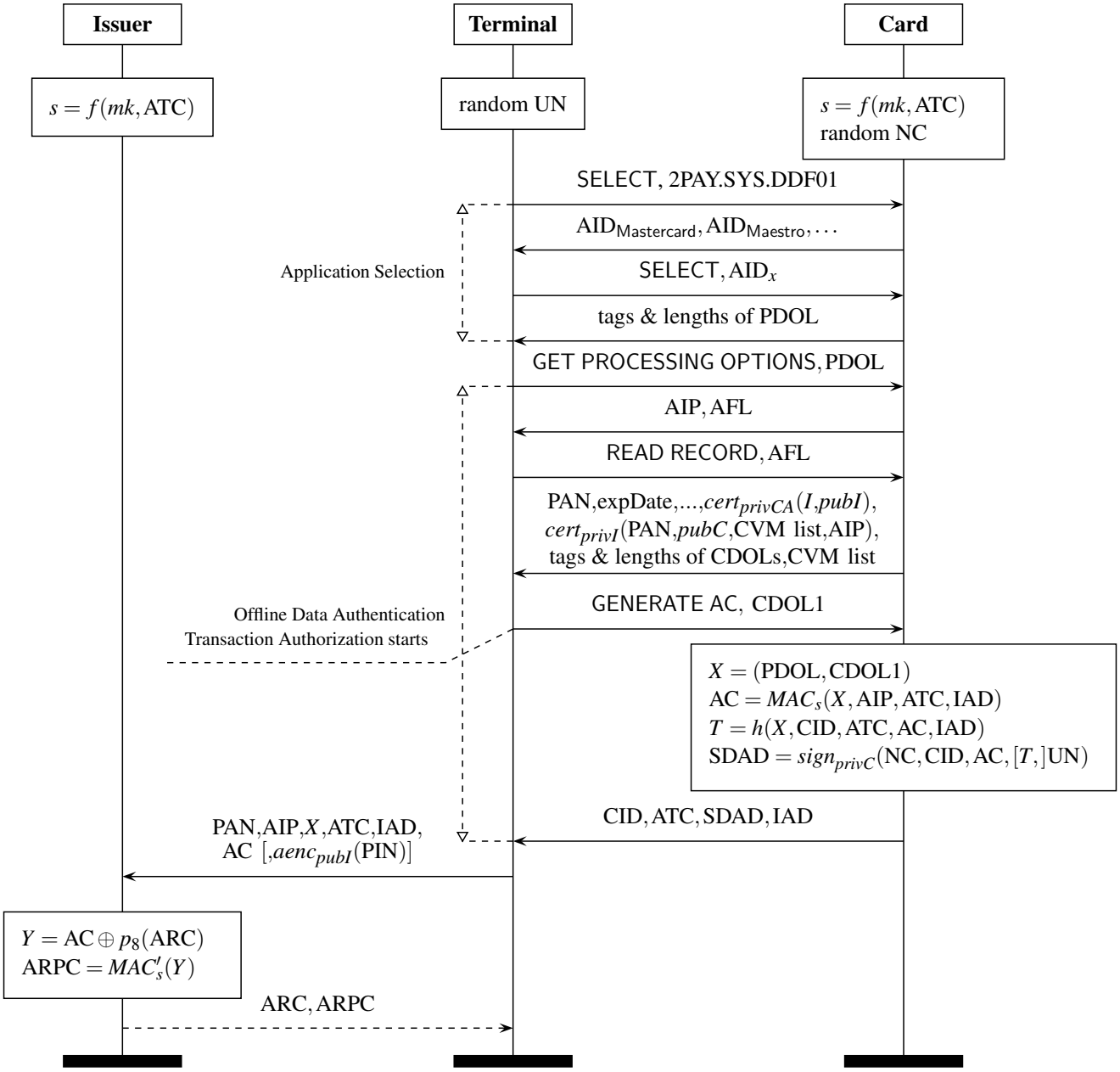

Figure 3: Overview of the Mastercard contactless transaction using the most common card authentication method, called Combined Dynamic Data Authentication (CDA). There are other two authentication methods, which we omit here for simplicity. Notation: $\oplus$ is exclusive-OR; f is a key derivation function; $(privC, pubC)$, $(privI, pubI)$, and $(privCA, pubCA)$ are the private/public key pairs of the card, the issuer, and the Certificate Authority, respectively; $cert_k(cont)$ is the PKI certificate on $cont$ signed with the private key k; $sign_k(m)$ is the signature on m with the key k; $aenc_k(m)$ is the asymmetric encryption of m with the key k; $MAC_k(m)$ and $MAC'_k(m)$ are cipher-based Message Authentication Codes (MAC) on m with the key k; $p_b(m)$ is the right-padding of m with b zero bytes. Note that there is some overlap between the Offline Data Authentication and the Transaction Authorization phases. This occurs when the terminal and the card agree on using the Combined Dynamic Data Authentication (CDA) method. For the sake of simplicity, we have omitted the middle entities (acquirer and payment network) that participate in the terminal-issuer exchanges before they reach their recipient.

1. *Static Data Authentication (SDA)*: the card transmits a signature by the card issuer on the card's static data such as the Primary Account Number (PAN), the card's expiration date, and the AIP. This signature, called the Signed Static Authentication Data (SSAD), is generated and stored on the card during production.

2. *Dynamic Data Authentication (DDA)*: in this method the terminal sends the INTERNAL AUTHENTICATE command with the Dynamic Data Object List (DDOL) as payload. The DDOL is a data object that must include the terminal's fresh number, called the Unpredictable Number (UN). The card replies with the Signed Dynamic Authentication Data (SDAD): a signature on its own fresh number NC and the DDOL.

3. *Combined Dynamic Data Authentication (CDA)*: this method also involves the SDAD, but includes additional transaction data in the signature such as the amount. No INTERNAL AUTHENTICATE command is used and instead the SDAD is later supplied by the card, if requested by the terminal's GENERATE AC command. This ODA method actually belongs, chronologically speaking, to another phase of the transaction, called the Transaction Authorization, which we describe later in Section 2.2.4.

The ODA method chosen is typically the last one (which is also the strongest one) in the above list that both the terminal and the card support. The ODA methods that the card supports are encoded within the AIP.

2.2.3 Cardholder Verification

The Cardholder Verification Methods (CVMs) are as follows:

1. *Online PIN*: the terminal sends to the card issuer the encryption of the PIN entered on the terminal's pad for verification.

2. *Consumer Device CVM*: the cardholder verification is performed on the consumer's device. This method is intended primarily for use with mobile payment apps such as Google/Apple Pay, where the cardholder is verified through biometrics such as fingerprint or face recognition.

