The unbearable lightness of PIN cracking

Omer Berkman[1] and Odelia Moshe Ostrovsky[2,3*]
[1] The Academic College of Tel Aviv Yafo, School of Computer Science
[2] Algorithmic Research Ltd. – Digital/Electronic Signatures, www.arx.com

[3] Tel Aviv University, School of Computer Science
omer@mta.ac.il, odelia@arx.com
Version 1.8

Abstract. We describe new attacks on the financial PIN processing API. The attacks apply to switches as well as to verification facilities. The attacks are extremely severe allowing an attacker to expose customer PINs by executing only one or two API calls per exposed PIN. One of the attacks uses only the translate function which is a required function in every switch. The other attacks abuse functions that are used to allow customers to select their PINs online. Some of the attacks can be applied on a switch even though the attacked functions require issuer's keys which do not exist on a switch. This is particularly disturbing as it was widely believed that functions requiring issuer's keys cannot do any harm if the respective keys are unavailable.

Key words: Security API, API attack, Financial PIN Processing API, HSM, Insider attack, Phantom Withdrawal, VISA PVV, IBM 3624, EMV

1 Introduction

Personal Identification Number (PIN) is the means used by a bank account holder to verify his/her identity to the issuing bank. When a PIN is entered by the card holder at a service point (e.g., an Automatic Teller Machine), the PIN and account number are sent to the verification facility (the issuing bank or other authorized entity) for verification. To protect the PIN on transit, it is formatted into a PIN block, the PIN block is encrypted under a transport key and the resulting Encrypted PIN Block (EPB) is sent for verification. As there usually isn't direct communication between the service point and the verification facility, the PIN goes through switches. Each switch decrypts the EPB, verifies the resulting PIN block format (so the format serves as some form of Message Authentication Code), re-formats the PIN block if necessary, and re-encrypts the PIN block with a transport key shared with the next switch (or the verification facility when arriving there). Switches may be part of other issuers' verification facilities or may be stand alone. There is generally no connection between a switch facility that handles an incoming EPB, and the issuer of the respective

* The work of this author was carried out as part of an MSc thesis in Tel Aviv University.

account number. Additionally, switches may be physically far from the issuer (for example, when a customer withdraws money overseas).

To protect the PIN and the encryption keys both in switches and in the issuer's environment, all operations involving a clear PIN are handled within a Hardware Security Module (HSM). Such operations are controlled by an application at the site using a cryptographic API. The Financial PIN Processing API is a 30-years old standard which includes functions for, e.g., PIN issuing, PIN verification, PIN reformatting, and PIN change.

The issuer's environment is usually physically separated into an issuing facility and an online verification facility. The issuing facility where customer PINs are generated and printed for delivery is usually isolated logically and physically from the rest of the issuer's environments. The verification facility as well as switches on the other hand, are in general significantly less protected than the issuing facility as they are required to be online. [1] requires that HSMs implementing the Financial PIN processing API separate the functionality required for the issuing facility from functionality required for the verification facility. The separation can be logical and/or physical. The reason for this requirement, is that much of the required functionality in the issuing facility is sensitive, and must not therefore be accessible in the verification facility which is much less secure. Switches are treated as verification facilities in this respect so HSMs in switches should only contain (at least logically) the functionality required for the verification facility.

In this paper we describe attacks on the Financial PIN Processing API, which result in discovering customers PINs. The attacks can be applied in switches as well as in verification facilities. The attacks require access (i) to the HSM in the attacked facility for executing API calls; (ii) to EPBs incoming to the attacked facility. Applying such attacks thus requires the help of an insider in the attacked facility. However, when the attacks are applied on a switch, one cannot relate to them as insider attacks. Since the switch, and the issuer whose EPBs are attacked on the switch, are unrelated, an insider of the switch facility is an outsider from the issuer's point of view. The issuer has no control, neither on the environment nor on the employees in the attacked system. We stress that our attacks only require the use of API functions (and only the ones approved for the verification facility) and **do not** assume that the attacker can perform operations such as loading known keys into the attacked HSM.

The first attack uses a single API function denoted "translate". The translate function allows to reformat an EPB in any PIN block format to an EPB in another PIN block format. It also allows to change the transport key which encrypts the PIN block. It is a required function in every switch, and exists also in verification facilities as part of the API. The attack on the translate function allows revealing the PIN packed in each EPB arriving to the attacked switch using one or two HSM calls, following a (one-time) preprocessing step of 20,000 HSM calls.

The attack is based on known weaknesses in the standard which are described (and referenced) later is this section, and in Section 3. Our contribution is two

fold: (i) we observe that these weaknesses imply a major security problem allowing to build a single (small) look-up table enabling to expose the contents (in particular the PIN) of any customer's EPB arriving to the attacked facility. (ii) we provide an efficient way to build such a table. Step (ii) uses ideas from [2–4].

The second attack has several variants some applicable to both switches and to verification facilities and some only to verification facilities. The attacks require the use of one of two API functions which are used for allowing customers to select their PINs online (these function are denoted "calculate offset" and "calculate PVV"). The most severe variants allow discovering a PIN from its EPB, discovering a PIN given its respective account number and setting a new value for a customer's PIN. Each of these variants can be performed in one or two HSM calls (and no preprocessing). We are not aware of previous attacks abusing these two functions.

