USENIX

THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

Zombie Cards Back Online: Reviving Expired Credit Cards for Contactless Payments

Raja Hasnain Anwar, Gerard DeCunha, and
Muhammad Taqi Raza, *University of Massachusetts Amherst*

https://www.usenix.org/conference/usenixsecurity26/presentation/anwar

This paper is included in the Proceedings of the
35th USENIX Security Symposium.

August 12–14, 2026 • Baltimore, MD, USA

ISBN 978-1-939133-58-8

Open access to the Proceedings of the
35th USENIX Security Symposium
is sponsored by

جامعة الملك عبدالله
للعلوم والتقنية
King Abdullah University of
Science and Technology

Zombie Cards Back Online: Reviving Expired Credit Cards for Contactless Payments

Raja Hasnain Anwar
University of Massachusetts Amherst

Gerard DeCunha
University of Massachusetts Amherst

Muhammad Taqi Raza
University of Massachusetts Amherst

Abstract

Contactless payment cards are widely assumed to stop working past their printed expiration dates, and many stakeholders rely on this assumption for authorization and access control. This paper shows that banks enforce the expiration as a transaction policy check, rather than an intrinsic property of the card, which allows an expired card to still initiate contactless payments. We demonstrate a practical "Zombie Card" attack that makes an expired card appear unexpired, allowing successful transactions despite the card being past its printed date. We evaluate the attack across real-world transaction configurations spanning multiple EMV kernels (Visa, Mastercard, and Discover), POS terminals, merchants, and five (5) major US banks. Our results show that Visa contactless transactions are susceptible to man-in-the-middle tampering due to a lack of effective integrity protection. We further find that banks often rely on the POS terminal's decisions and skip critical security checks during transaction authorization for faster payments. Across our trials, the attack remains operational under typical in-store conditions using commodity NFC transceivers and does not require specialized hardware. Together, these findings indicate that the outcome of a "zombie card" transaction is determined by how security responsibility is divided between terminals, card manufacturers, and issuers, and by how consistently issuers enforce card lifecycle state. Based on these findings, we propose countermeasures that span kernels, issuers, and payments to ensure end-to-end transaction integrity and security.

1 Introduction

Digital wallets (e.g., ApplePay and GPay) are increasingly used for in-store payments [39,51], and their global user base is expected to exceed 5.2 billion in 2026 [36]. However, this digital shift remains tied to card-based infrastructure, where physical cards remain central to consumer payments and account authorization [41]. The U.S. Federal Reserve reports that payment cards accounted for more than 60% of monthly payments in 2023, with credit cards accounting for 32% [29]. As of Q1 2024, 543.1 million credit cards were in circulation in the United States alone [33], generating $8.486 trillion in purchase volume [42]. In-store digital wallet penetration in the U.S. stood at only 28% in 2024 [38], meaning the majority of card-present transactions still originate from a physical card. Given that physical card security weaknesses propagate into the digital wallets [7], the security of the physical cards is central to the security of the broader payments ecosystem.

Prior work on payment cards has shown that security failures often arise not from broken cryptography, but from mismatches between user expectations, terminal-side checks, issuer-side authorization, and payment-network policy [7, 10–12, 44, 45]. This paper takes the same approach to examine one specific and underexplored assumption: that a payment card becomes inoperable once its printed expiration date has passed [15, 28]. Cardholders, merchants, and issuers broadly treat expiration as an intrinsic card property. However, as we show, it is enforced as a distributed policy with no mandatory end-to-end integrity binding. Hence, under the attack we present, an expired card that retains cryptographic capability can be revived for contactless transactions.

This work is motivated by documented patterns of improper expired-card handling. Although issuers instruct cardholders to destroy expired cards after replacement [5, 13, 15, 16], cardholders routinely underestimate this risk precisely because expired cards are assumed to be inactive [46,49]. This assumption is consistent with broader evidence that secure disposal practices are not universal: an AARP poll found that 34% of respondents shred sensitive paperwork only rarely, and 12% have never shredded such documents [3]. Similarly, a 2022 study attributes approximately 17% of identity theft cases to materials recovered from discarded waste [35]. Before examining how an "improperly discarded" expired (Zombie) card can be exploited for contactless transactions, we first describe the Europay, Mastercard, and Visa (EMVCo.) ecosystem and the protocol machinery that governs security and interoperability for card-present transactions worldwide.

A typical EMV® transaction involves several key partic-

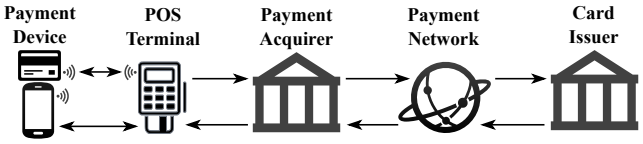


Figure 1: Overview of the EMV ecosystem where a Payment Device (physical card or digital wallet) and a POS terminal communicate over a direct NFC channel. Arrows represent the flow of transaction requests and responses.

ipants, each with a distinct role in the payment process, as shown in Figure 1. *Integrated Circuit Card* (ICC), or *Payment Device*, or simply "Card", is the consumer's payment device, commonly referred to as a smartcard or chip card. It can also be a mobile device emulating a payment card, i.e., a *digital wallet* [22]. The Card contains a microprocessor and memory that store both cardholder data and the payment application, including cryptographic keys used during transaction processing. *Point-of-sale (POS) Terminal* is the physical contactless interface that communicates with the card over NFC and connects to the payment network to process the transaction. *Acquirer* (bank) is the financial institution that processes card payments on behalf of the merchant and routes transaction requests through the appropriate payment network. *Payment Network*, such as Visa, Mastercard, or Discover, acts as the connection point between acquirers and issuers to route transaction messages. *Issuer* (bank) is the financial institution that provides the payment card to the cardholder and is ultimately responsible for authorizing or declining the transaction.

A central source of fragility is that the EMV contactless protocol is a selectively authenticated transaction flow. Many data objects are exchanged in plaintext between the card and terminal [2], and only a subset is later bound to cryptographic verification via Offline Data Authentication (ODA) and issuer-verified cryptograms. In consequence, an adversary who obtains a man-in-the-middle position on the NFC channel can tamper with decision-critical fields that are consumed by the terminal but are not end-to-end integrity protected. Prior NFC attacks [10–12, 44], including relay-based man-in-the-middle techniques, exploit this gap by manipulating transaction-relevant objects in transit while leaving cryptographic checks intact. The practical threat model is strengthened by the fact that the NFC channel can be accessed by commodity devices, and a malicious transaction can appear to the merchant as a normal contactless payment [12].

This paper focuses on a card lifecycle attribute that is widely assumed to be definitive: **card expiration**. Our key observation is that card expiration appears in multiple places and is consumed by different parties [23, 24]. In Visa Kernel 3 [24], the terminal evaluates processing restrictions using the Application Expiration Date, while the issuer extracts the expiry from Track 2 Equivalent Data for online authorization. Kernel 3 does not require these representations to be consistently bound, so the terminal's local expiration check is not an end-to-end invariant that the issuer can independently verify.

This motivates a question: if an expired card still retains valid cryptographic capability, can an attacker revive it by modifying only the terminal-consumed expiration input, without breaking ODA or issuer cryptogram verification? The Zombie Card attack answers this in the affirmative for Visa contactless transactions. The attack uses an NFC relay with a CardEmulator near the POS terminal and a POSEmulator near the target card, relaying APDUs end-to-end. The adversary intercepts the card's response carrying the Application Expiration Date and modifies it from a past date $D_{expired}$ to a future date D_{future} where $D_{future} > D_{transaction}$. Because Kernel 3 excludes the expiration date from the signature verification, this plaintext modification does not invalidate subsequent transaction processing.

We evaluate this end-to-end failure across real-world transactions spanning multiple EMV kernels (Visa, Mastercard, and Discover), POS terminals, merchants, and five major US banks. Our results show that Visa Kernel 3 [24] is particularly susceptible to man-in-the-middle tampering due to insufficient integrity protection for expiry-relevant data. We also observe that wallet CTQ settings steer "expired" transaction outcomes toward online authorization rather than a hard rejection, which increases reliance on issuer-side enforcement when the terminal suspects expiration. Critically, issuer behavior varies across banks, and our experiments show that some issuers rely on the POS terminal's decisions and skip critical security checks during transaction authorization, which allows the revived transaction to be authorized. Overall, these results show that secure protocol mechanisms do not guarantee secure card lifecycle enforcement when authority is fragmented and verification is inconsistent.

Contributions. This paper makes three notable contributions.

- **End-to-end analysis of expiration enforcement in EMV contactless.** We characterize how expiration is represented, consumed, and (not) bound to authenticated data across kernels, and why this makes expiration a policy outcome rather than an invariant.
- **Zombie Card attack.** We demonstrate a practical NFC man-in-the-middle attack that revives expired cards by rewriting the terminal-consumed Application Expiration Date during the contactless exchange, without breaking ODA or issuer-verifiable cryptograms.
- **Empirical validation and enforcement gaps.** We evaluate the attack across multiple EMV kernels, POS terminals, merchants, and five major US banks, and show that (i) Visa Kernel 3 provides a permissive surface for expiry manipulation and (ii) issuer-side authorization behavior is inconsistent, with some issuers effectively delegating expiration enforcement to terminal-side checks.

2 Background on EMV Contactless Payments

The EMV protocol is realized through a set of six kernels, each implementing a similar transaction flow while allowing design choices specific to different payment networks. A *kernel* defines how transaction data, card capabilities, and cardholder verification are processed and authenticated for a given payment network, such as Mastercard (Kernel 2 [23]), Visa (Kernel 3 [24]), or American Express (Kernel 4 [25]).

While the sequence of commands and responses is consistent to ensure interoperability across terminals and issuers, kernels differ in (i) how the terminal selects and parameterizes the transaction, (ii) which data objects are exchanged and when, and (iii) how authentication and risk-management decisions are expressed through protocol outputs. We primarily focus on the **Kernel 3 (Visa payWave, EMV mode)** [24], and reference other kernels only when necessary.

2.1 Physical Card vs. Digital Wallet

A contactless transaction begins with a payment device, i.e., a physical card or a digital wallet. A *physical card* stores the payment application and cryptographic keys in a secure element embedded in a plastic or metal card, whereas a *digital wallet* stores the same on the secure element of a mobile device. At the NFC interface, both participate in the same command–response exchange, leading the kernel to execute largely identical transaction flow for both form factors. In this paper, we use a generic term "Card" for any type of payment device participating in the EMV protocol.