3. *Paper Signature*: the cardholder signs (with a pen) the purchase receipt and the cashier checks it against the physical signature on the card's backside.

If applicable, typically when the amount is above the CVM-required limit, the terminal verifies the cardholder by choosing one (or two) of the above three methods. The choice depends on the card's list of supported CVMs, if supplied by the card. If this CVM list is not supplied (e.g., in Visa transactions), then the terminal proposes online PIN verification, and this proposal is encoded within the Terminal Transaction Qualifiers (TTQ) or a similar data object, depending on the kernel. The TTQ is typically part of the PDOL.

Notably relevant for our previous PIN bypass attack, and therefore this new attack, is the Consumer Device CVM (CDCVM). With respect to how and whether the CDCVM is used, the kernels can be divided into two groups:

The Visa group composed of the Visa, Discover, and UnionPay kernels, where the card's support for the CDCVM is announced to the terminal through the *cryptographically unprotected* CTQ or similar data object, depending on the specific kernel.

The Mastercard group composed of the Mastercard, American Express, and JCB kernels, where the card's support for the CDCVM is announced to the terminal through the *cryptographically protected* AIP and possibly additional data objects, depending on the specific kernel.

Our previous PIN bypass attack targets the cards within the Visa group, which is weaker than the Mastercard group in terms of the protection it offers. While the CDCVM is not meant for physical cards, attackers can abuse it by tricking the terminal into accepting this CVM for a purchase with a victim's physical card. The key point here is that, whenever an attacker convinces the terminal that the CDCVM was successfully performed, the latter wrongfully assumes that the actual verification was delegated to an external device and thus does not ask for the PIN. This is the essence of the flaw that our previous attack exploits.

Our new attack also exploits the Consumer Device CVM, but in combination with a flaw on EMV's application selection. This attack thereby targets the cards within the presumably better protected Mastercard group.

2.2.4 Transaction Authorization

Transaction authorization is implemented by having the card compute and transmit the Application Cryptogram (AC). This is a MAC-based cryptographic proof of the transaction, computed over the transaction details, the AIP, and the Application Transaction Counter (ATC, which is incremented on every transaction). Besides the AC and additional data that depends on the kernel, the card transmits:

- the Cryptogram Information Data (CID), which encodes the type of authorization being requested;
- the Application Transaction Counter (ATC);
- the Signed Dynamic Authentication Data (SDAD), if CDA was requested in the command payload; and

- the Issuer Application Data (IAD), which contains proprietary application data that is transmitted to the issuer.

The computation by the card (and verification by the issuer) of the AC uses a session key s, which is derived from the ATC and a symmetric key mk only known to the issuer and the card. The terminal therefore cannot verify the AC.

A transaction can be *authorized offline* by the terminal, sent *online* for *authorization* by the issuer, or *declined offline* by the card. The choice depends on factors including checks made by both the terminal and the card on transaction details such as the amount, the currency (transaction versus card's), the country (transaction versus issuer's), and the limit number of consecutive offline transactions. The most common type of transaction authorization is online by the issuer.

For transactions performed with the kernels within the Visa group, the AC is sent within the card's response to the GET PROCESSING OPTIONS. Typically, no Offline Data Authentication process is performed and no GENERATE AC command is used. For those kernels within the Mastercard group, the AC is transmitted in response to the GENERATE AC command.

If the transaction is to be authorized online by the issuer, then the AC is called the Authorization Request Cryptogram (ARQC) and the CID equals `0x80`. The actual authorization follows from a request-response exchange between the terminal and the issuer. The terminal's request carries the ARQC and the issuer's response is encoded in the Authorization Response Code (ARC). This exchange is not further specified by EMV.

If the transaction is to be accepted offline by the terminal, then the AC is called the Transaction Cryptogram (TC) and the CID equals `0x40` in this case. Also, the terminal is assumed to have already validated the transaction in the Offline Data Authentication phase. The transaction can be also declined offline, in which case the AC is called the Application Authentication Cryptogram (AAC) and the CID equals `0x00`.

Note that the AIDs are not *authenticated* by the card to the terminal. That is, the terminal has no cryptographic proof that the card supports the AIDs it advertised during the application selection phase. This turns out to be the new, fundamental security shortcoming that our attack exploits. Also note that EMV does not specify any mechanisms to match up the card's PAN with the advertised AIDs.

3 PIN Bypass via Card Brand Mixup

We describe our attack in detail here. We start in Section 3.1 by describing the threat model considered for this attack. We next give a step-by-step description in Section 3.2. Afterwards, in Section 3.3 we outline the hardware and software infrastructure we used in our proof-of-concept implementation and present the results of our experiments.

3.1 Threat Model

The threat model considered for this attack and for our formal analysis described in Section 4 is as follows:

1. The attacker has access to the victim's card.
2. The attacker has the capabilities of an active (so-called Dolev-Yao) attacker over the wireless channel between cards and terminals. Namely, the attacker can read, block, and inject messages on this channel.
3. The channel between the payment terminal and the banking infrastructure is *secure* in that it satisfies authenticity and confidentiality.

This models is realistic in practice. The attacker may access a victim's card that is lost or stolen. Indeed, in practice it may suffice simply to be physically close (within a few centimeters) to the victim's card. Moreover, as we will see in Section 3.3, using standard NFC-enabled smart phones one can carry out active man-in-the-middle attacks on the wireless channel.

3.2 Description of the Attack

As stated in [21, 6], the PIN verification *cannot* be bypassed for transactions where the payment terminal executes the Mastercard kernel (recall Figure 3). According to this kernel's specification [16], the AIP (specifically bit 2 of byte 1) is the only data object that indicates the card's support for on-device cardholder verification. Thus, modifying the AIP would lead to a declined transaction given that it is authenticated using the card's PK certificate, the Application Cryptogram (AC), and the Signed Dynamic Authentication Data (SDAD). We have validated this with several cards.

Unlike the AIP, the card's Application Identifiers (AIDs) are not protected. In fact, the AIDs are only used during the SELECT command exchanges. After these exchanges are completed, the terminal activates the corresponding kernel based on the AIDs received from the card. For example, if the preferred AID (or first, depending on the terminal's selection method) is $\text{AID}_{\text{Visa}} = \texttt{0xA0000000031010}$, then the terminal activates the Visa kernel. If the AID is instead $\text{AID}_{\text{Mastercard}} = \texttt{0xA0000000041010}$, then the terminal activates the Mastercard kernel.

Due to this lack of authentication of the AIDs, an attacker can maliciously replace them and thereby activate a desired kernel on the terminal. This is the fundamental security shortcoming that our attack exploits. An overview of the attack is displayed in Figure 4 and a step-by-step description follows.

1. *Activation of the Visa Kernel*: The terminal first activates the Visa kernel. For this, the attacker applies the trick just described, namely the replacement of the card's legitimate AIDs with AID_{Visa}.

