Breaking Real-World Implementations of Cryptosystems by Manipulating their Random Number Generation [*]

Yuliang Zheng
The Peninsula School of Computing and Information Technology
Monash University, McMahons Road, Frankston
Melbourne, VIC 3199, Australia
Email: `yzheng@fcit.monash.edu.au`
Fax: +61 3 9904 4124 Tel: +61 3 9904 4196

Tsutomu Matsumoto
Division of Artificial Environment Systems and
Division of Electrical and Computer Engineering
Yokohama National University
79-5 Tokiwadai, Hodogaya, Yokohama 240, Japan
Email: tsutomu@mlab.dnj.ynu.ac.jp
Fax: 045-338-1157 Tel: 045-338-1171/1173

May 6, 1997

Abstract

We suggest several methods that may allow one to completely break cryptosystems implemented in a portable hardware device such as a smart card. These cryptanalytic methods work by exploring the hardware in such a way that the pseudo-random number generator embedded in the device behaves predictably, even for a very short period of time. While these attacking methods will be discussed by standing in the position of an attacker, our genuine intention is to serve as an alarm for users, developers and researchers in information security.

[*]In *the Proceedings of the 1997 Symposium on Cryptography and Information Security*, pp.6B1-6, January 29 - February 1, 1997, Fukuoka, Japan. A less polished version entitled "Breaking Smart Card Implementations of ElGamal Signature and Its Variants" was presented at the rump session of Asiacrypt'96, Kyongju, Korea, November 5, 1996.

1 Cryptanalysis by Exploring Hardware Faults

On 25 September, 1996, Bellcore announced that D. Boneh, R. DeMillo (both of Bellcore) and R. Lipton (of Princeton University) have found that RSA signature generation/decryption key that is stored in a "tamper-proof" device such as a smart card may be extracted by an attacker who is in possession of the device [4]. The media release explains that the attack works by subjecting the tamper-proof device to certain types of physical stresses that would cause it to generate faulty computations. Technical details on the attack were later published in [7]. Prior to this, on 23 October 1996, six researchers from the National University of Singapore demonstrated in detail how an attacker may extract a secret RSA key stored in a "tamper-proof" device, again by subjecting the device to physical stresses [3].

The above attacks have been considered specifically for smart card implementations of public key cryptosystems (including decryption, signature, authentication and identification). A few weeks after Bellcore's media release, however, Biham and Shamir announced, with sufficient technical details, that practically all smart card implementations of private key cryptosystems (such as DES, IDEA, FEAL, ...) may be broken within the same attack model where an attacker may introduce hardware faults into a "tamper-proof" device [6]. In less than two weeks' time, Biham and Shamir pushed their cryptanalysis further to show that hardware faults may allow one to break a smart card implementation of a private key cryptosystem whose algorithmic details are not public (for instance, the Skipjack cryptosystem) [5]. A further improvement to their attack can be found in [1].

These attacks work primarily under the assumption that a computation process within a smart card may result in a faulty outcome, or an attacker may somehow flip a few bits in a secret key stored in a register. In this paper, we take a different approach. In particular, we concern ourselves with pseudo-random number generation which is a core part of most cryptosystems. Our cryptanalytic methods work by exposing a device to physical stresses in such a way that a pseudo-random number generator (PRG) embedded in the device behaves predictably. Although our discussions will be mainly about smart card implementations of ElGamal digital signature scheme, they can be easily generalized to virtually all cryptosystems that rely on randomness for the assurance of their security.

Some more recent advances in the above lines of cryptanalysis have been found in [8].

2 ElGamal Signature and Its Variants

ElGamal digital signature is based on the hardness of computing discrete logarithm over a large finite field. It involves two parameters public to all users:

1. p: a large prime.
2. g: an integer in $[1, \ldots, p-1]$ with order $p-1$ modulo p.

User Alice's secret key is an integer x_a chosen randomly from $[1, \ldots, p-1]$ with $x_a \not| \, (p-1)$ (i.e., x_a does not divide $p-1$), and her public key is $y_a = g^{x_a} \bmod p$.

