

# Security of Electronic Payments Services



**Erik Poll**

**Digital Security**

**Radboud University Nijmegen**

# Overview

## Some anecdotes & trends in e-banking fraud

- **skimming**
- **EMV (het nieuwe pinnen)**
- **online banking**
- **contactless payments**

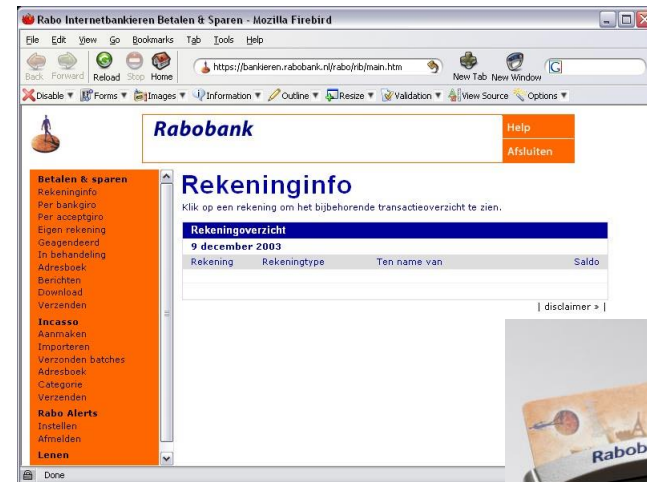
**incl. some of our own research**

**on more rigorous design and analysis**

**Joint work with PhD students Joeri de Ruiter and Fides Aarts, and MSc students Arjan Blom, Jordi van den Breekel, Georg Chalupar, Anton Jongsma, Robert Kleinpenning, Peter Maandag, and Stefan Peherstorfer.**

# Why look at electronic payments?

- Systems you are all familiar with
  - Obvious & longstanding target for cyber-criminals
- ⇒ long history of attacks & defences that we can learn from



# Skimming

# Skimming

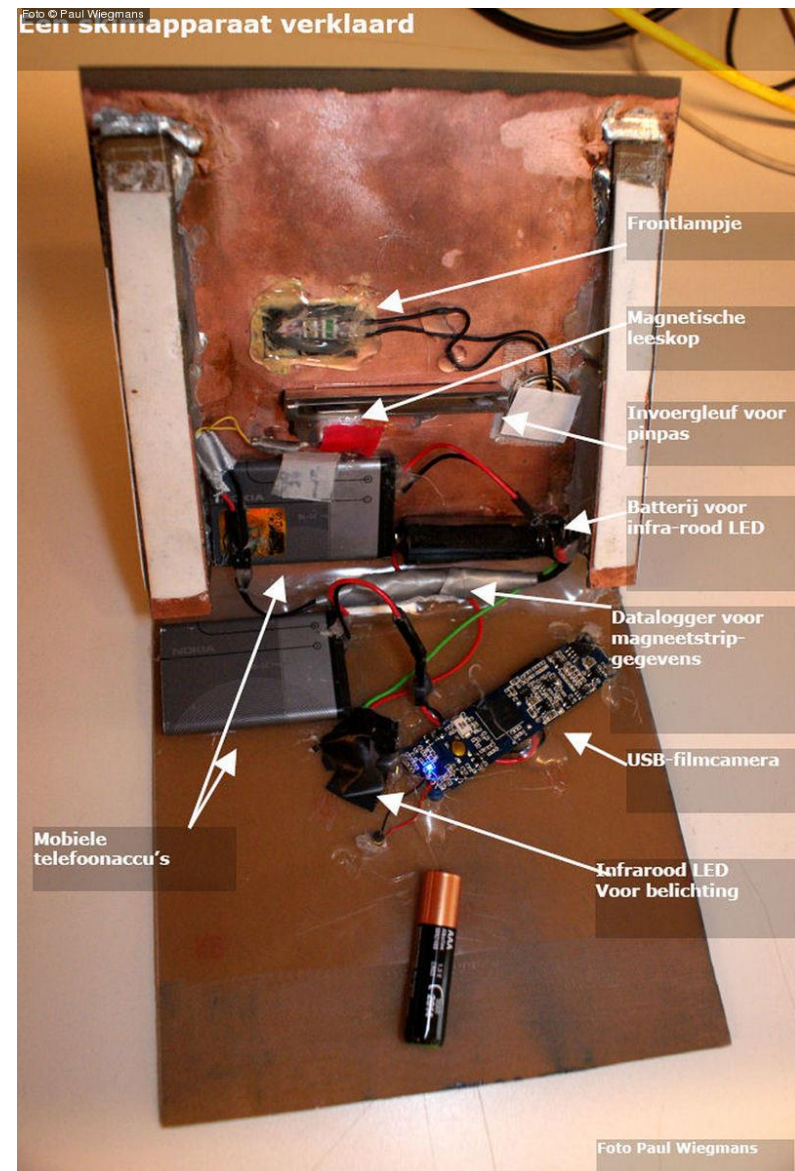
Mag-stripe on bank card contains digitally signed information



but... this info can be copied



# Example skimming equipment



# Skimming fraud in the Netherlands

2007 : 15 M€

2008 : 31 M€

2009 : 36 M€

2010: 19.7 M€ - better detection

2011: 38.9 M€

On a total of over 100 billion €, so fraud only around 0.03%

Hence migration to EMV (chip) cards moved forward from 2013 to 2011



# Does EMV reduce skimming?

- UK introduced EMV in 2006

	2005	2006	2007	2008
domestic	79	46	31	36
foreign	18	53	113	134

Skimming fraud with UK cards, in millions £

- Magstripes that are cloned can still be used in countries don't use the chip...
- Blocking cards for use outside EU (**geoblocking**) helps a lot!
- Skimmers have now moved to the US, and the US is (slowly) migrating to EMV



# Skimming fraud in the Netherlands

2007 : 15 M€

2008 : 31 M€

2009 : 36 M€

2010: 19.7 M€

2011: 38.9 M€

2012: 29 M€

2013: 6.8 M€

2014: 1.3 M€

2015: 1.7 M€

Migration to EMV in 2011, introduction geo-blocking in 2012


# Using a chip instead of a magstripe: EMV (Europay-MasterCard-Visa)



# EMV (Europay-MasterCard-Visa)

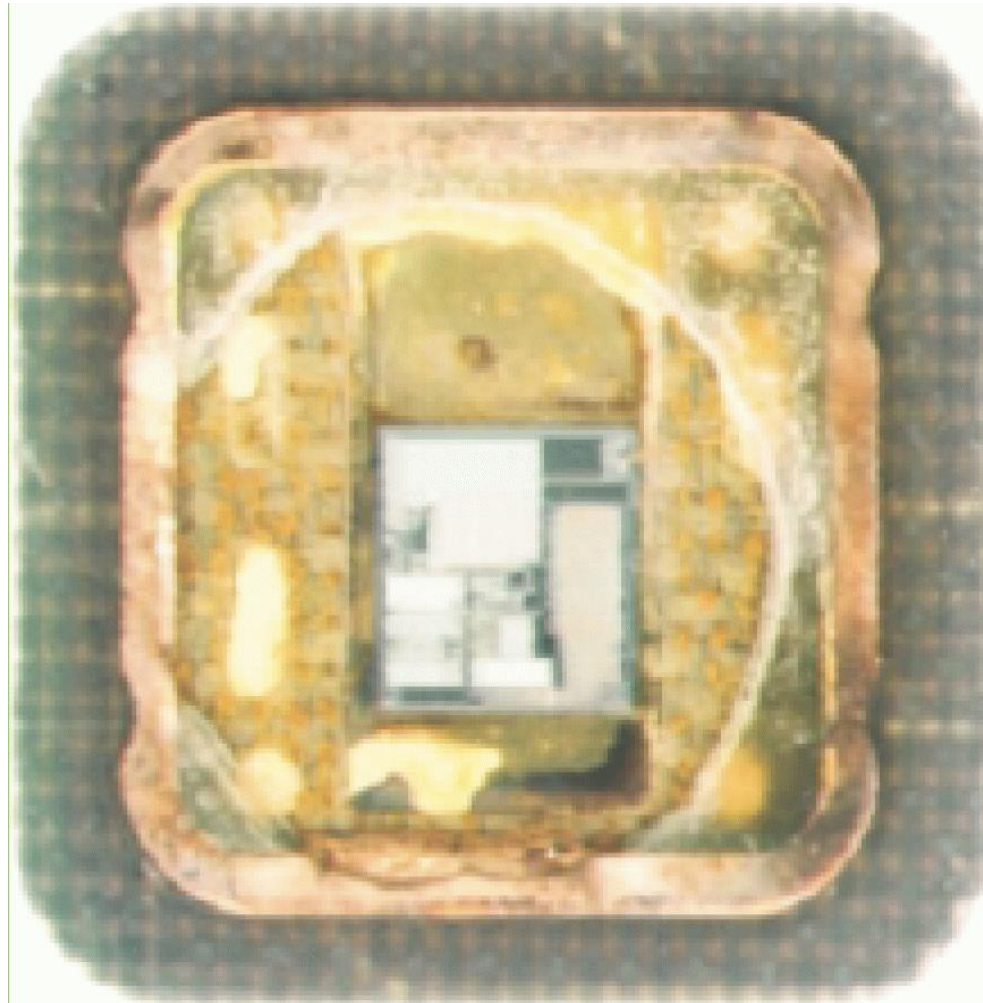
- Standard used by most chip cards for banking
- Specs controlled by  which is owned by