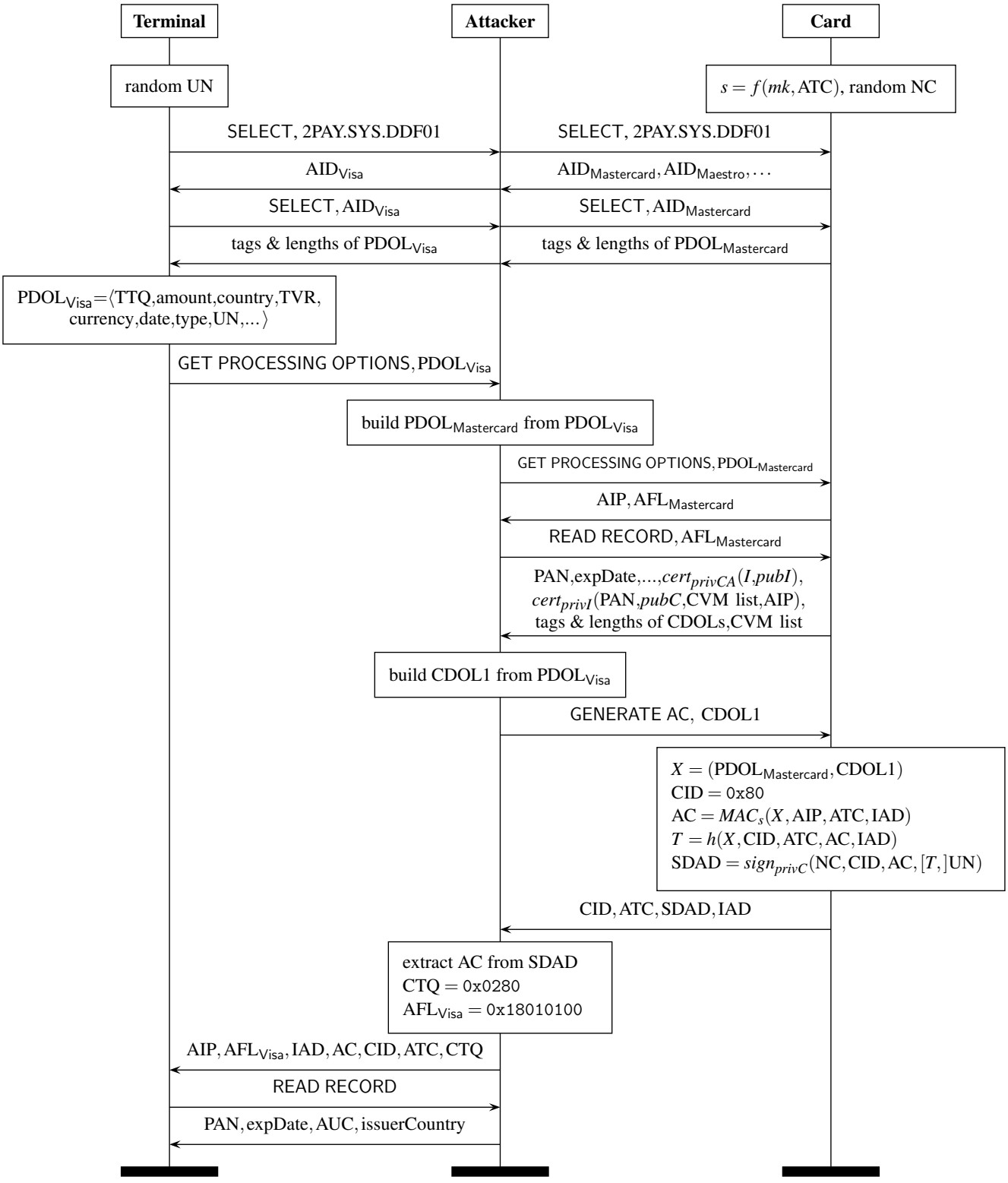

Figure 4: Overview of our PIN bypass attack for Mastercard, exploiting the card brand mixup. The attacker poses as (a) a card to the payment terminal and runs a Visa session with it, and (b) a payment terminal to the card with which it runs a Mastercard session. For simplicity, we have omitted the messages between the terminal and the issuer, which are the same as in Figure 3 but without the PIN block.

2. *Request Processing Options*: After the AID is negotiated, the attacker receives from the card the request (i.e., tags and lengths) for the Processing Data Object List (PDOL). The attacker forwards this request to the terminal with the addition of the request for the Terminal Transaction Qualifiers (TTQ) and all other processing data objects specified by the Visa kernel. The attacker's request also includes the data objects referenced by the First Card Risk Management Data Object List (CDOL1) specified by the Mastercard kernel, which usually are the Terminal Type (TT) and the Cardholder Verification Method Results (CVMR).

3. *Run the Mastercard Session*: Once the attacker has received the GET PROCESSING OPTIONS from the terminal, the attacker runs a Mastercard session with the card. The terminal is not involved during this step. The sub-steps are as follows.

   (a) The attacker builds and sends to the card the GET PROCESSING OPTIONS command along with the card's requested PDOL data, which is filled up from the terminal's command payload. The card responds to the attacker's command with the Application Interchange Profile (AIP) and the Application File Locator (AFL).

   (b) The attacker proceeds to read the card's records, using the received AFL. The relevant records collected are the PAN, the card's expiration date, the issuer country code, the Application Usage Control, and the CDOL1 tags and lengths.

   (c) The attacker builds and sends to the card the GENERATE AC command, whose payload is the CDOL1 data filled up with the PDOL data parsed from the payload of terminal's GET PROCESSING OPTIONS command. The CDOL1 typically is a superset of the PDOL. If the card supports CDA (i.e., bit 1 of byte 1 of the AIP is set), then the command should request CDA. Also, the bits 7 and 8 of the command's reference control parameter (i.e., byte 3) must be cleared and set, respectively. This tells the card that an ARQC is being requested (see [15], pp. 54–55).

   (d) From the card's response to the GENERATE AC command, the attacker collects the CID, the ATC, the IAD, and the AC or SDAD, depending on whether CDA was requested. If the SDAD is sent, then the attacker must extract the AC, using the card's Public Key (PK) (see [14], pp. 68–69).

   Using the received card's records, the attacker retrieves the card's PK using the following steps (see [14], pp. 60–65):

      i. retrieve the CA's PK from the CA's index,

      ii. retrieve the issuer's PK from the issuer's PK certificate, using the CA's PK, and

      iii. retrieve the card's PK from the card's PK certificate, using the issuer's PK.