Several papers ([5, 6, 2]) discuss a function denoted hereafter *change account number* that **is not** part of the Financial PIN Processing API but was added temporarily to the implementation of the API in a certain bank in order to enable changing all customers account numbers without re-issuing new PINs. The function accepts the customer's old account number, the customer's new account number and the customer's offset (a value used in the verification process - see Section 4.2) and retrieves a new offset to be used with the new account number. These publications show that the change account number function could have been used to discover customer PINs. One of our variants, reveals that the function calculate offset, which is part of the Financial PIN Processing API is as dangerous as the change account number function. The crucial observations are: (i) Although different, the input parameters of the two functions actually contain the same inherent information (in different form); and (ii) the functions use this information in a similar way. Therefore the two functions can be abused in a similar way to calculate a value that can be used to reveal the customer's PIN.

Both calculate offset and calculate PVV functions require issuer keys so it is quite surprising that they can be attacked in switches as switches do not contain issuer keys. Indeed, the design of each of the functions allow an attacker to use any key instead of the issuer's key when attacking these functions in a switch. The value of the key is not important, and the attacker does not need to know it. This is particularly disturbing as it is widely believed that functions requiring issuer's keys cannot do any harm if the respective keys are unavailable.

In some of the cases above, the attacked functions are not used by the application at the site. It is important in such cases to disable these functions (as well as other unused functions) if this capability is offered. Issuers certainly have the incentive to apply such measures in their verification (and issuing) facilities. However, it is not clear how to verify that switch facilities adhere to these measures.

The attacks apply to all common commercial HSMs implementing the API and affect all financial institutions. The attacks require one or two HSM calls per PIN discovery, enabling the discovery of several thousand (the exact number

depends on the HSM) customer PINs per attacked HSM per second. The attacks do not require the knowledge of any key value. The attacks apply also to the EMV standard ([7]) when on-line verification takes place.

The attacks abuse integrity and secrecy weaknesses in the financial PIN processing API, some of which are well known ([8, 9, 4, 3, 10, 2], see also Section 3). In fact, integrity is almost non existent in the standard. For example, we are able to trick API functions into accepting a customer's EPB together with an account number which is not the customer's.

The attacks described in this paper enable an attacker to apply serious attacks on issuing banks, such as simultaneous withdrawals of aggregate large sums of money. The attacks may also explain cases of phantom withdrawals where a cash withdrawal from an ATM has occurred, and neither the customer nor the bank admits liability.

To prevent the attacks described in this paper, worldwide modifications in ATMs, HSMs and other components implementing the Financial PIN processing API must be introduced.

Previous API-level attacks appear in [11–13]. Previous attacks on the Financial PIN Processing Standard appear in the references above as well as in [6].

The rest of the paper is organized as follows. In Section 2 we discuss the threat model. In Sections 3 and 4 we describe the attacks.

2 Threat model

We discuss in this section the requirements from an attacker.

A potential attacker is an insider of the attacked facility - a switch or a verification facility. Such an insider should have logical access to the HSM in the facility and should be able to generate API calls (the required API functions depend on the attack). In many cases this is easy as the HSM is connected to the organization's internal network. When this is not the case, the attacker can, for example, interfere with or masquerade as the legal application working with the HSM in the attacked facility.

All API functions use cryptographic keys. The standard does not specify how keys should be input to an API function but most implementations either keep keys outside the HSM encrypted by a master key, or keep them inside the HSM. In the first case, HSMs accept encrypted keys in each API call. In this case, an attacker is only required to record the desired encrypted key buffer from a real transaction. The same encrypted key can then be used in the attacker's API calls to the HSM. In the second case where keys are stored and managed inside the HSM, the attacker only needs to know the required key ID. In this case, however, the HSM may also handle user access rights to the keys. To use the required keys in such cases the attacker can, as before, interfere with or masquerade as the legal application working with the HSM in the attacked facility. In any case, we never assume that the attacker has any knowledge of the value of cryptographic keys.

In some of the attacks the attacker is required to generate EPBs which contain known PINs and which share a transport key with the attacked HSM. To do this, the attacker can use any banking card and enter a desired PIN at an ATM adjacent physically or logically to the attacked HSM. The attacker then needs to record the EPB when it arrives to the attacked facility. This can be done in various ways, i.e., by a program that reads the EPB on its way from the application to the HSM in the site. In the same way an attacker is able to record EPBs incoming to the switch, e.g., in order to expose the PINs they hide.

Transport keys sometimes change. However, parameters to the API functions that control the keys to be used in the API function come from the outside so the attacker can always use the same key. Additionally, when required, the attacker can use the translate function to translate an EPB encrypted with one transport key to an EPB encrypted with another.

3 Attack on the translate function

The attack we describe in this section, enables revealing for any EPB arriving to the attacked switch (or verification facility), the PIN that the EPB packs. The attack uses at most two API calls per EPB. The attack requires also a one-time preprocessing step consisting of 20,000 API calls (assuming the PIN is of length 4 as is normally the case). The attack uses the translate API function only.

