

# Weighing Down “The Unbearable Lightness of PIN Cracking”<sup>\*</sup>

Mohammad Mannan and P.C. van Oorschot  
School of Computer Science, Carleton University

**Abstract.** Responding to the PIN cracking attacks from Berkman and Ostrovsky (FC 2007), we outline a simple solution called *salted-PIN*. Instead of sending the regular user PIN, salted-PIN requires an ATM to generate a *Transport Final PIN* from a user PIN, account number, and a salt value (stored on the bank card) through, e.g., a pseudo-random function. We explore different attacks on this solution, and propose a variant of salted-PIN that can significantly restrict known attacks. Salted-PIN requires modifications to service points (e.g. ATMs), issuer/verification facilities, and bank cards; however, changes to intermediate switches are not required.

## 1 Introduction

Attacks on financial PIN processing APIs revealing customers’ PINs have been known to banks and security researchers for years (see e.g. [5], [4], [3]). Apparently the most efficient of these ‘PIN cracking’ attacks are due to Berkman and Ostrovsky [2].<sup>1</sup> However, proposals to counter such attacks are almost non-existent in the literature, other than a few suggestions; for example, maintaining the secrecy (and integrity) of some data elements related to PIN processing (that are considered security insensitive according to current banking standards) such as the ‘decimalization table’ and ‘PIN Verification Values (PVVs)/Offsets’ has been emphasized [4], [2]. However, implementing these suggestions requires modifications to all involved parties’ Hardware Security Modules (HSMs). Commercial solutions such as the `PrivateServer Switch-HSM` [1] rely mostly on ‘tightly’ controlling the key uploading process to a switch and removing ‘unnecessary’ APIs or weak PIN block formats. Even if the flawed APIs are fixed, or non-essential attack APIs are removed to prevent these attacks, it may be difficult in practice to ensure that all intermediate (third-party controlled) switches are updated accordingly. Thus banks rely mainly on protection mechanisms provided within banking standards, and *policy-based* solutions, e.g., mutual banking agreements to protect customer PINs.

One primary reason that PIN cracking attacks are possible is that actual user PINs, although encrypted, travel from ATMs to a verification facility. We seek a solution that precludes real user PINs being extracted at verification

---

<sup>\*</sup> Version: March 10, 2008. Contact author: [mmannan@scs.carleton.ca](mailto:mmannan@scs.carleton.ca).

<sup>1</sup> We encourage readers unfamiliar with financial PIN processing APIs and PIN cracking attacks to consult the longer version of this work [6].

facilities, and especially at switches (which are beyond the control of issuing banks), even in the presence of API flaws. While PIN cracking attacks get more expensive as the PIN length increases, it is unrealistic to consider larger (e.g. 12-digit) user PINs, for usability reasons. As part of our proposal, we assume that a unique random *salt* value of sufficient length (e.g. 128 bits) is stored on a user’s bank card, and used along with the user’s regular four-digit PIN (‘Final PIN’) to generate (e.g. through a pseudo-random function (PRF)) a larger (e.g. 12 digits) *Transport Final PIN* (TFP). This TFP is then encrypted and sent through the intermediate switches. We build our *salted-PIN* solution on this simple idea. We discuss several attacks on salted-PIN, and outline one variant of the original idea which is apparently resistant to currently known attacks. Our proposals require updating bank cards (magnetic-stripe/chip card), service-points (e.g. ATMs), and issuer/verification HSMs. However, our design goal is to avoid changing any intermediate switches, or requiring intermediate switches be trusted or compliant to anything beyond existing banking standards.

Salted-PIN provides the following benefits. (1) It does not depend on policy-based assumptions, and limits existing PIN cracking attacks even where intermediate switches are malicious. (2) It significantly increases the cost of launching known PIN cracking attacks; for example, the setup cost for the translate-only attack for building a complete Encrypted PIN Block (EPB) table now requires more than a trillion API calls in contrast to 10,000 calls as in Berkman and Ostrovsky [2]. (3) Incorporating service-point specific information such as ‘card acceptor identification code’ and ‘card acceptor name/location’ (as in ISO 8583) into a variant of salted-PIN, we further restrict attacks to be limited to a particular location/ATM.

## 2 Salted PIN

Here we present the salted-PIN proposal in its simplest form.

**Threat model and notation.** Our threat model assumes attackers have access to PIN processing APIs and transaction data (e.g. Encrypted PIN Blocks, account number) at switches or verification centers, but do not have direct access to keys inside an HSM, or modify HSMs in any way. Attackers can also create fake cards from information extracted at switches or verification centers and use those cards (perhaps through outsider accomplices). We primarily consider large scale attacks such as those that can extract millions of PINs in an hour [2]. We do not address attacks that are not scalable, such as *card skimming*, or cases where an accomplice steals a card and calls an insider at a switch or verification center for an appropriate PIN. The following notation is used:

- PAN* User’s Primary Account Number (generally 14 or 16-digit).
- PIN* User’s Final PIN (e.g. 4-digit, issued by the bank or chosen by the user).
- PIN<sub>t</sub>* User’s Transport Final PIN (TFP).
- Salt* Long-term secret value shared between the user card and issuing bank.
- $f_{\kappa}(\cdot)$  A cryptographically secure Pseudo-Random Function (PRF).



Assume the resulting EPB is  $E_{p_1p_2}$  which is the same as the one containing a TFP  $p_1p_20000000000$  with PAN 0. Now we can create all EPBs containing every 12 digit TFPs starting with  $p_1p_2$  from  $E_{p_1p_2}$ . For example, an EPB with  $p_1p_2q_3q_4 \dots q_{12}$  as the TFP can be generated through transforming  $E_{p_1p_2}$  using PAN  $q_3q_4 \dots q_{12}00$  (in ISO-0). Thus we can create all  $10^{10}$  EPBs with TFPs from  $p_1p_20 \dots 0$  to  $p_1p_29 \dots 9$ . Starting from a different  $p_1p_2$ , all  $10^{12}$  EPBs containing every 12 digit TFP can be generated.

