

Credit Without Debit: Formal Verification of Banking Payment Systems

Ali Hamza Malik

University of Massachusetts Amherst
Amherst, Massachusetts, USA
ahmalik@umass.edu

Muhammad Taqi Raza

University of Massachusetts Amherst
Amherst, Massachusetts, USA
taqi@umass.edu

Abstract

Banking payment systems, handling trillions of dollars annually, are widely assumed to provide strong security and access-control guarantees as transfers result in permanent movement of funds and are governed by strict operational rules. In this paper, we present a formal model of Automated Clearing House (ACH) payment processing that captures the delayed and asynchronous nature of transfers, the decentralized control of consumer accounts, and the temporary states visible to users before settlement. Using a model-based testing framework, we analyze whether fundamental guarantees such as authorization hold throughout a payment transfer's lifecycle. Our analysis reveals vulnerabilities in institutional choices permitted by Nacha rules: treating authorization as a one-time check and exposing provisional credits before access control and settlement complete. We identify ACH security flaws that lead to a new class of attacks which we label **Credit-Without-Debit (CWD)** for which we find three instances in practice, and validate them empirically in live U.S. banks and third-party financial platforms. We disclosed our findings and proposed remedies to all affected institutions. Follow-up testing at those institutions confirms that these instances are no longer possible.

CCS Concepts

• **Security and privacy** → **Logic and verification**; **Security services**; • **Applied computing** → **Electronic funds transfer**; • **Computing methodologies** → **Model verification and validation**.

Keywords

Automated Clearing House (ACH), banking security, payment systems, formal verification, model checking, access control and authorization, financial fraud, protocol verification

ACM Reference Format:

Ali Hamza Malik and Muhammad Taqi Raza. 2026. Credit Without Debit: Formal Verification of Banking Payment Systems. In *Proceedings of the 2026 ACM SIGSAC Conference on Computer and Communications Security (CCS '26)*, November 15–19, 2026, The Hague, Netherlands. ACM, New York, NY, USA, 14 pages. <https://doi.org/10.1145/3830454.3846614>

1 Introduction

The Automated Clearing House (ACH) network is central to U.S. retail, commercial, and government payment flows [46]. It supports direct deposit of salaries, bill payments, mortgage servicing, credit-card bill payments, bank-linked funding and withdrawal for digital wallets and payment apps such as Zelle [60], PayPal [51], Venmo [57], and CashApp [11].

Unlike credit-card networks, which rely on issuer guarantees and chargeback mechanisms, banking transfers move funds permanently between accounts and financial institutions. These transfers are irreversible once settled and are processed through multi-stage interbank clearing and settlement. Banking systems are therefore designed around strict operational rules and deferred settlement workflows, and are generally regarded as providing stronger security guarantees than card payments.

Security weaknesses in credit-card payment systems are well documented [4, 30, 34, 40, 54]. Card networks tolerate a measurable level of fraud by shifting liability, rather than enforcing absolute misuse prevention. Banking payment systems are assumed to provide stronger guarantees for correctness and access control because bank transfers move funds irreversibly and afford minimal recourse once settlement occurs. In practice, participants in bank-transfer workflows rely on several operational properties: debits and credits are expected to pair for both counterparties, account balances should remain consistent throughout a transaction, incomplete or pending transfers should not influence available balance, and seemingly complete transfers should not be reversed unexpectedly.

While the ACH network and overarching regulations do not necessitate such behaviors, they are similarly devoid of standardized requirements for addressing them. These operational assumptions are neither routinely scrutinized nor systematically verified, leaving security analysis of bank-transfer systems sparse. We investigate this gap by asking: *Can a systematic, model-based analysis of the Nacha Operating Rules reveal exploitable vulnerabilities that can be realized in operational ACH payment procedures?*

We study U.S. banking payment procedures defined by the Nacha Operating Rules [42], executed via the ACH network [37], and applied by banks and payment providers. To capture these behaviors, we construct a formal model of consumer ACH transactions grounded in the Nacha Rules (Section 4). The model preserves timing gaps, cross-institution boundaries, and user-visible intermediate states, and captures cases admitted under underspecified Nacha options observed in practice. Our analysis is an instance of the parameterized system verification problem (i.e., parameterized by the number of protocol participants), which is undecidable in general [19]. Achieving both soundness and completeness is impossible. We follow the conventional method of aiming for soundness



This work is licensed under a Creative Commons Attribution 4.0 International License. *CCS '26, The Hague, Netherlands*

© 2026 Copyright held by the owner/author(s).

ACM ISBN 979-8-4007-2871-6/2026/11

<https://doi.org/10.1145/3830454.3846614>

Table 1: Summary of three Credit-Without-Debit (CWD) instances identified by our analysis. All three were responsibly disclosed and have been resolved by institutions.

	A1: creditBLEED Attack	A2: creditMINT Attack	A3: creditRESET Attack
<i>Design rationale</i>	Provide immediate fund availability	Simplify account linking	Shorten credit-line recovery wait
<i>Implicit assumption</i>	Routing/account format $\Rightarrow$ debit capability	Inbound credit $\Rightarrow$ outbound debit	Internal posting $\Rightarrow$ ACH settlement
<i>Exploit</i>	ODFI credit without RDFI debit	Repeated micro-credits without debit recovery	Repeated credit-line restoration
<i>Impact</i>	Access funds on authorization	Repeatedly accumulate micro-deposits	Spend on prematurely restored credit line

instead of completeness. If our approach reports a violation, it is indeed a violation. We cannot, however, detect all violations. We subject this model to exhaustive bounded analysis (Section 5). We verify whether the expected guarantees are preserved at every stage of execution, including intermediate states between initiation and eventual settlement. Through this analysis, we reveal that ACH workflows fail to enforce fundamental transactional properties (Atomicity, Consistency, Isolation, and Durability, or ACID [58]) across institutional boundaries. Because authorization, internal posting, and interbank settlement are decoupled, attackers can exploit temporary state divergence to execute a class of logic vulnerabilities we term **Credit-Without-Debit (CWD)** attacks. In a CWD attack, an adversary secures a spendable credit at one institution while forcing the balancing debit to fail at another.

Our model-based testing reveals three practical CWD attacks (Section 6). We validate them through black-box experiments that check whether the property violation arises in production workflows. As we cannot exhaustively test every institution, we target institutions that account for a large share of ACH activity. ACH payment volume is concentrated among a small number of large banks. Nacha maintains a list of the top 50 U.S. financial institutions [47]. In 2025, the top 50 processed 30.7 billion ACH payments. Twenty billion of those payments were concentrated at four major banks: Chase, Bank of America, Wells Fargo, and Citibank. Findings at these institutions bear on a large fraction of U.S. ACH activity, even though our experiments cover only a subset of them. Our evaluation employs a fixed sample of authorized consumer accounts across two of these four banks, three major credit-card issuers, and three leading payment providers. *At the request of these financial institutions, their specific identities and the associated accounts have been anonymized.* Leveraging these accounts, we demonstrate the three CWD attacks identified during our analysis through standard user-controlled actions, including ACH debit initiations, micro-deposit verifications, and credit-card bill payments. Table 1 summarizes the rationale, assumptions, exploits, and impacts for each attack.

The three attacks operate as follows:

A1: creditBLEED Attack. ACH debit processing uses asynchronous authorization, where the originating bank (ODFI) provisionally credits the originator before interbank settlement. This step implicitly assumes the ODFI will later successfully retrieve funds from the receiving bank (RDFI). We exploit this by initiating an ACH debit to an RDFI account under a debit restriction. The ODFI authorizes the transaction and posts a fully spendable credit internally. When the ODFI attempts settlement, the debit fails due to access-control restrictions. By repeating this process, an attacker repeatedly accumulates spendable funds at the origin without a

corresponding debit succeeding at the target, producing a cross-institution imbalance until the RDFI return reverses the credit.

A2: creditMINT Attack. Third-party payment providers use micro-deposits to link an external bank account, treating successful receipt as proof of ownership. This verification only checks inbound credit acceptance, assuming it guarantees outbound debit capability. We exploit this by linking an account that accepts ACH credits but restricts debits. Each verification cycle triggers micro-deposit credits that post successfully, while the provider’s later debit to reclaim the amounts fails. These providers allow the verification process to be restarted. By repeating this cycle, an attacker repeatedly accumulates micro-deposits in the controlled account without the provider successfully recovering the funds.

A3: creditRESET Attack. Credit-card bill payments decouple internal posting from ACH settlement. When a cardholder submits a payment, the issuer posts it internally and restores the available credit before the underlying ACH debit settles. This posting implicitly assumes the issuer will successfully retrieve funds from the payer’s bank. We exploit this by paying a credit bill from a debit-restricted bank account. The issuer posts the payment and restores the credit line, enabling immediate spending, while the subsequent ACH debit fails and is returned. An attacker can submit multiple payments and repeatedly spend against the prematurely restored credit line.

Contributions. We contribute the following:

- We introduce a formal model of ACH payment procedures grounded in the Nacha Operating Rules, interbank settlement workflows, and operational constraints observed in consumer U.S. banking infrastructure.
- We develop a property-guided verification framework to analyze ACH-style batch-settlement systems (Section 3). The framework identifies violations arising from asynchronous authorization, cross-institution state divergence, and settlement timing.
- We demonstrate that these formally derived violations correspond to exploitable behaviors within our sample of evaluated institutions by validating three CWD instances on real U.S. banks and third-party payment providers.

Responsible Disclosure. We submitted disclosures to all affected banks and third-party payment providers on April 16, 2025, through their official responsible disclosure portals, providing reproduction steps, proof-of-concept demonstrations, and technical reports. All institutions acknowledged receipt. **A1** and **A3** were no longer viable by May 2025. **A2** remained exploitable through one third-party provider through December 2025 and was fully resolved by January 2026.

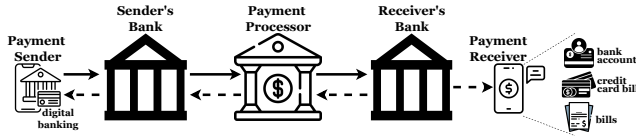


Figure 1: Overview of the ACH ecosystem where a sender initiates a payment to the receiver’s bank (RDFI). The sender’s bank (ODFI) forwards the request to an ACH Operator, which batches and routes requests between originating and receiving banks. Arrows show request and response flow.

