Improving PIN Processing API Security

R. Focardi and F. Luccio
Università di Venezia, Italy
focardi@dsi.unive.it

G. Steel
LSV, CNRS & ENS de Cachan, France
graham.steel@lsv.ens-cachan.fr

Abstract

We propose a countermeasure for a class of known attacks on the PIN processing API used in the ATM (cash machine) network. This API controls access to the tamper-resistant Hardware Security Modules where PIN encryption, decryption and verification takes place. The attacks are differential attacks, whereby an attacker gains information about the plaintext values of encrypted customer PINs by making changes to the non-confidential inputs to a command. Our proposed fix adds an integrity check to the parameters passed to the command. It is novel in that involves very little change to the existing ATM network infrastructure.

1 Introduction

In the international ATM (cash machine) network, users' personal identification numbers (PINs) have to be sent encrypted from the PIN entry device (PED) on the terminal to the issuing bank for checking. Issuing banks cannot expect to securely share secret keys with every cash machine, and so the PIN is encrypted under various different keys as it passes through the network. Typically, it will first be encrypted in the PED under a key shared with the server or *switch* to which the ATM is connected. The PIN is then decrypted and re-encrypted under the key for an adjacent switch, to which it is forwarded. Eventually, the PIN reaches a switch adjacent to the issuing bank, by which time it may have been decrypted and re-encrypted several times. The issuing bank has no direct control over what happens in the intermediate switches, so to establish trust, the internationally agreed standards ANSI X9.8 and ISO 9564 stipulate the use of tamper proof cryptographic hardware security modules (HSMs). In the switches, these HSMs protect the PIN encryption keys, while in the issuing banks, they also protect the PIN derivation keys (PDKs) used to derive the customer's PIN from non-secret validation data such as their personal account number (PAN). All encryption, decryption and checking of PINs is carried out inside the HSMs. To this aim, the HSMs have a carefully designed API providing functions for *translation* (i.e., decryption under one key and encryption under another) and *verification* (i.e. PIN correctness checking). The API has to be designed so that even if an attacker obtains access to the host machine connected to the HSM, he cannot abuse the API to obtain customer PINs.

In the last few years, several attacks have been published on the APIs in use in these systems [5, 6, 8]. Very few of these attacks directly reveal the PIN. Instead, they involve the attacker calling the API commands repeatedly with slightly different parameter values, and using the results (which may be error codes) to deduce the value of the PIN. High-profile instances of many PINs being stolen from a hacked switch has increased interest in the problem [1]. Recently, a Verizon Data Breach report and a subsequent article in the press confirmed publicly for the first time that PINs are being extracted from HSMs on a wide scale [3, 2].

PIN recovery attacks have been formally analysed before, but previously the approach was to take a particular API configuration and measure its vulnerability to combinations of known attacks [10]. In recent work, we proposed an extension to language based information flow analysis to take account of cryptographic primitives designed to assure data integrity, in particular MACs [7]. We showed how PIN processing APIs could be extended with MACs to counteract differential attacks, and showed how this revised API type-checked under our framework. However, that work was rather theoretical, and did not attempt to explain how our proposal could be put into practice. In particular, for our proposal to be

feasible in the short term, it needs to be adapted to take into account the constraints of the existing ATM infrastructure. In this paper we outline what we believe to be a practical scheme. We assess its impact on security and the amount of changes that would be required to the ATM network to put it in to practice.

We will not review the operation of PIN processing APIs in detail here. For understanding the abstract, it suffices to know that the attacks are caused by an attacker making illegitimate queries to the API, 'tweaking' the bits of the non-confidential parameters such as the customer's PAN and the *decimalisation table* (dectab). Interested readers are referred to existing literature for more details [5, 6, 8, 10].

2 The Fix

In another paper [7], we show how differential attacks can be countered by the use of MACs, which prevent the intruder from making arbitrary queries to the verification and translation APIs. Only queries whose parameters match the supplied MAC are processed. However, the infrastructure changes needed to add full MAC calculation to ATMs and switches are seen as prohibitive by banks [4]. We propose here a way to implement a weaker version of our scheme whilst minimising changes to the existing infrastructure. We lose some security, since our MACs now have an entropy of only 5 decimal digits ($2^{16} < 10^5 < 2^{17}$). We assess the effect of this change in section 2.4.

2.1 Ideal MAC-based integrity

The idea proposed in our paper [7] is to add a MAC of the non-confidential parameters required for PIN verification to the input to the PIN verification command. The PIN_Verify command checks the MAC before performing the PIN calculation. If the MAC check fails, the command halts without checking the PIN. This way we achieve 'robust declassification' [9], i.e. we declassify only the result of the legitimate comparison of the encrypted PIN block with the real PIN, and nothing else.

In our paper, we did not discuss how exactly this MAC should be calculated, and where it should be stored. Below we propose a way to store a MAC of the non-confidential inputs to the verification command on the card itself, using an existing mechanism.

2.2 CVC/CVV Codes

We observe that cards used in the cash machine network already have an integrity checking value: the card verification code (CVC) or value (CVV) is a 5 (decimal) digit number included on the magnetic stripe of most cards in use. It is, in effect, a MAC of the customer's PAN, expiry date of the card and some other data. The current purpose of the CVV is to make copying cards more difficult, since the CVV is not printed on the card and does not appear on printed POS receipts[1]. Below we give the algorithm for calculating the CVV. This is done at the card issuing facility. CVVs are checked at the verification facility.