PAN vs. Token. During a transaction, physical cards expose a *primary account number* (PAN), which is printed on the card, to the terminal. Digital wallets instead expose a digital replacement of PAN, called a *token* [30]. This token is provisioned and managed by the issuer's token service provider (TSP), which maintains the mapping to the underlying PAN [20]. To the terminal and acquirer, the token is indistinguishable from a PAN, so kernel execution is unchanged. Any behavioral differences typically arise from backend token lifecycle or issuer policy, not from kernel logic. Therefore, this paper uses "PAN" as a generic label for the account identifier (printed or tokenized) used in an EMV transaction.

Card Renewals. Since a wallet functions as a digital clone of a card, issuers can update (e.g., expiry date) and renew tokens over-the-air, often without the cardholder's involvement [21]. By contrast, when a physical card expires, the cardholder is required to request a physical card replacement.

2.2 APDUs and TLV Encoding

An EMV contactless transaction proceeds as a sequence of Application Protocol Data Unit (APDU) [2] command–response pairs between the terminal and the payment device. EMV application data is returned as Tag–Length–Value (TLV) objects within APDU responses. For example, the terminal can retrieve ICC application data using READ RECORD:

```
→  00 B2 01 0C 00
←  70 2A
     5A   08 4761739001010010
     5F24 03 240831
     5F34 01 01
     57   13 4761739001010010D24082211234567890F
   90 00
```

The card's response includes the Primary Account Number (PAN, tag #5A), the Application Expiration Date (tag #5F24), the PAN Sequence Number (tag #5F34), and Track 2 Equivalent Data (tag #57). Track 2 Equivalent Data encodes the PAN and an expiration date (as printed on the card) alongside additional track fields (e.g., service code), and it is commonly used by issuers as the carrier for account and expiry information in transaction authorization.

Observation 1: *Expiry Duality*. Expiry appears both as #5F24 (application date) and inside #57 (Track 2); they are consumed at different stages in the transaction flow.

Secrecy and Integrity. The APDU requests and data objects are typically transmitted in *cleartext* over NFC; integrity and authenticity are provided later in the transaction through offline data authentication and issuer cryptogram validation on certain critical data objects, rather than authenticating/protecting the entire APDU exchange.

2.3 EMV Transaction Flow

We now examine the stages of a contactless EMV transaction and dissect critical procedures and decision points that shape transaction execution and outcomes. Figure 2 summarizes the key message exchanges during the transaction flow.

2.3.1 Application Discovery & Selection

The transaction begins when the terminal polls and detects a card. The terminal first issues SELECT 2PAY.SYS.DDF01 ① to access the Proximity Payment System Environment (PPSE) on the card, which returns a list of supported applications ①', each identified by an Application Identifier (AID). The terminal then combines an AID returned by the card, and a Kernel ID supported by the terminal to form candidate combinations, and selects a combination according to its configuration and card-provided priority information.

The terminal issues SELECT AID for the chosen combination, e.g., Visa AID 00000011b ②. The card responds with Processing Options Data Object List (PDOL), a list of data tags that the card requires from the terminal to begin transaction processing ②', such as the authorized amount, terminal country code, and the Terminal Transaction Qualifiers (TTQ). TTQ is a capability and preference bitmap used by the kernel

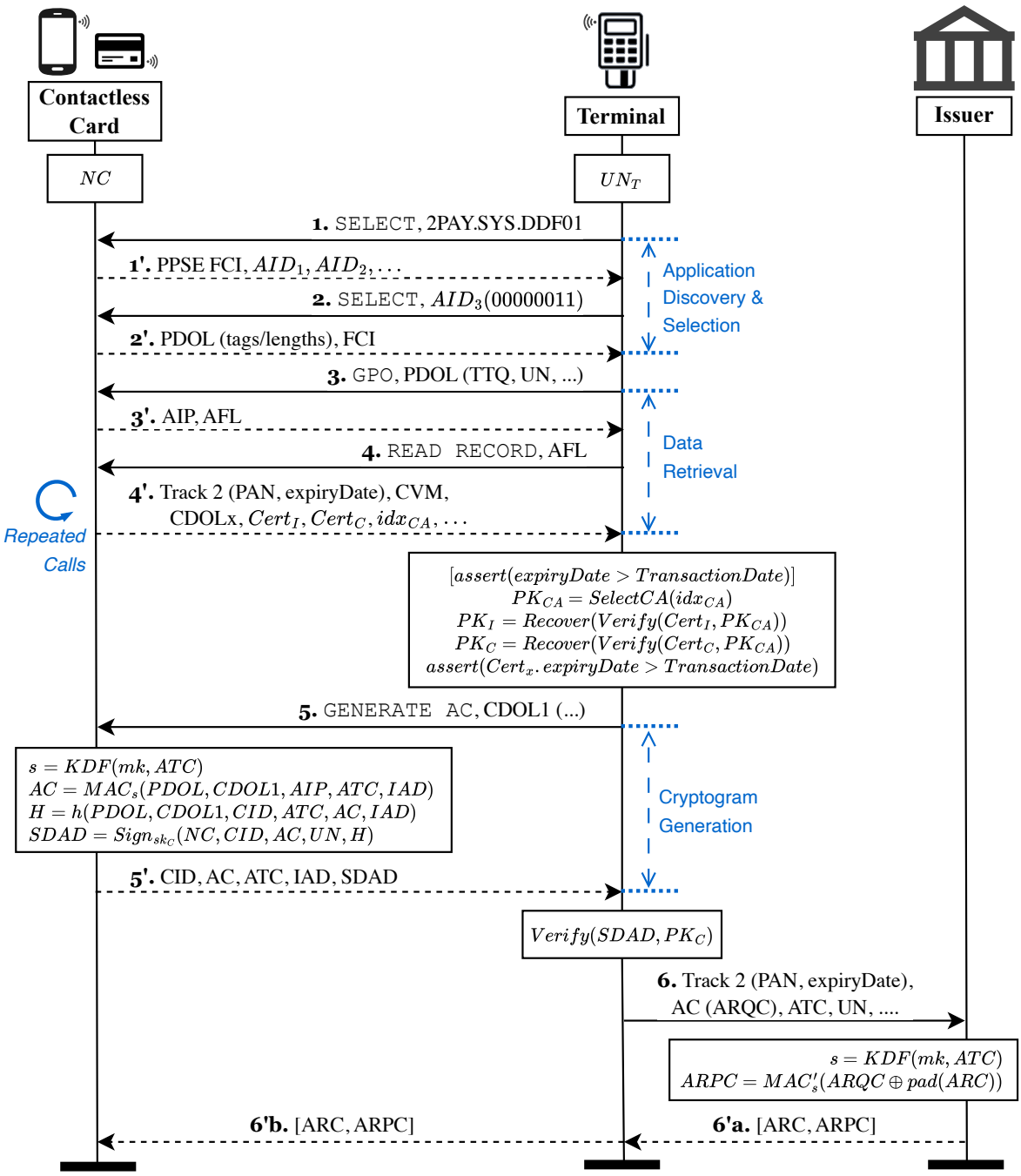


Figure 2: Overview of the EMV Kernel 3 (Visa) contactless transaction using fast Dynamic Data Authentication (fDDA) and online authorization. The terminal and Contactless Card (physical or digital wallet) exchange APDU request-response to exchange messages in TLV format and data objects as File Control Information (FCI). *Notation*: $\oplus$ denotes exclusive-OR; KDF is a key-derivation function; (sk_C, PK_C), (sk_I, PK_I), and (sk_{CA}, PK_{CA}) denote the private/public key pairs of the card, issuer, and certification authority; $\mathsf{Cert}_x(\cdot)$ denotes a PKI certificate signed for entity x; $\mathsf{Recover}(\cdot)$ denotes RSA certificate recovery and field extraction after signature verification; $\mathsf{Sign}_{sk_c}(m)$ denotes an RSA signature on message m; and $\mathsf{pad}(m)$ denotes right-padding of m with zero bytes. The application cryptogram (ARQC) is computed as $\mathsf{MAC}_s(\cdot)$ under a session key derived from issuer secrets and the application transaction counter (ATC), and is sent in the online authorization request together with issuer-visible transaction data (e.g., Track 2 Equivalent Data). For clarity, intermediate entities are omitted; though, the terminal messages to the Issuer are actually sent via Acquirer through the Payment Network.

and card to coordinate procedures such as cardholder verification methods (CVMs). These data objects are returned within the File Control Information (FCI), a structured set of TLV-encoded control and application data payload.

2.3.2 Application Initialization & Data Retrieval

After application selection, the Kernel processing begins with GET PROCESSING OPTIONS (GPO). The terminal supplies the requested PDOL data in the GPO command ③, and the card responds with parameters that govern the remainder of the transaction ③'. In particular, the response includes the Application Interchange Profile (AIP), which indicates supported security functions and offline authentication capabilities, and the Application File Locator (AFL), which identifies the records the terminal must read.

Guided by AFL, the terminal issues one or more READ RECORD ④ commands to retrieve data needed for risk management, cardholder verification (CVM), and authorization. These records include Track 2 Equivalent Data (#57), the Application Expiration Date (#5F24), and other control parameters used to decide whether the transaction is approved offline, declined, or sent online for authorization ④'.

2.3.3 Processing Restrictions

During transaction processing, the kernel applies a set of processing restrictions. One such restriction is the *Application Expired Check*, which is specified as implementation-conditional for readers that support *offline* transactions and may be handled **differently across kernels**. This check compares the Terminal Transaction Date with the Application Expiration Date, a 3-byte numeric value encoded in YYMMDD format when provided by the card.

In Kernel 3, the Application Expiration Date is returned by the card when Offline Data Authentication (ODA) is performed and may be omitted otherwise. If the application is determined to be expired, the terminal sets the "Expired application" bit (Byte 2, bit 7) in the Terminal Verification Results (TVR). The TVR is a 5-byte status register in which the terminal records the outcomes of all security and validation checks it executes during transaction processing. However, Kernel 3 specifies that the TVR forwarded to the issuer shall be set to all zeros (§ B.1.1 in [24]), effectively preventing the issuer from observing the terminal's local validation results.