2 Background

The ACH ecosystem (Figure 1) coordinates Originating Depository Financial Institutions (ODFIs), Receiving Depository Financial Institutions (RDFIs), Third-Party Senders (TPSs), and ACH Operators. ODFIs initiate transfers, RDFIs receive funds, Third-Party Senders offer payment services, and ACH Operators route batched transactions between banks. Nacha [44] governs the Automated Clearing House (ACH) [37] network and defines the Nacha Operating Rules for authentication, authorization, access control, and risk management. Banking payments settle in multiple stages over several days, unlike credit-card networks. The ODFI first authorizes the transaction, provisionally credits the originator, and then tries to pull the funds. The RDFI applies access control at that stage.

External Account Authentication. When a user links an external account, the bank verifies ownership through one of three methods. Manual-link requires the routing and account numbers. Web-login authenticates through the external bank’s portal, often through aggregators such as Plaid. Micro-deposits send two small amounts (\$0.01–\$0.99) to the external account, which the user confirms to prove ownership.

Transaction Authorization. After account verification, the user can initiate ACH credit (push) or debit (pull) transfers, bill payments, and recurring credit-card payments by selecting a verified account, amount, direction, and frequency. The ACH network classifies consumer entries using Standard Entry Class (SEC) codes. WEB covers internet- and mobile-initiated consumer entries. PPD covers prearranged consumer payments, including recurring bill payments. The ODFI validates the request, constructs an ACH entry with the routing number, account number, transaction code, and amount, and queues it in a batch for transmission to an ACH Operator. Batch processing separates initiation from settlement, which creates a temporal gap. Authorized transactions remain in flight until RDFI receipt.

Access Control. Upon receiving a transaction, the RDFI verifies account eligibility and transaction context. Credit entries proceed when the account is active and accepts deposits. Debit entries additionally require sufficient balance, valid authorization, and no fraud holds. The RDFI enforces authorization rules by SEC code and authorization type. Debits fail when the Receiver has revoked authorization before the RDFI posts the entry. Recurring authorizations remain valid until revoked. Single-entry authorizations do not cover later initiations after revocation. Failed transactions return to the ODFI, usually within two banking days, with reason codes such as insufficient funds, closed accounts, or unauthorized

debits. Some errors generate Notifications of Change (NOCs) that correct information without reversing funds. Provisional credit may be advanced before the RDFI completes access control. A pending transaction status therefore persists for one to three banking days. This timing gap between ODFI credit commitment and RDFI access-control enforcement is the structural feature our model captures.

3 Overview of ACH Model-Based Verification

We present a model-based framework (Figure 2) that analyzes ACH payment procedures for property violations. We extract formal models from the Nacha Operating Rules, extend them with adversarial capabilities, and apply model checking to detect security violations.

3.1 Threat Model

We consider an adversary with four capabilities. **C-1:** The adversary controls legitimate bank accounts at U.S. financial institutions, opened through standard procedures, and may assume the role of Originator, Receiver, or both. **C-2:** The adversary can obtain routing and account numbers for bank accounts from public sources (personal checks [18, 53, 56]) or data breaches [27], without online banking credentials [41]. **C-3:** The adversary can initiate or receive ACH transactions through consumer interfaces (banking portals, mobile apps, or third-party payment providers), controlling transaction parameters (payment type, SEC code, account type, and amount). **C-4:** The adversary can modify accounts under their own control through standard banking features such as holds, closures, and debit blocks [20].

We assume financial institutions faithfully apply the Nacha Operating Rules [42] and do not collude with adversaries. The ACH network reliably delivers messages within the specified timing windows. Victim account holders behave honestly and remain unaware of unauthorized transaction activity until they discover it later. Adversary-controlled accounts opened under **C-1** are assumed to pass know-your-customer (KYC) and account-opening checks. Anti-money-laundering (AML) monitoring, fraud engines, velocity controls, and investigator intervention are out of scope. The analysis targets authorization and settlement logic under transitions admitted by our Nacha-compatible reading.

3.2 Major Components of our Framework

Standards and Specification Analysis. We analyzed the Nacha Operating Rules [42, 43, 45], RDFI and ODFI documentation, SEC code specifications, and regulatory materials from banking authorities. The complete payment lifecycle spans multiple organizations (Figure 1), and rules for a single transaction appear across sections and appendices, which requires constant cross-referencing.

Scope Reduction. Building a complete formal model from the full Nacha Operating Rules is impractical: the Nacha Operating Rules [42] span 728 pages defining 25 SEC codes, 87 return codes, and 14 change notification codes, and modeling all elements yields an undecidable verification problem because of combinatorial transaction parameters and unbounded domains [19]. We therefore scope our model to *domestic, standard consumer ACH transactions* performed over the internet (e.g., *paying a credit-card bill*), which captures the dominant transaction volume and the primary consumer-fraud attack surface [34]. Section 4 refines this reduction to the

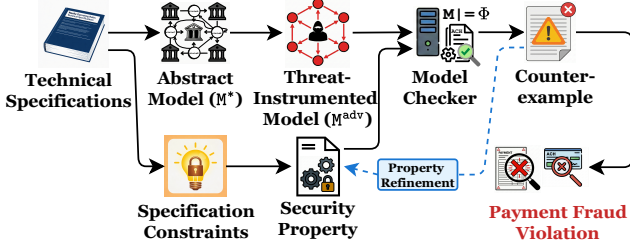


Figure 2: Overview of the ACH formal analysis framework. The process spans three phases: deriving formal models from Nacha rules, evaluating security properties via adversarial model checking, and empirically validating counterexamples in live banking environments.

WEB and PPD SEC codes. We model both direct ACH transfers and transactions started through third-party payment providers (e.g., PayPal, Venmo, CashApp), which under Nacha rules act as Third-Party Senders (TPSs) and use micro-deposit-based authentication for ownership checks.

Abstract Protocol Model. We model ACH payment procedures as a parameterized system of communicating FSMs. The system model M comprises five participant types: Originators O , an ODFI BO , the ACH Operator A , an RDFI BR , and Receivers R , with parameterized counts of Originators ($O^i, i \in [1, n]$) and Receivers ($R^j, j \in [1, m]$). We extend M with an adversarial machine embodying the threat capabilities from Section 3.1 to form M^{adv} , which is then checked against formal properties.

Model Checking Process ($M^{adv} \models \Phi$). We validate M^{adv} using a counterexample-driven loop. First, we check whether functional traces derived from the Nacha Operating Rules satisfy expected reachability properties (e.g., a submitted ‘entry’ reaches ‘settled’). When the model produces traces inconsistent with the specification, we update it and repeat verification until all traces align with the rules. The resulting model is both executable and specification-compliant (Section 5.2). Security requirements are encoded as safety invariants over the reachable states of M^{adv} . Model checking determines whether each invariant holds for the given instance or returns a counterexample showing a violation. Section 6 presents the resulting traces and their empirical validation.

Validating counterexamples. We validate each counterexample through black-box tests on real banking infrastructure (Section 5.2). For each counterexample, we configure accounts to match the trace preconditions, run the attack sequence through banking portals, and observe the resulting state changes. Because we lack access to internal bank logs or ACH trace files, we rely on user-visible artifacts in online banking dashboards. Successful reproduction confirms that the corresponding operational weakness (Section 6) is exploitable through standard consumer interfaces.

4 Formal Modeling of ACH Payments

In this section, we describe the design choices we make to formalize ACH payment procedures as a machine-checkable protocol model to support rigorous security analysis. We present the abstractions

and modeling strategies that capture the semantics of real-world ACH operations while enabling automated property verification.

4.1 Modeling Components

Modeling Scope. We restrict the formal model to domestic consumer ACH under the WEB and PPD SEC codes (threat assumptions in Section 3.1). The model comprises Originator, ODFI, Operator, RDFI, and Receiver FSMs, treats a Third-Party Sender as a specialized Originator, and encodes the U1 linking methods below, provisional credit, returns, and debit restrictions under bounded batch windows. We exclude rails and SEC codes outside this consumer surface (IAT, corporate and government codes unused in the CWD instances, wire, RTP, and card networks), FI-internal fraud and AML engines (Section 3.1), and return codes beyond those in the analysis traces. Same-Day ACH messages are admitted under the Boolean-window abstraction without modeling sub-day clocks.

Participant FSMs. The ACH payment process can be represented as communicating finite state machines among Originators (O), the ODFI (BO), the ACH Operator (A), the RDFI (BR), and Receivers (R), with each role spanning multiple states and transitions. We assign each participant type an independent FSM template so we can build the protocol modularly and reason compositionally. We extract and formalize each participant before composing the full system model M , which supports per-participant checks and end-to-end protocol verification. The same decomposition lets us adjust participant behavior in a controlled way when we instrument adversarial cases (Section 4.2).

For each participant, we identify the relevant operating rules from the Nacha Operating Rules [42] and translate the natural language specifications into formal state machines. Each FSM template $\mathcal{F}_k^l$ is a tuple $(S_k^l, s_0, V_k^l, \Sigma_k^l, \delta_k^l)$. Here S_k^l is a finite set of states, $s_0 \in S_k^l$ is the initial state, V_k^l is a set of local state variables, Σ_k^l is the message alphabet for inputs and outputs, and $\delta_k^l : S_k^l \times Guards \rightarrow S_k^l \times Actions$ defines guarded transitions. We write $\mathcal{F}_k$ with subscript k for the template class, where $\mathcal{F}_k \in \{O, R, BO, BR, A\}$, and we use superscript l for distinct instances of the same template.

States. States are defined by a combination of their names and associated variables. For instance, Boolean variables in BO such as *entry_validated* and *batch_submitted* record whether the ODFI has validated an incoming ACH entry and submitted the batch to the Operator, respectively. State transitions update both the FSM’s current state and state variables to capture protocol progression.

Transitions. We label transitions as *guard/action*. Guards are propositional formulas over V_i , received messages, and environment variables that trigger the transition, and actions specify variable updates and message transmissions. Non-empty guards are required for all transitions to ensure deterministic state progression when guard conditions are met. Environment variables are Boolean flags set non-deterministically by the model checker to explore alternative execution paths (e.g., optional account-related validation checks and payment settlement timing windows).