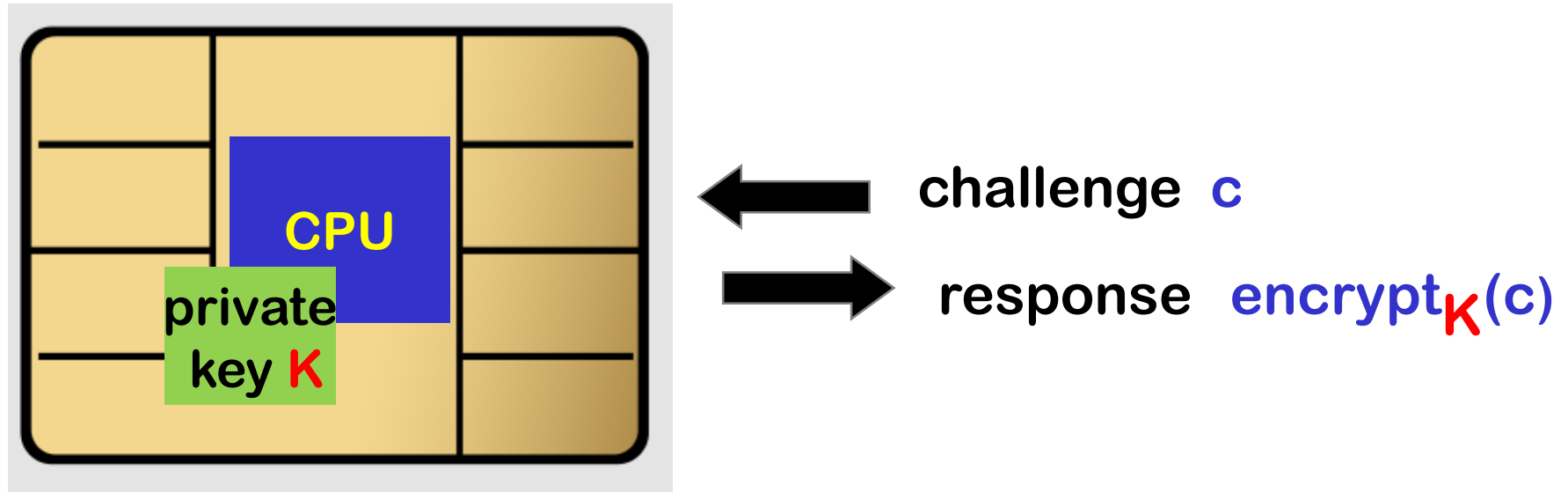
- Contact  and contactless version ))))
- The protocol makes cloning chips by eavesdropping impossible



# Unpacked smartcard with chip exposed



# Challenge-response protocol for authentication



The smartcard authenticates by proving it knows the secret key  $K$ , without revealing that key

=> key cannot be *eavesdropped*, like a password can be

=> response of the card cannot be *replayed*

The **card issuer** does not have to trust the **network**, the **terminal**, or the **card holder**

# Skimming 2.0

- In 2009, criminals put **tampered card readers** *inside* ABN-AMRO bank branches to skim cards
  - For **backwards compatibility**, the **chip** can report the **magstripe** data...
  - Both magstripe data and PIN code are sent plaintext from card to the reader
  - Criminals caught & convicted in 2011
- Cards have been improved to avoid this, and magstripe data should now be different from info on the chip



# Problem: complexity

EMV is not a single protocol, but a 'protocol toolkit suite' with *lots* of configuration options

- Original EMV specs : 4 books, > 700 pages
  - 3 types of cards (SDA, DDA, CDA), 5 authentication mechanism (online PIN, online PIN, offline encrypted PIN, signature, none), 2 types of transactions (offline, online), ....
- Additional EMV contactless specs: another 10 books, > 2000 pages
  - yet more modes and options....

## Sample sentence

“If the card responds to GPO with SW1 SW2 = x9000 and AIP byte 2 bit 8 set to 0, and if the reader supports qVSDC and contactless VSDC, then if the Application Cryptogram (Tag '9F26') is present in the GPO response, then the reader shall process the transaction as qVSDC, and if Tag '9F26' is not present, then the reader shall process the transaction as VSDC.”

# Complexity: example protocol flaw

Terminal can choose to do **offline PIN**

- ie. **terminal asks the card to check the PIN code**

The response of the card ('this PIN code is OK') is **not authenticated**

- ie. it is not cryptographically signed or MAC-ed

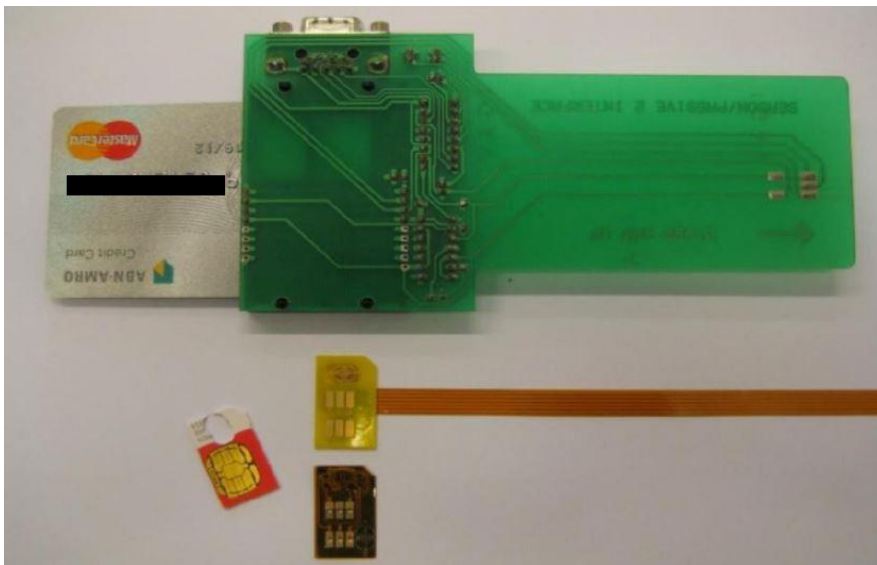
=> the terminal can be fooled by a **Man-in-the-Middle attack**

The transaction data will reveal the transaction was PIN-less, so the bank back-end will later know the PIN was *not* entered