Observation 2: *Non-cryptographic Expiry Check*. In *Kernel 3*, card expiry check is a terminal-side transaction date comparison; not a cryptographic proof from the card.

The terminal also evaluates the Card Transaction Qualifiers (CTQ) to guide subsequent processing. For example, the card may request that the reader proceed online if the application is expired, by setting Byte 1, Bit 4 ("Go online if application expired") to 1b [24].

Observation 3: *Online Expiration Check*. Through CTQ, the card can request online authorization when the terminal detects an expired application.

The results of processing restrictions are recorded as terminal indicators and propagated through the transaction flow. These indicators are consumed by subsequent procedures to decide the transaction processing path and outcome.

2.3.4 Offline Data Authentication (ODA)

Offline Data Authentication establishes card authenticity using a public key infrastructure (PKI) ④ & ④'. The terminal selects a Certification Authority (CA) public key (PK_{CA}) using the CA Public Key Index (idx_{CA}) provided by the card, and verifies the Issuer Public Key Certificate ($Cert_I$) to recover the Issuer public key (PK_I). It then verifies the ICC Public Key Certificate ($Cert_C$) to recover the ICC public key (PK_C). In both cases, certificate validity includes an expiration date check performed by the terminal against the transaction date.

It should be noted that the *Issuer* and *ICC* Certificate expiration dates are not the same as the Application Expiration Date. Excerpt from §10.2.5.3 in EMVCo. Book 2 [19]:

> *"– The expiry date of any issued IC Card shall be no later than the expiry date of the Issuer Public Key Certificate on that IC Card..."*

In other words, the certificate must remain valid for the entire lifetime of the card, but it is not required to expire with the card. Importantly, the cryptographic keys used in transactions, such as the card's private key (sk_C), do not encode any notion of expiry. Expiration is therefore an architectural property: temporal validity is enforced through certificates and terminal-side processing logic, not through the keys themselves.

Observation 4: *Card Expiry* $\leq$ *Certificate Expiry*. Validity periods of issuer and ICC certificates are independent of application expiry; the card's cryptographic keys can remain usable even after the application has expired.

The kernel specifies offline-capable terminals to support fast Dynamic Data Authentication (fDDA). In fDDA, the card generates Signed Dynamic Application Data ($SDAD = \text{Sign}_{sk_C}(\dots)$) over transaction-specific data, such as the terminal nonce (Unpredictable Number, UN_T), transaction amount, currency code, and more, which the kernel verifies using the recovered ICC public key (PK_C) [19]. Kernel rules determine the data included in SDAD, as well as whether and when it is performed along the transaction path.

2.3.5 Cryptogram Generation & Outcome Selection

The transaction culminates in an Application Cryptogram (AC) computed by the card ⑤. The terminal issues

GENERATE AC with a payload assembled per the card's Card Risk Management Data Object List 1 (CDOL1). CDOL1 specifies the ordered list of terminal-supplied data, typically the same transaction fields authenticated in SDAD. The card uses these elements to produce the AC as a Message Authentication Code ($AC = MAC(\dots)$). The card can generate three AC types indicated by the Cryptogram Information Data (CID): *Transaction Certificate* (TC) indicates offline approval by the card, *Application Authentication Cryptogram* (AAC) indicates an offline decline decision, and *Authorisation Request Cryptogram* (ARQC) requests online authorization from the issuer and is included in the authorization request.

The card returns (i) CID, (ii) AC (TC/AAC/ARQC) itself, (iii) Application Transaction Counter (ATC), and (iv) issuer-defined fields such as Issuer Application Data (IAD) (5'). The AC is computed with issuer-held symmetric keys, typically using a per-transaction session key derived from a master key and ATC ($s = KDF(mk, ATC)$).

2.3.6 Online Authorization

For transactions that proceed online, the terminal constructs an authorization request and forwards it to the issuer (6). Kernel 3 requires that Track 2 Equivalent Data be included in the request, from which the primary account number and the application expiration date are derived (**Observation 1: *Expiry Duality***). The request also typically includes the ARQC, and additional terminal and card data used by the issuer for cryptogram verification and risk assessment.

Upon receiving the authorization request, the issuer evaluates the transaction by verifying the account status. For digital wallets, this includes de-tokenizing the token via the token service provider (TSP) to recover the underlying PAN. The issuer then applies its risk assessment based on the PAN and transaction context to determine the authorization outcome. This backend processing is outside the EMV protocol and may vary across issuers per their policies.

If the issuer approves the transaction, it returns an authorization response (6'a) that includes an Authorization Response Code (ARC), a two-byte decision code indicating approval or decline, and an Authorization Response Cryptogram (ARPC), a MAC computed by the issuer under the same per-transaction session key to authenticate the response to the card (6'b).

3 Experimental Setup

In this section, we present the framework used to execute and evaluate the "Zombie Card" attack. We first establish a threat model centered on an active man-in-the-middle (MitM) adversary with the capability to intercept and modify NFC traffic between a card and a terminal. Subsequently, we detail our implementation of an end-to-end NFC relay and analysis methodology, which enables real-time study and manipulation of EMV protocol data.

3.1 Threat Model

We consider an **active MitM adversary** on the NFC channel between a card and a POS terminal. The adversary can observe, block, replay, modify, and inject APDU commands and TLV-encoded data. We assume the backend channel (acquirer, network, issuer) is authenticated and provides confidentiality and integrity for issuer-facing messages from the terminal.

The adversary does not possess **cryptographic secrets** and cannot compute valid application cryptograms (e.g., ARQC), and does not know cardholder verification secrets (e.g., PIN). The adversary's influence is, therefore, limited to protocol fields that are not cryptographically authenticated at the time they are exchanged between the card and terminal.

We assume the **target card** to be expired and replaced with a card bearing the *same card number* (PAN). This reflects a common practice whereby issuers provide a new card shortly before expiration, often preserving the same PAN while updating only the expiration date [5, 15, 16, 27, 40]. Our attacker model is bounded by the availability of the expired card; e.g., because the cardholder stores it, discards it without destruction, loses it, or otherwise fails to follow issuer guidance for handling expired cards [5, 13, 15, 16].

The adversary has **physical proximity** to the target card. This includes scenarios in which an expired card has been discarded, lost, or stolen, as well as scenarios where the attacker temporarily brings a relay or interception device within NFC range of the card. Importantly, a MitM position is required to modify transaction data in transit, even when the attacker has physical possession of the card, since the expiration date cannot be altered on the card itself.

The **POS terminal** is treated as a black box, and we do not assume any compromise of its firmware, secure elements, or backend infrastructure, i.e., the terminal conforms to the EMV Contactless specifications.

This threat model reflects realistic adversarial capabilities demonstrated in prior work [10–12, 44, 45].

3.2 Testbed Implementation

We implement a standard Man-in-the-Middle (MitM) relay between a legitimate contactless card and a functioning POS terminal, similar to the relay structure used in prior EMV studies [10–12]. The relay (shown in Figure 3) comprises two NFC-capable Android phones that run a custom application in two roles: Card Emulator and POS Emulator.

The POSEmulator phone is placed near the target card. It powers the card over NFC and forwards the terminal APDUs to the card. It returns the card responses back to the CardEmulator phone, placed near the POS terminal, over WiFi. This architecture allows us to intercept and modify specific

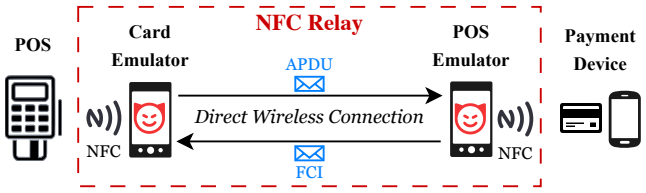


Figure 3: Architecture of NFC relay for contactless transactions. The Card Emulator captures APDU commands from the POS terminal and forwards them to the card via a POS Emulator. Card responses, i.e., FCI data objects, are relayed back to the terminal along the reverse path.

data objects and TLV fields in real-time before they reach the terminal. Figure 4 (ⓑ & ⓒ) shows the runtime views from both emulators illustrating the APDU commands and the relayed card responses.

Cards & Devices. Our testbed includes cards from major payment networks (Visa, Mastercard, Discover, and American Express) issued by various major U.S. banks. For some experiments, we also use Google Pay [32] and Apple Pay [8] digital wallets as payment devices. For experimentation, we implemented custom Android-based Card Emulator and POS Emulator running on OnePlus Nord 5 [43] devices. For real POS terminals that accept payments, we use SumUp Solo [48] and SumUp Plus [47] readers. Together, this setup provides a broad range of configurations for studying EMV transactions across different execution contexts.

Relay Overhead & Constraints. In our relay setup, each APDU round trip incurs an estimated additional latency of 20 (relay)—50 (relay+modify) ms, resulting in a per-transaction average latency of ~415 ms. The EMV contactless specification permits a response window of up to 500 ms per command before a timeout [22]. None of our experiments exceeds this threshold or cause a transaction timeout or failure, which is consistent with findings from prior studies [45, 50].

EMV® defines an optional Relay Resistance Protocol (RRP) [23], in which the terminal issues a timed challenge and the card must respond within a bounded window; a card or terminal pair that enforces RRP would detect the additional latency introduced by the relay and terminate the transaction. However, none of the physical cards or POS terminals included in our testbed implement RRP, which is consistent with the present reality: relay resistance remains optional and sparsely deployed [45].

3.3 Analysis Methodology

Our analysis is protocol-driven and centers on the NFC interface. Our relay records complete APDU command–response traces, including all transmitted TLV objects and parameters, and the transaction outcome. Using these traces, we identify terminal-facing data elements that influence transaction processing prior to, or independently of, cryptographic binding and that are not themselves cryptographically authenticated.

We interpret each observation by grounding it in the EMV specifications. For every manipulation, we trace how the modified fields are consumed by kernel procedures, how they affect processing restrictions and outcome selection, and which data elements are propagated into the online authorization request. Our primary reference is EMV Contactless Kernel 3 [24], together with EMV Security and Key Management (Book 2) [19], which we use to relate NFC-layer behavior to terminal decision logic. We apply the same analysis workflow to other contactless kernels, including Mastercard Kernel 2 [23], American Express Kernel 4 [25], and Discover Kernel 6 [26], to compare kernel-specific policies.