Alice's signature on a message m is composed of two numbers r and s which are defined as

$$\begin{aligned} r &= g^x \bmod p \\ s &= \frac{hash(m) - x_a \cdot r}{x} \bmod (p-1) \end{aligned}$$

where x is a random number from $[1, \ldots, p-1]$ with $x \not| \, (p-1)$. It should be stressed that x must be chosen independently at random every time a message is to be signed by Alice.

Given (m, r, s), one can verify whether $g^{hash(m)} = y_a^r \cdot r^s \bmod p$ is satisfied. (r, s) is regarded as Alice's valid signature on m only if the equation holds.

ElGamal signature scheme has been scrutinized extensively in the past decade. It has also been generalized to numerous variants, including the notable DSA and Schnorr signature scheme.

3 Breaking ElGamal Signature by Controlling the Pseudo-Random Number Generator

Some of the attacks, such as those in [3, 1], work under the assumption that an attacker can directly flip bits in an unknown secret decryption/signature generation key. In this work we take a different approach. In Particular, we focus on the generation of a pseudo-random number x, a crucial part of the signature scheme.

We consider two types of ElGamal signature cards:

1. smart cards that use a piece of program that is known to the public to generate pseudo-random numbers (smart cards with a software PRG).
2. smart cards that have a built-in hardware pseudo-random number generator (smart cards with a hardware PRG).

3.1 Cards with a Software PRG

For a smart card with a software PRG, it must have a status register to store data on its current status information S_{info}. To produce a fresh pseudo-random number, normally the piece of program would first fetch the current data S_{info} from the status register, calculate a number x from S_{info}, update the contents in the status register, and finally output x as an outcome. For an attacker who is in possession of User Alice's ElGamal card with a software PRG, the first thing he would do is to identify the location of the status register in a chip embedded in the card. This could be done by searching through public literature (some chip manufacturers publish the layout of chips), or simply assuming that the register would be located in the lower end of RAM (storing key or other important information in low address locations in RAM is apparently a practice adopted by many programmers [1]).

The attacker would proceed to expose that particular part of the smart card (i.e., the status register) to certain physical stresses (such as laser, focused heating or radiation etc). This would suppress the original S_{info} in the status register, and force it to temporarily turn into a constant data, say the all-one value. While the physical pressure is still effective, the attacker may supply the smart card with a message m and ask it to sign on the message. Assume that (r, s) is the corresponding signature produced by the smart card.

Now the attacker would be able to extract Alice's secret signature generation key x_a as follows:

1. calculates x_0 from the all-one value using the public algorithm for PRG.
2. extracts x_a:
$$x_a = \frac{hash(m) - s \cdot x_0}{r} \bmod (p-1)$$

3.2 Cards with a Hardware PRG

If Alice's signature card has a built-in hardware PRG, the attacher could use a technique similar to the one discussed above to force the output the PRG to turn into a known number x_0, say the all-one value. Some smart cards with a built-in hardware PRG have the following property: when they are exposed in certain "abnormal" physical environments, such as being supplied with a lower-than-normal voltage, their PRG would produce a predictable output x_0 [2].

Either case would lead to the immediate breaking of the signature scheme:

$$x_a = \frac{hash(m) - s \cdot x_0}{r} \bmod (p-1)$$

3.3 Relaxing the Assumption

In the above discussions we have assumed that the attacker may know directly the output x_0 of a software or hardware PRG. This requirement can be weakened if the attacker can ask a smart card to sign on two different messages. Assume that the attacker can force the PRG to produce a fixed, but unknown, output x_0. While maintaining the stresses on the card, the attacker asks it to sign on two different messages m_1 and m_2. He would get

$$(r_0 = g^{x_0} \bmod p, s_1 = \frac{hash(m_1) - x_a \cdot r_0}{x_0} \bmod (p-1))$$

and

$$(r_0 = g^{x_0} \bmod p, s_2 = \frac{hash(m_2) - x_a \cdot r_0}{x_0} \bmod (p-1))$$

as Alice's signatures on m_1 and m_2 respectively. From these data, the attacker would be able to find out

$$x_0 = \frac{hash(m_1) - hash(m_2)}{s_1 - s_2} \bmod (p-1)$$

and hence Alice's secret signature generation key x_a

$$x_a = \frac{hash(m_1) - s_1 \cdot x_0}{r_0} \bmod (p-1)$$

We believe that from an attacker's point of view, the two-message attack is more effective and easier to carry out than one that requires the attacker to force a PRG to produce a known outcome.