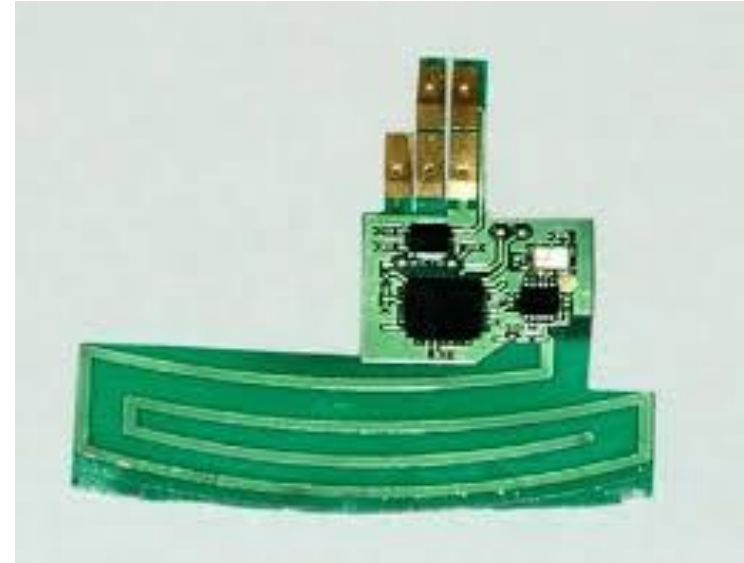
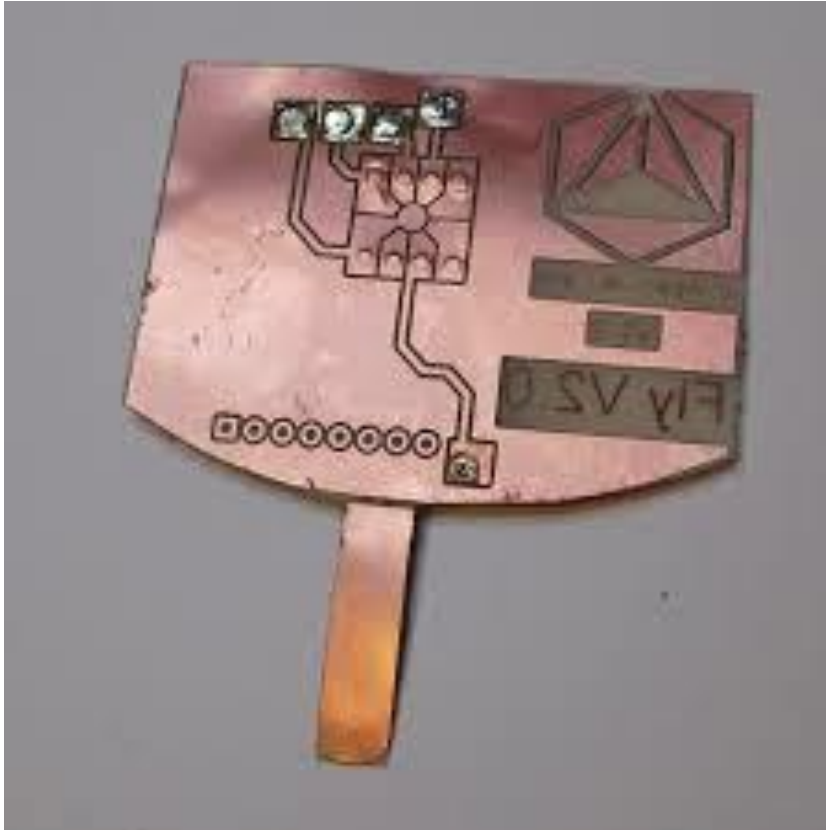
[Stephen Murdoch et al., *Chip & PIN is broken*, FC'2010]



# Our Man-in-the-Middle set-up

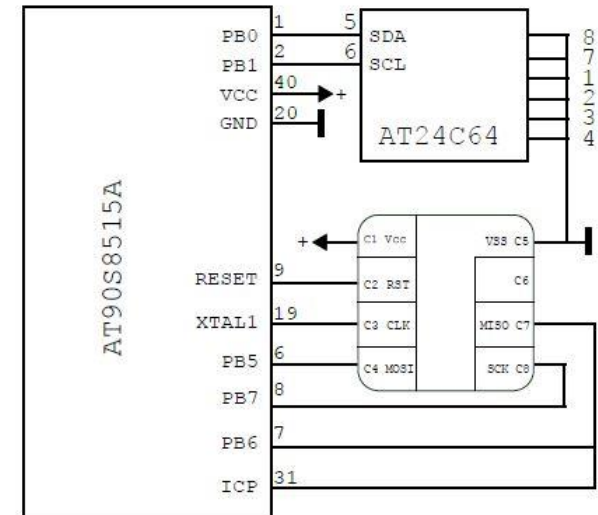
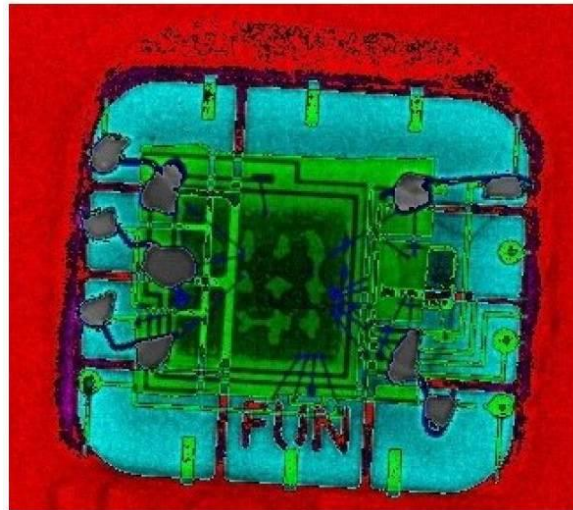
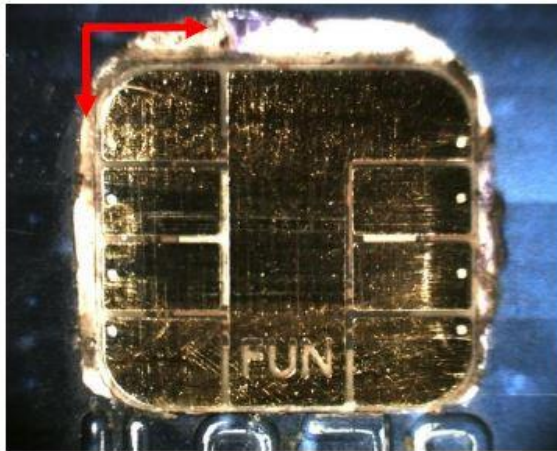


# More 'professional' equipment



# Criminal Man-in-the-Middle set-up

Chips from stolen cards inserted under another chip, which faked the PIN OK response



xray reveals  
green stolen chip under  
blue microcontroller

[Houda Ferradi et al., *When Organized Crime Applies Academic Results: A Forensic Analysis of an In-Card Listening Device*, Journal of Cryptographic Engineering, 2015]

# Complexity: example configuration flaw

Configuration mistake on the first generation contactless cards issued in the Netherlands:

- functionality to check the PIN code, which should only be accessible via the contact interface was also accessible via the contactless interface )))



Possible risk for DoS attacks, rather than financial fraud?

Flaw discovered by Radboud students Anton Jongsma, Robert Kleinpenning, and Peter Maandag.

# Complexity of EMV specs

- Moral of the story: specs too complex to understand
  - long documents
  - little or no discussion of security goals or design choices
  - little abstraction or modularity
- Who really takes responsibility for ensuring these specs are secure? EMVCo, the credit card companies behind EMVCo, or individual banks?
- Can we provide some scientific rigour?