To launch an attack, a valid EPB of a target customer is collected. The EPB is translated to ISO-1 (to decouple it from the target account, if not already in ISO-1), then to ISO-0 with PAN 0. The resulting EPB is then located on the EPB table (as created in the setup phase). The corresponding PIN from the table can now be used to exploit a card generated with the target’s PAN, and the attacker’s salt value (i.e.  $Salt_a$ ). The cost of this attack is at most two API calls and a search of  $O(10^{12})$ , i.e.,  $O(2^{40})$ .

In summary, the setup cost of this attack is about  $10^{12}$  API calls with a per account cost of two API calls plus a search of  $O(10^{12})$ . The same translate-only attack by Berkman and Ostrovsky [2] on the current implementation of PIN processing requires only about 10,000 API calls as setup cost, and a per account cost of two API calls plus a search of  $O(10^3)$ . More on this attack is discussed in the longer version of this work [6].

**Replay attack.** In this attack, an adversary  $M_i$  at a switch or verification center collects a valid EPB  $E_c$  for a target PAN  $A_c$ , and then creates a fake card with the account number  $A_c$  (and any salt value). Note that  $M_i$  here does not know the actual salt value or PIN for the target account. An accomplice  $M_a$  uses the fake card with any PIN at an ATM, and the ATM generates a false EPB  $E_a$ . At the switch/verification center  $M_i$  locates  $E_a$  in transfer, and replaces  $E_a$  with the previously collected correct EPB  $E_c$ . Thus the fake card will be verified by the target bank, and  $M_a$  can access the victim’s account. Note that this attack works against the basic variant of salted-PIN as well as current PIN implementations without requiring any API calls. Although quite intuitive, this attack has not been discussed elsewhere to our knowledge.

### 3.2 Service-Point Specific Salted-PIN

We now outline one variant of salted-PIN to practically restrict the above attacks by increasing the per account attack cost. If a fake bank card is created for a target account (e.g. through the attacks in Section 3.1), the card can be used from anywhere as long as it remains valid (i.e. the issuing bank does not cancel it). To restrict such attacks, we modify equation (2.1) as follows.

$$PIN_t = f_{Salt}(PAN, PIN, spsi) \quad (3.1)$$

Here  $spsi$  stands for *service-point specific information* such as a ‘card acceptor identification code’ and ‘card acceptor name/location’ as in ISO 8583 (Data Elements fields). The verification center must receive  $spsi$  as used in equation (3.1). Although any PIN cracking attack can be used to learn a TFP or build an EPB table, the table is valid only for the particular values of  $spsi$ . Also, the

replay attack may succeed only when the accomplice exploits a compromised card from a particular ATM. Thus this construct generates a localized TFP for each PIN verification, and thereby restricts the fake card to be used only from a particular location/ATM.

## 4 Conclusion

In the 30-year history of financial PIN processing APIs, several flaws have been uncovered. In this paper, we introduce a *salted-PIN* proposal to counter PIN cracking attacks from Berkman and Ostrovsky [2]. Our preliminary analysis indicates that salted-PIN can provide a higher barrier to these attacks in practice by making them considerably more expensive (computationally). Salted-PIN is motivated primarily by the realistic scenario in which an adversary may control switches, and use any standard API functions to reveal a user’s PIN; i.e., an attacker has the ability to perform malicious API calls to HSMs, but cannot otherwise modify an HSM. Salted-PIN is intended to stimulate further research and solicit feedback from the banking community. Instead of relying, perhaps unrealistically, on honest intermediate parties (who diligently comply with mutual banking agreements), we strongly encourage the banking community to invest efforts in designing protocols that do not rely on such assumptions which end-users (among others) have no way of verifying.

## Acknowledgements

This work benefited substantially from discussion and/or feedback from a number of individuals, including: Bernhard Esslinger of University of Siegen, Joerg-Cornelius Schneider and Henrik Koy of Deutsche Bank, especially regarding attacks on the simple version of salted-PIN; a reviewer from a large Canadian bank; Glenn Wurster; and anonymous reviewers. The first author is supported in part by an NSERC CGS. The second author is Canada Research Chair in Network and Software Security, and is supported in part by an NSERC Discovery Grant, and the Canada Research Chairs Program.

## References

1. Algorithmic Research (ARX). PrivateServer Switch-HSM. White paper. <http://www.arx.com/documents/Switch-HSM.pdf>.
2. O. Berkman and O. M. Ostrovsky. The unbearable lightness of PIN cracking. In *Financial Cryptography and Data Security (FC)*, Scarborough, Trinidad and Tobago, Feb. 2007.
3. M. Bond. Attacks on cryptoprocessor transaction sets. In *Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, Paris, France, May 2001.
4. M. Bond and P. Zielinski. Decimalisation table attacks for PIN cracking. Technical report (UCAM-CL-TR-560), Computer Laboratory, University of Cambridge, 2003.
5. J. Clulow. The design and analysis of cryptographic APIs for security devices. Masters Thesis, University of Natal, Durban, South Africa, 2003.
6. M. Mannan and P. van Oorschot. Weighing down “The Unbearable Lightness of PIN Cracking” (extended version). Technical report, School of Computer Science, Carleton University (2008). [http://www.scs.carleton.ca/research/tech\\_reports/](http://www.scs.carleton.ca/research/tech_reports/).