4 Prelim Study: Kernel Tolerance to Data Modifications

To study the kernel behavior, we conduct a preliminary study with the hypothesis that TLV fields critical to terminal decision-making are not integrity protected over the NFC channel. If so, a relay positioned between the POS and card could rewrite selected fields to manipulate transactions without triggering a kernel error or a transaction decline.

We evaluate this hypothesis using the relay setup in § 3.3. We execute $1.00 transactions with ***active*** *physical cards* and *mobile wallets* (Apple Pay [8] and Google Pay [32]) across multiple issuers and kernels. Since exhaustive testing across all issuers, cards, and configurations is costly, we adopt a two-stage evaluation strategy. For each kernel, we first perform a *screening* pass using a small set of modifications, namely the CDCVM flag and application expiry. If a kernel consistently rejects transactions under a given modification, we treat that modification as *blocked* and do not pursue further experiments for that kernel–edit pair. When a modification does not trigger an immediate integrity failure, we expand the evaluation to examine its effect on transaction processing. This is why our evaluation is most extensive for Kernel 3: it is the only one in which most modifications survived screening frequently enough to warrant deeper investigation. Table 1 summarizes the outcomes of these transactions under different settings.

4.1 CDCVM Signal Flip

Digital wallets typically rely on *Consumer Device Cardholder Verification Method* (CDCVM), where the user authenticates on-device (e.g., FaceID/TouchID). The wallet then conveys CDCVM result to the terminal via bit-level signaling, i.e., within CTQ/TTQ, depending on the kernel. Using our relay, we evaluate two modifications on contactless transactions: (i) forcing CDCVM→ 0 despite successful on-device authentication, and (ii) forcing CDCVM→ 1 to show success when no on-device authentication was performed.

Kernel 2 (Mastercard). Attempts to flip the CDCVM bit in Kernel 2 transactions consistently resulted in trans-

Table 1: Card-to-POS data field modifications for different issuers and EMV kernels. Each row corresponds to a distinct live transaction configuration tested at least three times across different times of day at $1.00, the minimum amount supported by our POS setup. Since Kernels 2, 4, and 6 failed on modifications; subsequent experiments focus on Kernel 3.

Issuer	Source	Kernel	CDCVM	Expiry	Pymnt.
Bank A	ICC	2	0	future	**Fail**
			0	past	**Fail**
		3	0	future	**Pass**
			0	past	**Fail**
			1 ↑	–	**Pass**
			1 ↑	future	**Pass**
			1 ↑	past	**Fail**
	Wallet	3	1	future	**Pass**
			1	past	**Pass**
			0 ↓	–	**Pass**
			0 ↓	future	**Pass**
			0 ↓	past	**Pass**
Bank B	ICC	3	0	future	**Fail**
			1 ↑	future	**Fail**
Bank C	ICC	2	0	future	**Fail**
			0	past	**Fail**
		3	0	future	**Pass**
			0	past	**Fail**
			1 ↑	–	**Pass**
			1 ↑	future	**Pass**
			1 ↑	past	**Fail**
	Wallet	3	1	future	**Pass**
			1	past	**Pass**
			0 ↓	–	**Pass**
			0 ↓	future	**Pass**
			0 ↓	past	**Pass**
Bank D	ICC	6	0	future	**Fail**
			1 ↑	future	**Fail**
Bank E	ICC	4	0	future	**Fail**
			1 ↑	future	**Fail**

CDCVM: no arrow = unmodified; ↑ = upgraded to CDCVM; ↓ = downgraded to classical CVM.
Pymnt: **Pass** = transaction succeeded; **Fail** = transaction failed.

action failure. Kernel 2 prevents CDCVM manipulation by cryptographically binding the AIP carrying CDCVM to both the Signed Dynamic Application Data (SDAD) and the Application Cryptogram (AC). Any modification to the CDCVM flag in the AIP invalids the MAC (for the issuer) and the SDAD (for the terminal), causing transaction failure. This behavior is consistent with prior research work on CVM bypass [12, 31, 45].

Kernels 3/4/6. For Kernels 3, 4, and 6, the CDCVM result is carried in plaintext data objects that our relay could modify without breaking terminal-side integrity checks. Concretely, Kernel 3 encodes CDCVM in CTQ [24], while Kernels 4 and 6 use Mobile CVM Results [25,26]; online transaction authorization where ODA is not enforced or does not cover these fields, flipping the CDCVM bit does not necessarily invalidate any signature at the terminal. However, the modification is frequently *non-decisive* because the EMV protocol falls back to alternative CVMs (PIN) rather than error out, and issuer-side authorization can still rely on cryptographically bound state (e.g., IAD within the AC) to accept or decline transactions irrespective of the terminal's local checks [24–26].

Finding 1: *In-flight Bit Flip*. In several EMV kernels, critical signal bits, like CDCVM, can be modified in flight without triggering an integrity error in the terminal.

4.2 Application Expiration Handling

We next test modification of the application expiration date, a field central to terminal-side processing restrictions. Using ***active*** *physical cards* in our relay setup, we alter the expiration date to both future and past values and observe how the kernel interprets the resulting state during transaction processing.

Kernel 2 (Mastercard). For Kernel 2 transactions, modifying only #5F24 expiry encoding while leaving the Track 2 encoding unchanged (recall **Observation 1: *Expiry Duality***) caused the transaction to fail. In practice, Track 2 Equivalent Data (#57) is forwarded in the online authorization request, and any modification to it is detected by the issuer and results in a decline. Kernel 2, therefore, requires the terminal to perform a consistency check during READ RECORD parsing and treats a mismatch between expiry representations as a card data error, triggering a decline rather than falling back to online authorization.

Finding 2: *Expiration Date Consistency*. When the kernel enforces consistency between 5F24 and Track 2 expiry, relay edits are trivial to detect and reject.

Kernel 4 (AmEx). In Kernel 4, the Application Expiration Date is a mandatory record element retrieved during READ RECORD processing, and Kernel 4's ODA scheme cryptographically binds this static record data as part of the data authenticated by the issuer's signature. As a result, any MitM modification of #5F24 induces a hash mismatch during signature validation, and the transaction terminates.

Kernel 6 (Discover). Instead of "fast" fDDA, Kernel 6 uses Combined Dynamic Data Authentication (CDA) (see Figure 8 in Appendix A), which is completed *after* the terminal reads records and performs terminal action analysis, it issues GENERATE AC, and the card returns SDAD. The terminal verifies SDAD by checking a recovered transaction data hash that binds the PDOL values and the card-returned TLVs. Any relay modification of a TLV (or of referenced record data needed to validate the certificate chain and recover PK_C) causes the locally computed hash to diverge from the signed value, leading to CDA failure and transaction decline.

Finding 3: *Expiration Date Validation*. Kernels 4 and 6 cryptographically bind expiry-relevant data, so relay edits fail signature verification.

Kernel 3 (Visa). Kernel 3 exhibits a split between expiry *logic* and integrity enforcement. When we modify #5F24 to a *past* date, the terminal's processing restrictions typically trigger a local decline. In contrast, modifying #5F24 to a *future* date allows the restriction check to pass. Interestingly, these modifications did not consistently trigger an integrity failure and instead exposed a failed downstream policy. Since these experiments showed positive signs, we investigate further.

For *Digital Wallets (Apple Pay / Google Pay)*, the expiration date change (either past or future) has no impact on the transaction outcome. This is because, in Kernel 3, a failed application-expired check is not a hard stop: the kernel consults CTQ and, specifically, Byte 1, Bit 4 ("Go online if application expired") to choose between offline decline and online processing. In our traces, we observe that the wallets consistently set this bit, so even when we modify #5F24, the reader deterministically selects the "online required" path and defers the decision to issuer authorization rather than local decline.

Finding 4: *CTQ in Expiration Check*. For Visa Kernel 3, wallet CTQ settings steer "expired" transaction outcomes toward online authorization, rather than a hard rejection.

Taken together, these observations indicate that **Kernel 3 offers the most permissive surface for relay-driven transaction manipulation, since several decision-critical TLVs remain mutable along the observed data path without triggering integrity and transaction failure.**

5 Using Zombie Cards for Contactless Transactions

Recall that card expiry appears in two distinct places (**Observation 1: *Expiry Duality***). In Kernel 3, the terminal evaluates processing restrictions using the Application Expiration Date, while the issuer extracts the expiry from Track 2 Equivalent Data carried in the online authorization request [24]. Kernel 3 does not require these two representations to be consistently bound end-to-end, so "card expiration" is not an invariant across the terminal-to-issuer path (**Finding 2: *Expiration Date Consistency***). Whether the issuer revalidates expiry at authorization is outside EMV scope and therefore varies across banks.

This distinction matters because the terminal's expiry decision is a local policy check rather than a cryptographic property of the card (**Observation 2: *Non-cryptographic Expiry Check***). Offline Data Authentication (ODA) authenticates PKI artifacts and their validity periods, but those lifetimes constrain certificates and key management, not application expiry semantics [19]. As a result, certificates and the underlying card keys can remain valid beyond the printed or application expiry date, meaning an *expired* card can still produce valid ODA evidence and issuer-verifiable cryptograms (AC) (**Observation 4: *Card Expiry* $\leq$ *Certificate Expiry***).

Taken together with our ability to bypass expiry checks (§4.2), we ask whether **a "zombie" expired card that still has valid cryptographic capability can also bypass the expiration checks to conduct successful transactions**.

5.1 Attack Description

The lack of end-to-end consistency for card expiration enables a practical bypass in Visa contactless transactions. An attacker can *revive* an expired card by modifying the expiration date that the terminal consumes for its local checks. This causes the terminal to continue processing the transaction even though the underlying card is expired. The attack proceeds as follows:

1. Relay Setup. The attacker positions a CardEmulator near the POS terminal and a POSEmulator near the target card to relay APDUs and data objects end-to-end so that the terminal executes the normal contactless transaction flow against what it believes is a physical card. Figure 4 shows our attack setup with two smartphones placed between the physical card and the terminal to create an NFC relay.