# Formalisation of EMV in F#

```
open Crypto
open Data
open System
open F#

let sk = rsa_keygen ()
let pk = rsa_pub sk

// ICC Application Cryptogram Master Key
let MAC = hex_keygen ()

// Checksum between card and terminal
let uric = "00000000000000000000000000000000"
let addressC = HexToURic uric

// Events used in queries
type event =
| TransactionInit of bool * bool * bool
| TransactionFinal of bool * bool * bool
| CardSuccess of bool
| TermSuccess of bool
| TermInit of bool
| TermFinal of bool
| TermInitDA of bool
| TermFinalDA of bool

let tr : event list = []

// Create the ISO element for the card
let card_create_sdad ssp =
  ra.ssp s1 (ASDU.sspByBytes ssp)
let card_create_certificate sk data =
  ra.encrypt sk data

// Construct AC message
let construct_ac ssp ssk data cbs_enabled =
  // If DA is enabled, add a signature over the data in the AC
  if cbs_enabled then
    ra.sign ssk data
  else
    []

let construct_ac_mac data =
  ra.encrypt MAC data

// Perform the actual transaction
let card_transaction (c, atc, (PIC, PIC)) d pdd1 ssp force_online =
  let (ac_type, cbs_requested, cbs12) = ASDU.parse_generate_ac d in
  let (mac, cvr, nonce) = cdb11 in
  let mac = construct_ac mac (mac, nonce, atc) in
  if ac_type = Data.AAC then
    begin
      let send c (ASDU.generate_ac_response Data.AAC atc mac (construct_ac_sig ssk (Data.AAC, atc,
        pdd1, cdb11, mac)))
      let (pin, cbs_requested, cbs12) = ASDU.parse_generate_ac (Net.rev c) in
      let send c (ASDU.generate_ac_response Data.E atc mac (construct_ac_sig ssk (Data.TC, atc, pdd1,
        cdb11, cdb12, mac)))
      let send c (ASDU.generate_ac_response Data.AC atc mac (construct_ac_sig ssk ()))
      failwith "Unexpected command"
    end
  elif ac_type = Data.E then
    begin
      if force_online then
        let send c (ASDU.generate_ac_response Data.RAC atc mac (construct_ac_sig ssk (Data.RAC, atc,
        pdd1, cdb11, mac)))
      let (ac_type, cbs_requested, cbs12) = ASDU.parse_generate_ac (Net.rev c) in
      if ac_type = Data.TC then
        let send c (ASDU.generate_ac_response Data.TC atc mac (construct_ac_sig ssk (Data.E, atc,
        pdd1, cdb11, mac)))
      // Receive RRR
      let pdd = ASDU.parse_select_application_response (Net.rev c) in
      let pdd_items = [] in
      // Get processing options
      let send c (ASDU.get_processing_options pdd_items)
      let (agp, ar) = ASDU.parse_get_processing_options_response (Net.rev c) in
      let (cbs_enabled, cbs_enabled, cbs_enabled) = ssp in
      // Read files
      let send c (ASDU.read_record)
      let (sdat, cert) = ASDU.parse_read_record_response (Net.rev c) in
      // Perform DA authentication if this is the highest supported authentication method
      if cbs_enabled = false then
        if cbs_enabled = false then
          begin
            let result_sda = ra.verify_no_fail pk (ASDU.sspByBytes ssp) sdat in
            log tr (Nothing)
          end
          // No DA method supported
          log tr (Nothing)
        else
          log tr (Nothing)
      else
        if cbs_enabled = false then
          begin
            // Perform DA authentication if this is the highest supported authentication method
            let (pic, ssp, ssk) = ra.decrypt_pk_cert in
            let result_sda = ra.verify_no_fail pic (ra.mac (ac_type, atc, (pdd1_items, cdb11)) signature) in
            log tr (TerminalCDM(result_sda))
          end
          log tr (Nothing)
        else
          begin
            // Construct transaction
            log tr (Nothing)
            and
            elif ac_type = Data.AAC then
              begin
                // Short transaction
                log tr (Nothing)
              end
            else
              failwith "Unexpected AC type"
            end
            elif ac_type = Data.AAC then
              log tr (Nothing)
            elif cbs_enabled = false then
              log tr (Nothing)
            else
              failwith "Unexpected AC type"
            log tr (TransactionInit(cbs_enabled, cbs_enabled, cbs_enabled))
          end
        end
      end
    end
  end
end

// Perform PIN verification
if pin_enabled then
  begin
    let pin = utf8 (str "1234") in
    let send c (ASDU.verify_pin)
    let response = ASDU.parse_verify_response (Net.rev c) in
    log tr (TerminalSuccess(response))
  end
  log tr (Nothing)
end

// Perform the actual transaction
let nonce = addressC () in
let cdb11 = (mac, cvr, nonce) in
let cdb12 = (vtr) in
// DA is performed if this is supported
if online_enabled then
  let send c (ASDU.generate_ac Data.AAC cbs_enabled cdb11)
  else
    let send c (ASDU.generate_ac Data.TC cbs_enabled cdb11)
end

// Construct Application Transaction Counter
let atc = addressC () in
// Generate event for initialization of transaction
log tr (TransactionInit(cbs_enabled, cbs_enabled, cbs_enabled))
// GET PROCESSING OPTIONS command
let pdd1 = ASDU.parse_get_processing_options (Net.rev c) in
// Send response with ASP and AFV
let send c (ASDU.get_processing_options_response ssp ar)
// Send response
let send c (ASDU.read_record_response (card_create_sdad ssp), (card_create_certificate sk (PIC, ssk),
  (sp)))
// Perform DA if enabled
let msg = result_sda (c, atc, (PIC, PIC)) in
log tr (Nothing)
// Perform PIN verification if requested
let msg = card_pin_verify (c, atc, (PIC, PIC)) msg in
// Perform the actual transaction
card_transaction (c, atc, (PIC, PIC)) msg pdd1 ssp force_online
log tr (Nothing)

let card () =
  // Set up channel between card and terminal
  let c = Net.listen addressC in
  let sk = rsa_keygen () in
  let pk = rsa_pub sk in
  #if F#
  let (cbs_enabled, cbs_enabled, cbs_enabled) = (concat (Net.rev c) in
  let (cbs_enabled, cbs_enabled, cbs_enabled) = (Bytes2bool sdat_enabled,
  bytes2bool cbs_enabled) in
  #else
  let (cbs_enabled, cbs_enabled, cbs_enabled) = (Net.rev c) in
  #endif
  (* card process (PIC, PIC) c (cbs_enabled, cbs_enabled, cbs_enabled) *)
  #if F#
  #endif
  let terminal () =
    // Set up channel between card and terminal
    let c = Net.connect addressC in
    #if F#
    // Get card options from network
    let (pin_enabled, online_enabled) = (concat (Net.rev c) in
    let (pin_enabled, online_enabled) = (Bytes2bool pin_enabled, bytes2bool online_enabled) in
    #else
    let (pin_enabled, online_enabled) = (Net.rev c) in
    #endif
    // Let (pin_enabled, online_enabled) = (false, true) in
    // Create T00 Book 3, (RRR)
    let cvr = ssk cvr in
    // Create DA
    let cvr = ssk cvr in
    // Initialize transaction dependent values
    let amount = addressC () in
    let terminal_currency_code = "9020" in // Netherlands
    let transaction_currency_code = "9010" in // Euro
    // Select application
    let send c (ASDU.select_application)
  end
end
```