4. *PIN Bypass*: At this point, our PIN bypass attack on Visa is applied. That is, the attacker injects a CTQ data object valued `0x0280`, which instructs the terminal that online PIN verification is not required and that the Consumer Device CVM was performed (see [17], pp. 69–70).

   Together with the CTQ, the attacker supplies the AIP, an artificial AFL with value `0x18010100`, the AC, the IAD, and all other data objects specified by the Visa kernel.

5. *Transmit Records*: In response to the terminal's READ RECORD command, which is `0x00B2011C00` due to the artificial AFL, the attacker replies with the PAN, the expiration date, the Application Usage Control (AUC), and the issuer country.

3.3 Carrying out the Attack

To demonstrate our PIN bypass attack, we developed a proof-of-concept Android application, comprising roughly 3,700 lines of Java code. On the merchant side, we used the payment kit commercialized by SumUp: an EMV and PCI DSS (Payment Card Industry Data Security Standard) certified company licensed under the UK's Financial Conduct Authority. The kit costs about 50 USD and includes a card reader, which works with both contact and contactless cards, and a back-end mobile application available for iOS and Android devices. The SumUp card reader is PCI PTS (Payment Card Industry PIN Transaction Security) certified. Figure 5 displays the components of our testing environment.

Our attack is implemented using two Android phones, connected through a relay channel built using TCP/IP server-client communication over WiFi. One phone runs our app in POS Emulator mode (Device 4 in Figure 5) and the other phone runs our app in Card Emulator mode (Device 3 in Figure 5). Both devices must support NFC and run Android 4.4 KitKat (API level 19) or later. Moreover, the Card Emulator device must support Android's host-based card emulation [2] so that the phone can launch the NFC payment service implemented by our app. The actual man-in-the-middle functionality runs on the POS Emulator device (although this choice is irrelevant) and the Card Emulator acts as the proxy for the relay channel.

Using our app, we successfully bypassed PIN entry for transactions with four different cards: two Mastercard credit cards and two Maestro debit cards. A video demonstration of the attack and other information can be found at [1].

The results of our experiments are summarized in Table 1. Some of these transactions were performed with the

Brand	Card	Amount (CHF)	Processed with the Visa kernel	Bypassed PIN
Visa	Visa Credit	200	NA	Yes
	Visa Debit	100	NA	Yes
	V Pay	100	NA	Yes
Mastercard	Maestro$^{(1)}$	400	Yes	Yes
	Maestro$^{(1)}$ on Google Pay	1	Yes	NA
	Maestro$^{(1)}$ on Apple Pay	1	Yes	NA
	Maestro$^{(2)}$	200	Yes	Yes
	Mastercard Debit$^{(3)(*)}$	10	Yes	NA
	Mastercard Debit$^{(3)}$ on Google Pay	1	Yes	NA
	Mastercard Debit$^{(3)}$ on Apple Pay	1	Yes	NA
	Mastercard Credit$^{(4)}$	100	Yes	Yes
	Mastercard Credit$^{(5)}$	100	Yes	Yes

Legend:

NA: not applicable (1) to (5): each of the five different physical cards we tested

$(*)$: card for which we unsuccessfully attempted our PIN bypass for a 100 CHF transaction but the terminal requested to insert the card to complete the transaction using the contact chip instead

Table 1: Summary of the transactions during our experiments. All of these transactions were authorized online and were subsequently debited from the cardholder's account and credited to the merchant's account. For some cards, we performed multiple transactions and we show here the one with the highest value.

Google/Apple Pay apps using non-Visa cards. Such transactions do not require PIN verification and thus no bypass is needed, yet they showcase unauthentic uses of the Visa kernel.

Critical here is that the transactions in Table 1 were *all* authorized online by the issuer. Moreover, this was without any adversarial intervention beyond the terminal-card interaction and despite the different views between the terminal and the issuer on the AID selected for the transaction. The EMV protocol does not unambiguously specify what transaction data is sent to the issuer for authorization. Clearly, since our attack is possible, the AID and any other kernel-identifying data is either not sent, or not checked by the issuer. We cannot however confirm that this is the case for all EMV implementations in terminals.

Our card brand mixup suggests that merchants (in particular, their terminals) accepting Visa cards can also be fooled into accepting other EMV card brands, like Mastercard, even if they would not normally accept them. This could result in violations of contracts, market regulations, sanctions, embargoes, and credit card fees. Note that our attack could even be done in collusion with the merchant to evade taxes or fees. Another scenario where criminals might exploit our card brand mixup attack is the following. They might perform a high-value transaction with their own Mastercard-branded card turned into a Visa and then request reimbursement, claiming a terminal malfunction or fraud based on the fact that they do not own a Visa card. To support their claim, on the purchase receipt both the 'Visa' label and the Visa AID will be printed, which looks suspicious under scrutiny.

Usability and Scope. Our attack requires minimal hardware to carry out, namely two NFC-enabled Android phones, which can be purchased for under 300 USD. This represents a one-time investment for the criminals, and might even be unnecessary when they can use their own phones. In addition, the use of this hardware is inconspicuous since only one phone need be visible during payment and it easily escapes detection by store clerks since our app's appearance is very similar to legitimate payment apps such as Google/Apple Pay.

For our attack to work, clearly the authorization request must reach the card issuer. For this, it is necessary that the merchant's acquirer routes the request to either:

- a payment network that matches the real card brand, regardless of what the terminal thinks the brand is, or
- a payment network that handles transactions with cards of different brands, including Mastercard and Visa.

It is likely that the SumUp acquirer employs the first approach. The second approach is enforced by legal means in

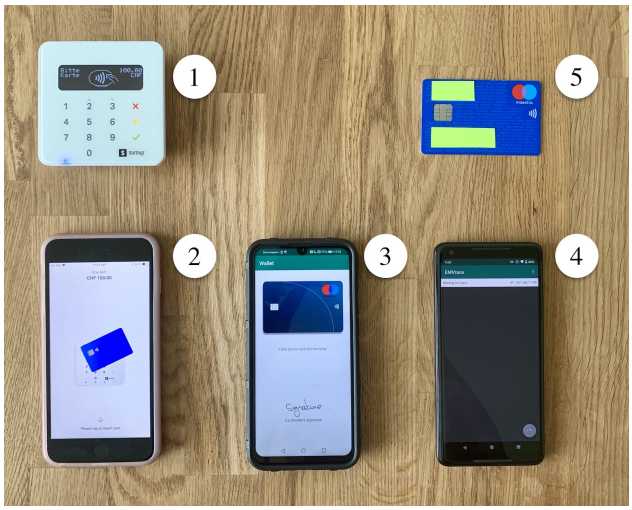

Figure 5: Setup of the testing environment for our proof-of-concept implementation, displaying the following devices: (1) SumUp Plus Card Reader, (2) mobile phone running the SumUp app and connected over Bluetooth to the SumUp reader, (3) Android phone running our app in Card Emulator mode, (4) Android phone running our app in POS Emulator mode, and (5) contactless card. Note that the device (2) is not part of the attacker's equipment since in an actual store this device and (1) would be the payment terminal. In this scenario, the devices (3) and (4) would be the attacker's equipment and (5) would be the victim's card.

