Authenticating Secure Tokens Using Slow Memory Access

John Kelsey Bruce Schneier
{schneier,kelsey,schneier}@counterpane.com
Counterpane Systems, 101 East Minnehaha Parkway, Minneapolis, MN 55419

Abstract

We present an authentication protocol that allows a token, such as a smart card, to authenticate itself to a back-end trusted computer system through an untrusted reader. This protocol relies on the fact that the token will only respond to queries slowly, and that the token owner will not sit patiently while the reader seems not to be working. This protocol can be used alone, with "dumb" memory tokens or with processor-based tokens.

1 Introduction

Smart cards have been used in many applications that require that the data be secured from the cardholder. In the Modex stored-value smart card system, for example, the smart card stores a particular monetary value. An attacker who can successfully change the value stored in the card can essentially print money.[1] In GSM phones, smart cards provide identity information used to prevent cloning and in billing. An attacker who can modify that information can charge cellular phone calls to another account [BGW98]. Smart cards used to protect satellite television can be attacked to obtain free services [McC96].

These cards are vulnerable to a large class of attacks [And94, Sch97, Row97, Sch98], from reverse-engineering [AK96] and protocol failures [AN95] to side-channel attacks [Koc96, Koc98, KSWH98, DKL+99] and fault analysis [BDL97, BS97]. In all of these attacks, the attacker can defeat the security of the card by breaching its secure perimeter and learning the confidential information stored within. With reverse-engineering, the attacker defeats the tamper-reisistance measures directly; with side-channel and fault-analysis attacks, the attacker exploits weaknesses in the physical security to learn information within the secure perimeter.

[1]For other examples, see [CP93], [SK97], and [RKM99].

There is another class of applications for smart cards—applications in which the value of a successful attack is much smaller. These cards might provide micropayment information, low-security access control, or act as payment vehicles in circumstances with low marginal cost of goods (pay television, public transportation, etc). In these applications we are less concerned with individual fraud, and more concerned with an organized attack to create and distribute fake cards. Someone who sneaks onto the subway or watches a satellite movie without paying isn't going to affect the service provider's bottom line, but someone who is able to counterfeit access tokens that allow everyone to sneak onto the subway or watch the movie could collapse the entire system. Hence, the primary threat is not from an isolated attack against a single card, but an attack that can be scaled to multiple cards.

The threat is from untrusted card readers that the user may stick his card into. In this paper, we propose a low-tech authentication protocol that is useful in this sort of situation. Our protocol makes use of what we will call a "slow memory device": a device that responds to queries by revealing the contents of different memory locations, but one that necessarily takes several seconds to do so.

The protocol relies on the fact that the cardholder does not have infinite patience. If he puts his smart card into a reader and nothing happens for several seconds, he will likely pull the card out and try again. If nothing happens again, he will find another reader. The slow response of the card ensures that a fraudulent reader can only do so much damage before the cardholder removes his card.

This protocol makes no cryptographic assumptions, and is independent of any cryptography that may or may not be in the system. It can be implemented by itself, or in conjunction with cryptographic controls.

The rest of the paper is organized as follows. In Section 2 we describe the slow memory device and its functionality. In Section 3, we describe our authentication protocol. In Section 4 we discuss variants and extensions to this basic protocol, and in Section 5 we discuss applications.

2 The Slow Memory Device

This protocol assumes the existence of a slow memory token. By this we mean a token—a smart card, a Dallas Semiconductor iButton, etc.—that acts as a memory device, but never responds to a request in less than t seconds ($t = 10$, for the purposes of this example). It is impossible for a terminal to get more information during that time; the token's electronics are such that it simply cannot respond to requests faster.

This memory token has m memory locations, each w bits wide ($w = 64$ for the purposes of this example). The token does not need a processor, nor does it need to implement any cryptographic primitives in order to execute this protocol.

3 The Basic Protocol

There are three parties in this protocol:

- **The Token:** The slow memory device.
- **The Terminal:** The untrusted card reader.
- **The Trusted Machine:** A trusted computer connected to, and possibly remote from, the Terminal.

Our authentication protocol is simple. A user inserts his Token into a Terminal. The Terminal now needs to prove to a Trusted Machine that the Token is currently inserted into the Terminal.

The Terminal is only marginally trusted, and could be malicious. In order to complete the protocol correctly the Terminal must be connected, via a secure data link, to a Trusted Device. The Trusted Device keeps an exact copy of the contents of the Token; this is a shared secret that the Terminal does not know.

Our protocol is as follows:

(1) The holder of the Token inserts it into the Terminal.

(2) Terminal reads the header information from the Token.

(3) Terminal sends this information over an encrypted link to the Trusted Device.

(4) Trusted Device generates a random challenge and sends it back over the encrypted link to the Terminal. This challenge consists of a list of n memory locations on the Token.

(5) Terminal reads the n memory locations from the Token, XORs them all together, and sends the result back over the encrypted link to the Trusted Device.