Essence of protocol in functional programming language F#



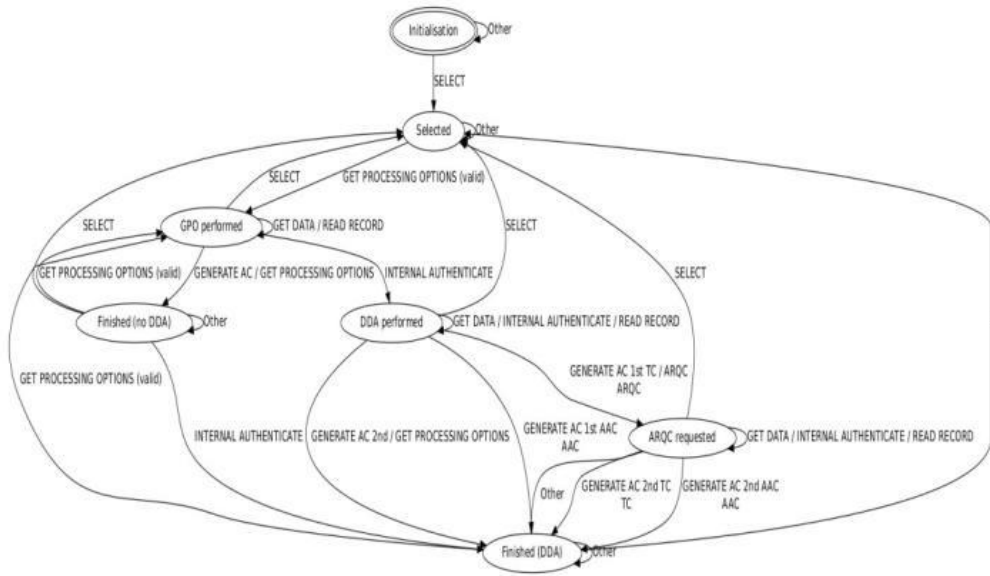
# Formal Analysis of EMV

- Essence of EMV (all variants!) can be formalized in less than 700 lines of F# code
- This model be analysed for security flaws using **ProVerif** tool
- No new attacks found, but existing attacks inevitably (re)discovered

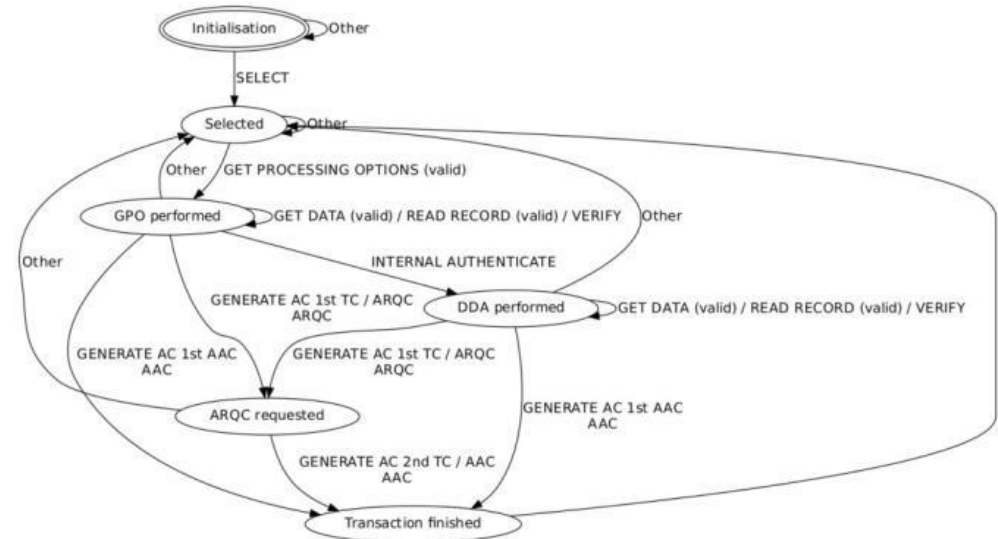
[Joeri de Ruiter and Erik Poll, *Formal Analysis of the EMV protocol suite*, TOSCA 2012]

This still leaves the question if the software implementing these standards is correct!

# State machine inference: automated testing



Volksbank   
implementation



Rabobank   
implementation

We can automatically infer the state machine of an EMV smartcard, using only black-box testing, in 30 minutes.

No security flaws found, but lots of differences between cards!

[Fides Aarts et al., *Formal Models of bank cards for free*, SECTEST 2014]

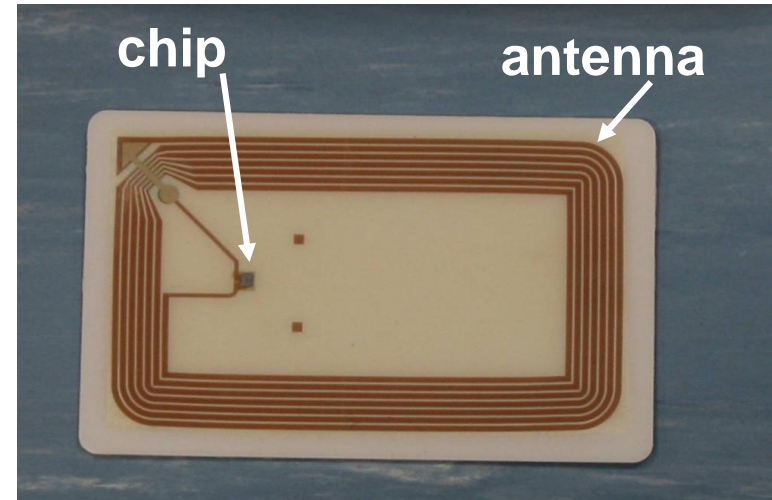


# Contactless & mobile payments



# RFID & NFC

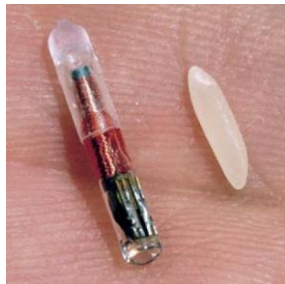
- **RFID (Radio Frequency Identification)** aka contactless smartcard  
smartcard chip with an antenna