some countries, making the scope of our card brand mixup attack very broad. For example, in the US, the 2010 federal law known as the *Durbin Amendment* [10] legislates that all domestic debit transactions must be given the choice, if so opted by the merchant, the cardholder, or the card (through the AIDs), to be routed to a common payment network, called the US Common Debit Network. This network forwards authorization requests to the card issuer, regardless of the card brand. Thus, if the victim's card is a Mastercard-branded debit card issued in the US and the merchant is also in the US, our attack should be effective by using the Visa US Common Debit AID `0xA0000000980840` instead of AID_{Visa} = `0xA0000000031010` during the application selection phase. This replacement would also deceive the terminal into running the flawed Visa kernel.

Other countries like Australia and New Zealand are also pushing for similar approaches for routing debit transactions to local payment networks as opposed to global ones. The Electronic Funds Transfer at Point of Sale (EFTPOS) system is an example of such an initiative in these countries.

Unsuccessful Attempts. We attempted to execute our attack to pay with a Mastercard card in a Discover and a UnionPay transaction, as these two kernels are similar to the Visa kernel. We did not succeed in either case. In these tests, we observed that the terminal did not pass the selection phase and requested us to insert the card or to try with another card. This suggests that the usage of cards of these brands over the contactless interface might be restricted in Switzerland, where we carried out our experiments.

We have performed additional tests on other payment terminals, including two by SIX.[2] From our disclosure process with Mastercard we learned that none of these transactions were routed to the Mastercard network, and so the SIX acquirer presumably routed the authorization requests to the Visa payment network, which flagged the card as non-Visa and declined the transaction.

Clearly the EMV standard should specify an unambiguous, *cryptographic* mechanism to detect and avoid mismatches between the AID and the PAN, in terms of the card brand they advertise. In the next section we analyze countermeasures that achieve this.

4 Countermeasures

This section discusses countermeasures to our card brand mixup attack. After reviewing our previous EMV model and our new extensions (Sections 4.1 and 4.2 respectively), we present both formally-verified countermeasures at the kernel level (Section 4.3) and countermeasures already implemented at the network level by Mastercard (Section 4.4).

4.1 Previous EMV Model

To design and verify kernel-level countermeasures to our attack, we extend our previous model [4] of the EMV contactless protocol. We developed this model focusing on the following three security properties:

1. The **issuer accepts** all transactions accepted by the terminal.
2. All accepted transactions are **authenticated to the terminal** by the card and, if authorized online, the issuer.
3. All accepted transactions are **authenticated to the issuer** by both the card and the terminal.

The first property expresses a causality of accept and decline events: whenever the terminal accepts a transaction, so will the issuer (or equivalently, the issuer will not decline it). For the authentication properties, we use *injective agreement* [25, 9]. In short, an agreement property validates that whenever the agent, whom the transaction must be authenticated to, reaches a state where the transaction is accepted, then that agent observes the same transaction details as the authenticating agent does. The transaction details to agree on for the properties are: the PAN, the AIP, the CVM, the

[2] https://www.six-group.com/

Parameter	Values	Comments
Brand	- Mastercard - Visa	Brand of the card used
Strongest ODA method supported by the card	- SDA - DDA - CDA	Mastercard cards only
Processing mode	- DDA - EMV	Visa cards only
Strongest CVM supported by the card	- No PIN - Online PIN	Mastercard cards only
Transaction value	- Low - High	Whether CVM is required

Table 2: Parameters that define target configurations.

ATC, the AC data input (i.e., X in Figure 3), the AC itself, and the IAD.

We specify a generic model of the EMV contactless protocol that allows for the analysis of transactions performed with the Visa and Mastercard kernels. The remaining four kernels can be modeled by one of these, which is their group representative as per the two groups introduced in Section 2.2.3.

Our analysis methodology, which we used in both our previous work and the current work, is structured by *target configurations*. A target configuration is a choice of up to four parameters (depending on the kernel) from Table 2. A target model is derived from the EMV contactless protocol model and allows any execution of the latter while only assessing the security of accepted transactions sharing the same target configuration.

The use of multiple configurations enables one to focus the security analysis on those transactions of interest, defined by the corresponding choice of target configurations. For example, one might be interested in whether authentication to the terminal holds for high-value transactions performed using the Mastercard kernel and cards supporting DDA as the Offline Data Authentication method and online PIN as the Cardholder Verification Method. Further details can be found at [4].

4.2 Extended Model with PAN-based Routing

Our previous model of the EMV contactless protocol specifies the terminal-issuer channel in a way that these two parties always agree on the kernel used for online-authorized transactions. In other words, we assumed that the transaction authorization request is routed to a payment network that only processes cards of the brand determined by the kernel used (or equivalently the AID chosen during the application selection phase). For example, if the transaction was processed with the Visa kernel, then the authorization request is routed to a network that handles Visa cards only.

This modeling assumption means that Mastercard cards can only be used for transactions performed using the Mastercard kernel. Clearly, our brand mixup attack demonstrates otherwise. That is, in some cases the authorization request reaches the card issuer, even when the card is not of the brand determined by the kernel used by the terminal. We have extended our previous model with a more general model of routing, where the terminal routes the authorization to the payment network determined by the card's PAN. The employed modeling techniques are standard ones, but we generalized the formalization of our previous model to consider this PAN-based routing choice.

In Table 3 we summarize the results of our analysis, conducted using our extended model. All target models have 56 Tamarin rules and about 800 lines of code on average. Remarks 1 and 2 in the table indicate authentication issues, which were first identified by the original model (see [6], Table 2, p. 11).

Remarks 3 and 4 indicate the newly discovered lack of authentication of the AID and the CVM used in the EMV contactless transaction. This is the underlying flaw that leads to our card brand mixup attack. For each of the affected target models, our Tamarin analysis reveals an accepted transaction where the following statements hold:

- the card used was a Mastercard,
- the terminal ran the transaction using the Visa kernel,
- no cardholder verification was performed, and
- if the transaction value was high, then the CDCVM was successfully performed from the terminal's perspective.

We remark that our current findings do not contradict those from our previous work. Our claim in [6] is that the Mastercard protocol is secure, whereas in this paper we show that Mastercard cards are not secure. In fact, as we have explained, our attack is possible precisely because one can use Mastercard cards for transactions not performed with the Mastercard protocol!

