"All Your Keybit Are Belong To Us" The Truth about Blackbox Cryptography

Rüdiger Weis[1] and Stefan Lucks[2]*

[1] cryptolabs Amsterdam
ruedi@cryptolabs.org
[2] Theoretische Informatik, University of Mannheim,
lucks@th.informatik.uni-mannheim.de

Abstract. In this paper we discuss techniques of stealing key-bits using black box cryptography. We start with simple techniques misusing e.g. "random" initial vectors (IV) or padding. In the main part we discuss a new secret-key cryptosystems leaking information to their designers. Internally, an attacker can use the secret key cryptosystems as a public-key cryptosystem under a hard-coded public key. Reverse-engineering might discover the information leakage, but it cannot be exploited to learn anything about the keys or plaintexts. On the other hand, the system designer (or rather the owner of the secret key) can easily decrypt cipher texts. We conclude that before using a cryptographic device, its design has to be evaluated carefully. Even software implementations can be risky, especially if the source code and design documents are not open for inspections.

Key words:
secret-key cryptography, trusted hardware, black-box encryption, trapdoors, open source.

* Supported by DFG grant Kr 1521/3.

1 Introduction

The research in trapdoors for black-box ciphers has been boosted by the US governments "clipper" initiative. Apart from clipper's key escrow [4], people speculated about additional trapdoors. Even after the failure of clipper, possible trapdoors in black-box devices remain a threat: many – if not most – commercial encryption packages come to their users as black boxes, either as hardware devices, or as software packages without source code. And the situation might get even worse since there are plans to integrate blackbox systems in *almost all* digital devices (e.g. for "legal interception" or "digital rights management").

We want to show in this paper that there is a wide variety of misusing blackbox systems. Especially we want to demonstrate the possibility of systems provide "security" for malicious blackbox designer even in a strong cryptographical model.

2 Reengenering Secure Kleptography

When using a (secure) cryptosystem, one expects no information leakage about either plaintexts or keys. But it may be in the interest for a cryptosystem-designer (say "Dora") to gather such information by eavesdropping. Thus, cryptographers have long been on guard about trapdoors hidden in cryptosystems.

Young and Yung [8] have an approach called "MONKEY"

- Distinguish between three entities:
  - the designer Dora,
  - the reverse engineer,
  - the ordinary user.
- Dora can recover the secret key and then decrypt the ciphertexts.
- For the ordinary user, the encryption scheme appears like an ordinary secret-key cipher, i.e., the trapdoor is hard to detect.

The MONKEY methodology provides block ciphers where Dora can find the key via a known plaintext attack. The reverse engineer, say "Richie", can't learn anything about the key (provided the cryptographic primitives used by MONKEY are secure). But ciphertext blocks, leak some information about the plaintext to Richie. ([8] mention this. The leakage is limited to one bit of information per block of plaintext.[1]

A interesting additional requirement is that a reverse engineer can detect the existence of a trapdoor, but gains no further advantage[2]. Here, we deal with ciphers similar to MONKEY, but we hinder Richie from gathering any information about either plaintexts or keys.

Black box encryption means that the user is unaware of what the encryption scheme (either a in hardware or in software) really does. Reverse engineering may be possible.[3]

Below, we assume that Dora is tempted to build a trapdoor into her cryptosystem (or is "persuaded" to do so). We also assume that Dora hesitates to weaken the cryptosystem's security against reverse engineers such as Richie. In principle, this is public-key cryptography. The trapdoor should not be too obvious, which rules out trivial solutions (adding an encryption of the key under Dora's public key, evaluating a hidden public-key operation for every secret-key encryption operation, ...).

3 Related Work and Open Questions

Blaze, Feigenbaum, and Leighton [2] discuss the construction of secret-key trapdoor ciphers. They point out the equivalence of such cryptosystems and public-key cryptosystems: Dora's design is her public key. A message M is sent to Dora by choosing a key K, encrypting M under K and sending the ciphertext C. Now Dora can recover M from C, but, if the construction is secure, only Dora and those who know K.

The authors of MONKEY [8] also considered designing *public-key* cryptosystems leaking information subliminally and securely [5–7].