- **NFC (Near Field Communication)**  
compatible standard for mobile phones



# Other RFID uses



# Contactless has security disadvantage: *eavesdropping*

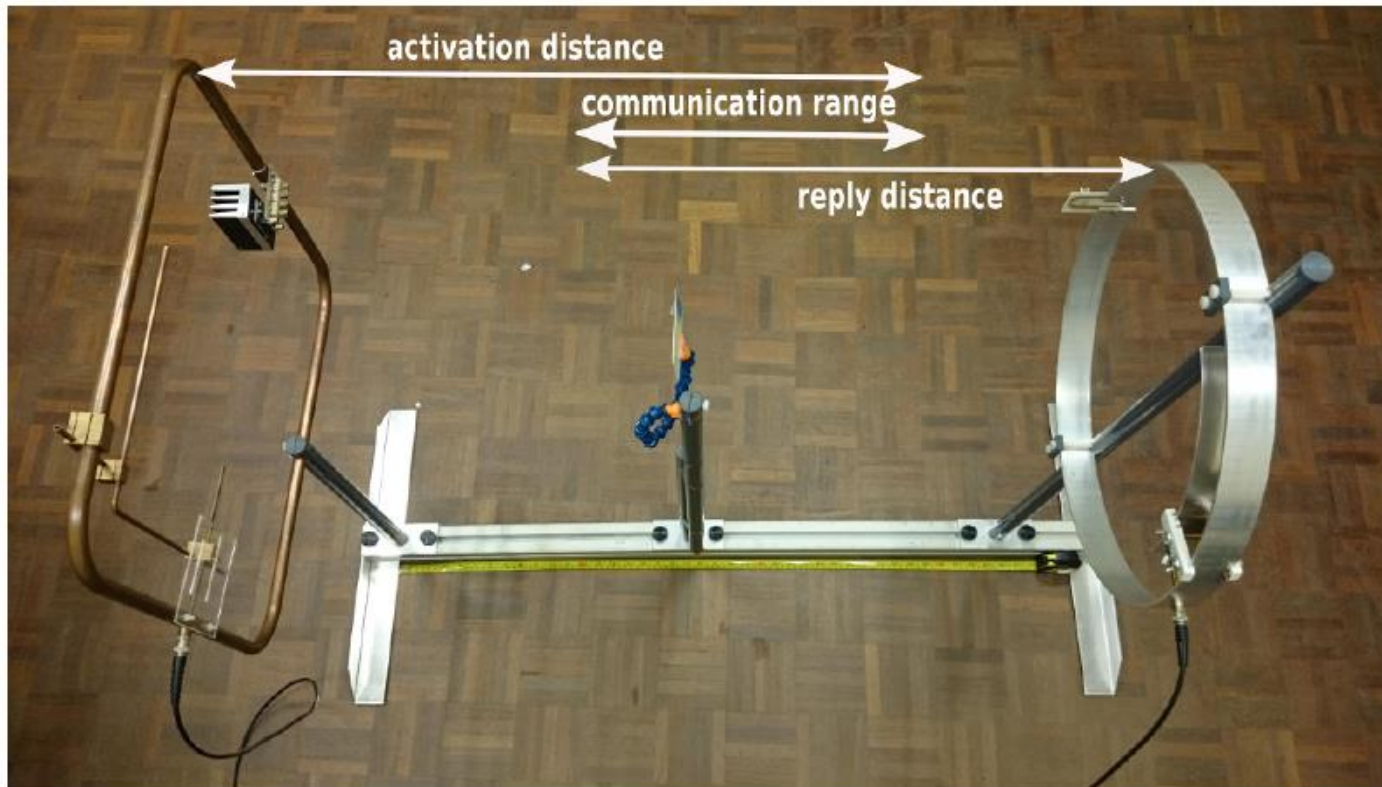
## passive attacks

- **eavesdropping** on communication between passport & reader
- possible from **many meters** if card is held to normal reader

## active attacks

- **unauthorised access** to tag without owner's knowledge
- possible up to **≈50 cm**
  - activating RFID tag requires powerful field!
- aka **virtual pickpocketing**  
incl. **relay attack**

# Our powerful antenna



**Bank card can be used at 50 cm max,  
but width of gate at 80 cm works better**

**[Rene Habraken et al., An RFID Skimming Gate Using Higher Harmonics, RFIDSec 2015]**

# Are relay attacks a risk?

- No interesting criminal business model?
  - attacker runs risk of being caught
  - attacker can only steal 25 euro max
  - attacker has to buy something (eg buy chocolate bar at vending machine),
  - or get the money in his own bankaccount, but then he can be traced
- The impact of someone spying to see you PIN and then stealing your bank card is much greater than digital pickpocketing!

# Online banking



# Internet banking fraud in Netherlands

2008	2.1 M€
2009	1.9 M€
2010	9.8 M€ (7100€ per incident)
2011	35 M€ (4500€ per incident)
2012	34.8 M€
2013	9.6 M€
2014	4.7 M€
2015	3.7 M€

[Source: NVB & Betaalvereniging]

- Better detection of suspicious transactions
- Better detection of money mules



# Strong (2-factor) Authentication



- Authentication with hand-held card reader using EMV bank card and PIN code, as second factor
  - intended use: **internet banking** and **online shopping**
  - specification (by MasterCard) secret but reverse-engineered




- Some silly technical flaws, eg sending a fixed challenge 000000 to the smartcard instead of the random number the user types in
- Remaining problems: still prone to **phishing attacks** (eg by phone) and **Man-in-the-Browser attacks**

# e-banking using EMV-CAP



This reader can be trusted.  
But can the user understand  
the meaning of these numbers?

Computer display of  
*cannot* be trusted  
(despite )



→ 23459876  
← 123654

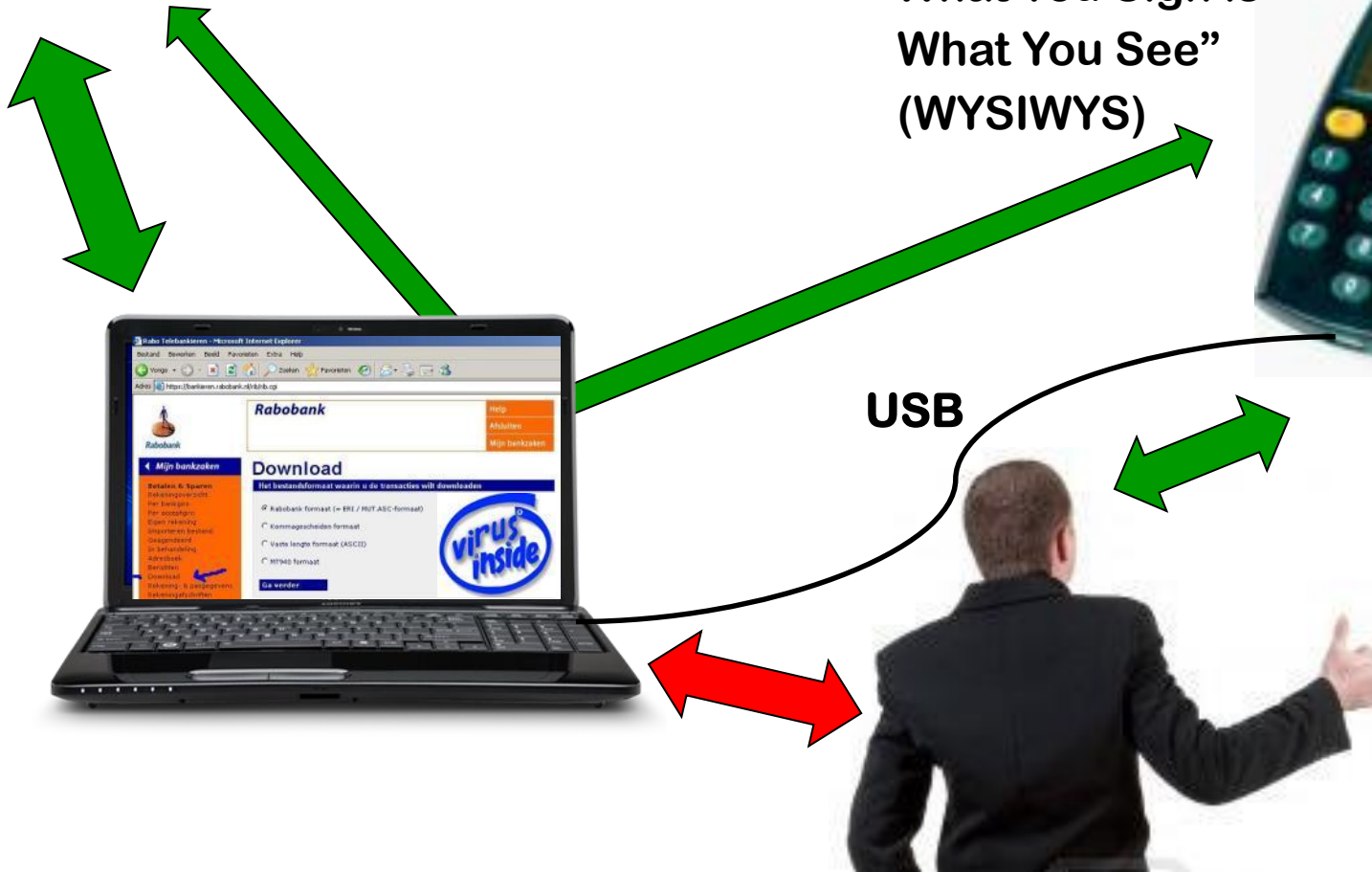


# e-banking using USB-connected e.dentifier



This display can be trusted & understood

“What You Sign is What You See” (WYSIWYS)



# Flaw in USB-connected e.dentifier2

It's possible to press the OK button via the USB cable...

So malware on an infected PC could change all the transaction details and press OK!