4.3 Verified Countermeasures

In [6], we proposed two fixes to the PIN bypass attack on Visa. These fixes are:

1. The terminal must always set the bit 1 of byte 1 of the Terminal Transaction Qualifiers (TTQ).
2. The terminal must always verify the Signed Dynamic Authentication Data (SDAD).

No.	Target model	Properties		
		issuer accepts	**auth. to terminal**	**auth. to issuer**
1	Visa_EMV_Low_PaynetPAN	✓	×(1)	×(1)
2	Visa_EMV_High_PaynetPAN	✓	×(1)	×(1)
3	Visa_DDA_Low_PaynetPAN	×(2)	×(2)	✓
4	**Visa_DDA_High_PaynetPAN**	✓	✓	✓
5	Mastercard_SDA_OnlinePIN_Low_PaynetPAN	×(2)	×(2)	×(3)
6	Mastercard_SDA_OnlinePIN_High_PaynetPAN	✓	✓	×(3,4)
7	Mastercard_SDA_NoPIN_Low_PaynetPAN	×(2)	×(2)	×(3)
8	Mastercard_SDA_NoPIN_High_PaynetPAN	–	–	–
9	Mastercard_DDA_OnlinePIN_Low_PaynetPAN	×(2)	×(2)	×(3)
10	Mastercard_DDA_OnlinePIN_High_PaynetPAN	✓	✓	×(3,4)
11	Mastercard_DDA_NoPIN_Low_PaynetPAN	×(2)	×(2)	×(3)
12	Mastercard_DDA_NoPIN_High_PaynetPAN	–	–	–
13	Mastercard_CDA_OnlinePIN_Low_PaynetPAN	✓	✓	×(3)
14	Mastercard_CDA_OnlinePIN_High_PaynetPAN	✓	✓	×(3,4)
15	Mastercard_CDA_NoPIN_Low_PaynetPAN	✓	✓	×(3)
16	Mastercard_CDA_NoPIN_High_PaynetPAN	–	–	–

Legend:

✓: property verified ×: property falsified –: not applicable

(1): disagrees with the card on the CVM (2): disagrees with the card on the AC

(3): disagrees with the terminal on the AID (4): disagrees with the card on the CVM

Table 3: Analysis results for the EMV contactless protocol where the authorization is routed to a payment network determined by the brand indicated by the PAN. Each target model is named according to the corresponding target configuration.

The above fixes ensure that high-value transactions processed with the Visa kernel use Visa's secure configuration (DDA on online authorizations), where the card is requested to supply the SDAD and the terminal verifies it. As can be observed in our results (Table 3, Line 4), we have verified that this configuration, and by extension the two fixes listed above, **prevents one from turning a Mastercard card into a Visa card**. The fixes work because of the kernel-specific format of the data that cards sign to produce the SDAD. Namely, the Visa protocol specifies that the input to the SDAD has the header `0x95` for online authorizations (see [17], p. 128), whereas the Mastercard kernel specifies the usage of the `0x05` header (see [16], p. 310 and [14], p. 73). In other words, no SDAD generated by a Mastercard card will pass the verification by a terminal running the Visa kernel for transactions requiring online authorization.

Additionally, we propose the following novel EMV-wide countermeasures that kernels can implement internally to guarantee secure online-authorized transactions, without having to rely on Visa-specific countermeasures.

1. All transactions must have the card generate the SDAD and the terminal verify it.
2. The selected AID must be part of the input to the SDAD.

Our first countermeasure generalizes the two fixes we proposed in [6], listed earlier in this section. The second countermeasure defends precisely against the card brand mixup attack that we have described in this paper. We have produced machine-checked security proofs for these countermeasures, using our extended model. This means that they effectively prevent the card brand mixup attack as well as both PIN bypass attacks. Note that the second countermeasure will be costly as it requires reissuing cards.

4.4 Countermeasures by Mastercard

We shared our countermeasures with Mastercard, as part of the disclosure process, and learned from them the following:

1. Mastercard acquirers are required to include the AID in the authorization data, allowing issuers to check the AID against the PAN.

2. Mastercard has other data points in the authorization request that can be used to identify our attack.

As a result of the disclosure process and once Mastercard learned that not all issuers check the AID or these other data points, they implemented these checks on their network. Our interaction with Mastercard also provided us additional insights on how certain terminals, such as the ones from SIX, can detect a mismatching AID and PAN and thus decline the transaction from the start.

With the mentioned checks in place, we again attempted our attack. This time it failed: the terminal requested the insertion of the card into the terminal and the entry of a PIN. Our experiments therefore provide evidence that these checks, deployed now by Mastercard, prevent our Mastercard-Visa mixup attack.

5 Related Work

In this section we review some of the related work on EMV (in)security, focusing on other practical attacks against the payment standard. As can be seen in our and others' work, the EMV contactless protocol is a prime target for hackers, given the ease of eavesdropping and modifying transaction data on the NFC channel. Widely available hardware such as mobile phones, Arduino boards, and Raspberry Pi boards can easily be used for these attacks.

Ten years ago, Murdoch *et al.* [27] reported the first PIN bypass attack against the EMV payment system.[3] The authors demonstrated that, for transactions where the card verifies the PIN entered on the terminal's PIN pad, a man-in-the-middle can simply reply with the "PIN verified" response to any PIN entered, right or wrong. The security flaw leading to this attack is the lack of authentication of the card's response to the terminal's PIN verification request, used in offline Cardholder Verification Methods (CVMs). Our prior research [6] showed that this flaw still exists in old cards that support neither asymmetric cryptography nor online PIN verification.

Ferradi *et al.* [18] described the forensic analysis of a series of credit card fraud events where criminals used 40 modified cards and carried out 7,000 fraudulent transactions, totaling about 600,000 Euros. The technical flaw that was presumably exploited by these criminals is that of [27].

Barisani *et al.* [5] presented a PIN harvest attack, also against EMV contact cards. Their attack works by downgrading the card's list of supported CVMs to a Plaintext PIN-only list. The authors showed that the protection against modification that the Offline Data Authentication (ODA) offers to the CVM list can be bypassed by setting the card-sourced Issuer Action Code (IAC)-Denial object to zero. This prevents the terminal from declining transactions with ODA failure. The terminal's selection of the CVM is determined by the card's list of supported CVMs. The authors found out that, even if this list is authenticated to the terminal during the offline authentication of the card, the list can be downgraded to a Plaintext PIN-only list. This is possible by setting the Issuer Action Code (IAC)-Denial data to zero, which prevents the terminal from declining the transaction.

EMV's specification v4.3 [15] (p. 115) recommends using a non-zero Terminal Action Code (TAC)-Denial object, which results in ODA-failing transactions being declined and thus prevents the PIN harvest of [5]. Indeed, during the (contactless) tests we performed using our app, all the transactions where we modified the IAC-Denial object were declined. We exposed a similar PIN harvest attack in [6].