PAN	Exp date	Service code	0 pad
16 digits max	4 digits	3 digits	9 digits max
Block B1	Block B2		

Note the partition of the CVV plaintext field into two blocks, B1 and B2. To construct the CVV, a two-part DES key is required. Call the two 64-bit parts key K1 and key K2. The hexadecimal CVV root is constructed as

[1] The CVV/CVC is not to be confused with the CVC2 or CVV2, which is printed on the back of the card and designed to counteract 'customer not present' fraud when paying over the Internet or by phone.

$$CVV_{hex} \quad = \quad enc(K1, dec(K2, enc(K1(enc(K1, B1) \oplus B2))))$$

The 5 digit decimal is constructed using the Visa decimalisation scheme:

1. Extract all decimal digits from CVV_{hex}, preserving their order from left to right.
2. Left justify the result
3. Reduce any remaining digits by 10
4. Left justify the result and append it to the result of 2
5. The CVV is the first 5 digits (from left to right) of the result.

2.3 Packing the MAC into the CVC/CVV

Our proposal is to pack more information into the CVV at issue time, and to use this as a MAC at the verification facility. In our formal scheme, we included the data of the decimalisation table and card offset. Observe that with the maximum 16 digits of the PAN being used, we still have 9 digits of zeros in the final field of block B2. Our idea is to use the CVV calculation method twice, in the manner of a hashed MAC or HMAC function. We will calculate the CVV of a new set of data, containing the decimalisation table and offset or PVV and a code for the PIN block format. Then we will insert the result of the original CVV calculation to produce a final 5-digit MAC.

Our second CVV, which we will call CVV', contains the following fields:

Dectab	Offset/PVV	PIN block format	original CVV	0 pad
16 digits max	4 digits	1 digit	5 digits	6 digits max
Block B1'	Block B2'			

We calculate CVV' in the same way as the standard CVV. This makes for easy upgrade from the original infrastructure, because CVV generation and verification commands are already available in HSMs so will need minimal changes to the firmware. The PIN_Verify command of the HSM must be changed to check CVV' before performing a verification test. The PIN test is only performed if the CVV' of the inputs matches the supplied CVV'. Of course, the API of the HSM must not make available the functionality to allow the creation of CVV's on arbitrary data.

The scheme is practical because ATMs and switches generally already send the CVV from the ATM to the issuing bank, so can easily be adapted to send CVV'. In fact, many ATMs blindly send all the 'Track 2 data' from the magnetic card - this includes the PAN, expiry date, and CVV. Under most schemes there is still space on the magnetic stripe for a further 5 digit code. Chip based cards should have no problem storing a further 20 bits of data. So, we could use CVV' and the original CVV', thus allowing CVVs to be checked separately if required.

One could use the same keys for calculating the CVV' as for the CVV, or one could use different keys. Either way, the verifying HSM needs access to these keys. The use of different keys could be motivated by a desire to be able to check CVVs, and so to an extent to verify card authenticity, in ATMs, without giving them the keys used to create CVV's.

2.4 Security of the CVV Based Scheme

In our formal scheme we assume a perfect, collision free scheme. However, in proposing a scheme with a 5-digit MAC value, we are admitting the possibility of brute-force attacks. To guess the CVV' for a given set of parameters should take an average of 50 000 trials. So, an attack like the dectab attack which previously required 15 calls to the API will now take 750 000 calls. HSMs typically perform something of the order of 1000 PIN verifications per second, so this change moves the expected attack time for a single PIN from 0.015 seconds to 750 seconds, or 12.5 minutes, making the 'lunch hour hack' scenario of [6] worth an expected 4 or 5 PAN/PIN pairs.

3 Conclusions

We have described a version of our MAC based scheme for ensuring integrity of queries to PIN processing APIs that is easy to implement and does not require wholesale changes to the ATM infrastructure. This is at the cost of some security, since the CVV codes can be cracked by brute force, but its implementation in the short term would make attack scenarios far less profitable. In the medium term we feel that the full MAC scheme should be used. The cost of this should be weighed against the cost of a complete overhaul of the way PIN processing is carried out in the ATM network.

References

[1] Hackers crack cash machine PIN codes to steal millions. The Times online. `http://www.timesonline.co.uk/tol/money/consumer_affairs/article4259009.ece`.

[2] PIN Crackers Nab Holy Grail of Bank Card Security. Wired Magazine Blog 'Threat Level'. `http://blog.wired.com/27bstroke6/2009/04/pins.html`.

[3] Verizon Data Breach Investigations Report 2009. Available at `http://www.verizonbusiness.com/resources/security/reports/2009_databreach_rp.pdf`.

[4] R. Anderson. What we can learn from API security (transcript of discussion). In *Security Protocols*, pages 288–300. Springer, 2003.

[5] O. Berkman and O. M. Ostrovsky. The unbearable lightness of PIN cracking. In Springer LNCS vol.4886/2008, editor, *11th International Conference, Financial Cryptography and Data Security (FC 2007), Scarborough, Trinidad and Tobago*, pages 224–238, February 12-16 2007.

[6] M. Bond and P. Zielinski. Decimalization table attacks for pin cracking. Technical Report UCAM-CL-TR-560, University of Cambridge, Computer Laboratory, 2003. `http://www.cl.cam.ac.uk/TechReports/UCAM-CL-TR-560.pdf`.

[7] M. Centenaro, R. Focardi, F. Luccio, and G. Steel. Type-based analysis of PIN processing APIs. Available from `http://www.dsi.unive.it/~focardi/typing-PIN-full.pdf`, 2009.

[8] J. Clulow. The design and analysis of cryptographic APIs for security devices. Master's thesis, University of Natal, Durban, 2003.

[9] A.C. Myers, A. Sabelfeld, and S. Zdancewic. Enforcing robust declassification and qualified robustness. *Journal of Computer Security*, 14(2):157–196, May 2006.

[10] G. Steel. Formal Analysis of PIN Block Attacks. *Theoretical Computer Science*, 367(1-2):257–270, November 2006.