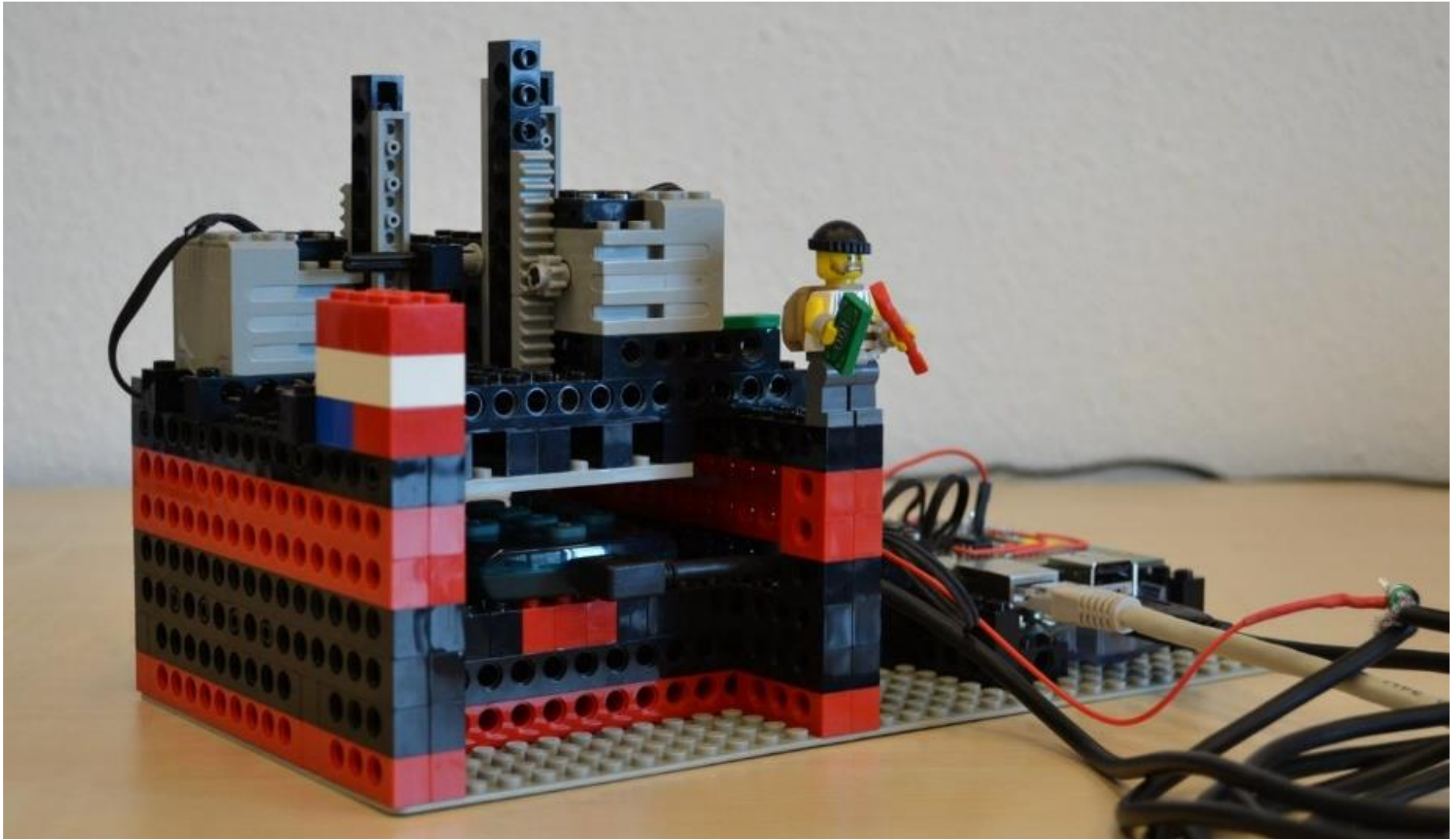
Flaw found with manual analysis.

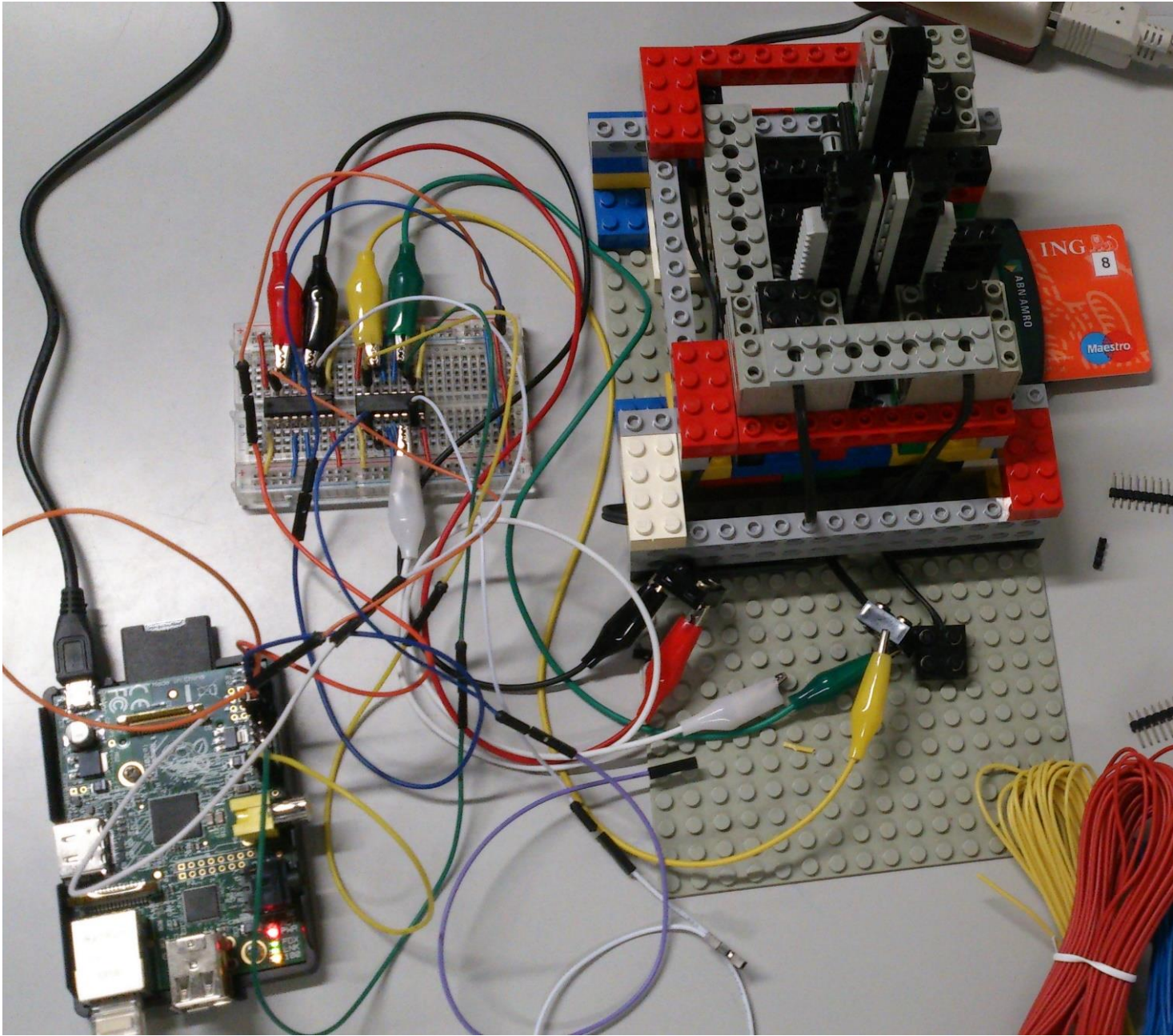
Could we automate this?