Another PIN-related issue for EMV was observed by Emms *et al.* in [13]. The authors reported that some Visa contactless cards issued in the UK do not request PIN verification for non-GBP transactions. We note that this is unlikely to be exploited with modern cards and terminals. The reasons are two-fold: (1) the current Visa kernel establishes that if the terminal requires cardholder verification for a given transaction, then the card must offer at least one method to do so, and (2) Emms *et al.*'s observation seems to work only for transactions in EMV's magstripe mode, which is now deprecated.

Various relay and other NFC attacks have been presented in hacking conferences, such as [24, 31, 29, 21]. In particular, [29] presents a relay attack implementation that uses two Software Defined Radio (SDR) boards, which offer a faster and more controlled relay channel than the ones implemented using mobile phones over WiFi, according to the authors. However, the transmission speed of WiFi-based relay channels has not been an issue in any of our tests using our Android app.

Galloway and Yunusov [21] were the pioneers in bypassing PIN verification for modern Visa contactless cards. Their man-in-the-middle attack, implemented using wired Raspberry Pi boards, modifies both the Terminal Transaction Qualifiers (TTQ) before delivering it to the card and the Card Transaction Qualifiers (CTQ) before transmitting it back to the terminal. The authors did not however weaponize their attack in a way that it could be inconspicuously used in real stores.

Galloway recently showed [20] that it is possible, still in 2020, to clone a card and use the clone for swiped transactions. The author shows that the cloning can be made effortlessly, using the MSR605 magnetic card reader/writer, which costs around 100 USD. This research also shows that the data used to create the counterfeit magstripe cards can be read from the EMV interfaces (both NFC and contact chip)

[3] BBC News coverage at https://youtu.be/1pMuV2o4Lrw

with a skimmer device. The data needed is part of the Track 1 and Track 2 Equivalent Data objects, provided by the card during an EMV session. Back in 2008, Drimer *et al.* [12] also demonstrated cloning from EMV chip data to magstripe; thus this problem has remained unfixed even after 12 years.

As explained throughout this paper, our card brand mixup attack builds on our previous work [6]. In a nutshell, there are three main differences between our previous work and this new work. First, the card brand mixup attack is completely novel and exposes a serious weakness in EMV that permits payments with Mastercard cards for fraudulent Visa transactions. Second, we have extended our previous model of the issuer and of the terminal-issuer channel to support the completion of online transactions where the terminal and issuer do not observe the same card brand. We have used our extended model to verify our new fixes that prevent the card brand mixup. Finally, concerning the implementation of our attack, nearly 1,000 lines of Java code in our software instrument the NFC message modifications specific to the attack as well as the required cryptographic mechanisms such as the retrieval of PKs from PK certificates. Our PIN bypass attack on Visa does not require any of these mechanisms.

6 Conclusions

We have identified a serious, easily exploitable vulnerability in the EMV contactless protocol, namely the Application Identifiers (AIDs) are not authenticated to the payment terminal. The AIDs define what instance (a.k.a. kernel) of the protocol must be activated for the transaction. As a result, an adversary can maliciously replace the legitimate AIDs to deceive the terminal into activating a flawed kernel.

We have shown how to exploit this vulnerability using a man-in-the-middle attack that tricks the terminal into transacting with a Mastercard card, while believing it to be a Visa card. This card brand mixup, in combination with our recently developed PIN bypass attack [6] on Visa, results in a novel, critical attack where criminals can bypass the PIN for Mastercard cards. The cards of this brand were previously presumed protected by PIN. Shockingly, this is even possible for transactions that are authorized online in which the terminal and the card issuer do not agree on the payment card's brand.

To carry out our exploit, we developed a proof-of-concept Android application and successfully tested our attack on a real-world payment terminal. For example, we bypassed the PIN in a transaction for 400 CHF with a Maestro debit card. We have also extended our formal model of EMV by modeling the terminal-issuer channel in a way that allows for communication even when these agents disagree on the card brand. We used our extended model to formally verify that the ready-to-deploy fixes applicable to the Visa kernel that we proposed in [6] are an effective countermeasure to our Mastercard-Visa mixup attack. Additionally, we have specified and verified two new intra-kernel countermeasures that can be implemented on the Mastercard kernel without relying on Visa's defenses. Furthermore, Mastercard has implemented an alternative defense mechanism at the network level, which we have experimentally confirmed as effective against our attack.

Acronyms

AAC Application Authentication Cryptogram. 6

AC Application Cryptogram. 5, 6, 8, 11, 12

AFL Application File Locator. 3, 8

AID Application Identifier. 1–3, 6, 8–14

AIP Application Interchange Profile. 3, 5, 6, 8, 10

ARC Authorization Response Code. 6

ARQC Authorization Request Cryptogram. 6, 8

ATC Application Transaction Counter. 5, 6, 8, 11

AUC Application Usage Control. 8

CA Certificate Authority. 3, 4, 8

CDA Combined Dynamic Data Authentication. 4, 5, 8, 11

CDCVM Consumer Device CVM. 5, 8, 11

CDOL Card Risk Management Data Object List. 3, 8

CID Cryptogram Information Data. 5, 6, 8

CTQ Card Transaction Qualifiers. 3, 5, 8, 13

CVM Cardholder Verification Method. 3, 5, 10–13

CVMR Cardholder Verification Method Results. 8

DDA Dynamic Data Authentication. 5, 11, 12

DDOL Dynamic Data Object List. 5

IAC Issuer Action Code. 13

IAD Issuer Application Data. 6, 8, 11

MAC Message Authentication Code. 1, 4, 5

NFC Near Field Communication. 2, 3, 6, 8, 13, 14

ODA Offline Data Authentication. 3–6, 11, 13

PAN Primary Account Number. 1–3, 5, 6, 8, 10–13

PDOL Processing Data Object List. 3, 5, 8

PK Public Key. 3, 6, 8, 14

RID Registered Application Provider Identifier. 1

SDA Static Data Authentication. 5, 11

SDAD Signed Dynamic Authentication Data. 5, 6, 8, 11, 12

SDR Software Defined Radio. 13

SSAD Signed Static Authentication Data. 5

TAC Terminal Action Code. 13

TC Transaction Cryptogram. 6

TT Terminal Type. 8

TTQ Terminal Transaction Qualifiers. 5, 8, 11, 13

UN Unpredictable Number. 5

References

[1] The EMV Standard: Break, Fix, Verify. https://emvrace.github.io/. Accessed: February 2021.

[2] Host-based card emulation overview. https://developer.android.com/guide/topics/connectivity/nfc/hce. Accessed: August 2020.