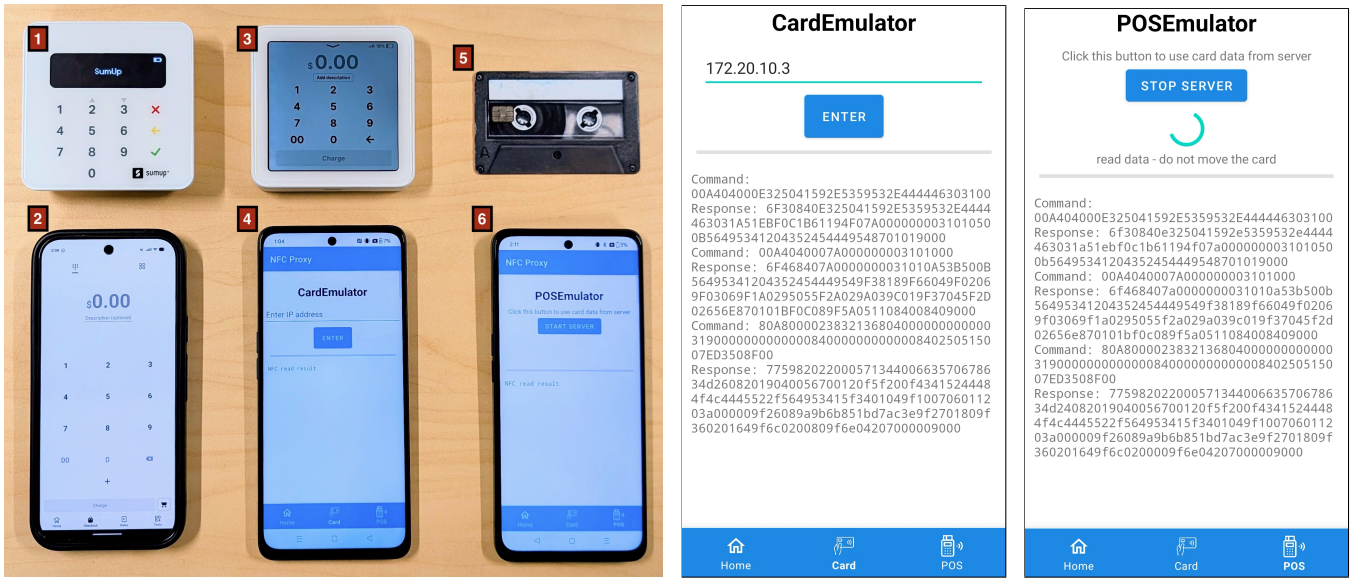
2. Payload Injection. The attack triggers during the Data Retrieval phase. The terminal issues READ RECORD commands based on the AFL to fetch data for risk management (refer to ④ & ④' in Figure 2). The card responds with unprotected and plaintext data objects.

- The relay *intercepts* the response containing the Application Expiration Date and Track 2 Data.
- The adversary *modifies* the #5F24 from the past date D_{expired} to a future date D_{future} (i.e., $D_{\text{future}} > D_{\text{transaction}}$).
- Under normal conditions, an expired card triggers the terminal to set the "Expired Application" bit (Byte 2, bit 7) in TVR. In Kernel 3, all bits of TVR are set to 0; the terminal checks are not communicated with the issuer.

3. Neutralizing Integrity Checks. The Kernel excludes #5F24 from the SDAD. However, the Track 2 (#57) is sent to the issuer, so the adversary ensures not to disturb it. The Application Expiry Date modification does not invalidate the subsequent fDDA signature verification, as the modified plaintext date is never bound to the card's RSA signature.

4. Certificate Validity. The terminal performs Offline Data Authentication (ODA) to verify the card's legitimacy.

- The terminal validates the Issuer and ICC Public Key Certificates ($Cert_I$ & $Cert_C$) using the CA Public Key and checks their expiration dates against the Transaction Date.
- As per **Observation 4: *Card Expiry* $\leq$ *Certificate Expiry***, banks frequently issue certificates extending beyond the card's printed expiry to accommodate reissue overlaps. Thus, the expired card successfully completes fDDA verification using valid certificates despite the application being logically retired.



(a) Experimental setup of the practical attacks (b) Card Emulator (c) POS Emulator

Figure 4: Experimental setup and emulator roles. **(a)** Physical NFC relay setup showing: (1) a SumUp Plus [47] terminal controlled via an external mobile app, (2) the companion app connected to the Plus terminal, and (3) a SumUp Solo [48] terminal operating independently, which together represent the non-attacker payment infrastructure. The attacker's equipment consists of (4) the card emulator phone that presents card data to either terminal (1) or (3), and (6) the POS emulator phone that reads APDU data from the victim card. Device (5) is the victim's physical card used in our experiments. **(b)** View from the card emulator (4), which interacts with the real POS terminal during transaction execution. **(c)** View from the POS emulator (6), which reads APDU data from the victim card and modifies and forwards it to the POS emulator. A demo video of the practical attack can be found at [1].

5. Online Authorization. The card uses its valid symmetric keys (derived from the Issuer Master Key *mk*) to generate a valid cryptogram (TC/AAC/ARQC).

- The authorization request payload forwarded to the issuer contains the valid ARQC, the active PAN, and the "zero" TVR (refer to ⑥ in Figure 2).
- The transaction succeeds against issuers who validate transactions at the *Account Level* (checking if the PAN exists and the account is open) rather than the *Instrument Level* (checking if the specific card instance, identified by the ATC or expiry date, is active).

Consequently, the issuer sees an active account and a transaction that *passed* all terminal checks. Trusting the terminal's (manipulated) pre-validation, the issuer authorizes the transaction on the new account context, unknowingly processing a payment from an expired card.

5.2 Practical Attacks using Expired Cards

We evaluate both controlled and real-world transactions to characterize how acceptance behavior changes under different configurations. All evaluated cards are **expired** and have been replaced by the issuer with a new card that retains the same PAN but has a different expiration date.

In the lab, we use SumUp Plus [47] and SumUp Solo [48] terminals registered under a *Professional Services* merchant category (marked as *Services* in Table 2) to obtain repeatable online authorization conditions across multiple amounts. We then validate that the same effects manifest outside the lab by conducting small purchases at "retail" and "grocery" merchants on campus.

For each issuer–kernel combination, we perform three runs. First, a *baseline* pass executes the transaction through the relay without any modification, verifying that the relay itself does not perturb acceptance. Second, the *in-lab modification* pass enables in-flight rewriting of the target data elements during the READ RECORD exchange on the SumUp terminals, recording both the POS and issuer outcomes. Third, the *in-the-wild validation* pass repeats the same modification at real-world retail and grocery merchants to confirm that the observed behavior is not specific to our SumUp testbed.

We record two **outcome signals**: (i) whether the POS/kernel proceeds past local checks and initiates an online authorization request ("Success: POS"), and (ii) whether the issuer ultimately authorizes the transaction ("Success: Issuer"). We interpret these jointly: a terminal-side bypass without issuer approval is a partial break that still provides evidence of an integrity gap, while issuer approval confirms

Table 2: *Summary of transactions using expired or replaced physical cards.* Green (g) labels (✓, ✗) indicate outcomes that match the expected behavior for the corresponding scenario. Red (r) labels (✓, ✗) indicate deviations from expected behavior. Blue (b) labels (✓, ✗) indicate outcomes that require further investigation and may depend on the payment scenario and issuer configuration.

Issuer	Kernel #	Card #	Expired	Altered	Merchant Type	Amount ($)	Transaction Success	
							Terminal	Issuer
Bank A	3	1	Yes	No	Services*	1.00	✗ (g)	✗ (g)
				Yes	Services*	1.00	✓ (r)	✓ (r)
						100.00	✓ (r)	✓ (r)
						500.00	✓ (r)	✓ (r)
					Retail	2.79	✓ (r)	✓ (r)
					Grocery	3.19	✓ (r)	✓ (r)
Bank B	3	2	Yes	No	Services*	1.00	✗ (g)	✗ (g)
				Yes	Services*	1.00	✓ (r)	✗ (b)
						100.00	✓ (r)	✗ (b)
						500.00	✓ (r)	✗ (b)
Bank D	6	4	No**	No	Services*	1.00	✓ (b)	✓ (b)
				Yes	Services*	1.00	✗ (g)	✗ (g)
		5	No	No	Services*	1.00	✓ (g)	✓ (g)
				Yes	Services*	1.00	✗ (g)	✗ (g)

**Services: Indicates transactions performed on our own POS terminal registered as 'Professional Services'.*
*(4) No**: Card not expired but automatically replaced by the bank because of less than 3 months of validity.*

an end-to-end policy bypass.

Table 2 summarizes end-to-end outcomes on POS terminals across multiple transaction amounts and merchant categories. In-the-wild (retail and grocery) purchases serve to verify external validity, confirming that the modifications observed under controlled lab conditions also succeed against the commercial POS deployed in real retail environments; they do not constitute an ecosystem-wide measurement study across merchants, terminal vendors, or issuer configurations.

5.3 Findings and Discussion

Across all tests, the dominant factors that determine attack success or failure are: (i) the EMV kernel in use and whether expiry data fields are cryptographically bound to authenticated protocol outputs; (ii) issuer-side lifecycle enforcement, especially whether authorization is tied only to the active account and PAN or also to the specific card instrument and expiration date; and (iii) whether terminal-side validation results are visible to the issuer via TVR. In contrast, transaction amount[1], merchant category, and POS terminal brand did not independently determine the transaction outcome. Our key findings are discussed below.

[1]Regional limits or restrictions on the card (e.g., lock), can affect whether additional cardholder verification is required for a successful transaction.

F1: Kernel 3 enables expired card revival. For an expired Visa card, the unmodified transaction is rejected as expected. After we apply our MitM modification to the expiry date that the terminal uses for its local check, the same expired card completes transactions across multiple amounts and merchant categories (**Bank A**). We observe this both in our controlled experiments (including higher-value purchases) and at real merchants for small purchases. The bypass is therefore not tied to a specific amount or a special merchant setup. Once the terminal accepts the (modified) expiry value, the transaction proceeds normally as an online authorization request.

This behavior reflects Kernel 3 design choices that prioritize flexibility and transaction continuation over end-to-end integrity. The attacker can modify the terminal-facing expiry (and, in the same attack surface, other decision inputs such as CDCVM indicators) without triggering an authentication failure at the terminal. A key compounding factor is observability: Kernel 3 forwards a TVR of all zeros, which removes a standard channel by which the issuer could learn that terminal-side validation or risk conditions were bypassed. Similarly, Kernel 3's preference for fDDA and its CTQ-driven "go online" fallback further shifts enforcement away from authenticated terminal decisions and toward backend heuristics.