Messages and Communication Channels. For tractability, we model only the critical message fields related to authorization and settlement (e.g., account numbers, routing numbers, transaction codes, and amounts). We abstract ACH communication through two

channels. Channel C_{info} transmits payment instructions, acknowledgments, and settlement data between participants (Originator $\leftrightarrow$ ODFI, ODFI $\leftrightarrow$ Operator, Operator $\leftrightarrow$ RDFI). Channel C_s executes fund transfers during settlement.

Communication channels are modeled as *shared resources* that enable asynchronous message passing between participants. When a sender transmits a message, the receiver can immediately access it. The sender invokes $C_{info}.send(receiver, msgData)$ to append the message to the channel. The receiver invokes $C_{info}.rcv(sender, receiver, msgData)$ as a transition guard that matches and accepts the $msgData$ when its type and sender fields match the guard parameters. Under this abstraction, the channels omit queueing and reordering beyond FSM guards.

System Composition. The complete system model $\mathcal{M}$ composes all participant FSMs, message channels, and an environment controller:

$$\mathcal{M} = (\mathcal{E}, O^1, \dots, O^{n_O}, BO, \mathcal{A}, \mathcal{BR}, \mathcal{R}^1, \dots, \mathcal{R}^{n_R}, C_{info}, C_s)$$

where $\mathcal{E}$ controls global timing variables in V_G , and superscripts denote FSM instances. We instantiate n_O Originators and n_R Receivers to model multi-party scenarios (we set n_O and n_R during analysis, see Section 5.2). We bound the model so that the state space remains finite. Each Originator initiates at most two transactions, each batch contains at most three entries, and we model three batch processing windows per verification run. Under these bounds, the model checker explores every state reachable in the finite instance, which keeps verification tractable while still representing multi-party batch flows. Completeness is relative to these bounds and to the fixed participant counts used in analysis (Section 5.2). Counterexamples found inside the bound remain witnesses under at least one Nacha-compatible reading of underspecified options. Absence of other violations outside the bound is not claimed.

4.2 Adversarial Instrumentation

We extend the base protocol model $\mathcal{M}$ with adversarial capabilities to obtain the threat-instrumented model $\mathcal{M}^{adv}$. We instrument Originators (O^i) and Receivers (R^j) with nondeterministic state transitions that capture the adversarial behaviors defined in Section 3.1. Nondeterministic transitions let the model checker explore all attack strategies permitted by the protocol specification within the bounds of Section 4.1.

We encode capability C-1 by instrumenting any Originator (O^i) or Receiver (R^j) instance to act adversarially. The adversary controls FSM state transitions by initiating account linking requests from O^i to ODFI BO or account authorization requests from R^j to O^i at any point during execution, subject to protocol phase constraints (e.g., *linking must precede debit initiation*). The adversary sets the request parameters to values within finite domains defined by ACH account types (checking, savings), transaction codes (debit, credit), and routing and account number formats. For example, a malicious Originator O^0 may link a debit-restricted consumer checking account that still accepts ACH credits.

We encode capability C-2 as nondeterministic choice of routing and account identifiers from a finite victim-identifier domain. Adversarial Originator instances are restricted to authentication paths that do not require online-banking credentials. For example, a malicious Originator O^0 may swap the routing and account numbers of

a legitimate account $\mathcal{R}^1$ to redirect funds. This encoding matches the threat model in Section 3.1.

For capability C-3, $\mathcal{M}^{adv}$ allows Originators (O^i) to select payment initiation parameters nondeterministically: payment type (credit or debit), account type (checking or savings), payment direction, amount, and SEC code from $\{WEB, PPD\}$. Corporate and government SEC codes are excluded because they fall outside the consumer WEB and PPD attack surface evaluated in this work. The model checker explores all feasible combinations of these adversarial choices within the constraints imposed by Nacha Operating Rules [42]. For capability C-4, $\mathcal{M}^{adv}$ allows adversary-controlled Originator and Receiver accounts to transition nondeterministically into holds, closures, and debit blocks during execution.

4.3 Addressing Underspecification

Natural language specifications, such as the Nacha Operating Rules [42], often contain ambiguous, conflicting, or underspecified instructions that block precise formal modeling. We briefly describe the underspecifications encountered during model extraction and our resolutions below.

(U1) Undefined authentication standards. Consider the *linkExternalAccount* operation, which authorizes Originator O^i (e.g., via a mobile banking app) to initiate a debit payment ($\mathcal{R}^j \xrightarrow{\$} O^i$) from a Receiver $\mathcal{R}^j$. While authentication is required, Nacha Article 2, Subsection 2.5.17.5 [42] states:

“An Originator of a debit WEB Entry must establish and implement **commercially reasonable methods** of authentication to verify the identity of the Receiver of the debit WEB Entry.”

The Nacha Operating Rules do not define what constitutes **commercially reasonable methods of authentication** or enumerate acceptable security factors (e.g., knowledge-based, possession-based), leaving critical implementation decisions to each institution.

This ambiguity affects a second critical procedure. When an Originator links the ODFI with Third-Party Senders (e.g., PayPal, Venmo) through the *verifyAccount* operation, Nacha Article 2, Subsection 2.2.1 [42] imposes the same vague requirement:

“The ODFI must utilize a **commercially reasonable method** to verify the identity of an Originator or Third-Party Sender at the time the ODFI enters into an Origination Agreement with the Originator or Third-Party Sender.”

The Nacha Operating Rules thus underspecify two key authentication procedures: *linkExternalAccount* authorizes Originators to debit Receiver accounts, while *verifyAccount* establishes Third-Party Sender agreements. Both rely on the undefined “commercially reasonable” standard despite different trust requirements. To resolve U1, we surveyed bank and provider documentation. Institutions deploy three linking mechanisms: (i) *manual-link*, where users input the routing number (ABA) and account number printed on checks, (ii) *web-login*, where users authenticate through their

Process	Timing Ambiguity/Underspecification
Stop Payment	Unlike the fixed 3-day rule for recurring entries, Single/Subsequent Entries require only a “reasonable opportunity to act,” which is subjective (Subsection 3.7.1.2) [42].
Return Deadline	The deadline is “opening of business” on the second Banking Day (Section 3.8) [42], which is not defined by a specific time of day and varies across banks.
Banking Day	Dual-reference definition (Section 8.15) [42] allows settlement to occur on a “scheduled date” even if the RDFI is closed/not a Banking Day (Section 7.6) [42].
Reversals	The 24-hour transmission window is triggered by the “discovery” of the error (Subsection 2.9.3) [42], which is a subjective event lacking a standardized timestamp.
Recredit	An RDFI must recredit a consumer account “promptly” upon receipt of a valid claim (Subsection 3.11.1) [42]. The rules do not define “promptly” in hours or days.

Table 2: Timing-related underspecifications in Nacha Operating Rules. Each ambiguity uses subjective language or context-dependent references that resist formal encoding.

external bank’s online portal, either directly or via aggregators such as Plaid [52], and (iii) *micro-deposits*, where the provider sends two small ACH credits (typically \$0.01–\$0.99) and users confirm ownership by reporting the exact amounts. Third-party payment providers (e.g., PayPal, Venmo) combine manual linking with micro-deposits to verify ownership.

Our model $\mathcal{M}$ incorporates all three authentication mechanisms. During the *linkExternalAccount* transition, the Originator FSM (O^i) non-deterministically selects manual-link, web-login, or micro-deposit authentication. Each selection triggers a distinct subprotocol encoding required Originator knowledge and capabilities.

Third-Party Senders (e.g., PayPal, Venmo), modeled as specialized Originators, must complete micro-deposit verification through *verifyAccount()* before initiating debits. Both *linkExternalAccount* and *verifyAccount* transitions allow non-deterministic authentication selection. The model checker explores all reachable executions of $\mathcal{M}$ under these authentication choices, ensuring complete coverage of the authentication behaviors within the bound.

(U2) Temporal definition ambiguities. ACH payment processing relies on precise timing conditions for settlement deadlines, processing windows, and event triggers. The Nacha Operating Rules nonetheless contain temporal ambiguities that complicate logical modeling. Table 2 catalogs five timing-related underspecifications identified during model extraction. These ambiguities follow a common pattern: deadlines and triggers use subjective terms (“reasonable”, “promptly”, “discovery”) or context-dependent references (“opening of business”, “Banking Day”) rather than explicit clock readings.

As an example, consider the Banking Day definition, which affects return deadlines and settlement timing. Nacha Article 8, Section 8.15 [42] states:

“**with reference to a Participating DFI**, any day on which such Participating DFI is open to the public during any part of such day for carrying on substantially all of its banking functions, and, **with reference to an ACH Operator**, any day on which the applicable facility of such ACH Operator is being operated.”

This dual-reference definition creates ambiguity when computing return and Notification of Change (NOC) transmission deadlines. When an institution observes a non-standard holiday closure while the ACH Operator remains operational, it is unclear which entity’s Banking Day governs deadline calculations. The network operates on Eastern Time while participating DFIs span all U.S. time zones, which further compounds this ambiguity. Our analysis targets logical vulnerabilities in ACH procedures rather than timing-dependent exploits. ACH transaction behavior, however, depends on timing events, which we encode using Boolean event variables (*event_started*, *event_expired*) instead of explicit clock ticks (Section 4.4). The temporal ambiguities cataloged in Table 2 prevent direct translation of timing constraint windows into formal logic. We resolve these ambiguities through two modeling decisions that preserve the logical ordering of payment operations.

First, we assume all participants operate within a common time zone and share a calendar. This choice removes inter-institutional timing discrepancies. Second, we replace subjective deadline language (“reasonable opportunity”, “promptly”) with a deterministic rule: actions initiated before the end-of-day complete within the current Banking Day, and actions initiated after the end-of-day complete at the start of the next Banking Day. These abstractions preserve temporal ordering, which is enough for identifying logical flaws in authentication, authorization, and settlement flows, while avoiding state-space explosion.

(U3) Conflicting error conditions. While modeling *BatchProcessing* state transitions for the ACH Operator, we find contradictory file rejection conditions about the message parameters in two different locations. Appendix 2, Part 2.4 [42] specifies that files or batches will be rejected for invalid transaction codes or non-numeric amount fields:

“The ACH Operator **will either pend or Reject the entire File or any batch(es)** meeting one or more of the error conditions identified below: When the Transaction Code in an Entry Detail Record is invalid, ... When the Amount field in an Entry Detail Record is non-numeric.”