As we mentioned earlier, on its way for verification, the PIN is formatted into a PIN block and the result is encrypted using a transport key to generate an Encrypted PIN Block (EPB). The attack is based on the following weaknesses: (i) the fact that there exists an API function which translates an EPB in one approved PIN block format to another; (ii) weaknesses in the approved PIN block format themselves.

Specifically, [14] describes four different PIN Block formats. ISO-0, ISO-1, ISO-2, and ISO-3, which differ in whether the customer's account number and/or random data is involved in the format in addition to the PIN itself. ISO-0 uses only account number, ISO-1 uses only random data, ISO-2 uses neither account number nor random data, and ISO-3 uses both account number and random data. [1] approves ISO-0, ISO-1, and ISO-3 for online PIN transactions. ISO-2 is **not** approved for online PIN transactions since an EPB based on ISO-2 (and on a given transport key) has only 10,000 possible values (assuming the PIN is of length 4) enabling the use of a look-up table. The specific weaknesses we use are the following:

1. The translate API function allows reformatting an EPB from any of the approved formats (ISO-0, ISO-1, or ISO-3) to another ([4, 2, 3]).
2. For a particular account number, the ISO-0 format is as weak as ISO-2: an EPB based on ISO-0 and a particular account number has only 10,000 possible values enabling the use of a look-up table ([4, 8, 2]).
3. The ISO-1 format is independent of any account number ([2]).
4. As mentioned in the introduction, the format of PIN block serves as a form of Message Authentication Code (MAC). The weakness is that in ISO-0 format,

digits of the PIN are XORed with digits of the account number, making it impossible to correctly authenticate neither ([2–4]).

We start by showing that weaknesses 1-3 degrade the system to the strength of ISO-2, the weak and thus non-approved PIN block format.

Given an EPB in any of the approved formats (ISO-0, ISO-1, or ISO-3) we use the translate capability to convert it to ISO-1 (if it is not already in ISO-1) and then back to ISO-0 but with a chosen account number. By doing this we untie the link between the customer's account number and the customer's PIN and create a fabricated link between a chosen account number and this customer's PIN. This abuse was discussed in [8], [4] and [2].

Fixing the chosen account number to some value A, and applying the above to all EPBs arriving to the attacked switch ensures that all resulting EPBs are based on account number A, thus degrading their strength to that of ISO-2. This new observation has extremely serious implications on the security of the Financial PIN Processing API, as it implies that a single look-up table of size 10,000 is all that is required in order to discover the PIN packed in **every EPB arriving to the attacked switch**, regardless of its account number.

We now describe how to build the required table. Specifically, we show how to generate a table of 10,000 EPBs, where the ith EPB, $1 \leq i \leq 10,000$ contains the PIN whose value is i, and such that each EPB in the table is formatted in ISO-0 using account number A.

One obvious way such a table can be generated is by brute force - generating 10,000 EPBs by ATMs: For each i, $1 \leq i \leq 10,000$ use a card with any account number and type PIN value i. When the respective EPB arrives at the attacked HSM, translate it to ISO-0 using account number A (by one or two calls to the translate function depending on the format of the incoming EPB). Using different account numbers when generating EPBs via ATMs would make it harder to discover the attack.

We now describe a much more practical method of generating the table. This method requires generating only 100 EPBs by ATMs. To build the 10,000-entry table we use weakness 4 to generate 100 of the required EPBs *from each* of the 100 EPBs generated by ATMs. The details are described below.

We start by describing the ISO-0 PIN block format. Denote the PIN $P_1P_2P_3P_4$ and the respective account number $A_1A_2 \ldots A_{12}$ (only 12 digits of the account number are used in the ISO formats). The PIN block is the XOR of two 16-hexadecimal digits blocks. An *original block* containing the PIN ("F" stands for the hexadecimal value F)

0	4	P_1	P_2	P_3	P_4	F	F	F	F	F	F	F	F	F	F

with an *account number* block containing the account number

0	0	0	0	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8	A_9	A_{10}	A_{11}	A_{12}

When an API function receives an EPB in ISO-0 as a parameter, it also receives its associated account number. To use the PIN packed in the EPB, the

function decrypts the EPB, and XORs the result with the account number block to recreate the original block. It then authenticates the result by verifying that the values of the first two digits of the original block are 0 and 4, that the last 10 digits are hexadecimal F and that the PIN is composed of decimal digits.

Weakness 4 - the fact that two digits of the PIN are XORed with two digits of the account number - is used for generating the table. The attacker generates in ATMs 100 EPBs packing, respectively, PIN values $0000, 0100, \ldots, 9900$. Each of these EPBs is formatted in ISO-0 and associated with account number $00A_3 \ldots A_{12}$ where the values $A_3, \ldots, A_{12}$ are immaterial to the attack and can be different for each PIN value to make the attack more innocent. (It is also possible to generate the 100 EPBs in ATMs using completely arbitrary account numbers and then change the account number of each EPB to the desired one using the translate function.)