[1] Also, MONKEY depends on block cipher with a non-standard block size (e.g., 63 bit) as a tool to realize a block cipher with a standard block size (e.g. 64 bit) leaking the key.

[2] MONKEY[8] violates this.

[3] Even so called "tamperproof" devices often provide poor protection against reverse engineering [1, 9].

They introduced the notion of a SETUP ("Secretly Embedded Trapdoor with Universal Protection"). A secretly embedded trapdoor (a public key) is used to leak secret information without the user's knowledge. [5–7] exploit the probabilistic nature of the underlying public-key schemes. This methodology is not very useful in the case of (deterministic) secret-key encryption.

Rijmen and Preneel [3] describe block ciphers with embedded trapdoors. They hide some structure in large S-boxes, presumably generated at random. (The main point of [3] is to be alert if a cipher uses fixed "random" values.) The S-boxes are difficult to distinguish from truly random S-boxes, but the designer exploits the structure for linear or differential attacks. Hence, Richie can hardly find out that a trapdoor exists, quite in contrast to MONKEY and to our approach.[4]

Open Questions

The previous approaches leave especially two questions:

1. *How hard is it, to find the trapdoor?*
   Recall that – as stressed in [3], too – a trapdoor cryptosystem can in principle be used as a public-key cryptosystem. Many promising ideas for public-key cryptosystems have been proposed, and most of them where broken later. (Also, if the designer mounts a chosen plaintext attack, the transcript of the attack might might help the reverse engineer to detect the trapdoor.)
2. *Are there attacks on the cipher without exploiting the hidden trapdoor?*
   It is usual in cryptography, to distrust a new cipher until it has withstood years of intense cryptanalysis. How much cryptanalytic attention will the above question catch, for a cipher known to be designed with a hidden trapdoor (and not much used, for this reason)?

The current paper is less ambitious than [3]. Richie may well detect the trapdoor – but not use it for his own purposes. The current

[4] But since – as stressed in [3], too – a trapdoor cryptosystem can in principle be used for public-key encryption, [3] describe a new kind of public-key encryption scheme, and its security deserves much further study.

paper is also more ambitious, defining generic schemes, i.e. use arbitrary primitives as cryptographic tools (say, a block cipher and a public-key cryptosystem). We argue that our schemes are secure if the primitives are secure.

4 Randomness and Subliminal Channels

Many protocol designers think that it is a golden rule to bring in randomization into protocols. But if the encryption device chooses "random values" on its own, it may be trivial for the cryptosystem-designer to establish a subliminal channel.[5]

It is easy to see that e.g. the random padding in the new PKCS#1 versions (≥ 2.0) can be used to to transmit many bits in each public key encrypted message. Think about the normal hybrid encryption scenario where a usually 128 bit session key is transmitted with at least 1024 bit RSA key.[6]

Sometimes, though, even deterministic secret key encryption needs to make use of randomness. E.g., usual standard modes of operation for an n-bit block cipher such as *cipher block chaining*, *cipher feedback*, and *output feedback* depend on a n-bit "initialization vector" (IV), often transmitted in the clear. In the remainder of this section, we give a concrete example and describe a device generating n-bit "random" IVs to establish a subliminal channel.

Let K' be an additional secret key, hardwired into the device. The IV is generated as follows:

1. let $Y = E_{\text{pub}}(K)$; by $y_1, \ldots, y_m \in \{0,1\}$ we denote the m bits of Y (we assume $m > n$, e.g., $m = 1024$);
2. choose $n-1$ random bits $(r_1, \ldots, r_{n-1}) \in \{0,1\}^{n-1}$;
3. depending on $(r_1, \ldots, r_{n-1})$ and on K', pseudorandomly generate $(z_1, \ldots, z_m) \in \{0,1\}^m$;
4. compute $p = \bigoplus_{1 \leq i \leq m} z_i y_i$ and use $(r_1, \ldots, r_{n-1}, p) \in \{0,1\}^n$ as IV.

[5] Note that there are a lot of additional problems if blackbox devices are used for key generation.

[6] In the case of DLP algorithms (e.G. ElGamal) there is even more "random" involved in the protocols.