Part 2.5, however, states that error conditions R19 (Amount Field Error) and R36 (Return of Improper Credit Entry) will never cause file or batch rejection, directly contradicting the criteria in Part 2.4. Because the Nacha Operating Rules do not fix the correct ACH Operator behavior, we model all admissible *BatchProcessing* transition behaviors; that is, we include transition conditions for both variants. This choice lets us analyze every potential scenario involving payment processing.

Similarly, for any rules and guidelines where alternative behaviors are explicitly or implicitly defined (e.g., *optional parameters* or use of keywords such as “may” or “should” in the Nacha Operating

Rules), we model all candidate behaviors in $\mathcal{M}$, as long as none are evidently weak or insecure. Using environment variables, we ensure that the model checker explores and analyzes all candidate behaviors of the ACH payment system within the bounded domains until a counterexample is found. During our analysis, the model checker can non-deterministically set the values of these variables at runtime, or we can control the variables' values through security properties to focus on specific execution paths.

4.4 Modeling Time-Dependent Behavior

ACH processes transactions in batches at scheduled intervals. Many state transitions depend on timing constraints, but modeling time using discrete clock ticks causes state-space explosion [15]. We model timing windows as Boolean variable pairs: *event_started* and *event_expired*. For example, *settlement_started* represents settlement window start and *settlement_expired* marks expiration. Intermediate states are captured through logical conjunction: an ongoing settlement window is *settlement_started* $\wedge$ $\neg$ *settlement_expired*.

Consider the ACH requirement: "If an originator O^i initiates an ACH credit on Wednesday by 5:00 p.m. for Friday settlement, the RDFI must make funds available to Receiver $\mathcal{R}^j$ by 9:00 a.m. Friday". We encode this as:

$$\begin{aligned} \text{complete} &:= \text{achCredit}(O^i, \mathcal{R}^j, e) \wedge \text{transfer_started}(e) \\ &\wedge \text{settlement_started}(e) \wedge \neg \text{settlement_expired}(e) \\ &\implies \text{FundsAvailable}(\mathcal{R}^j, e) \end{aligned}$$

The model uses a single global return-window phase as an abstraction of per-entry windows. The formal property in Section 5.1 quantifies per entry, while the model phase scopes globally. This abstraction is sound for the safety properties of interest because no reachable state distinguishes per-entry from global window closure. The Boolean-window abstraction preserves initiation, provisional access, and return or settlement order. It omits real-time deadlines, cross-timezone skew, and Same-Day clock precision.

4.5 Batch Processing

Batch processing represents a critical ACH component but is challenging to encode in symbolic model checkers that lack dynamic data structure support. We encode batch membership within individual transaction states rather than explicit containers. Each transaction e_i maintains $\sigma_{e_i} = e_i.\text{payment_state}$ over the domain:

$$\sigma_{e_i} \in \{\text{not_submitted}, \text{submitted}, \text{batched}, \text{pending_settlement}, \text{settled}, \text{rejected}, \text{returned}, \text{reversed}\}$$

Batch membership is determined by identifying all transactions in *submitted* state. Batch transmission is modeled as a synchronized global transition triggered by collection window closure (*batch_expired*). At closure moment, batch B_t is: $B_t = \{e_1, e_2, \dots, e_n \mid e_i \in \mathcal{T} \wedge \sigma_{e_i} = \text{submitted}\}$. The model enforces atomicity through synchronized state transition:

$$\forall e_i \in \mathcal{T} \left((\sigma_{e_i} = \text{submitted} \wedge \text{batch_expired}) \rightarrow \sigma'_{e_i} = \text{batched} \right)$$

Detailed batch processing procedures (transaction flow, ACH Operator processing, RDFI handling, reversal mechanisms) are in Appendix C.

5 Property-Guided Model Analysis

This section formalizes the three ACH security requirements we verify in this paper. We state each property and its formal definition, then outline how the model and properties are encoded in nuXmv for model checking.

5.1 Properties of Secure ACH Transactions

Before analyzing the formal model $\mathcal{M}^{adv}$, we require security properties that guide the model checker. We therefore reviewed the Nacha Operating Rules [42] and distilled three high-level security requirements for ACH payments within the scope of this paper. ACH payments involve three principal entities: the Originator, the Receiver, and their respective financial institutions. The corresponding security properties address authorization between account holders (the Originator and Receiver), transaction integrity, and reliable fund transfer.

The first property we examine is that authorization granted by the Receiver remains valid throughout the full transaction lifecycle, from initiation through settlement. This property concerns linking procedures, in which an Originator or TPS gains authorization to initiate payments from an external bank account. ACH batch processing introduces a three-to-five-day settlement window during which authorization status may change. If authorization is revoked *after* payment initiation but *before* settlement completes, the system must prevent the transfer from finalizing. We capture this requirement as a safety invariant over two named state predicates that span the entry's submission-to-settlement lifecycle.

Φ_1 (Continuous Authorization). Every settled entry must have been submitted while authorized and must not have been revoked between submission and settlement:

$$\begin{aligned} \forall o, r, E \quad \text{EntrySettled}(E, o, r) \implies \\ \text{Authorized}(E, o, r) \wedge \neg \text{Revoked}(E, o, r) \end{aligned}$$

Here, *EntrySettled*(E, o, r) holds when the ODFI has posted a provisional credit for entry E to Originator o 's account on behalf of Receiver r , committing liquidity regardless of the ultimate network-side disposition of the entry. This is an ODFI-local commitment event. The predicate holds as soon as the ODFI advances funds, and it remains true even if the underlying ACH debit is subsequently returned. *Authorized*(E, o, r) holds when, at the moment Originator o submits entry E to the ACH pipeline, Receiver r 's authorization for o is in force. *Revoked*(E, o, r) holds when Receiver r revokes authorization for o between submission of E and its terminal state. A violation enables unauthorized transfers by exploiting the gap between authorization at initiation and fund transfer at settlement. We assume banks are honest in our model.

An RDFI-side debit restriction that blocks settlement therefore yields counterexamples in the same predicate pattern as mid-window revocation, because the ODFI path can advance while the receiver never grants a debit that survives through settlement. The formal predicate *Authorized*(E, o, r) subsumes both the Receiver's legal authorization status under Nacha rules and the RDFI-side operational debit eligibility of the account. This abstraction is conservative.

An RDFI-side debit restriction prevents the ODFI from completing the debit regardless of whether the Receiver's standing authorization agreement remains in force, so treating the restriction as an authorization failure yields no false positives in the model.

The second property requires the modeled ACH procedures to preserve monetary value across all transactions. For each settlement cycle, the number of credit entries must equal the number of debit entries within each entry-amount class. We map the entry files as a finite set of amount classes $K = \{\text{micro}, \text{regular}\}$, where all entries within class k carry the same nominal amount a_k . The model can therefore express monetary conservation as per-class equality of credit and debit counts over a finite set of states, which keeps the property checkable as a finite-state safety invariant.

Φ_2 (Zero-Sum Invariant). For any complete cycle, the number of credit entries must equal the number of debit entries within each entry class:

$$\forall c. \quad \text{CycleComplete}(c) \Rightarrow \bigwedge_{k \in K} \text{CountCredits}(c, k) = \text{CountDebits}(c, k)$$

$\text{CycleComplete}(c)$ holds when all entries in cycle c have reached a terminal state, including *settled*, *returned*, or *reversed*. The symbols $\text{CountCredits}(c, k)$ and $\text{CountDebits}(c, k)$ count the credit and debit entries of class k in cycle c , respectively. Because every entry in class k carries amount a_k , per-class count equality implies per-class amount equality, and summing over K recovers the aggregate-equality requirement that monetary preservation demands. A violation allows an attacker to extract funds, for example by accepting credits while preventing the corresponding debits.

The third property requires resources (funds, credit limits) to be released only after the underlying payment reaches final settlement.

Φ_3 (Settlement Finality). Resources dependent on entry E may be released only after E 's return window has closed:

$$\forall r, E \quad \text{ResourceAvailable}(r, E) \Rightarrow \text{ReturnWindowClosed}(E)$$

Settlement in the ACH model is operationally defined as “the return window closed without an intervening return,” so requiring window closure subsumes settlement as the irrevocability condition. The predicate $\text{ResourceAvailable}(r, E)$ captures the moment r (e.g., credit limit, account balance) becomes accessible, which begins at internal posting and may precede external settlement by several days. $\text{ReturnWindowClosed}(E)$ indicates that the ACH return period elapsed without a return. A counterexample is a path in which resources become available while the entry remains provisional.

5.2 Implementation and Validation Setup

This subsection describes the nuXmv encoding, the sanity checks that validate $\mathcal{M}^{adv}$ against Nacha-conformant traces, and the empirical setup used to reproduce Φ_i counterexamples. Artifacts for

formal verification are available at <https://doi.org/10.5281/zenodo.22701199>[36]. Table 1 summarizes our findings.

Model Composability. We construct the abstract ACH model by consulting the Nacha Operating Rules [42, 43], encoding five participant FSMs (O, BO, A, BR, R) as composable nuXmv modules communicating over two shared single-slot buses (`c_info_channel` and `c_dollar_channel`), which implement the abstract channels C_{info} and $C_{\$}$. The top-level composition instantiates two adversarial Originators, two adversarial Receivers, one ODFI, one ACH Operator, and one RDFI, along with the environment controller ($\mathcal{E}$).

Verification Setup. We first sanity-check $\mathcal{M}^{adv}$ against selected functional traces from the Nacha Operating Rules in scope, including successful WEB/PPD settlement, authorized returns, and micro-deposit credit posting under credit-accepting configurations. When a returned trace contradicts a Nacha-mandated outcome in scope, we treat it as a modeling error and correct the transition system until these traces conform. We then model-check $\mathcal{M}^{adv}$ against Φ_1 , Φ_2 , and Φ_3 using nuXmv [12]. The INVARSPEC engine applies IC3/PDR over the parameterized model and either establishes the invariant for this finite instance or returns a counterexample π_i . We interpret each π_i as a witness that a Φ_i violation is reachable in $\mathcal{M}^{adv}$ under a Nacha-compatible reading of underspecified options.