We complete our description by showing how the attacker generates an EPB containing PIN value $xyuv$ for any decimal values x, y, u, v:

To generate an EPB that packs PIN value $xyuv$, the attacker uses the EPB packing $xy00$ which was generated by ATM. It reformats this EPB to ISO-1 but instead of using the original account number $00A_3 \ldots A_{12}$ the attacker provides the translate function with account number $uvA_3 \ldots A_{12}$.

The translate function decrypts the EPB and gets a block which is the XOR of the original block

0	4	x	y	0	0	F	F	F	F	F	F	F	F	F	F

and the original account number block

0	0	0	0	0	0	A_3	A_4	A_5	A_6	A_7	A_8	A_9	A_{10}	A_{11}	A_{12}

Note that the function only sees the decrypted block - the XOR of these two blocks. It then XORs the decrypted block with the following:

0	0	0	0	u	v	A_3	A_4	A_5	A_6	A_7	A_8	A_9	A_{10}	A_{11}	A_{12}

to get

0	4	x	y	u	v	F	F	F	F	F	F	F	F	F	F

This resulting block will be authenticated (its two first digits are 0 and 4, its 10 last digits are hexadecimal F, and the PIN consists of decimal digits). Consequently, PIN value $xyuv$ will be packed in an EPB in ISO-1 PIN block and returned. The attacker can now translate this EPB to ISO-0 with the desired account number A.

4 Attacks on functions allowing PIN change

In the Financial PIN Processing API, the PIN is verified using one of two approved methods - the IBM 3624 or the VISA PIN validation value (PVV) methods. In both methods the input to the verify function is as follows:

– An EPB containing the PIN presented by the customer.
– The customer's account number.
– A four decimal digits *customer's verification value* (called *offset* in the first method and PVV in the second).

This customer's verification value is not secret. It is kept either in a database as part of the customer details (together with the customer's account number), or on the customer's card.

Denote by P the PIN packed in the EPB, by A the customer's account number, and by V the customer's verification value.

The verify function decrypts the EPB, authenticates it by verifying the PIN block format, extracts P from the EPB, and verifies whether $V = f(P, A)$ where f is a function which depends on an issuer's secret key. The function f is different in the two methods.

In order to allow customers to select their PIN online, the Financial Processing API contains two functions (one for each method) that allow recalculating the customer's verification value when the customer's PIN changes. The functions are denoted *calculate offset* and *calculate* PVV. Both functions receive the following input:

– An EPB containing the customer's chosen PIN .
– The customer's account number.

The functions return $V = f(P, A)$ where P, A, V and f are as before. We note that in both functions, the value V is pseudo random as a result of using the random issuer's key in f.

The main weakness in both functions regardless of f is that the new PIN supplied to the function (packed in an EPB) is not bound to the old PIN. Note that since an attacker can carry out the attack by directly using the API, this binding must be checked by the function itself and not by the application at the site. Note also that it **would not** be enough to add the functions a parameter consisting of an EPB that packs the customer's old PIN (and means to verify it, i.e., the respective PVV) since the attacker can record a customer's real EPB on its way for verification, and use it as the additional parameter.

We describe below attacks on the calculate PVV function. These attacks do not use any properties of the respective f (except for assuming that the value V is pseudo random). Thus, all our attacks on calculate PVV apply also to calculate offset. We also describe attacks specifically on calculate offset. These attacks use properties of the respective f and are more severe than the general attacks.

To attack the calculate PVV, we are required to send it an EPB and an account number. In all cases, the EPB would be generated by one customer and the account number would belong to another.

To force the calculate PVV function to accept the non-matching parameters we use the translate function to reformat the EPB to ISO-1 (which is not linked to any customer) before sending it to the calculate PVV function. Because ISO-1

does not depend on account number, there would be no inconsistency between the EPB parameter and the account number parameter. Note that restricting the calculate PVV function to accept only EPBs with a certain format would not thwart the attacks, as we can reformat the EPB to that format. We also note that the attacks can be applied (though in a more restricted form) even if PIN block reformatting capability is disabled.

When attacking the calculate PVV function (respectively the calculate offset function) with an EPB denoted E and an account number A, we use a shorthand $V = PVV(E, A)$ (respectively, $O = offset(E, A)$). As all EPBs are converted to ISO-1 before using the function, we will not mention the reformatting any more.

4.1 Attacks on the calculate PVV function

Attacking the calculate PVV function in a switch Consider all customer EPBs arriving to the attacked switch. The attack discovers for each such EPB (and its associated account number) a list of other EPBs having the same PIN (with high probability). It requires one or two HSM calls per attacked EPB. We note that the attack can be applied also in verification facilities.

We use a table of 10,000 entries. The table is indexed by values of computed PVVs. Each entry of the table contains customer EPBs (and their associated account numbers). Initially all entries are empty.

We choose a fix account number B. We show how to attack any customer's EPB arriving to the switch. Denote by E_c the customer's EPB.

1. $V = PVV(E_c, B)$.
   The computed PVV value V equals $f(P_c, B)$ where P_c is the PIN packed in the customer's EPB.
2. Add the customer's EPB E_c to the table entry corresponding to the resulting PVV value V.
   For example, if the value of P_c is 1234 and V is 5678 then E_c will be added to table entry 5678.