Since Dora knows K', she can compute $(z_1, \ldots, z_m)$ from $(r_1, \ldots, r_{n-1})$. With, say, $2m$ independent IVs $(r_1, \ldots, r_{n-1}, p)$, she gets $2m$ vectors $(z_1, \ldots, z_m) \in \{0,1\}^m$.

With high probability, m of these $2m$ vectors are linearly independent, which allows Dora to compute $(y_1, \ldots, y_m) = Y = E_{\text{pub}}(K)$ by solving a system of linear equations. Given Y, she can find K.

Note that knowing a couple of ciphertexts (or actually a couple of IVs) suffices for Dora to determine the key K.

The Security of the Proposed Scheme

Recall that the bit p is determined by K' and $(r_1, \ldots, r_{n-1})$. If, for two IVs generated independently according to the above scheme, the $n-1$ bits $r_1, \ldots, r_{n-1}$ collide, then the n-th bit p collides, too. Hence, the probability to get colliding IVs increases slightly. For $n \geq 128$, this effect is hard to detect.

The inner products p are indistinguishable from random bits for anyone not knowing K', except when two independently generated strings $(r_1, \ldots, r_{n-1})$ collide. (Here, we assume that the pseudorandom generator used in Step 3 is secure.) K' is only known to Dora – and possibly to Richie. Without reverse engineering it is hard to find out that the system leaks key bits to Dora.

K' can be found by reverse engineering. Given m IVs, it is possible for Richie to learn Y. But if E_{pub} is secure, he cannot find K.

5 Two Reengeneering Secure Stream Ciphers

A *stream cipher* is a function

$$f : \{0,1\}^k \times \mathbb{N} \to \{0,1\}^*.$$

Given a key

$$K \in \{0,1\}^k \text{ and } n \in \mathbb{N},$$

f produces an n-bit string

$$f_K(n) \in \{0,1\}^n.$$

We say, f is *secure* if it is infeasible for every adversary to distinguish $f_K(n)$ from a random n-bit without knowing K. (More general definitions of stream ciphers exist, but are not considered here.)

Given a n-bit plaintext P and a key K, the ciphertext is the bitwise XOR

$$f_K(n) \oplus P.$$

We call $f_K(\cdot) \in \{0,1\}^*$ the *cipher stream.*

Public Key Encryption

We also need a *public encryption algorithm* E_{pub}. Given $X \in \{0,1\}^k$ and depending on Dora's public key, E_{pub} generates a value $Y \in \{0,1\}^m$; we write $Y := E_{\text{pub}}(X)$. Using her secret key, Dora can recover X from Y.

E_{pub} may be either deterministic or probabilistic. If both

$$Y := E_{\text{pub}}(X) \text{ and } Y' := E_{\text{pub}}(X),$$

then $Y = Y'$ is *not* guaranteed if E_{pub} is probabilistic.

Consider an adversary A knowing E_{pub}, including Dora's public key, but not knowing Dora's private key. The public encryption algorithm E_{pub} is *secure*, if it is infeasible for A to distinguish the values Y generated by E_{pub} from random values in $\{0,1\}^m$, assuming the corresponding inputs X and – if applicable – the internal coin flips of E_{pub} are unknown to A.

Example

We describe a public-key cryptosystem satisfying our demands. Set $m = 1024$ and let N be a 1024-bit RSA modulus. We first stretch the k-bit input X to a 1024-bit value X_{stretch}, "looking like" a random value in $\mathbb{Z}_N$. This can be done e.g. by using X as the seed to pseudorandomly generate $m - k - 1$ bits (the most significant bit of X_{stretch}, is set to 0). We RSA-encrypt X_{stretch} and get a ciphertext $Y_N \in \mathbb{Z}_N$. To get a uniformly distributed value $Y \in \{0,1\}^m$, we apply the "probabilistic bias removal" method from [7] on Y_N.

Other choices of public-key encryption algorithms would do the job as well, e.g. Rabin encryption to save encryption time, or an Elliptic-Curve based cryptosystems such as SMALL1 [8] to keep m small.