(6) The Trusted Device verifies this information; if it checks out, it believes that the Token is currently inserted into the Terminal.

Note that there are no cryptographic primitives in the protocol: the only mathematical operation involved is XOR. There are no encryption functions, one-way hash functions, or message authentication codes used in the protocol.

The Token might have 1000 64-bit memory locations. Each time read request comes in, it takes ten seconds to respond, and without reverse-engineering the device and tampering with its internals, this can't be made any faster. Assuming ten seconds per communications exchange, and setting $n = 1$, steps (2) and (3) take ten seconds, step (4) takes another ten seconds, and step (5) takes another ten seconds. This gives us a total of 30 seconds per transaction. Now, this can be extended a little bit without the user knowing. A malicious Terminal might ignore the protocol completely, and simply query the Token (repeat step (5)). Maybe the Terminal can perform six queries instead of one, doubling the transaction time, before the Token owner removes his card.

To provide adequate security, we need to account for possible malicious behavior by the Terminal in how many transactions are allowed, keeping the probability of success acceptably low even if most Terminals are corrupt.

3.1 Reasonable Values for n

Suppose the Token has m memory locations, that each instance of the protocol requests the XOR of n random locations, and that $n << m$. Assuming an attacker eavesdrops on each authentication, he will learn the contents of n memory locations, not all of them different, after each transaction. The average number of authentications that the Token can accomplish before the attacker has a better than

0.5 chance of impersonating the Token (that is, being able to respond correctly to a random query), is:

$$log(n)/(log(m) - log(m - n))$$

For example, for $m = 1000$ and $n = 5$, an attacker who eavesdrops on 322 authentications has a better than 0.5 probability of being able to impersonate the Token by answering a random challenge correctly.

This, of course, is not the most effective attack. A fraudulent Terminal will specifically query the Token to learn the memory locations that it does not know. Hence, a malicious Terminal can learn the entire contents of a Token in m/n queries. So for the parameters above, a malicious terminal that conducts 100 fraudulant transactions will have a better than 0.5 probability of being able to impersonate the Token. The Token owner, though, would have to be convinced to allow the Token to be used in 100 ineffectual transactions.

4 Extensions

4.1 A Button on the Token

Ideally, we'd have some user-interface mechanism on the Token, like a pushbutton. The Token is willing to perform only once per button-push. Alternatively, the Token buzzes or lights up once per memory query answered. This would make multiple queries much harder for the Terminal to make, since the Token owner could detect that the protocol was not proceeding as specified.

4.2 Reducing Storage Requirements in the Trusted Device

As written, the Trusted Device must store a complete copy of the Token's memory location. If this is too much memory, the Trusted Device could store the Token's ID information and a secret key known only to it (and not the token). The Token's memeory locations would then be the memory address encrypted with this secret key, and would have to be loaded onto the Token by the Trusted Device.

4.3 CRC Hardware on the Token

If the Token can afford CRC hardware, then queries can be handled using this alternate protocol:

(1) The holder of the Token inserts it into the Terminal.

(2) Terminal reads the header information from the Token.

(3) Terminal sends this information over an encrypted link to the Trusted Device.

(4) Trusted Device generates a random challenge and sends it back over the encrypted link to the Terminal. This challenge consists of a list of n memory locations on the Token.

(5a) The Terminal sends the Token a request for the n memory locations.

(5b) The Token goes through the motions (and delay) of sending it out internally, but only outputs the 32-bit CRC of the n requested memory locations.

(6) The Trusted Device verifies this information; if it checks out, it believes that the Token is currently inserted into the Terminal.

Each memory location can now be 32 bits long, and even one unknown memory location in the query string prevents an attacker from succeeding in an impersonation attack. Note that this system works best if there are lots of Terminals under different entities' control. If a Token only interacts with one Terminal every time it executes the protocol, then this system doesn't work very well.

4.4 Incrementing Values in the Token and Trusted Device

If we're worried about an attacker reverse-engineering and making lots of copies of the Token, then we have each query of memory location cause that memory location to increment by one, modulo 2^{32}, or rotate left one bit, or whatever else is cheap enough to be implemented. Note that this update must occur on both the Token and the Trusted Device, and assumes that there is either only one Trusted Device or that the different Trusted Devices can communicate with each other securely to synchromize these updates.

This variant does not help against impersonation attacks. What it does do is to make continued synchronization of those counterfeit Tokens very expensive and complex. Now, if any memory location in the challenge string has been changed, the duplicated Token fails to give the correct answer.

4.5 Reducing the Amount of Trust in the Trusted Device

A given Trusted Device may be only partially trusted. Instead of having it store a complete listing of the Token's memory locations, it can be given a series of precomputed challenges in order and the right responses to be expected. (If we use the previous extension, this will work only for small numbers of different Trusted Devices.)