The requirement on forcing the PRG to produce a fixed but unknown output can be further weakened. More specifically, we can consider a situation where an attacker might be able to force the PRG to produce "significantly" biased outcomes, namely, to manipulate the PRG so that it produces some outcomes (say all $x_0 < 10,000$) more often than the others. With such a smart card, the task to extract the secret key x_a is essentially reduced to a statistical experiment: by supplying the smart card with a sufficient number of different messages m_i, the attacker would be bound to eliminate the equivocalness surrounding the secret key x_a.

3.4 Further Extension

The cryptanalytic methods presented above are also applicable to all the variants of ElGamal signature scheme, including those based on elliptic curves. In addition, the general idea of manipulating a PRG in a smart card can also be readily adapted to compromise smart card implementations of other types of cryptographic primitives that rely on pseudo-random numbers for their security. These primitives include many types of digital signature schemes, authentication and identification protocols.

We also note that the attacks described above are not limited to hardware implementations. A cryptosystem implemented in software or in a combination of hardware and software may be attacked in a similar way, if an attacker could manage to manipulate pseudo-random number generation employed by the system.

4 The Importance of Un-Compromiseable Pseudo-Random Generation

While attacks using register faults or computation process errors, as is the case for cryptanalytic methods discussed in [7, 6] and other related papers, may be foiled by using hardware fault tolerance technology and result verification, malicious manipulation of pseudo-random number generation may be more difficult to handle, as it cannot be detected by fault tolerance circuitry or result verification alone. Technically, embedding an un-compromiseable pseudo-random number generator into a LSI chip seems as challenging as making a truly tamper-resistant LSI chip.

This observation is closely related to a fundamental gap between a security solution in an abstract mathematical model and its implementation using real-world technology. While the abstract solution may have some desirable features such as provable security, these features would rely on a number of mathematical or non-mathematical assumptions, such as the pseudo-randomness of sequences employed and the intractability of certain underlying mathematical problems. The abstract model would tend to capture mathematical assumptions better than those non-mathematical. Examples of non-mathematical assumptions include that a secret key algorithm or signature generation procedure can be executed uninterruptedly with no intermediate results or partial keys being leaked, that modular exponentiation can be done in a constant fraction of time regardless of the size of an exponent, that a secret key can be truly kept secret, that a pseudo-random number generator behaves consistently and honestly, to name just a few. The breaking-down of a non-mathematical assumption in a real-world implementation would lead to the breaking-down of the security solution: timing attacks and hardware fault attacks are perhaps the best examples to support this assertion. Indeed, an important question is whether and how the gap between an abstract mathematical model and real-world technology can be bridged at all.

References

[1] R. Anderson. A serious weakness of DES. (posted to cypherpunks@toad.com, 2 November 1996).

[2] R. Anderson and M. Kuhn. Tamper resistance — a cautionary note. In *Proceedings of the 1996 Usenix Electronic Commerce Conference*, November 1996.

[3] F. Bao, R. Deng, Y. Han, A. Jeng, T. H. Nagir, and D. Narasimhalu. A new attack to RSA on tamperproof devices. (posted to cypherpunks@toad.com, 23 October 1996).

[4] Bellcore. Now, smart cards can leak secrets. (press release, available at http://www.bellcore.com/PRESS/ADVSRY96/smrtcrd.html, 25 September 1996).

[5] E. Biham and A. Shamir. The next stage of differential fault analysis: How to break completely unknown cryptosystems. (email message to a group of researchers including one of the present authors, 30 October 1996).

[6] E. Biham and A. Shamir. Research announcement: A new cryptanalytic attack on DES. (posted to cypherpunks@toad.com, 18 October 1996).

[7] D. Boneh, R. DeMillo, and R. Lipton. On the importance of checking computations. (extended abstract, available at http://www.bellcore.com/SMART/, 31 October 1996).

[8] S. Moriai. A fault-based attacks of rc5. In *Proceedings of the Workshop on Design and Evaluation of Cryptographic Algorithms*, pages 117–126, Kikai Shinko Kaikan, Tokyo, Japan, November 1996.