Most of our constructions use a block or stream cipher under a key K *and* a public encryption operation to encrypt K. We assume the secret-key cipher and the public encryption algorithm to **behave independently**, i.e. we assume the infeasibility of finding any statistical relationship between secret-key encryption under an unknown key K and the public encryption of K. [7]

SETUP and QUASI-SETUP

Next, we need to define what we want to realize with our tools. Our formal approach bears much resemblance to the notions of a SETUP [6] and a QUASI-SETUP [8].

A **secret-key SETUP** is a symmetric black-box cryptosystem C (inaccessible to its regular user but potentially accessible to a reverse engineer, such as Richie), made by a designer Dora. It has secret specifications, and we require that:

1. The sizes of keys, plaintexts and ciphertexts is publicly known. The key may be loaded by the user.
2. C doesn't use public-key operations in real time. (Public-key operations and Dora's public key may be used e.g. directly after a key change.)
3. Dora's private key is not contained in C and only known to her.
4. The output of C is secure, based on the security of the underlying cryptographic tools. At the same time, it is possible for Dora to recover the key, given some number of ciphertexts (the corresponding plaintext may be known to or chosen by Dora).
5. The fact that C leaks key bits is undetectable to everyone except Dora and to a successful reverse engineer.
6. After the discovery of the specifications of C (e.g., by reverse engineering) no one except Dora can determine past or future keys or gather information about unknown plaintexts, given the corresponding ciphertexts.

[7] This condition is technically necessary, but we would expect it to hold anyway for a secure and efficient secret-key cipher and a secure practical public-key cryptosystem.

Note that our definition of a secret-key SETUP is essentially the same as the definition of a QUASI-SETUP, except for item 6. [8]

Some cryptosystems described below satisfy the requirements for secret-key SETUPs completely, while others – we informally call these *relaxed secret-key SETUPs* – only *almost* satisfy the requirements (but serve the same purpose).

The Secret-Key SETUP Stream Cipher SC1

The design of the stream cipher SC1 is quite straightforward. When $\mathrm{SC1}(K, n)$ is called, the n-bit string $\mathrm{SC1}_K(n)$ is evaluated as follows:

1. $Y := E_{\mathrm{pub}}(K)$;
2. if $n < m$ then output the first n bits of Y;
3. else output the m bits of Y and the $n - m$ bits of $f_K(n - m)$.

If Dora happens to know the first m plaintext-bits, she can recover Y from the ciphertext. This allows her to find K, to compute $f_K(n - m)$ and hence to find the remaining plaintext-bits.

The usefulness of the secret-key SETUP for Dora depends on Dora being able to guess the first m bits of plaintext. Many practical communication protocols start with the exchange of some "magic constants" or otherwise predictable values, e.g. as SMTP headers.

The Relaxed Secret-Key SETUP Stream Cipher SC2

Now we allow real-time public-key operations (and thus consider a relaxed secret-key SETUP). The relaxed restriction is: *one* public-key operation is done in parallel with *many* secret key operations.

Assume Dora knows (or can reasonably guess) $m_p \geq m$ consecutive bits of plaintext, not necessarily at the beginning of a long stream of plaintext bits. We fix a value M, such that M bits can be encrypted under f_K, while in parallel one evaluation of E_{pub} can be done without suspiciously slowing down the complete encryption device (hence $M \gg m$).

[8] The requirement for a QUASI-SETUP is less restrictive [8]: *"After the discovery of the specification of C (e.g., after reverse-engineering of the device), no one except the designer can determine past or future keys, though the successful reverse-engineer may be able to determine some fraction of plaintexts via known-plaintext attacks."*

For $i \in \{1, 2, \ldots\}$ (* generate $m + M$ bits of cipher stream *):

1. compute $Y_i := E_{\text{pub}}(K + i)$;
2. take the m bits of Y_i, followed by the last M bits of $f_K(iM)$.

If Dora is lucky, one value Y_i in the cipher stream exists such that all the m bits of Y_i can be recovered by bit-wise XOR from the ciphertext and the known bits of plaintext. This allows Dora to recover $K + i$ and thus K. If she knows K; she can compute the plaintext bits unknown to her before. If the starting point for the m_p known bits of plaintext is chosen at random,

$$\text{Prob[Dora is lucky]} \geq \frac{m_p - m + 1}{M + m}.$$

(Note that Dora is lucky if the first known plaintext bit is at most $m_p - m$ bits ahead of the first bit of Y_i. And if $m_p \geq M + 2m - 1$, an m-bit value Y_i is guaranteed to exist, such that the first known plaintext bit is at most $M + m$ bits ahead of the first bit of Y_i.)