5 Security Analysis

The slow memory protocol is designed to frustrate a particular kind of attack. It is intended to provide security against an insecure reader trying to collect enough information from a token to be able to impersonate it. It does not provide security against someone reverse-engineering the token and cloning it (although the extension described in Section 4.2 considerably frustrates that attack in most circumstances), nor does it provide security in the event that the back-end database (the Trusted Device in the protocol) is compromised.

A more extensive security analysis will be in the final paper.

6 Applications

A complete discussion of applications, along with their security ramifications, will be in the final paper.

The most obvious applications are things like non-duplicable keys for locks and alarms, free passes or one-day (or k-day) coins, and login keys. In these applications, we are less concerned with a single person hacking the authentication device than the same single person being able to distribute a large number of them.

One of the most beneficial aspects of this protocol is that it works well with cryptography, even though it has no cryptography itself. We can use a message authentication code such as HMAC [BCK96] in addition to this protocol, and if the MAC breaks, the memory trick still works. Or course, all Trusted Devices must be trusted with copies of the memory maps.

7 Conclusions

Security countermeasures must be commensurate with the actual threats. In this paper we have presented a low-tech security solution that helps mitigate a specific threat, and can be used in conjunction with other cryptographic countermeasures.

8 Acknowledgments

The authors would like to thank Chris Hall for his helpful comments. Additionally, the authors would like to thank Niels Ferguson, who broke one of our MACs and subsequently inspired this research.

References

[And94] R. Anderson, "Why Cryptosystems Fail," *Communications of the ACM*, v. 37, n. 11, Nov 1994, pp. 32–40.

[AK96] R. Anderson and M. Kuhn, "Tamper Resistance – A Cautionary Note," *Second USENIX Workshop on Electronic Commerce Proceedings*, USENIX Press, 1996, pp. 1–11.

[AN95] R. Anderson and R. Needham, "Programming Satan's Computer," *Computer Science Today: Recent Trends and Developments*, LNCS #1000, Springer-Verlag, 1995, pp. 426–440.

[BCK96] M. Bellare, R. Canetti, and H. Karwczyk, "Keying Hash Functions for Message Authentication," *Advances in Cryptology — CRYPTO '96 Proceedings*, Springer-Verlag, 1996, pp. 1–15.

[BDL97] D. Boneh, R.A. Demillo, R.J. Lipton, "On the Importance of Checking Cryptographic Protocols for Faults," *Advances in Cryptology—EUROCRYPT '97 Proceedings*, Springer-Verlag, 1997, pp. 37–51.

[BGW98] M. Briceno, I. Goldberg, D. Wagner, "Attacks on GSM security," work in progress.

[BS97] E. Biham and A. Shamir, "Differential Fault Analysis of Secret Key Cryptosystems," *Advances in Cryptology—*

CRYPTO '97 Proceedings, Springer-Verlag, 1997, pp. 513–525.

[CP93] D. Chaum and T. Pederson, "Wallet Databases with Observers," Advances in Cryptology — CRYPTO '92 Proceedings, Springer-Verlag, 1993, pp. 391–407.

[DKL+99] J.-F. Dhem, F. Koeune, P.-A. Leroux, P. Mestre, J.-J. Quisquater, and J.-L. Willerns, "A Practical Implementation of the Timing Attack," *CARDIS '98 Proceedings*, Spriger-Verlag, 1999, to appear.

[Koc96] P. Kocher, "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems," *Advances in Cryptology—CRYPTO '96 Proceedings*, Springer-Verlag, 1996, pp. 104–113.

[Koc98] P. Kocher, "Differential Power Analysis," available online from `http://www.cryptography.com/dpa/`.

[KSWH98] J. Kelsey, B. Schneier, D. Wagner, and C. Hall, "Side Channel Cryptanalysis of Product Ciphers," *ESORICS '98 Proceedings*, Springer-Verlag, 1998, pp. pp 97–110.

[McC96] J. McCormac, *European Scrambling Systems*, Waterford University Press, 1996.

[RKM99] C. Radu, F. Klopfert, and J. De Meester, "Interoperable and Untraceable Debit-Tokens for Electronic Fee Collection," *CARDIS '98 Proceedings*, Springer-Verlag, 1999, to appear.

[Row97] T. Rowley, "How to Break a Smart Card," The 1997 RSA Data Security Conference Proceedings, RSA Data Security, Inc., 1997.

[Sch97] B. Schneier, "Why Cryptography is Harder than it Looks," *Information Security Bulletin*, v. 2, n. 2, March 1997, pp. 31–36.

[Sch98] B. Schneier, "Cryptographic Design Vulnerabilities," *IEEE Computer*, v. 31, n. 9, September 1998, pp. 29–33.

[SK97] B. Schneier and J. Kelsey, "Remote Auditing of Software Outputs Using a Trusted Coprocessor," *Journal of Future Generation Computer Systems*, v.13, n.1, 1997, pp. 9–18.