Empirical Validation. We confirm each counterexample π_i by executing the corresponding steps on live consumer accounts in our evaluation sample. The sample consists of legally held consumer accounts at two of the four target banks, three leading credit-card issuers, and three dominant payment platforms, selected to represent the institutional actors involved in the three CWD instances. For each counterexample we match the trace preconditions and record user-visible balances, timestamps, status transitions, fund availability, return timing, and terminal states.

6 Security Vulnerabilities in ACH Payments

We identify ACH security failures that lead to a new class of attacks called **Credit-Without-Debit (CWD)**, for which we find three instances in practice. Each instance is linked to a model counterexample that violates a security invariant. These failures arise from permissive Nacha rule interpretations, which some institutions adopt as operational practices. Empirical analysis verifies the presence of these weaknesses in the institutions evaluated.

6.1 A1: creditBLEED Attack

This attack exploits a delegation gap in ODFI-side access-control checks observed in our evaluation. The ODFI treats routing number validation alone as grounds to initiate debits, without verifying account accessibility at the RDFI. An adversary links a victim's account using valid routing and account numbers, which the ODFI accepts and forwards to the ACH operator. The ODFI grants provisional credit within one to two days, creating a two-day window during which the adversary can extract funds. When targeting debit-restricted or dormant accounts, the RDFI returns the transaction days later (e.g., with return code R16 when the account is blocked for debits), resulting in a financial loss to the ODFI after the adversary has withdrawn the provisionally credited amount.

Figure 3 summarizes this creditBLEED flow: the ODFI initiates debits from routing-validated credentials, grants provisional credit

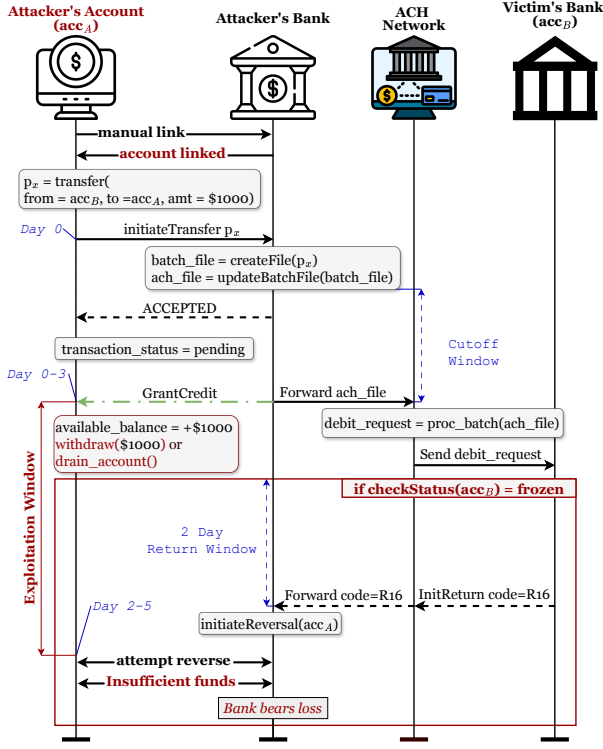


Figure 3: creditBLEED (A1) instance of Credit-Without-Debit in ACH debit processing.

Figure 4 shows the external account linking interface. (a) is the 'Add an external account' form, which prompts the user to enter their account information, including the routing number, account number, and account nickname. (b) is the confirmation message and account details, showing that the user has successfully added an external bank account and displaying the account details.

Figure 4: External account linking interface. The bank app prompts for routing and account numbers (a), then displays the linked external account (b) without verifying account access at the target institution.

during the settlement window, and learns RDFI-side debit restrictions only when a late return arrives after funds may already have been extracted.

Counterexample analysis. Checking Φ_1 against M^{adv} returns a counterexample π_1 . Tracing the state transitions in π_1 , an Originator submits an entry E targeting a debit-restricted Receiver account, the ODFI accepts the submission after $CheckRoutingValidity(routing_number) = VALID$, the entry advances to *AUTHORIZED*, the ODFI grants provisional credit, and several

Figure 5 shows two screenshots. (a) is the 'Schedule a Transfer' interface, which allows the user to schedule a transfer from Bank B to Bank A for an amount of \$1,000.00. (b) is the Bank A banking dashboard, which shows a transaction history for April 10, 2025, including an online transfer from Bank B for \$1,000.00.

Figure 5: (a) ACH Debit scheduling interface. (b) Bank A banking dashboard shows funds received before settlement.

days later the RDFI returns R16 (Account Frozen/Blocked) and the entry transitions to *RETURNED*. $EntrySettled(E, o, r)$ holds at the moment the ODFI posts the provisional credit, while $Authorized(E, o, r)$ does not hold at the RDFI. The Receiver's account never granted continuous debit authorization through settlement, as the R16 return indicates.

The violation arises because the ODFI's submission gate verifies only network-level reachability, namely whether the routing number maps to an active institution in the ABA directory, and does not verify whether the RDFI-side account permits the requested debit. The ODFI commits liquidity at routing-validation time and learns the access-control restriction only when a return code arrives during settlement. A1 is assigned to Φ_1 because π_1 shows that ODFI liquidity commitment under routing-only checks can settle while RDFI-side debit authorization never holds at settlement time. **Root-cause analysis.** The ODFI validates only routing and account number format when initiating a debit, without checking whether the target account is locked, debit-restricted, or otherwise restricted. It then grants provisional credit immediately, assuming settlement will succeed, even though actual fund transfer takes up to five days. Restriction status is discovered only when the RDFI rejects the debit during settlement, briefly treating a restricted account as if it were debit-able.

Permissive Nacha condition. Nacha rules do not require ODFIs to query RDFI-side account status before initiating a debit. ODFIs commit liquidity at routing-validation time under this rule set. Restriction status becomes known only when a return code arrives during settlement. That timing gap yields π_1 .

Empirical assessment. Figure 5 documents the online-dashboard trace for our controlled reproduction of A1. Our empirical test used a dormant account under researcher control to demonstrate the attack without affecting third parties. We tested this vulnerability using Bank A (ODFI) and Bank B (RDFI). A dormant Bank B account was linked as the debit target using only its routing and account numbers (Fig. 4), and an active Bank A account served as the credit destination. An external ACH debit was then submitted from Bank A to Bank B (Fig. 5a). Under capability C-2, A1 applies when the originating institution grants provisional credit as observed in our sample. When the targeted account is not debit-restricted, the attack

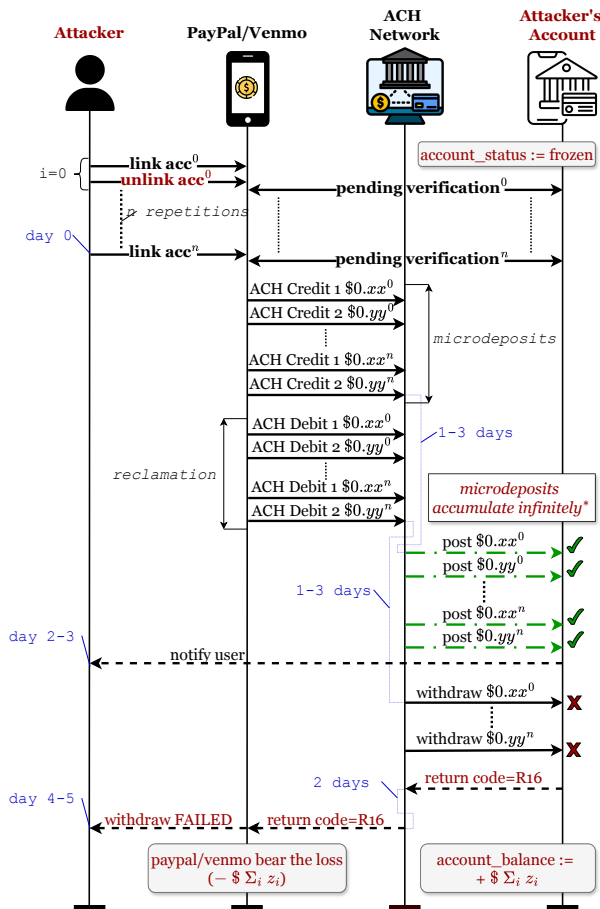


Figure 6: creditMINT (A2) instance of Credit-Without-Debit in micro-deposit verification.

transfers funds from the victim to the attacker's account without additional verification or control.

Without access to internal logs, we performed a black-box test using only the banks' online dashboards. We observed the following. (i) After submission, Bank A showed the debit as pending with no immediate error. (ii) Three days later, Bank A made the funds available before settlement (Fig. 5b). (iii) Bank B did not debit the debit-restricted account. (iv) Several days later, Bank B returned R16 with a message indicating the account was blocked for debits, and Bank A reversed the credit. These results confirm that account restrictions at Bank B were enforced only during settlement, not at initiation, allowing provisional funds to be temporarily available.