Even if Dora is not as lucky as above, she may determine K when knowing all but the first δ bits or the last δ bits for small δ, since there are only 2^δ choices for Y_i.

The Security of SC1 and SC2

The security of a secret-key SETUP stream cipher SC$*$ depends on the following:

1. Without knowing the internal specifications of SC$*$, the fact that it leaks key bits cannot be detected.
2. Even when knowing the internal specifications, one cannot learn anything about either key or plaintexts.

In the case of SC1 and SC2, it is sufficient to show that the cipher stream is indistinguishable from a stream of random bits, even for Richie. If we assume that

1. the stream cipher f is secure,
2. the public encryption function E_{pub} is secure, and
3. f and E_{pub} behave independently,

then SC1 and SC2 are secure.

Summary of Results

The secret-key SETUP stream cipher **SC1** and the relaxed secret-key SETUP stream cipher **SC2** are useful for Dora, if she can mount a known plaintext attack. For SC1, Dora has to guess the first m bits of plaintext. For SC2, Dora must be able to guess a long string of plaintext bits, not necessarily at the beginning of the plaintext.

6 Conclusions

This paper introduces the notion of a secret-key SETUP. We provide several examples how to leak the key in a secret-key setting. The leakage is hard to detect without reverse engineering, and even a reverse engineer gains no information about plaintexts or keys.

It is well known, that black-box implementations of *public-key* cryptosystems can be used to leak secret information secretly and subliminally [5–7]. Most such techniques exploit the probabilistic nature of public-key cryptography. In this paper, we consider strictly deterministic devices for secret-key encryption. If the device where allowed to generate "random" keys for its own use, or to generate some "random" IVs needed in many modes of operation, subliminally leaking information would could be implemented more effectively.

As described in [8] and here, even in a strictly deterministic setting and without being suspicious for a black-box analyst, such devices can be built to leak information securely. The notion of security defined here, improves on the one from [8], who still allow the reverse engineer to learn a significant amount of information about plaintexts, given the ciphertexts (namely, one bit of information about each plaintext block).

The focus of this paper is methodological, to demonstrate the ease of designing such cryptosystems. This demonstrates the danger of black-box cryptography. Whenever possible, the user (or a trusted consultant) should *scrutinize the design* of a cryptographic device. Else, a black box device should only be trusted if its origin is trustworthy. Extensive statistical testing is not an alternative to examining design and implementation.

References

1. R. Anderson, M. Kuhn, "Tamper Resistance - a Cautionary Note", 2nd Usenix Workshop on Electronic Commerce, 1996, pp. 1–11.
2. M. Blaze, J. Feigenbaum, F. Leighton, "Master-Key Cryptosystems", Manuscript, based on a Rump-Session Talk at Crypto '95.
   Online: "`ftp://ftp.research.att.com/dist/mab/mkcs.ps`".
3. V. Rijmen, B. Preneel, "A Family of Trapdoor Permutations", Fast Software Encryption 1997, Springer LNCS1267, pp. 139–148.
4. M. Roe, "How to Reverse Engineer an EES Device", Fast Software Encryption 1994, Springer LNCS 1008, pp 305–328.
5. A. Young, M. Yung: "The Dark Side of Black Box Cryptography", Crypto '96, Springer LNCS, pp. 89–103.
6. A. Young, M. Yung: "Kleptography: Using Cryptography against Cryptograpy", Eurocrypt '97, Springer LNCS, pp. 62–74.
7. A. Young, M. Yung: "The Prevalence of Kleptographic Attacks on Discrete-Log Based Cryptosystems", Crypto '97, Springer LNCS.
8. A. Young, M. Yung: "Monkey: Black-Box Symmetric Ciphers Designed for MONopolizing KEYs", Fast Software Encryption 1998, Springer LNCS 1372, pp. 122–133.
9. R. Weis, M. Kuhn, Tron, A. Bogk, "Hacking Chipcards", Workshop CCC'97, Hamburg 199