The computed PVV value V depends only on P_c, B, and on the key k used by the function f. Since the attack is applied in a switch, k is not the issuer's key as required, but some other arbitrary value (not known to the attacker). The value of k is immaterial to the attack since all that we require is that the value V be a pseudo random function of P_c, B, and k, which is the case from our assumption on f. Since B and k are fixed, V can be regarded as a pseudo random function of P_c only.

Suppose we have performed the above with many EPBs. What actually happens in steps 1 and 2 is that all EPBs that pack the same PIN value are thrown into the same table entry. Since the process is random, a table entry may be empty, may contain EPBs corresponding to a single value of PIN, or may contain EPBs corresponding to several PIN values. Combinatorically, the process is equivalent to throwing balls (PINs) to bins (table entries) and asking questions

on the number of balls (distinct PINs) in each bin. It can be shown that when the number of balls and bins is the same (10,000 in our case) the average number of balls in a non-empty bin is less than 2. In other words, EPBs that ended in the same table entry correspond to less than 2 distinct PINs on the average.

Since the probability that EPBs in the same table entry correspond to more than a single PIN is still high, we are still not done. For each entry in the table we repeat the procedure with respect to the EPBs in that entry using a different fixed account number C. EPBs from a given table entry that end in the same table entry again, have high probability of having the same PIN.

Attacking the calculate PVV function in a verification facility This attack reveals for any account number associated with the attacked issuer, the PVV that corresponds to this customer's account number and a chosen PIN. Replacing the verification value on the card or in the database (depending on the system) enables withdrawing money from the customer's account using the chosen PIN. The attack requires one HSM call per account number attacked. In addition, it requires generating by ATM an EPB that packs a known PIN. This single EPB will be used to attack the account numbers of all customers.

We start by generating an EPB in an ATM that packs a chosen PIN. This EPB, denoted E_a (for attacker's EPB) is used to attack the account numbers of all customers. For each customer's account number A_c compute $V = PVV(E_a, A_c)$.

The computed PVV value V corresponds to the customer's account number and the chosen PIN. Since the attack takes place in the verification facility, the required issuer's key is used, and the PVV is valid.

It remain to explain how the attacker can replace the customer's original PVV used by the system by the PVV computed in the attack.

According to [1], the clear PVV can be stored on the card's magnetic stripe or in a PVV database. In case the PVV is stored on both, the PVV is taken from the database. In many implementations the PVV is stored only on the card as long as the customer uses the initial PIN generated by the issuer.

Setting the customer's PVV to the computed PVV can be done as follows:

Case 1: The PVV is stored only on the card. Generate a card containing the customer's details and set the PVV value on the magnetic stripe to the PVV that was calculated by the attacker. In this case the fabricated card and the customer's original card will both be valid at the same time. It is important to note that in this case, issuing a new PIN to a customer will not prevent the attack as the fabricated card with the false PVV will remain valid.

Case 2: The PVV entry of this customer exists in the PVV database. In this case the attacker needs write access to the PVV database. The attacker can then do one of the following:

- Delete the PVV entry (and then apply the steps described in Case 1).

– Set the customer's entry in the PVV database to the PVV that was calculated by the attacker. If the entry does not exist - create it. In this case the fabricated card will be the only valid card.

4.2 Attacks on the calculate offset function

The specific function f in calculate offset is $V = g(A) - P$ where P is the PIN packed in the EPB parameter, A is the account number parameter, V is the returned offset, g is a function that depends on an issuer's key and computes a 4 decimal digits number, and "$-$" is minus modulo 10 digit by digit.

Attacking the calculate offset function in a switch The attack reveals for each customer's EPB arriving to the attacked switch, the PIN it packs. It requires one or two HSM calls per attacked EPB. In addition, it requires generating by ATM an EPB that packs a known PIN. This single EPB will be used to attack all EPBs arriving to the attacked switch. We note that the attack can be applied also in verification facilities.

Choose a fixed account number B. Generate an EPB in an ATM that packs a chosen PIN and denote it E_a. Compute $O_1 = Offset(E_a, B)$. For each customer's EPB arriving to the attacked switch compute $O_2 = Offset(E_c, B)$ where E_c is the customer's EPB.

Denote by P_a and P_c the values of PINs packed in the attacker's and customer's EPBs, respectively. We thus have $O_1 = g(B) - P_a$ and $O_2 = g(B) - P_c$. Since the value of P_a is known, the value of P_c can be trivially calculated. Note that the value of $g(B)$ is immaterial, so the attack can be applied in a switch which does not contain the required issuer's key.

Attacking the calculate offset function in a verification facility This attack reveals for every customer account number associated with the attacked issuer, the respective customer's PIN. It requires one or two HSM calls per attacked account number. In addition, it requires generating by ATM an EPB that packs a known PIN. This single EPB will be used to attack all account numbers associated with the issuer.

Generate an EPB in an ATM that packs a chosen PIN. Denote the EPB by E_a. Given any customer's account number A_c compute $O = Offset(E_a, A_c)$.