6.2 A2: creditMINT Attack

Many payment providers verify bank account ownership through micro-deposits. They send small ACH credits and ask users to report the amounts. To settle these credits, the provider tries to withdraw the same amounts. Successful credits are mistakenly treated as implicit authorization for debits. In practice, debit permission is enforced only at withdrawal, not during verification. Attackers can exploit this by linking restricted or debit-blocked accounts. Deposits

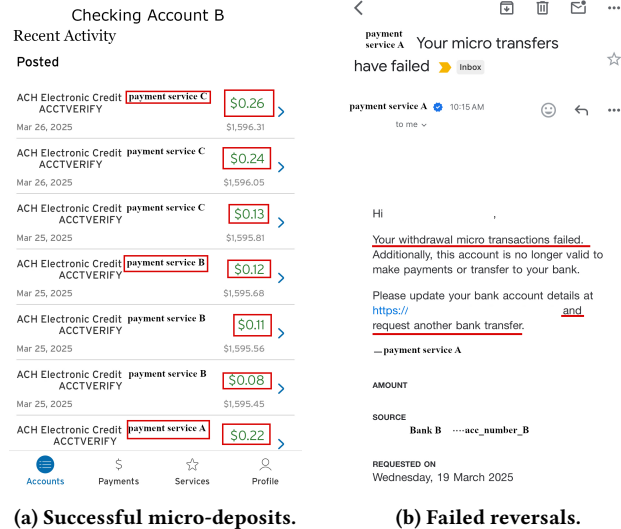
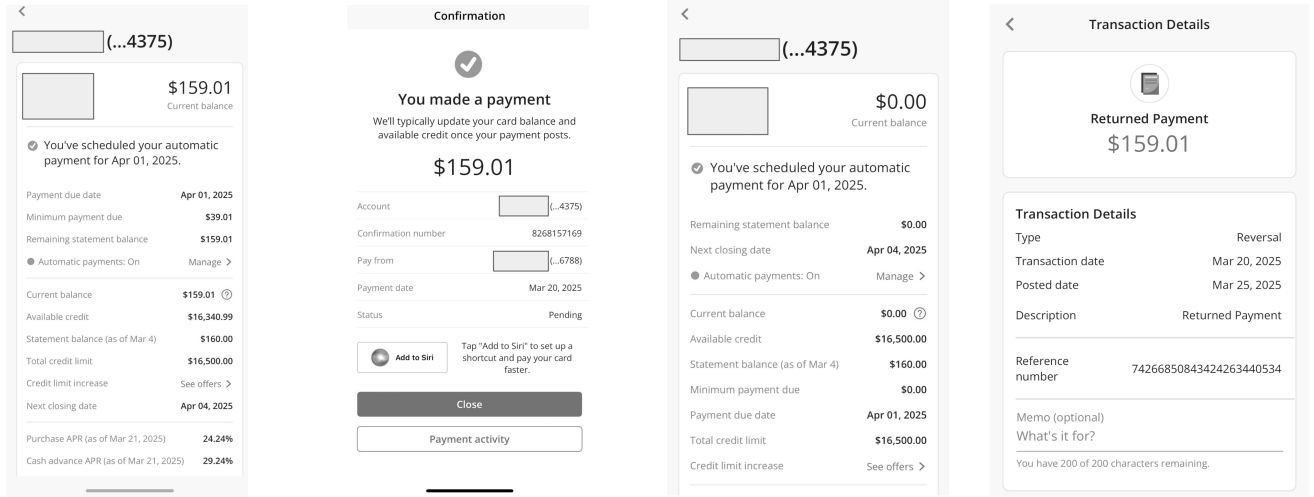


Figure 7: Empirical validation of micro-deposit accumulation: (a) multiple micro-deposit credits posted to a debit-restricted account; (b) corresponding debit withdrawal failures.

succeed, but withdrawals fail, which allows repeated micro-deposits to accumulate without repayment. Figure 6 depicts the resulting asymmetric verification: micro-credits complete and are treated as proof of control, while matching debits fail under RDFI-side restrictions, enabling accumulation across repeated cycles.

Counterexample analysis. Checking Φ_2 against $\mathcal{M}^{adv}$ returns a counterexample π_2 . Along π_2 , a Third-Party Sender posts micro-class credits to a target account through the symbolic function *ProcessCredit*(*account*, *amount*), the credits complete without return codes, the account transitions to *LINKED*, and the matching micro-class debits fail with R08 (Payment Stopped) or R16 (Account Frozen for Debits). At the close of the cycle, $\text{CountCredits}(c, \text{micro}) > \text{CountDebits}(c, \text{micro})$, which violates the per-class equality required by Φ_2 . The violation arises because the verification subprotocol decides linkage on credit success alone, while ACH accounts support asymmetric permission configurations that allow inbound credits while blocking outbound debits. Credit success at the RDFI does not imply debit eligibility, and the provider commits to a *LINKED* state before observing any debit outcome. The formal counterexample π_2 captures the single-cycle violation, in which micro-class debits fail while credits complete within one verification cycle. The empirical attack amplifies this through iterative cycle cancellation and restart, where each new cycle issues additional credits without recovering prior deposits. We discuss the underlying root cause and the corresponding permissive condition below.

Root-cause analysis. The micro-deposit flow assumes credit success implies debit capability, but debit permission is enforced only at withdrawal after credits are already posted. Payment providers further allow verification restarts without rate limiting, which enables repeated cycles that accumulate micro-deposit credits with no corresponding debit recovery.



(a) Payment scheduled on linked ACH-debit-blocked account.

(b) Issuer shows pending until internal posting completes.

(c) Paid balance with limit restored before bank account debits.

(d) Bank returns the ACH debit from restricted account.

Figure 8: Empirical trace of premature posting for a credit card payment funded from a linked bank account with ACH-debit restrictions. The panels follow the scheduled payment, issuer-side progression from pending to accepted while the ACH debit is still outstanding, posting as paid with restored available credit before the restricted bank account debits, and an ACH return from the originating bank after about six business days, once the issuer had already treated the payment as final.

Permissive Nacha condition. Nacha rules do not require a pre-flight debit-capability check before verification credits post. Payment providers learn debit eligibility only by submitting an actual debit and observing the response. Micro-class credits in π_2 can then accumulate without matched debits.

Empirical assessment. Figure 7 provides the main screenshot evidence. We evaluated the micro-deposit verification flow against three major payment providers (A, B, and C in the left panel, Fig. 7a) by linking a bank account configured with ACH-debit restrictions. All three credited the account with micro-deposits, but every corresponding reversal failed, with the right panel illustrating repeated failed withdrawal attempts (Fig. 7b). Because payment providers detect debit failure only after posting credits, repeatedly canceling and re-initiating verification caused each reset to issue new micro-deposits, enabling multiple accumulation cycles per verification reset. This behavior produces a re-triggerable credit accumulation with no matched debits. This issue is further amplified by providers that do not impose timeouts on repeated reversal failures. Without effective safeguards, adversaries can exploit this loophole to amass funds from payment providers through repeated credit cycles.

6.3 A3: creditRESET Attack

Credit-card payment systems decouple internal posting from ACH settlement but operate under the assumption that both stages will ultimately align. The issuer restores the user's available credit once the payment posts internally (typically within two days), even though the ACH debit funding the payment typically settles only after several more days (usually five to seven days). This creates a structural mismatch. Posting is an issuer-local bookkeeping event, while settlement depends entirely on the external bank's access-control policy. Although the design improves user experience by

making credit available quickly rather than forcing customers to wait through the full ACH settlement window, it creates an exploitable posting-before-settlement window in our sample. An attacker can exploit this optimistic posting model by paying the card from an ACH-restricted account. Once the payment is submitted, the credit card issuer prematurely treats it as settled and restores the credit limit, which allows the attacker to spend against funds that were never actually paid.

Counterexample analysis. Checking Φ_3 against M^{adv} returns a counterexample π_3 . Along π_3 , an entry E transitions to *POSTED* at $t_{posting}$, the issuer's $CanWithdraw(account, amount)$ guard evaluates to *true* based on $InternalBalance(account)$, the resource (restored credit limit or available balance) becomes accessible at t_{avail} , and an ACH return arrives at t_{return} before the return window expires. The state $ResourceAvailable(r, E)$ holds while $ReturnWindowClosed(E)$ is still false, leaving a four- to five-day window during which the entry can still be reversed. The violation occurs at the transition *POSTED* $\rightarrow$ *WITHDRAWAL_AVAILABLE* which consults only internal ledger state and never blocks on external settlement confirmation. The issuer treats internal posting as evidence of finality even though the return window remains open, so resources are released against funds that may still be reversed.

Root-cause analysis. The issuer restores the credit limit upon internal posting, treating it as settlement confirmation despite having no guarantee the underlying ACH debit will succeed. Because ACH failures surface only days later, the issuer commits to a financial state before funds are confirmed.

Permissive Nacha condition. Nacha rules lack a live settlement-status flag that issuers can consult before releasing resources. Issuers infer settlement only retrospectively, from the absence of a return before the return window closes. Entry E can therefore reach $ResourceAvailable(r, E)$ while returns remain admissible, as in π_3 .

Empirical assessment. Figure 8 documents the dashboard trace, beginning with payment against an ACH-debit-blocked linked account (Fig. 8a). We tested three credit-card accounts. For each card, we linked a debit-restricted bank account and paid the full card balance. Upon submission, each issuer stated that payments “*take a couple of days to process*” and that the balance and available credit would update “*once your payment posts*” (Fig. 8b). In all cases, the issuer posted the payment within two business days, marked it paid, showed a zero balance, and restored the full credit limit while the ACH debit remained unsettled (Fig. 8c), which allowed immediate new spending. Because posting is internal and independent of ACH settlement, we repeated the cycle: submit, wait for posting, spend the restored credit. Repeated payments temporarily increased available credit. When the ACH debit failed after six business days (Fig. 8d), the issuer reversed the payment but did not assess interest for the paid interval. This approach also permitted interest-free cycles without additional spending.

7 Remedies and Design Implications

This section outlines remedies informed by our formal analysis and the operational weaknesses identified in our sample.

Continuous Authorization Enforcement. Authorization in ACH payments should be treated as a persistent predicate rather than a one-time check at initiation. ACH debits must remain conditionally authorized until settlement succeeds. Any change in the debit eligibility of the source account (e.g., debit restriction or account status update) during the settlement window should invalidate the transaction and revoke provisional credits. Banks should revalidate authorization at posting and again before settlement finalizes, rather than assuming monotonic validity.

Delayed Exposure of Provisional Credits. User-visible balances and spendable funds should not reflect provisional credits until the corresponding debit has passed an enforceable access-control point. If provisional crediting is necessary (e.g., early fund availability), such credits should be placed in an isolated state, not usable for withdrawals, transfers, bill payments, or credit-line restoration. This isolation keeps in-flight transactions separate from user spending and blocks exploitation of intermediate states.

Explicit Zero-Sum Enforcement Across Institutions. Cross-institution settlement workflows must enforce a zero-sum invariant throughout execution, not only at final reconciliation. Any workflow that exposes funds at one institution without a guaranteed, enforceable debit at another should be treated as inconsistent. In practice, banks and third-party payment providers are required to track debits and credits as a single logical transfer and prohibit user access until both sides reach a committed state.

Symmetric Verification in Account Linking. Micro-deposit-based account linking must verify both inbound credit acceptance and outbound debit capability before an account is marked usable. Failed debit recovery must be treated as verification failure, and repeated credit issuance for retries must be prevented.