[Arjan Blom et al., *Designed to Fail: A USB-Connected Reader for Online Banking*, NordSec 2012]



# Our Lego hacker



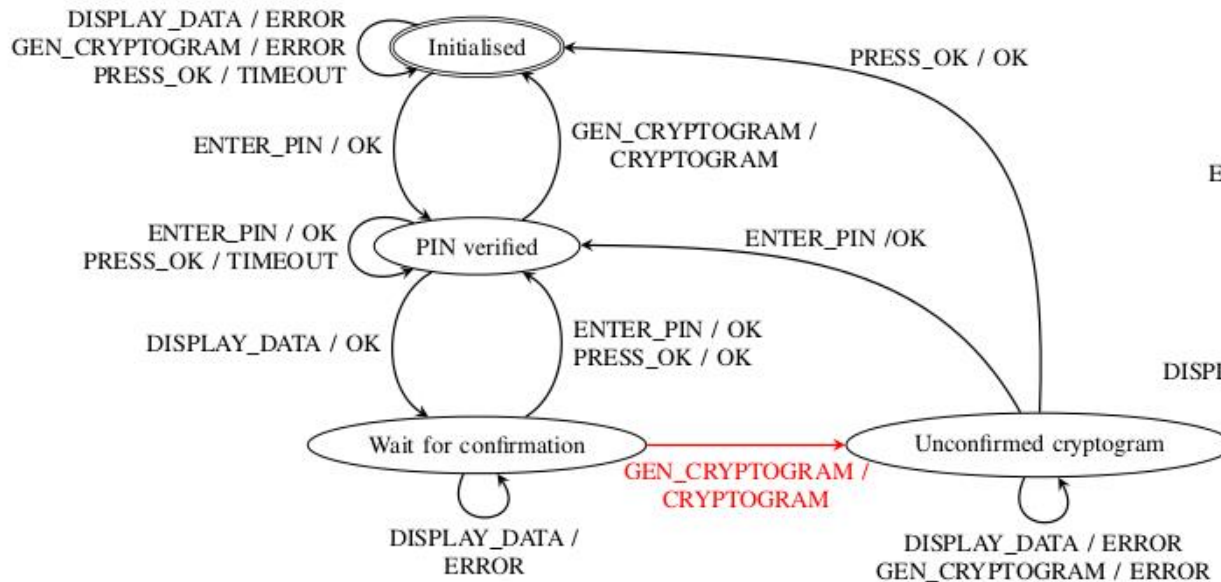


# Our Lego hacker

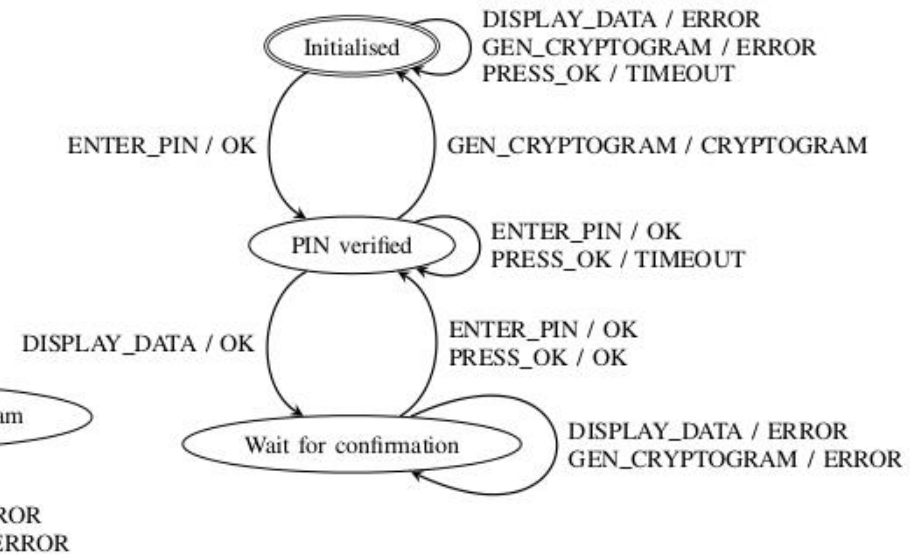


# Automatic reverse engineering using Lego

State machines automatically inferred by our Lego robot



state machine of old, flawed device

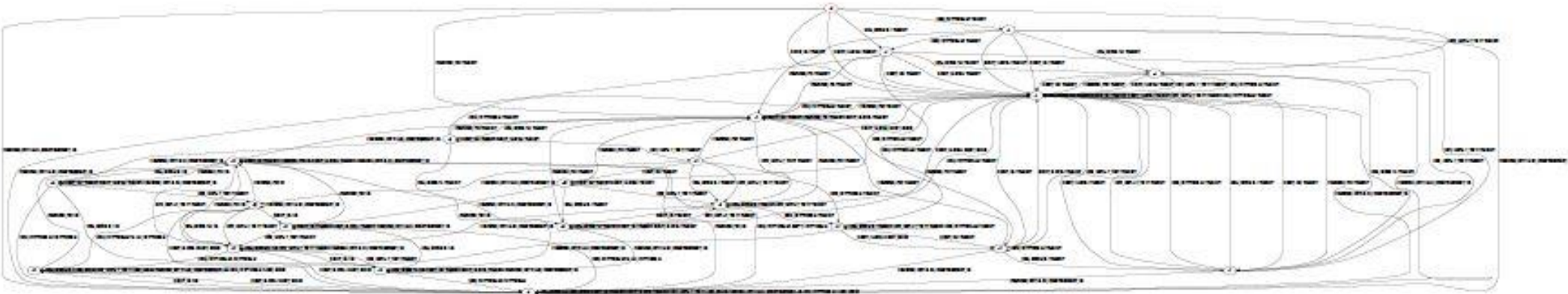


state machine of new device

[Georg Chalupar et al., *Automatic reverse engineering using Lego*, Workshop on Offensive Technologies, WOOT 2014]



# Aaargh!



full state machine inferred for new, fixed e.dentifier2

Do you think the designer of this protocol and the person who implemented it are confident that it is secure?



# Conclusions



# Cyber criminals

- Not script kiddies & hobbyists, but **creative, skilled, well-funded branch of organized crime**
- Cyber crime is increasingly professional
  - trend: *cyber crime as a service*

# Technology vs business models

- A technical weakness in a system (a 'hack') need not be a (big) problem:  
  
namely, if there is no **good & scalable criminal business model**
- Example bottleneck in e-banking fraud business model:  
**Recruiting money mules for bank accounts to receive stolen money**
- Example highly successful business model:  
**Ransomware**

# Trend: prevention -> detection & reaction

- Instead of trying to *prevent* problems, trying to *detect* and *respond* to problems may be more (cost) effective way to improve security.
- Example: breaking into a Dutch house, which huge glass windows on the ground floor, is trivial. Only the risk of *detection* and the *reaction* then (ie. getting caught) is deterring criminals.
- Example: banks have combatted skimming fraud & online banking fraud with better detection.

Note: this is often related to making the criminal business model less attractive.

# Assurance? Cover-Your-Ass security?

*Assessment & assurance of security is really hard!*

## 1. How do you assess the risk?

And then assure security accordingly?

risk = probability x impact

## 2. Who is *really* taking responsibility for doing this?

- Individuals in organisations are often just interested in covering their own ass... (And rightly so, from their own point of view!)

Example: Who is assessing the security of e-payment solutions?

The banks? Their suppliers? The scheme owners Mastercard and Visa? EMVCo? The regulators, eg. the European Central Bank (ECB)?  
*Or do they all assume someone else will?*

# Why e-banking security is an easy problem!

1. **Fraud with e-banking is easy to *measure*!**
  - => *trends* in cybercrime are easy to spot!
  - => **economic decisions** to invest in security easy to justify
2. There is an obvious & motivated party to take these decisions, namely the bank
  - though there is the risk of **liability shift** to customers, and the risk of Cover-Your-Ass security decisions

In many other settings these points do not hold! Eg: if a bank has 3 million euro fraud with e-banking, it knows what to spend on better security. But how much can can the security of say electronic health records or the electricity grid cost?

- More generally: **many security problems persists because of economic disincentives**

# Conclusions

- Cyber crime is a highly professional & well-funded branch of organised crime
- Scalable criminal business models may be more dangerous than technical security flaws
- Detection & reaction may be more useful than prevention
- Security assessment is hard!
  - The only way to do it : **think like an attacker & try to hack the system.**
  - But: Who can & wants to do it? Who is interested & economically motivated in improving security?

*Thanks for your attention! Questions?*