Denote by P_a the value of PIN packed in the attacker's EPB, by P_c the required customer PIN, and by O_c the customer's offset (stored in the issuer's database or on the magnetic stripe of the card). We thus have $O = g(A_c) - P_a$. Since P_a is known, $g(A_c)$ can be easily computed. We also know that $O_c = g(A_c) - P_c$. Since $g(A_c)$ is known and since the value of O_c is not secret (it can be recorded during a transaction or read from the database or from the card) the customer's PIN P_c can be trivially calculated. Note that the exact nature of g is immaterial, but the attack requires the real value of $g(A_c)$ so it needs to be applied in the verification facility were the required issuer's key exists.

5 Conclusions

We have shown in this paper that the Financial PIN processing API is exposed to severe attacks on the functions translate, calculate offset and calculate PVV inside and outside of the issuer environment.

The attacks we describe provide explanations to possibly many unexplained Phantom Withdrawals. The attacks are so simple and practical that issuers may have to admit liability not only for future cases but even retroactively. The attacks can be applied on such a large scale (in some of the variants up to 18,000,000 PINs can be discovered in an hour) that such liability can be enormous.

As some of the attacks apply to switches, which are not under the issuers control, countermeasures in the issuers environment do not suffice. To be protected from this attack, countermeasures in all verification paths to the issuer must be taken. As this is unrealistic, solutions outside the standard must be sought.

We have also shown that the common assumption that all sensitive operations are limited to the issuing facility and that separation of the issuing and verification facilities prevent severe attacks is wrong.

It is well known that when several PIN block formats are available the security of the whole system degrades to the security of the weakest PIN block format. The attacks demonstrate that reformatting capability between different PIN block formats allows an attacker to abuse weaknesses of both formats. Therefore enabling reformatting is more dangerous than using the weakest format. We have also shown that the ISO-1 format is extremely weak and thus should be immediately removed from the list of allowed interchange transaction formats.

Another interesting insight from the attacks described is that the offset and the PVV values may reveal as much information as the PIN itself. One possible remedy is treating the offset and the PVV as secret values.

The changes require worldwide modifications in ATMs, HSMs and other components implementing the PIN processing API.

In addition to all implementation of this API, systems applying the EMV standard ([7]) and using online (rather than off-line) PIN verification are also vulnerable to the attacks.

References

1. VISA: PIN security requirements (2004) http://partnernetwork.visa.com/st/pin/pdfs/PCI_PIN_Security_Requirements.pdf.
2. Clulow, J.: The design and analysis of cryptographic APIs. Master's thesis, University of Natal, South Africa (2003) Available through http://www.cl.cam.ac.uk/~jc407.
3. Bond, M., Clulow, J.: Extending security protocols analysis: New challenges. In: Automated Reasoning and Security Protocols Analysis (ARSPA). (2004) 602–608

4. Bond, M., Clulow, J.: Encrypted? randomised? compromised? In: Workshop on Cryptographic Algorithms and their Uses. (2004)
5. Anderson, R.: The correctness of crypto transaction sets. In: Security Protocols, 8th International Workshop. (2000)
6. Andersson, R.: Why cryptosystems fail. Communications of the ACM **37**(11) (1994) 32–40
7. EMV: Integrated circuit card specifications for payment systems (2004) Available through http://www.emvco.com.
8. Anderson, R., Bond, M., Clulow, J., Skorobogatov, S.: Cryptographic processors - a survey. Proceedings of the IEEE **94**(2) (2006) 357–369
9. Bond, M.: Understanding Security APIs. PhD thesis, University of Cambridge (2004) Available through http://www.cl.cam.ac.uk/~mkb23/research.html.
10. Bond, M., Zielinski, P.: 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.
11. Longley, D.: Expert systems applied to the analysis of key management schemes. Computers and Security **6**(1) (1987) 54–67
12. Rigby, S.: Key management in secure data networks. Master's thesis, Queensland Institute of Technology, Australia (1987)
13. Steel, G., Bundy, G.: Deduction with XOR constraints in security API modelling. In: Proceedings of the 20th Conference on Automated Deduction (CADE 20). (2005)
14. ISO: Banking – personal identification number (PIN) management and security – part 1: Basic principles and requirements for online PIN handling in ATM and POS systems (2002)

Appendix

Terminology

ADD10 –	Adding modulus 10 digit by digit.
EPB –	Encrypted PIN block.
Final PIN –	The PIN (Personal Identification Number) associated with a given PAN. It is entered by the cardholder when using the card.
Issuer PIN key –	The key used by the issuer to calculate a Natural PIN for each customer.
Issuer PVV key –	The key used by the issuer to calculate a PVV for each customer's PIN.
Natural PIN –	Used in IBM3624 calculation method as an intermediate PIN. It is the result of encrypting the customer's PAN with an issuer's PIN key.
Offset –	A value that is added (modulus 10) to the "Natural PIN" in order to get the "Final PIN" (used only in IBM3624 calculation method). When customers choose a new "Final PIN" only the offset is re-calculated and kept in clear.
PAN –	Primary Account Number, which is equivalent to a card number.
Phantom Withdrawals –	A cash withdrawal from an ATM where money has been withdrawn, and neither the customer nor the bank admit liability.
PVV –	PIN Verification Value. This value is calculated from the "Final PIN" presented by the customer and compared to a pre-calculated value, which is kept in clear on the customer's card or in the issuer's PVV database. When customers choose a new "Final PIN" the PVV is re-calculated and kept in the PVV database. When an entry for a customer's PAN exists in the PVV database, the PVV on the card is ignored.
SUB10 –	Subtracting modulus 10 digit by digit.