These design choices reflect systemic tradeoffs that prior work [7, 34, 44] has identified: the split between terminal-

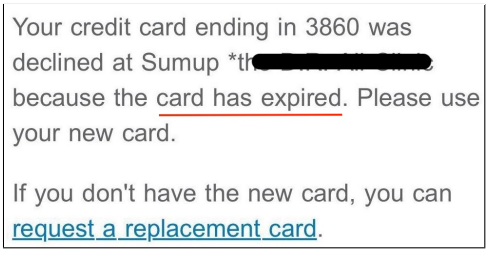


Figure 5: Email received by the Issuer bank after the transaction from an expired card reaches it.

side policy checks and flexible issuer-side authorization paths exists partly to preserve backward compatibility and to avoid false declines in a large ecosystem where strict end-to-end coupling would require coordinated changes across issuers, acquirers, and terminal vendors.

F2: Kernel design determines whether expiry values are integrity protected. Kernel design largely determines how security policy is realized in contactless EMV, and our results show substantial variability in baseline integrity enforcement even within the same EMV protocol family. Different kernels enforce materially different transaction-flow philosophies and priorities (latency, online fallback, or data consistency), which leads to inconsistent treatment of the same data fields.

In particular, Kernel 3 permits an expiry manipulation because the expiry value used for the terminal's local check is not cryptographically bound to the authenticated data that the terminal verifies. As a result, our MitM can modify it without triggering an integrity failure. In contrast, other kernels reject the same modification because expiry-relevant record data are included in authenticated inputs (e.g., signature verification or transaction hash bindings), making in-flight edits detectable.

For instance, Kernel 6 is more sensitive to modification: for **Bank D** on Kernel 6, transactions fail under in-flight edits even when the unmodified transaction succeeds. This suggests additional integrity or consistency checks in some kernel variants (Figure 8, Appendix A).

F3: Issuer outcomes diverge. The issuer authorization is outside the EMV scope. EMV specifies how the cryptogram is generated and verified, but it does not require issuers to enforce a uniform authorization mechanism. As a result, issuers implement their own policies and priorities. For instance, for Kernel 3, **Banks A and B** authorization diverges once the POS sends the transaction online. Bank A approves across our tested conditions, which indicates that its logic only verifies the application cryptogram and the active PAN. Bank B, in contrast, consistently declines transactions from an old expired card and asks to use the newly replaced card (Figure 5).

Kernel 3 further weakens the issuer's ability to act as a reliable fallback. Kernel 3 specifies that the TVR forwarded to the issuer is all zeros, so the issuer does not see the terminal's risk and validation outcomes. Here, the bank also prioritizes faster transaction settlement and does not re-evaluate all verifications that the terminal performed locally.

The key takeaway is that "distributed" expiry enforcement is brittle: unless the issuer treats expiry and card lifecycle as explicit authorization predicates, it cannot reliably serve as a second line of defense when terminal checks are bypassed.

F4: Card replacement semantics expose lifecycle ambiguity beyond EMV. Beyond expired cards, Table 2 also includes a card (**# 4**) that was *not yet expired* but was automatically replaced by the issuer shortly before its printed expiration date. A common expectation is that once a replacement is issued, the old card should stop working. In our experiments, both the old card (**# 4**) and the replacement card (**# 5**) continued to complete transactions to the same underlying account.

Even in cases where the kernel is rigorous about in-flight integrity (e.g., Bank D on Kernel 6 rejecting data manipulation, see Table 1), issuer card lifecycle enforcement can still be weak. The key point is that "replaced" is an issuer-managed state that may not be reflected in the on-card fields the kernel evaluates at the NFC interface, and EMV does not mandate uniform handling of this state across authorization paths.

This reinforces our core takeaway: expiry safety is primarily a lifecycle enforcement problem. The issuer should sit at the center of decision-making to oversee lifecycle management for its cards and tokens, and serve as a second line of defense when terminal-side checks fail or are bypassed. More broadly, issuers should not blindly delegate decisions to other entities; they should enforce and re-evaluate their policies explicitly during authorization.

F5: How digital wallets differ from "Zombie" cards?. Although this paper focuses on physical contactless cards, digital wallets provide a useful contrast. Digital wallets typically refresh token attributes and validity through issuer- and TSP-coordinated lifecycle updates, which reduces the chance that an expired underlying credential remains usable in the same way. This reinforces our core takeaway: expiry validation is primarily a lifecycle enforcement problem. Systems that rely on a central entity as a second line of defense, with explicit mechanisms to update and retire credentials, are less likely to inherit terminal-local policy gaps.

6 Proposed Countermeasures

We propose countermeasures that map directly to the issues identified in §5.3. We focus on changes that close the specific gaps our experiments exposed across kernels, terminals, and issuer authorization, as well as enhance the general security and reliability of the EMV protocol.

CM0: Preventing Attack Preconditions. The most direct mitigation is to eliminate the physical preconditions that make the attack feasible. First, cardholders should follow issuer guidance for expired-card disposal by cutting through the EMV chip and magnetic stripe, or by returning the card via an approved channel [5, 13, 15, 16]. Second, issuers and terminal

vendors should deploy the EMV® Relay Resistance Protocol (RRP) [23], which bounds the permissible round-trip time for card responses during a contactless transaction.

CM1: Integrity protection for expiry-critical APDUs and TLVs. Kernels and terminals should identify critical data fields and ensure their integrity protection over NFC (addresses *F1* & *F2*). Concretely, the data elements that drive terminal validity decisions, including Application Expiration Date, and CVMs, should be cryptographically bound to an issuer-verifiable signature, authenticated using ODA, or sourced from a kernel-defined transaction hash binding that fails under in-flight modification.

CM2: Expiry-representation consistency at the earliest verifiable point. Where a kernel exposes multiple expiry representations, terminals should perform an explicit consistency check between the expiry representation consumed by the terminal and carried in issuer-facing data, and produce issuer-visible evidence when they diverge (addresses *F1* & *F2*). This recommendation directly targets the Kernel 3 gap in which the terminal accepts a modified application expiry while the issuer may authorize based on different fields. Enforcing this check early also reduces kernel-to-kernel disparity by making consistency an invariant.

CM3: Expiry and replacement as hard authorization predicates. For online authorizations, issuers should treat the presented expiry as part of the credential identity and authorize against a (PAN, expiration date) tuple, rather than PAN alone (addresses *F3* & *F4*). Issuers should decline transactions when the presented expiry does not match the issuer's currently valid credential for that PAN and lifecycle state. This closes the "PAN continuity" behavior in which an obsolete card remains usable solely because the PAN remains active on the account.

CM4: Encode "replaced" as a revocation state across all authorization paths. When an issuer issues a replacement card, the prior card should transition to a state that causes a consistent decline on subsequent authorizations (addresses *F4*). This revocation semantics should apply uniformly across merchant categories and amounts, including low-value paths that are otherwise optimized for acceptance, to avoid creating a preferential channel in which replaced credentials remain usable.

CM5: Preserve terminal validation signals to the issuer. Kernels and payment networks should ensure that terminal risk and validation outcomes are meaningfully propagated to the issuer (addresses *F1* & *F3*). In particular, forwarding an all-zero TVR removes a standard mechanism for issuer-side compensating controls and impedes issuer-side monitoring. A minimal corrective requirement is that expiry-related terminal outcomes be reflected in issuer-visible data, either by preserving the actual TVR or by introducing an equivalent issuer-visible indicator with unambiguous semantics.

CM6: Fail closed on authenticated-data inconsistencies, rather than falling back online. When a kernel or terminal detects an inconsistency in authenticated static data that affects validity decisions, it should terminate the transaction locally rather than continuing via online fallback (addresses *F2*). This constrains the extent to which "go online" can serve as a universal recovery path for integrity failures that are, in effect, validity violations, and it aligns terminal behavior with the security intent of offline authentication checks.

7 Related Work

We study expiry enforcement in EMV contactless as an end-to-end property spanning the card, kernel, terminal, acquirer/network routing, and issuer risk engines. Prior work has analyzed EMV protocol properties, relay and terminal-logic attacks, and kernel-specific integrity gaps. Our work complements these threads by treating expiry as an end-to-end invariant with adversarially controllable inputs, rather than a purely local validity check.

EMV® Protocol Security and Formal Analysis. A foundational line of work models EMV's protocol suite and its security properties, emphasizing how gaps between specifications, implementations, and ecosystem behaviors lead to exploitable inconsistencies [11]. Systematizations organize adversary models, protocol properties, and recurring failure patterns across kernels and deployments [11, 34]. Formal analyses also illustrate that EMV security depends on which fields are authenticated end-to-end versus treated as local policy, with early work by de Ruiter and Poll demonstrating how formal reasoning exposes subtle authentication and agreement failures [17]. Empirical and systems-oriented work similarly highlights that security outcomes depend on how terminals, acquirers, networks, and issuers interpret protocol artifacts and risk signals, not only on the underlying cryptographic primitives [11, 34]. Beyond core EMV, work has shown that mismatches between specifications (including ISO-facing interpretations) and deployments can compromise integrity [4], and that independently introduced features can interact in ways that break assumptions and create new vulnerabilities [44].

Contactless Relay and Kernel Attacks. A second major thread targets the contactless device–terminal channel, including relay and on-path manipulation. Practical relay attacks using commodity devices demonstrate that NFC proximity assumptions can be violated at low cost [50], motivating relay-resistance mechanisms and evaluations of their practicality [14, 45]. Within contactless kernels, multiple attacks arise from mismatches between what is authenticated and what terminals accept as policy inputs. Emms et al. showed that acceptance logic and CVM handling can be bypassed in high-value foreign-currency scenarios [18]. Basin et al. identified cross-network misuse enabled by insufficient binding between payment network and issuer validation [10], and later showed that inducing specific authentication failures can push terminals into insecure fallbacks that remain accepted

under common configurations [12]. Collectively, these results underscore the security impact of terminal action codes, kernel configuration choices, and partially authenticated data objects [34]. We build on this insight by modeling the device–terminal exchange as a policy-carrying channel and focusing on how expiry-related fields can be modified in-flight without breaking the protocol.