Specification-Level Clarification. Payment network standards must clarify (i) whether authorization is ongoing or a one-time event, (ii) whether provisional credits are user-visible, and (iii) how settlement failures are handled in user-visible state. Ambiguities

in standard definitions permit implementations that are insecure despite being compliant with the specification.

8 Related Work

To our knowledge, this work is the first formal analysis of consumer ACH banking security. Closely related work covers credit-card systems, financial systems, and formal network-protocol analysis.

Security analysis of payment systems. Prior work analyzes payment vulnerabilities through reverse-engineering and testing of EMV [6, 10, 40, 48–50, 59], personal payment platforms [2, 5, 8, 33, 34, 61], UPI [30, 31, 35, 39], mobile money [13, 16, 17, 24, 25, 54], and digital wallets [1, 4, 26, 32]. ACH return-code exploitation appears in fraud advisories, but those treatments document exploitation without identifying design-level causes. We instead use formal verification to find the violations that enable such attacks.

Systemic cyber risk in financial systems. Prior work studies market failures and cross-institution cyber cascade risk [3, 7, 9, 14, 21–23, 28, 29, 38, 55]. We instead focus on protocol- and workflow-level properties of ACH-style batch settlement.

Formal verification for network protocol security. Finite-state techniques analyze message-based protocol security [15], though rich protocol classes remain undecidable [19]. We reuse safety invariants over permitted behaviors and target ACH batch-settlement workflows within a rules-derived finite bound.

9 Conclusion

We present a formal security analysis of domestic consumer ACH payment procedures by modeling ACH processing with adversarial capabilities and verifying three core security invariants: continuous authorization, zero-sum integrity, and settlement finality. Counterexamples identify a **Credit-Without-Debit (CWD)** attack class with three instances in debit authorization, micro-deposit verification, and premature credit-card payment posting. The formal guarantees are relative to $\mathcal{M}^{adv}$ and the threat assumptions in Sections 3.1 and 4. Black-box experiments on our evaluation sample confirm the instances in those deployments. Coordinated disclosure led to remediation at affected institutions. Follow-up testing shows that the reported instances are no longer reproducible.

References

- [1] Sharad Agarwal and Marie Vasek. 2025. Card-Not-Present Fraud resulting from Smishing Attacks: An Experimental Study. In *Proceedings of the 2025 New Security Paradigms Workshop*. 108–121.
- [2] Waqas Ahmed, Aamir Rasool, Abdul Rehman Javed, Neeraj Kumar, Thippa Reddy Gadekallu, Zunera Jalil, and Natalia Kryvinska. 2021. Security in next generation mobile payment systems: A comprehensive survey. *IEEE Access* 9 (2021), 115932–115950.
- [3] Naeem AllahRakha. 2024. Rethinking digital borders to address jurisdiction and governance in the global digital economy. *Int. J. Law Policy* 2, 1 (2024).
- [4] Raja Hasnain Anwar, Syed Rafiul Hussain, and Muhammad Taqi Raza. 2024. In wallet we trust: bypassing the digital wallets payment security for free shopping. In *33rd USENIX Security Symposium (USENIX Security 24)*. 541–558.
- [5] Esther Edem Archibong, Bliss Utibe-Abasi Stephen, and Philip Asuquo. 2024. Analysis of cybersecurity vulnerabilities in mobile payment applications. *Archives of Advanced Engineering Science* (2024), 1–12.
- [6] Shane Balfe and Kenneth G Paterson. 2008. e-EMV: emulating EMV for internet payments with trusted computing technologies. In *Proceedings of the 3rd ACM workshop on Scalable trusted computing*. 81–92.
- [7] Giuliana Birindelli and Antonia Patrizia Iannuzzi. 2024. The systemic importance of cyber risk in banks. In *Systemic Risk and Complex Networks in Modern Financial Systems*. Springer Nature Switzerland Cham, 301–321.

- [8] Sriramulu Bojjagani, VN Sastry, Chien-Ming Chen, Saru Kumari, and Muhamad Khurram Khan. 2023. Systematic survey of mobile payments, protocols, and security infrastructure. *Journal of Ambient Intelligence and Humanized Computing* 14, 1 (2023), 609–654.
- [9] Antoine Bouveret. 2017. *Cyber Risk, Market Failures, and Financial Stability*. Working Paper WP/17/185. International Monetary Fund.
- [10] Sergiu Bursuc, Ross Horne, Sjouke Mauw, and Semen Yurkov. 2023. Provably unlinkable smart card-based payments. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*. 1392–1406.
- [11] CashApp. 2026. CashApp. <https://cash.app/>. Accessed: 2026-02-04.
- [12] Roberto Cavada, Alessandro Cimatti, Michele Dorigatti, Alberto Griggio, Alessandro Mariotti, Andrea Micheli, Sergio Mover, Marco Roveri, and Stefano Tonetta. 2014. The nuXmv Symbolic Model Checker. In *CAV*. 334–342.
- [13] Sen Chen, Lingling Fan, Guozhu Meng, Ting Su, Minhui Xue, Yinxing Xue, Yang Liu, and Lihua Xu. 2020. An empirical assessment of security risks of global android banking apps. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*. 1310–1322.
- [14] John J Chung. 2017. Critical infrastructure, cybersecurity, and market failure. *Or. L. Rev.* 96 (2017), 441.
- [15] Edmund Clarke, William Klieber, Miloš Nováček, and Paolo Zuliani. 2012. *Model Checking and the State Explosion Problem*. 1–30. doi:10.1007/978-3-642-35746-6_1
- [16] Hesham Darvish and Mohammad Husain. 2018. Security analysis of mobile money applications on android. In *2018 IEEE International Conference on Big Data (Big Data)*. IEEE, 3072–3078.
- [17] Alioune Diallo, Jordan Samhi, Tegawendé Bissyandé, and Jacques Klein. 2024. (In) Security of Mobile Apps in Developing Countries: A Systematic Literature Review. *arXiv preprint arXiv:2405.05117* (2024).
- [18] Discover Bank. 2025. What do the numbers on a check mean? <https://www.discover.com/online-banking/banking-topics/what-do-numbers-on-a-check-mean/>. Accessed: 2025-10-21.
- [19] Nancy A. Durgin, Patrick Lincoln, and John C. Mitchell. 2004. Multiset rewriting and the complexity of bounded security protocols. *Journal of Computer Security* 12 (2004), 247 – 311. <https://api.semanticscholar.org/CorpusID:2523975>
- [20] Electronic Code of Federal Regulations (eCFR). 2026. 12 CFR Part 1005 : Electronic Fund Transfers (Regulation E). <https://www.ecfr.gov/current/title-12/chapter-X/part-1005>. See §1005.10(c) on stop payment and revocation of authorization for preauthorized EFTs. Accessed: 2026-04-25.
- [21] Alessandro Fedele and Cristian Roner. 2022. Dangerous games: A literature review on cybersecurity investments. *Journal of Economic Surveys* 36, 1 (2022), 157–187.
- [22] Oliver Gulyas and Gabor Kiss. 2023. Impact of cyber-attacks on the financial institutions. *Procedia Computer Science* 219 (2023), 84–90.
- [23] Eldar Haber and Tal Zarsky. 2016. Cybersecurity for infrastructure: a critical analysis. *Fla. St. U. L. Rev.* 44 (2016), 515.
- [24] Sameer Hayikader, FN Hadi, and Jamaludin Ibrahim. 2016. Issues and security measures of mobile banking apps. *International Journal of Scientific and Research Publications* 6, 1 (2016), 36–41.
- [25] Daojing He, Shihao Li, Cong Li, Sencun Zhu, Sammy Chan, Weidong Min, and Nadra Guizani. 2020. Security analysis of cryptocurrency wallets in android-based applications. *IEEE network* 34, 6 (2020), 114–119.
- [26] Xenia Hofmeier, David Basin, Ralf Sasse, and Jorge Toro-Pozo. 2025. SoK: Attacks on Modern Card Payments. *arXiv preprint arXiv:2504.03363* (2025).
- [27] Shaharyar Khan, Ilya Kabanov, Yunke Hua, and Stuart Madnick. 2023. A Systematic Analysis of the Capital One Data Breach: Critical Lessons Learned. *ACM Transactions on Privacy and Security* 26, 1, Article 3 (2023), 29 pages. doi:10.1145/3546068
- [28] Antonis Kotidis and Neeltje van Horen. 2023. The Propagation of Cyberattacks through the Financial System. *ECB Working Paper Series* (2023).
- [29] Brenden Kuerbis and Farzaneh Badieli. 2017. Mapping the cybersecurity institutional landscape. *Digital Policy, Regulation and Governance* 19, 6 (2017), 466–492.
- [30] Renuka Kumar, Siddharth Sharma, and Ponnuram Kumaraguru. 2020. Security Analysis of Unified Payments Interface and Payment Apps in India. In *29th USENIX Security Symposium*.
- [31] K Krithiga Lakshmi, Himanshu Gupta, and Jayanthi Ranjan. 2019. UPI based mobile banking applications—security analysis and enhancements. In *2019 Amity International Conference on Artificial Intelligence (AICAI)*. IEEE, 1–6.
- [32] Hyung-Jin Lim, Sokjoon Lee, Moonseong Kim, and Woochan Lee. 2025. Comparative Analysis of Security Features and Risks in Digital Asset Wallets. *Electronics* 14, 12 (2025), 2436.
- [33] Junru Lin, Mingzhe Liu, Songze Li, and Xuechao Wang. 2025. SecurePay: Enabling Secure and Fast Payment Processing for Platform Economy. *arXiv preprint arXiv:2505.17466* (2025).
- [34] Jiadong Lou, Xu Yuan, and Ning Zhang. 2021. Vulnerabilities in Emerging Personal Payment Systems. In *30th USENIX Security Symposium*.
- [35] Yash Madwanna, Mayur Khadse, and BR Chandavarkar. 2021. Security issues of unified payments interface and challenges: case study. In *2021 2nd international conference on secure cyber computing and communications (ICSCCC)*. IEEE, 150–154.
- [36] Ali Hamza Malik and Muhammad Taqi Raza. 2026. *Artifacts for: Credit Without Debit: Formal Verification of Banking Payment Systems*. doi:10.5281/zenodo.22701199
- [37] James McAndrews. 1994. The automated clearinghouse system: moving toward electronic payment. *Business Review* (1994), 15–23.
- [38] Ntokozo F Miya and Nazeer Joseph. 2025. Banking on resilience: 20 years of cybersecurity evolution. *South African Journal of Information Management* 27, 1 (2025), 2019.
- [39] Deepthi Mungara, Harshini Sri Ramulu, and Yasemin Acar. 2025. Security and Privacy Advice for {UPI} Users in India. In *34th USENIX Security Symposium (USENIX Security 25)*. 6085–6103.
- [40] Steven J. Murdoch and Ross Anderson. 2014. Security Protocols and Evidence: Where Many Payment Systems Fail. In *Financial Cryptography and Data Security*. 21–32.
- [41] Nacha. 2022. The Basics of Authentication in the ACH Network. https://www.nacha.org/system/files/2022-11/The_Basics_of_Authentication_in_the_ACH_Network_2022_Update.pdf. 2022 Update. Accessed: 2026-01-15.
- [42] Nacha. 2025. *2025 Operating Rules : Basic Edition*. Nacha. PDF (Basic Edition rules).
- [43] Nacha. 2025. *2025 Operating Rules : Basic Edition: Appendixes*. Nacha. PDF (Basic Edition appendixes; record formats/return codes/specifications).
- [44] Nacha. 2025. ACH File Details. <https://achdevguide.nacha.org/ach-file-details>. ACH Guide for Developers (Payments Innovation Alliance), ©.
- [45] Nacha. 2025. *Nacha Operating Rules : Basic Edition: Front Matter, Policy Statements, and Formal Interpretations*. Nacha. PDF (Basic Edition front matter).
- [46] Nacha. 2026. ACH Payments Fact Sheet. <https://www.nacha.org/content/ach-payments-fact-sheet>. Accessed: July 29, 2026.
- [47] Nacha. 2026. Nacha's Top 50 ACH Originators and Receivers of 2025 Now Available; Total ACH Payment Volume in 2025 Exceeded 42 Billion. <https://www.nacha.org/news/nachas-top-50-ach-originators-and-receivers-2025-now-available-total-ach-payment-volume-2025>. Accessed: July 29, 2026.
- [48] Mahshid Mehr Nezhad, Feng Hao, Gregory Epiphaniou, Carsten Maple, and Timur Yunusov. 2025. SoK: Security of EMV Contactless Payment Systems. *arXiv preprint arXiv:2504.12812* (2025).
- [49] Charles Olivier-Anclin, Ioana Boureanu, Liqun Chen, Christopher Newton, Tom Chothia, Anna Clee, Andreas Kokkinis, and Pascal Lafourcade. 2025. Who Pays Whom? Anonymous EMV-Compliant Contactless Payments. In *34th USENIX Security Symposium 2025*.
- [50] Andreas Pashalidis and Chris J Mitchell. 2004. Using EMV cards for single sign-on. In *European Public Key Infrastructure Workshop*. Springer, 205–217.
- [51] Paypal. 2026. Paypal. <https://www.paypal.com/us/home>. Accessed: 2026-02-04.
- [52] Plaid. 2026. Plaid Financial Service. <https://plaid.com/>. Accessed: 2026-02-04.
- [53] PNC Bank. 2024. How to Read a Check. <https://www.pnc.com/insights/personal-finance/spend/how-to-read-a-check.html>. Accessed: 2025-10-21.
- [54] Bradley Reaves, Nolen Scaife, Adam Bates, Patrick Traynor, and Kevin R. B. Butler. 2015. Mo(bile) Money, Mo(bile) Problems: Analysis of Branchless Banking Applications in the Developing World. In *24th USENIX Security Symposium*. 17–32.
- [55] B Rowe, Igor D Pokryshevskiy, Albert N Link, and Douglas S Reeves. 2013. Economic analysis of an inadequate cyber security technical infrastructure. *National Institute of Standards and Technology Planning Report* (2013), 13–1.
- [56] U.S. Bank. 2025. U.S. Bank Routing number : What is an ABA routing number? <https://www.usbank.com/bank-accounts/checking-accounts/checking-customer-resources/aba-routing-number.html>. Accessed: 2025-10-21.
- [57] Venmo. 2026. Venmo. <https://venmo.com/>. Accessed: 2026-02-04.
- [58] Sky Pelletier Waterpeace and Nikolay Ivanov. 2026. SoK: Evolution, Security, and Fundamental Properties of Transactional Systems: [Systematization of Knowledge Paper]. Association for Computing Machinery, New York, NY, USA. doi:10.1145/3750555.3811904
- [59] Ming-Hour Yang, Yu-Shan Hsu, and Hsiao-Chuan Hsu. 2025. Enhanced EMV Security: Preventing Credit Card Fraud from a Distance. *IEEE Access* (2025).
- [60] Zelle. 2026. Zelle. <https://www.zelle.com/>. Accessed: 2026-02-04.
- [61] Xianyi Zheng, Lulu Yang, Jiangang Ma, Gang Shi, and Dan Meng. 2016. TrustPAY: Trusted mobile payment on security enhanced ARM TrustZone platforms. In *2016 IEEE Symposium on Computers and Communication (ISCC)*. IEEE, 456–462.