PIN Block Formats

We describe below only PIN block formats and functionality used in this chapter. The description is reproduced from [18]. The PIN block is composed of 16 hexadecimal digits (64 bits). For simplicity of exposition we take the PIN to be of length 4, although the attacks work for any PIN length.

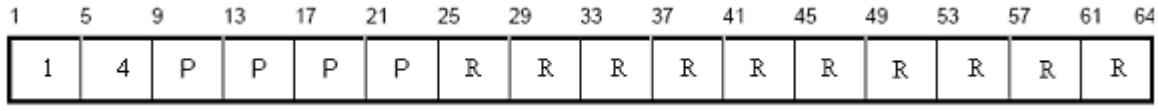
ISO-0 PIN block format

The PIN block is formed by combining, in XORing an 'original PIN Block' comprising PIN digits, with a mask formed from the least significant 12 digits (excluding the check digit) of the Primary Account Number (PAN).

0	4	P	P	P	P	F	F	F	F	F	F	F	F	F	F	Original PIN Block
0	0	0	0	A	A	A	A	A	A	A	A	A	A	A	A	PAN
0	4	P	P	X	X	X	X	X	X	X	X	X	X	X	X	PIN Block

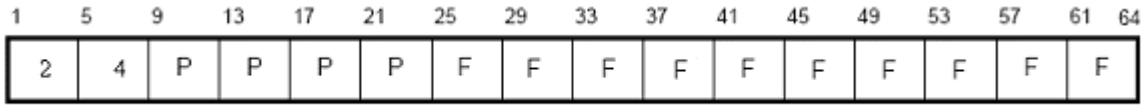
0 in the "Original PIN Block" stands for ISO-0 format.
4 in the "Original PIN Block" stands for the PIN length.
P = PIN digit (permissible values 0 to 9).
F is the Fill digit (must be hexadecimal F).
This format is used for online interchange transactions.

ISO-1 PIN block format



1 stands for ISO-1 format.
4 stands for the PIN length.
P = PIN digit (permissible values 0 to 9).
R = Random. (permissible values 0 to F).
This format is used for online interchange transactions.

ISO-2 PIN block format



2 stands for ISO-2 format.
4 stands for the PIN length.
P is the PIN digit (permissible values 0 to 9).
F is the Fill digit (must be hexadecimal F).
This format is used for the physical personalization process (when creating the card) and is not approved for online interchange transactions.

ISO-3 PIN block format

The PIN block is formed by combining, in XORing an ‘original PIN Block’ comprising PIN digits, with a mask formed from the least significant 12 digits (excluding the check digit) of the Primary Account Number (PAN).

3	4	P	P	P	P	F	F	F	F	F	F	F	F	F	F	Original PIN Block
0	0	0	0	A	A	A	A	A	A	A	A	A	A	A	A	PAN
3	4	P	P	X	X	X	X	X	X	X	X	X	X	X	X	PIN Block

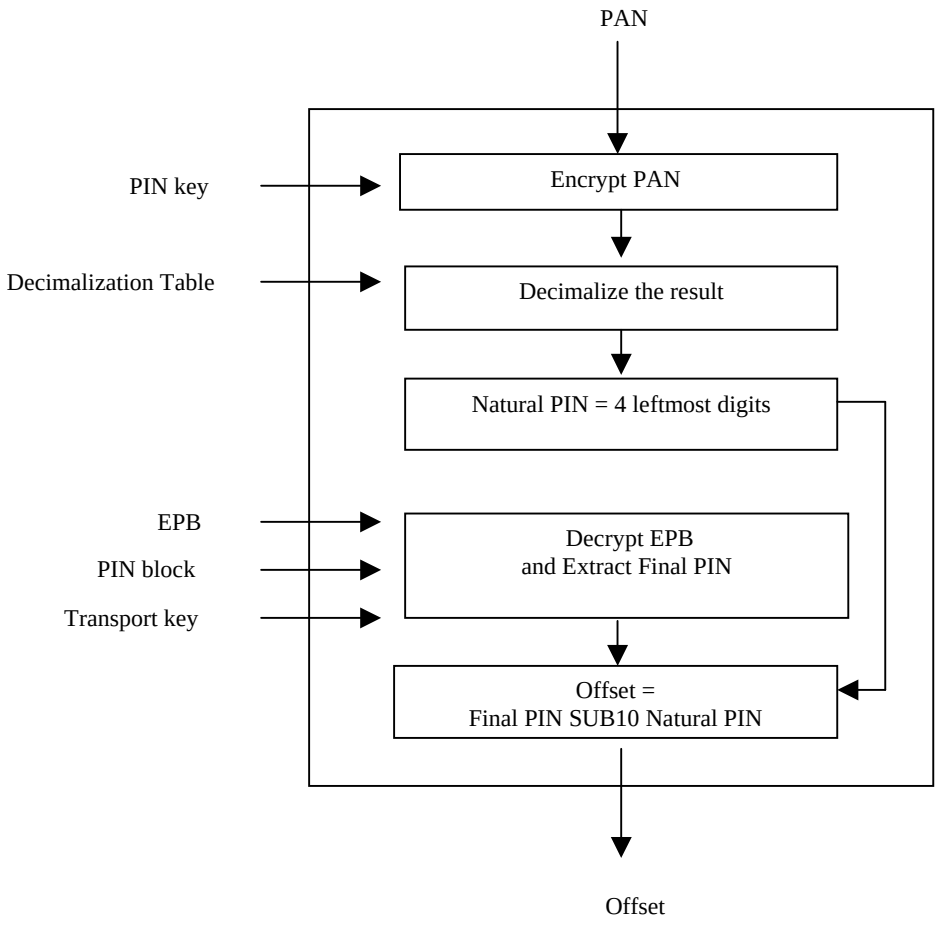
3 in the “Original PIN Block” stands for ISO-3 format.
4 in the “Original PIN Block” stands for the PIN length.
P is the PIN digit (permissible values 0 to 9).
F is the Fill digit (permissible value A to F).
This format is used for online interchange transactions.