Wallets and Tokenization. Digital wallets and tokenization shift the security focus to delegated authentication, authorization, and lifecycle control. Prior work demonstrates attacks that exploit inconsistent enforcement across devices, cloud services, and payment backends [9], and shows that multi-party state machines can contain vulnerable intermediate states that are hard to validate end-to-end [37]. Related work on wallet ecosystems shows that lifecycle and policy signals can be inconsistently enforced across wallets, token service providers, and issuers when treated as distributed policy rather than end-to-end verified invariants [7]. Although tokenized settings differ from physical cards and are often proprietary, they reinforce the same lesson: lifecycle decisions are security-critical and must remain robust under partial trust.

Positioning of Our Work. Prior work typically analyzes either (i) cryptographic authentication and agreement properties of the EMV protocol, or (ii) specific kernel/terminal logic failures such as relay enabling conditions, CVM, or fallback behavior [10–12, 34, 44]. In contrast, *Zombie Card Attack* centers on an operational invariant: *expiry must be enforced consistently across the card–terminal exchange and issuer authorization flow*. We show empirically that expiry can degrade into a policy-only attribute whose enforcement depends on where it appears, whether it is integrity protected, and how kernels and terminals propagate it into authorization requests. This enables attacks that selectively rewrite expiry-carrying data objects to revive expired credentials for contactless payments, without cryptographic key compromise. Relative to relay and CDCVM/CVM manipulation, our focus is on bypassing card lifecycle enforcement via expiry-related integrity gaps and downstream interpretation. Relative to induced authentication failure attacks [10, 12], our attacks do not require brand-routing or direct authentication failures; we exploit the fact that expiry is not uniformly bound into end-to-end authenticated data across kernels and issuers.

8 Conclusion

This paper shows that card expiration in EMV contactless payments is not an intrinsic property of the card, but a transaction policy outcome that depends on how terminals and issuers interpret and validate transaction data fields. We present a practical NFC man-in-the-middle attack that revives expired physical cards by rewriting the terminal-consumed expiration date during the contactless exchange, without breaking standard cryptographic checks. Our evaluation across multiple EMV kernels, POS terminals, merchants, and major US issuers demonstrates that Visa Kernel 3 is particularly susceptible due to insufficient integrity protection for expiry-relevant data, while issuer-side authorization behavior varies substantially across banks. These results highlight a systems-level lesson: secure protocol mechanisms do not guarantee secure lifecycle enforcement when responsibility is fragmented, and verification is inconsistent. Effective mitigations require treating expiration and replacement as issuer-validated lifecycle predicates, ensuring that expiry state is represented consistently across the transaction paths, and binding any terminal-enforced expiry inputs to authenticated data so that NFC-layer tampering is detectable.

Acknowledgments

We thank the USENIX Security reviewers and shepherd for their valuable feedback to improve this paper. We extend thanks to Dr. Ahsen Ali for assisting us with merchant POS registration.

Ethical Considerations

We recognize that certain tests and attack evaluations may pose risks to financial institutions and merchants. To conduct our study responsibly, we adopt the following safeguards. First, all attack scenarios are executed using our own credit cards and devices, with the researchers acting as both attacker and victim; no material risk to third parties was anticipated in this setting. For in-the-wild validation, we informed the specific merchants about the nature and purpose of the experiments in advance, performed at most two transactions per merchant, and purchased goods at each location. Our use of standard EMV cards and commercial POS terminals falls within normal cardholder and merchant use and does not require special authorization under the applicable card network terms of service; we did not attempt to access, modify, or interfere with any backend system, and our modifications were limited to the NFC channel between our own card and the terminal during our own transactions. Second, we do not dispute, or report any transactions resulting from our tests; all charges are accepted and paid in full, ensuring that no fraud is committed against issuers, acquirers, or merchants. Third, we do not release any tooling or source code used in our experiments to prevent misuse and limit the risk of replication by malicious actors.

Since this work did not involve human subjects, IRB approval was not required. The man-in-the-middle setup used in our experiments follows standard practice in prior EMV security research [10–12], and we did not receive objections from any affected stakeholder regarding the conduct of the research itself.

Disclosure. We disclosed our findings to VISA and the impacted banks in May 2025, and again reached out in Decem-

ber 2025. Our report included (a) a step-by-step reproduction guide, (b) full APDU traces, and (c) a video demonstration illustrating the authorization bypass. As per the last update, the VISA report has passed initial triage and is undergoing attack reproduction by their red team. We made further contact with all notified stakeholders during the rebuttal cycle. As of the time of acceptance, neither Visa nor the notified banks has provided any update on the status or nature of mitigations.

Open Science

Our study is grounded in publicly available EMV contactless specifications and in widely deployed payment networks. We evaluate real-world behavior using commodity payment infrastructure, including standard EMV cards (including expired cards), digital wallets, commercial POS terminals, and off-the-shelf smartphones. Our methodology and threat model follow established practice in prior empirical and formal analyses of EMV protocols and kernels, and we position our findings as part of the broader body of EMV security assessment research.

To support reproducibility, we aim to provide detailed experimental methodology, parameter settings, and sufficient protocol-level detail for independent validation. This includes (i) the list of tested devices and configurations, (ii) the transaction scenarios exercised (e.g., amount ranges, merchant categories, online vs. offline authorization paths when applicable), and (iii) sanitized examples of the relevant protocol messages and fields that underpin our analysis.

Release of Artifacts. We strongly support open-science practices and the open sharing of knowledge. To improve reproducibility, we release sanitized transactions logs for successful transactions at https://doi.org/10.5281/zenodo.20437876 [6], and we provide step-by-step protocol-level detail sufficient for independent validation in the body of the paper and its appendix.

Considering the sensitivity and impact of our implemented artefacts, i.e., NFC relay app, we have decided not to release the relay/MITM implementation used to induce the observed failures as part of an artifact evaluation. The code provides a direct, reusable capability to modify and relay NFC payment interactions and could materially lower the barrier to operational fraud. At the time of acceptance, we are awaiting relevant stakeholders' validation and remediation, and we do not consider the issues to be fully resolved across the ecosystem. Releasing exploit-grade tooling before mitigations are widely deployed would create avoidable risk to merchants, issuers, and cardholders, and would be inconsistent with responsible disclosure norms for payment systems. This decision is consistent with prior research that withheld exploit-capable artifacts while still disclosing full methodology, findings, and traces [11].

References

[1] EMV Expired Card Attack. https://khwarizmilab.github.io/emvexpiredcards/. Accessed: 2026-06-10.

[2] ISO/IEC 7816-4: Identification cards – Integrated circuit cards – Part 4: Organization, security and commands for interchange. https://www.iso.org/obp/ui/#iso:std:iso-iec:7816:-4:ed-4:v1:en, 2020. Accessed: 2025-03-08.

[3] AARP. Aarp's annual operation: Stop scams to feature document-shredding events and other activities in communities nationwide. https://www.aarp.org/press/releases/2018-4-16-aarp-annual-operation-stop-scams-feature-document-shredding-events-other-activities-communities-nationwide.html, April 2018. Accessed May 20, 2026.

[4] Nicholas Akinyokun and Vanessa Teague. Security and privacy implications of nfc-enabled contactless payment systems. In *Proceedings of the 12th international conference on Availability, reliability, and Security*, pages 1–10, 2017.

[5] American Express. What to know about credit card expiration dates. https://www.americanexpress.com/en-us/credit-cards/credit-intel/credit-card-expiration-date/, 2024. Accessed: 2025-12-30.

[6] Raja Hasnain Anwar, Gerard DeCunha, and Muhammad Taqi Raza. Zombie cards back online: Reviving expired credit cards for contactless payments, May 2026. doi:10.5281/zenodo.20437876.

[7] Raja Hasnain Anwar, Syed Rafiul Hussain, and Muhammad Taqi Raza. In wallet we trust: bypassing the digital wallets payment security for free shopping. In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 541–558, 2024.

[8] Apple Inc. Apple Pay. https://learn.applepay.apple/why-apple-pay, 2025. Accessed: 2026-01-05.

[9] Xiaolong Bai, Zhe Zhou, XiaoFeng Wang, Zhou Li, Xianghang Mi, Nan Zhang, Tongxin Li, Shi-Min Hu, and Kehuan Zhang. Picking up my tab: Understanding and mitigating synchronized token lifting and spending in mobile payment. In *USENIX Security Symposium*, pages 593–608, 2017.

[10] David Basin, Ralf Sasse, and Jorge Toro-Pozo. Card brand mixup attack: bypassing the {PIN} in {non-Visa} cards by using them for visa transactions. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 179–194, 2021.

[11] David Basin, Ralf Sasse, and Jorge Toro-Pozo. The emv standard: Break, fix, verify. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 1766–1781. IEEE, 2021.

[12] David Basin, Patrick Schaller, and Jorge Toro-Pozo. Inducing authentication failures to bypass credit card {PINs}. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 3065–3079, 2023.

[13] Chase. What to do with expired or old credit cards, 2025. Accessed: 2025-12-30. URL: https://www.chase.com/personal/credit-cards/education/basics/what-to-do-with-old-credit-cards.

[14] Tom Chothia, Flavio D Garcia, Joeri De Ruiter, Jordi Van Den Breekel, and Matthew Thompson. Relay cost bounding for contactless emv payments. In *International Conference on Financial Cryptography and Data Security*, pages 189–206. Springer, 2015.

[15] Citi. What to know about credit card expiration dates. https://www.citi.com/credit-cards/understanding-credit-cards/credit-card-expiration-date, 2025. Accessed: 2025-12-30.

[16] CreditOne Bank. What happens when your credit card expires? https://www.creditonebank.com/articles/why-credit-cards-have-expiration-dates, 2024. Accessed: 2025-12-30.

[17] Joeri De Ruiter and Erik Poll. Formal analysis of the emv protocol suite. In *Joint Workshop on Theory of Security and Applications*, pages 113–129. Springer, 2011.

[18] Martin Emms, Budi Arief, Leo Freitas, Joseph Hannon, and Aad 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*, pages 716–726, 2014.

[19] EMVCo. Integrated Circuit Card Specifications for Payment Systems – Book 2 – Security and Key Management. https://www.emvco.com/specifications/book-2-security-and-key-management/, November 2011.

[20] EMVCo. EMV payment tokenisation specification – technical framework. https://www.emvco.com/terms-of-use/?u=wp-content/uploads/documents/EMVCo-Payment-Tokenisation-Specification-Technical-Framework-v2.3.pdf, April 2020.