A Open Science

This paper’s core contributions rely on (i) the formal ACH model and property specifications encoded in our model checker, (ii) the counterexample traces that witness the three CWD instances, and (iii) the empirical validation methodology and observation logs.

Formal model and verification artifacts. The formal model of ACH payment processing—including the finite-state machine

definitions, temporal logic property specifications, and nuXmv input files—is available at <https://doi.org/10.5281/zenodo.22701199>[36]. The repository includes the complete model source, property files, and scripts to reproduce all counterexamples reported in the analysis.

Empirical validation. Our empirical evaluation uses real consumer bank accounts owned by members of the research team. Raw account screenshots and transaction logs cannot be shared publicly because they contain personally identifiable financial information. The paper provides the redacted screenshots documenting the user-visible state changes for each CWD instance (A1: creditBLEED Attack, A2: creditMINT Attack, and A3: creditRESET Attack).

Proprietary Data. The models are primarily derived from the Nacha Operating Rules [42, 43, 45], for which Nacha holds the rights. Access to these rules requires registration or purchase, and Nacha maintains the official legal version. We therefore do not include them in the repository. We also drew on several documents and public materials from the anonymized institutions. Due to legal constraints and the anonymization requirements of those institutions, we do not include these materials in the repository.

B Ethical Considerations

Our work raises two primary ethical considerations: the potential harm of conducting vulnerability experiments on live financial infrastructure, and the responsible disclosure of identified weaknesses to affected institutions.

Attack Evaluation. In Section 6, we show that vulnerabilities identified within our sample of evaluated institutions can be exploited to access funds or credits before proper settlement. We conducted all attacks against accounts under our own control to avoid collateral harm to third parties. The banks and third-party payment providers bearing the financial losses were also informed promptly. No third-party accounts were affected at any point during our experiments.

We assess the ethical conduct of our work through two frameworks proposed for computer security research. From a consequentialist view, the benefit of exposing the vulnerabilities identified in our evaluation outweighs the minimal, self-contained risk incurred during testing. From a deontological view, confining experiments to accounts under our control ensures no third party is subjected to harm without consent, satisfying our duty of care to others.

Disclosure. Disclosure protects users of financial infrastructure who may be targeted through the vulnerabilities we identified and helps affected institutions address the identified weaknesses. On April 16, 2025, we submitted disclosures to each relevant institution (banks and payment providers) through their official responsible disclosure portals, providing detailed reproduction steps, proof-of-concept demonstrations with screenshots, and technical reports. We later provided additional evidence, and received confirmation that our reports had been received and were under internal review. By May 2025, A1 and A3 were no longer viable, though A2 remained viable. As of December 2025, A2 was still exploitable through one affected third-party payment provider, and by January 2026, all affected payment providers had resolved A2.

We did not receive detailed technical responses about remediation approaches. The acknowledgment of vulnerabilities and the

behavioral changes observed in several systems following our disclosure are consistent with active remediation efforts. The absence of detailed technical engagement reflects a broader challenge in financial-sector vulnerability disclosure where legal constraints limit communication.

C Modeling Payment Batch Processing

Section 4.5 defines batch membership as a transaction-state predicate and models batch transmission as a synchronized transition at collection-window closure. This appendix extends that definition with the operational stages that occur before and after the atomic transmission step. Under Nacha Article 2, Subsection 2.2.1, an ODFI does not send each payment entry as soon as it arrives. Entries instead accumulate during a collection¹ window and are sent together when the window closes or ends.

Collection-window lifecycle. Two Boolean flags control the window lifecycle: *batch_started* signals window opening, while *batch_expired* signals closure. ODFIs close the collection window at predetermined intervals, typically once per business day, and send all accumulated transactions in one batch. Transactions submitted after the window closes stay in *submitted* until the next collection window closes.

Transaction flow during collection window. During the collection window, Originators O^i submit payment entries that move from *not_submitted* to *submitted*. For each entry, the ODFI BO validates it against Nacha data-format rules. Valid entries stay in *submitted* until batch transmission, and invalid entries move at once to *rejected*. When *batch_expired* becomes true, every *submitted* transaction moves atomically to *batched*.

ACH operator processing. After batch transmission, the model covers the ACH Operator clearing and routing steps. Transactions that pass the Operator’s data-acceptance checks move from *batched* to *pending_settlement*, which marks readiness for RDFI handling. Entries that fail validation are rejected or returned under the applicable Nacha return reason codes.

RDFI processing and error remediation. RDFI handling of *pending_settlement* transactions is modeled non-deterministically so the model can cover multiple outcomes from account state and exception cases. RDFI handling is not atomic like batch transmission: it depends on account-level state and can end in settlement, return, or other dispositions under Nacha Article 4.

The model includes reversals, which let Originators correct erroneous entries. A reversal introduces a compensating transaction e_{rev} with an offsetting amount:

$$\text{Reverse}(e_i) \Rightarrow \exists e_{rev} : \text{amount}(e_{rev}) + = -\text{amount}(e_i)$$

The system enforces temporal constraints so reversals occur within five banking days after settlement, as Nacha requires. Batch headers for reversal entries must carry the literal label *REVERSAL*, as the Nacha Operating Rules require.

¹A collection window is a fixed period during which transactions are accumulated before being processed as a batch.