The Financial PIN Processing API:

The Financial PIN Processing API is a set of functions used for securing PIN handling. We describe below a subset of these functions used in our attacks as reproduced from [18]. Different HSMs may implement a given function in slightly different ways, for example by executing two HSM calls rather than one.

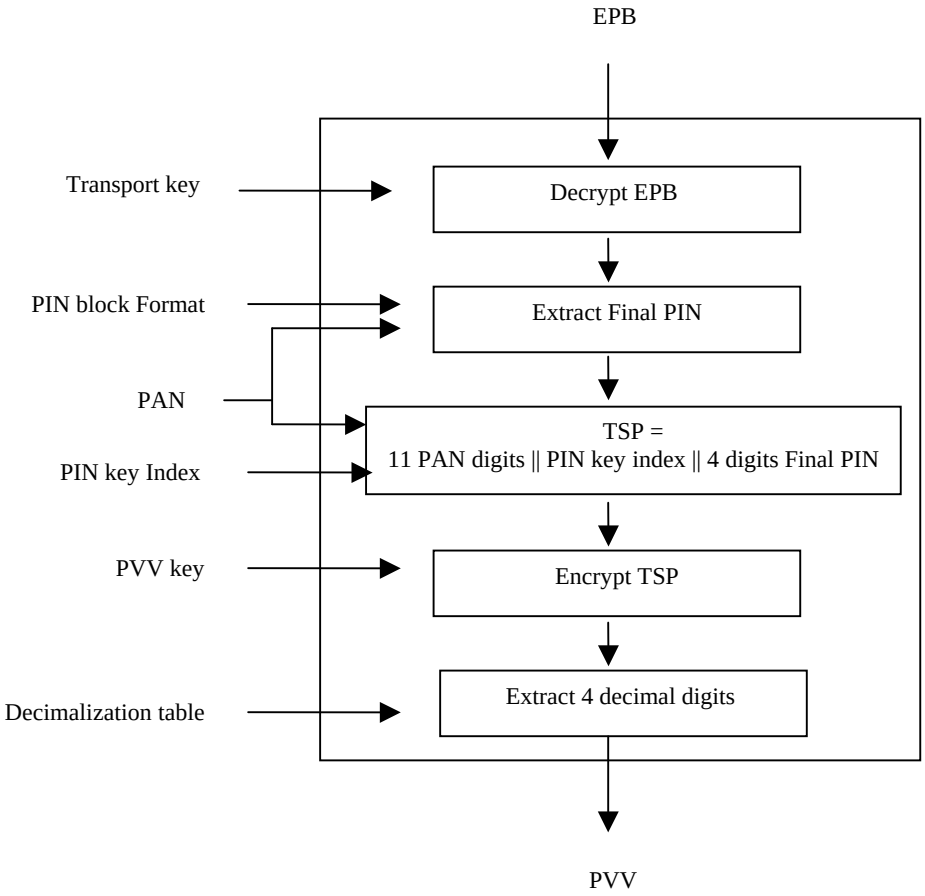
Calculate Offset:

This function calculates the customer's offset given a customer's PAN and an EPB that packs the customer's "Final PIN". It first calculates the customer's "Natural PIN" from the customer's PAN using a PIN key. Then it calculates the offset by subtracting (modulus 10) the "Natural PIN" from the customer's "Final PIN" that is extracted form the EPB.
This function is known as "IBM3624 PIN offset calculation method" ([18]).



Calculate PVV:

This function calculates the PVV given a customer's PAN and an EPB that packs the customer's "Final PIN". It concatenates the PIN to the customer's PAN, and then encrypts the result using a PVV key. Four decimal digits are extracted from the encrypted result (details omitted) and constitute the PVV value.
This function is known as "VISA PIN validation value" method.



Translate:

This function translates an input EPB from one format to another and/or from one transport key to another.