[21] EMVCo. EMV payment tokenisation – a guide to use cases. https://www.emvco.com/terms-of-use/?u=wp-content/uploads/documents/EMVCo-Payment-Tokenisation-A-Guide-To-Use-Cases-v2.1.pdf, August 2021.

[22] EMVCo. Contactless Specifications for Payment Systems – Book A – Architecture and General Requirements. https://www.emvco.com/specifications/book-a-architecture-and-general-requirements-11/, June 2023.

[23] EMVCo. Contactless Specifications for Payment Systems – Book C-2 – Kernel 2 Specification. https://www.emvco.com/specifications/book-c-2-kernel-specification/, June 2023.

[24] EMVCo. Contactless Specifications for Payment Systems – Book C-3 – Kernel 3 Specification. https://www.emvco.com/specifications/book-c-3-kernel-specification/, June 2023.

[25] EMVCo. Contactless Specifications for Payment Systems – Book C-4 – Kernel 4 Specification. https://www.emvco.com/specifications/book-c-4-kernel-specification/, June 2023.

[26] EMVCo. Contactless Specifications for Payment Systems – Book C-6 – Kernel 6 Specification. https://www.emvco.com/specifications/book-c-6-kernel-specification/, June 2023.

[27] Experian. What happens when your credit card expires? https://www.experian.com/blogs/ask-experian/what-happens-when-credit-card-expires/, 2021. Accessed: 2025-12-30.

[28] Experian. What happens when your credit card expires?, 2026. Accessed: 2026-04-30. URL: https://www.experian.com/blogs/ask-experian/what-happens-when-credit-card-expires/.

[29] Federal Reserve Financial Services. 2024 findings from the diary of consumer payment choice, 2024. Accessed May 20, 2026. URL: https://www.frbservices.org/news/research/2024-findings-from-the-diary-of-consumer-payment-choice.

[30] US Payments Forum. EMV payment tokenization primer and lessons learned. https://www.uspaymentsforum.org/wp-content/uploads/2019/06/EMV-Payment-Tokenization-Primer-Lessons-Learned-FINAL-June-2019.pdf, June 2019.

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

[32] Google. Google Pay. https://pay.google.com, 2025. Accessed: 2026-01-05.

[33] Helcim. U.S. credit card statistics and trends 2025. https://www.helcim.com/guides/credit-card-statistics-and-trends/, 2024. Accessed May 20, 2026.

[34] Xenia Hofmeier, David Basin, Ralf Sasse, and Jorge Toro-Pozo. Sok: Attacks on modern card payments. *arXiv preprint arXiv:2504.03363*, 2025.

[35] Javelin Strategy & Research. 2022 identity fraud study: The virtual battleground. https://javelinstrategy.com/2022-Identity-fraud-scams-report, 2022. Accessed May 20, 2026.

[36] Juniper Research. Digital wallet users to exceed 5.2 billion globally by 2026. Press release, August 2022. Accessed: 2026-01-06. URL: https://www.juniperresearch.com/press/digital-wallet-users-exceed-5bn-globally-2026/.

[37] Jiadong Lou, Xu Yuan, and Ning Zhang. Messy states of wiring: Vulnerabilities in emerging personal payment systems. In *30th USENIX Security Symposium*, pages 3273–3289, 2021.

[38] McKinsey & Company. State of consumer digital payments in 2024. https://www.mckinsey.com/industries/financial-services/our-insights/banking-matters/state-of-consumer-digital-payments-in-2024, 2024. Accessed May 20, 2026.

[39] Sila Money. The future is now: How digital wallets are transforming fintech. URL: https://www.silamoney.com/ach/the-future-is-now-how-digital-wallets-are-transforming-fintech.

[40] Nasdaq. Credit card expiring? here's what to expect. https://www.nasdaq.com/articles/credit-card-expiring-heres-what-to-expect, 2022. Accessed: 2025-12-30.

[41] FDIC Consumer News. A closer look at mobile banking: More uses, more users. *FDIC*. URL: https://www.fdic.gov/consumers/consumer/news/cnwin18/mobilebanking.html.

[42] Nilson Report. US card spending at merchants — 2023. https://nilsonreport.com/articles/us-card-spending-at-merchants-2023/, 2024. Accessed May 20, 2026.

[43] OnePlus. OnePlus Nord 5. https://www.oneplus.com/global/nord-5/specs, 2025. Accessed: 2026-01-05.

[44] George Pavlides, Anna Clee, Ioana Boureanu, and Tom Chothia. More is less: Extra features in contactless payments break security. In *34th USENIX Security Symposium*. USENIX Association, 2025.

[45] Andreea-Ina Radu, Tom Chothia, Christopher JP Newton, Ioana Boureanu, and Liqun Chen. Practical emv relay protection. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 1737–1756. IEEE, 2022.

[46] SumUp. What to do with old credit cards. https://www.sumup.com/en-us/business-guide/what-to-do-with-old-credit-cards/, 2024. Accessed: 2026-05-10.

[47] SumUp. Plus card reader. https://www.sumup.com/en-us/plus-card-reader/, 2025. Accessed: 2025-12-30.

[48] SumUp. Solo card reader. https://www.sumup.com/en-us/solo-card-reader/, 2025. Accessed: 2025-12-30.

[49] The Motley Fool. Here's what to do with old, expired credit cards. https://www.fool.com/money/credit-cards/articles/heres-what-to-do-with-old-expired-credit-cards, 2023. Accessed: 2026-05-10.

[50] Jordi van den Breekel and B Asia. Relaying EMV contactless transactions using off-the-shelf android devices. *BlackHat Asia, Singapore*, 2015.

[51] Fernando Zandona. Cash is no longer king: Why the rise of digital wallets will shift the payment landscape. *Financial IT*, February 2022. URL: https://financialit.net/blog/cash-no-longer-king-why-rise-digital-wallets-will-shift-payment-landscape.

A Supporting Figures

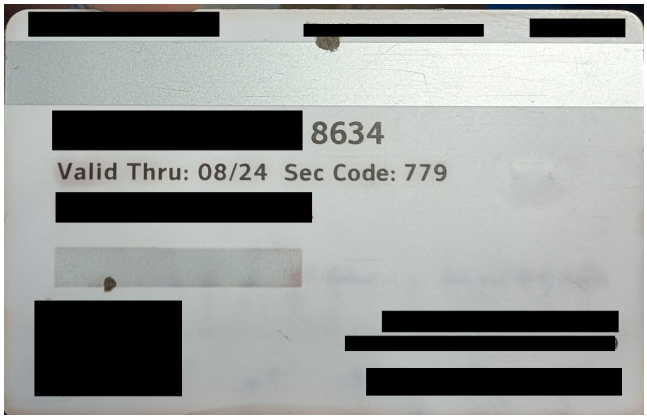


Figure 6: An expired card used in our experiments. The card is issued by Bank A through the Visa payment network and ends in 8634.

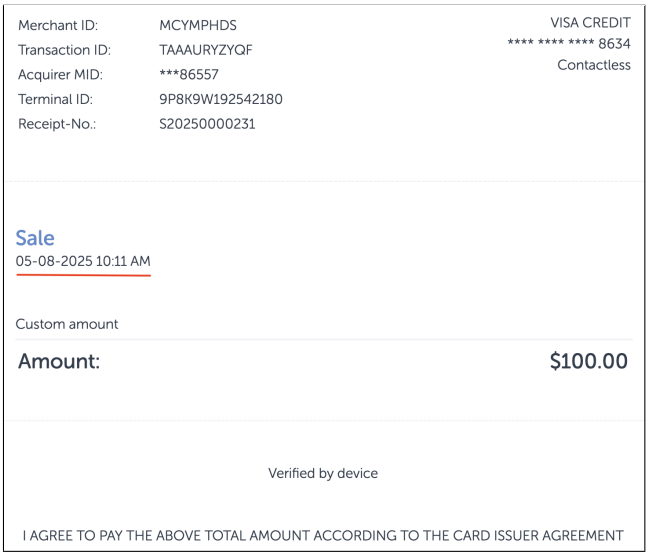


Figure 7: Receipt for the test transaction made on the expired card ending in 8634 (shown in Figure 5).

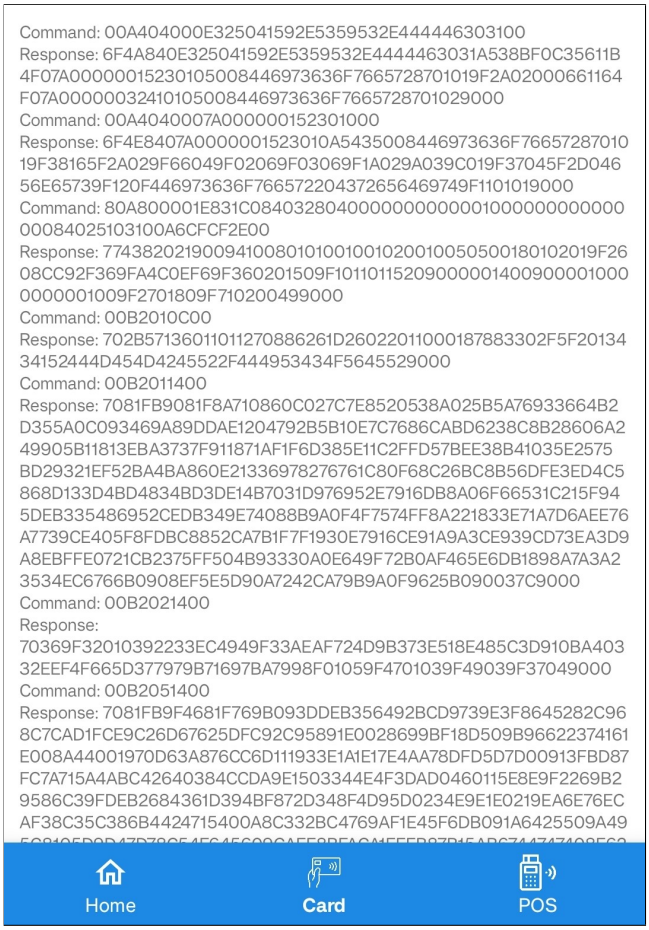


Figure 8: Conventional Combined Dynamic Data Authentication (CDA) flow observed in Discover (Kernel 6) transaction that declines the transaction when a data object, e.g., #5F24, is modified. This exchange differs significantly from the fast Dynamic Data Authentication (fDDA) shown in Figure 4.