[3] A model of EMV with PAN-based routing. https://github.com/EMVrace/EMVerify-PAN-routing. Accessed: February 2021.

[4] A Tamarin model of EMV. https://github.com/EMVrace/EMVerify. Accessed: February 2021.

[5] Andrea Barisani, Daniele Bianco, Adam Laurie, and Zac Franken. Chip & PIN is definitely broken: Credit Card skimming and PIN harvesting in an EMV world. In *Defcon*, volume 19, 2011.

[6] David A. Basin, Ralf Sasse, and Jorge Toro-Pozo. The EMV standard: Break, Fix, Verify. In *42nd IEEE Symposium on Security and Privacy (S&P 2021)*, 2021.

[7] Thomas Bocek, Christian Killer, Christos Tsiaras, and Burkhard Stiller. An NFC relay attack with off-the-shelf hardware and software. In Rémi Badonnel, Robert Koch, Aiko Pras, Martin Drasar, and Burkhard Stiller, editors, *Management and Security in the Age of Hyperconnectivity - 10th IFIP WG 6.6 International Conference on Autonomous Infrastructure, Management, and Security, AIMS 2016, Munich, Germany, June 20-23, 2016, Proceedings*, volume 9701 of *Lecture Notes in Computer Science*, pages 71–83. Springer, 2016.

[8] Tom Chothia, Flavio D. Garcia, Joeri de Ruiter, Jordi van den Breekel, and Matthew Thompson. Relay cost bounding for contactless EMV payments. In *Financial Cryptography and Data Security - 19th International Conference, FC 2015, San Juan, Puerto Rico, January 26-30, 2015, Revised Selected Papers*, pages 189–206, 2015.

[9] Cas Cremers and Sjouke Mauw. *Operational Semantics and Verification of Security Protocols*. Information Security and Cryptography. Springer, 2012.

[10] Chris Dodd and Barney Frank. Dodd-Frank Wall Street Reform and Consumer Protection Act. https://www.govinfo.gov/app/details/PLAW-111publ203, July 2010.

[11] Saar Drimer and Steven J. Murdoch. Keep your enemies close: Distance bounding against smartcard relay attacks. In *Proceedings of the 16th USENIX Security Symposium, Boston, MA, USA, August 6-10, 2007*, 2007.

[12] Saar Drimer, Steven J. Murdoch, and Ross J. Anderson. Thinking inside the box: System-level failures of tamper proofing. In *2008 IEEE Symposium on Security and Privacy (S&P 2008), 18-21 May 2008, Oakland, California, USA*, pages 281–295. IEEE Computer Society, 2008.

[13] Martin Emms, Budi Arief, Leo Freitas, Joseph Hannon, and Aad P. A. van Moorsel. Harvesting high value foreign currency transactions from EMV contactless credit cards without the PIN. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014*, pages 716–726, 2014.

[14] EMVCo. EMV Integrated Circuit Card Specifications for Payment Systems, Book 2, Security and Key Management, Version 4.3. https://www.emvco.com/wp-content/uploads/documents/EMV_v4.3_Book_2_Security_and_Key_Management_20120607061923900.pdf, November 2011.

[15] EMVCo. EMV Integrated Circuit Card Specifications for Payment Systems, Book 3, Application Specification, Version 4.3. https://www.emvco.com/wp-content/uploads/documents/EMV_v4.

3_Book_3_Application_Specification_20120607062110791.pdf, November 2011.

[16] EMVCo. EMV Contactless Specifications for Payment Systems, Book C-2, Kernel 2 Specification, Version 2.9. https://www.emvco.com/wp-content/uploads/documents/C-2-Kernel-2-V2.9-final_3.pdf, March 2020.

[17] EMVCo. EMV Contactless Specifications for Payment Systems, Book C-3, Kernel 3 Specification, Version 2.9. https://www.emvco.com/wp-content/uploads/documents/C-3-Kernel-3-v2-9.pdf, March 2020.

[18] Houda Ferradi, Rémi Géraud, David Naccache, and Assia Tria. When organized crime applies academic results: a forensic analysis of an in-card listening device. *J. Cryptographic Engineering*, 6(1):49–59, 2016.

[19] Lishoy Francis, Gerhard P. Hancke, Keith Mayes, and Konstantinos Markantonakis. Practical relay attack on contactless transactions by using NFC mobile phones. *IACR Cryptology ePrint Archive*, 2011:618, 2011.

[20] Leigh-Anne Galloway. It only takes a minute to clone a credit card, thanks to a 50-year-old problem. Link, 2020.

[21] Leigh-Anne Galloway and Tim Yunusov. First contact: New vulnerabilities in contactless payments. In *Black Hat Europe 2019*, 2019.

[22] Mastercard Inc. Annual Report 2019. https://s25.q4cdn.com/479285134/files/doc_financials/2019/ar/2019-Annual-Report-on-Form-10-K.pdf, 2020.

[23] Visa Inc. Annual Report 2019. https://s24.q4cdn.com/307498497/files/doc_downloads/Visa-Inc.-Fiscal-2019-Annual-Report.pdf, 2020.

[24] Eddie Lee. NFC hacking: The easy way. In *Defcon*, volume 20, pages 63–74, 2012.

[25] Gavin Lowe. A hierarchy of authentication specification. In *10th Computer Security Foundations Workshop (CSFW '97), June 10-12, 1997, Rockport, Massachusetts, USA*, pages 31–44, 1997.

[26] Simon Meier, Benedikt Schmidt, Cas Cremers, and David A. Basin. The TAMARIN prover for the symbolic analysis of security protocols. In *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*, pages 696–701, 2013.

[27] Steven J. Murdoch, Saar Drimer, Ross J. Anderson, and Mike Bond. Chip and PIN is broken. In *31st IEEE Symposium on Security and Privacy, S&P 2010, 16-19 May 2010, Berleley/Oakland, California, USA*, pages 433–446, 2010.

[28] Benedikt Schmidt, Simon Meier, Cas J. F. Cremers, and David A. Basin. Automated analysis of Diffie-Hellman protocols and advanced security properties. In *25th IEEE Computer Security Foundations Symposium, CSF 2012, Cambridge, MA, USA, June 25-27, 2012*, pages 78–94, 2012.

[29] Haoqi Shan and Jian Yuan. Man in the NFC. In *Defcon*, volume 25, 2017.

[30] Luigi Sportiello and Andrea Ciardulli. Long distance relay attack. In *Radio Frequency Identification - Security and Privacy Issues 9th International Workshop, RFIDsec 2013, Graz, Austria, July 9-11, 2013, Revised Selected Papers*, pages 69–85, 2013.

[31] Jordi van den Breekel. Relaying EMV contactless transactions using off-the-shelf Android devices. In *BlackHat Asia, Singapore*, 2015.