

Monkey: Black-Box Symmetric Ciphers Designed for MONopolizing KEYS

Adam Young¹ and Moti Yung²

¹ Dept. of Computer Science, Columbia University
ayoung@cs.columbia.edu.

² CertCo New York, NY, USA.
moti@certco.com, moti@cs.columbia.edu

Abstract. We consider the problem of designing a black-box symmetric cipher that leaks information *subliminally* and *exclusively* to the designer. We show how to construct a cipher which we call ‘Monkey’ that leaks one key bit per output block to the designer of the system (in any mode). This key bit is leaked only if a particular plaintext bit is known to the designer (known bit/message attack which is typically available in plain ASCII). The attack is of kleptographic nature as it gives a unique advantage to the designer while using strong (e.g., externally supplied) keys. The basic new difficulty with the design of spoofable block ciphers is that it is a *deterministic function* (previous attacks exploited randomness in key generation or message encryption/signature), and the fact that we do not want easy (statistical) observability of the spoofing (e.g., the variability of ciphertexts should be noticeable when keys change etc.).

We distinguish between three entities: the designer, the reverse-engineer and the user. We show a design methodology that assures that: (1) if the device is not reverse-engineered, the attack is secure (namely, the cipher is good) and undetectable, (2) if the device is reverse-engineered, then the reverse-engineer learns at most one plaintext bit from every ciphertext (but no past/future keys), and (3) the designer learns one plaintext bit and one key bit from each ciphertext block (say in ECB mode). The method is therefore highly robust against reverse-engineering.

Key words: design methodologies for symmetric ciphers, secret cryptographic algorithms, spoofing, kleptographic attacks, trust, software vs. tamper-proof hardware designs, tamper-proof reverse engineering, public scrutiny.

1 Introduction

The US government has proposed recently a classified secret block cipher called SkipJack as part of the Clipper Initiative. Furthermore, since the mid 80’s the NSA’s Commercial COMSEC Endorsement Program has been active trying to base cryptography for secure computer and communication based on tamper-proof devices (see [Sch], page 598). The motivation of this paper is to investigate

the possibilities of designing secret symmetric ciphers with sophisticated trapdoors that are hard to detect and are immune to reverse engineering, and at the same time maintain the basic properties of block ciphers. The issue is essentially methodological, as it points at potential non-trivial leakage attacks which are possible with black-box cipher designs as opposed to public designs. (Our goal is neither to undermine SkipJack, nor to claim any concrete attack – we merely point at what we believe is a threat of secret designs that is beyond giving trivial known advantages.)

We first note that it is easy to mount attacks on secret devices, e.g. by fixing their keys. Such trivial advantage of easily reverse-engineered secret cipher (reverse engineering has been shown to be a concrete possibility, recently and can be done by a company with a well-equipped micro-electronic laboratory). This risks the designer's unique advantage (of getting other past/ future keys). Thus, one may argue that such designs will not be put to use (e.g., by an agency which is concerned about losing out to the resourceful companies). On the other hand, with a unique advantage, even after reverse engineering, a designer of a secret algorithm will have less hesitation to put it in general use. The above simplistic attack is also easily detectable when encryptions under supposedly “different keys” turn out to be identical. Another attack is by adding an encryption of the key under a secret designer key to ciphertexts; this will be easily noticeable due to data expansion. It will not be possible to classify such design as a block cipher. Yet another attack is by designing secret devices using pseudorandomness known to the attacker. However, block ciphers are deterministic functions such that when given the same input with the same key, the same result is expected. Thus, one may employ pseudorandomness for encryption which is derived from the key and a message, but then the encryption depends strongly on the key which is unknown to the designer (attacker). Ignoring the key or using only a partial key is statistically detectable.

Moving ahead, we then noticed that when we resort to known message attacks, we may once in a while leak key bits, so such attacks can be more powerful in attacking. This leakage should not destroy the quality of the cipher (e.g., make it insecure w.r.t. differential or linear cryptanalytic attacks or other statistical attacks). We would like to go even further and have the cipher be immune to reverse engineering, which is a characteristic of “kleptographic attacks” on black box devices (e.g. tamper resistant hardware). This means, for example, that the pseudorandom function's permanent key is not exposed after reverse engineering. If it were exposed, then the secret keys used with other devices would be exposed. We hope that the above discussion reveals some of the reasons behind the methodology we propose herein.

There has been much recent work on designing cryptosystems to leak secret information securely and subliminally to the designers. The basic notions underlying these attacks as well as tools that accomplish them were developed in [YY96,YY97a,YY97b]. Specifically, they introduced the notion of a SETUP attack, where SETUP stands for a “Secretly Embedded Trapdoor with Universal Protection”. In their attack it is a secretly embedded trapdoor (public key) that

is used to securely leak the secret information out of the cryptosystem. Their attacks are geared specifically towards public key systems and exploit randomness and subliminal channels [S94] in key generation, message encryption, and signing. Here, we show how to perform these attacks on deterministic symmetric block ciphers that are secret. We will propose a design we call “Monkey”¹.

The ‘Monkey’ is a general design (a methodology). For concreteness and clarity of presentation we give an algorithm which uses an 80 bit key and has a block size of 64 bits.

The design has an aspect which is inherently more challenging than SETUP attacks. Namely, we want to allow a strong cipher and the use of the actual large key space (that is, to design a real block cipher)—and block ciphers in turn are deterministic algorithms (implementing a permutation which is length-preserving). The attacker cannot control the choice of (strong and random) keys used by the cipher (we want to allow external source keying). Further, the attacker cannot know when it has access to mount the attack (we cannot assume some partial control over the device operation). In all the early work on SETUPS, the fact that the public key algorithms (key generation or signing) were probabilistic in nature was exploited. Thus, the natural question is: how can we design kleptographic attacks in this fully deterministic setting? In the present paper, we show that if we are able to mount minimalistic known-plaintext attacks (known bit per message), then a close approximation to a SETUP attack (called Quasi-SETUP) can be performed on a symmetric cipher with secret specification (the notion of Quasi-SETUP may be of independent interest). The attack gives exclusive advantage to the designer as in a SETUP and has strong protection against reverse engineering (which is needed given the potential of reverse engineering of tamper resistant devices which has been demonstrated recently in some settings [AK96]).

2 Background and Definitions

Our attack bears a resemblance to setup attacks let us recall what is a setup and define our quasi-setup attack. The following is the definition of a regular setup [YY97a]:

Definition 1. *Assume that C is a black-box cryptosystem with a publicly known specification. A (regular) SETUP mechanism is an algorithmic modification made to C to get C' such that:*

1. *The input of C' agrees with the public specifications of the input of C .*
2. *C' computes efficiently using the attacker’s public encryption function E (and possibly other functions as well), contained within C' .*

¹ We call our method “Monkey”, because: (1) it allows the attacker to [Mon]opolize [key]s, (2) it “monkeys around with the users secret key” in a way that a trusted cipher shouldn’t, (3) jokingly, we can say that the design reassures that one has to be a monkey to still trust secret unscrutinized designs, and finally (4) the naming follows a “recent tradition” of calling cipher designs after beasts of various kinds!

3. *The attacker's private decryption function D is not contained within C' and is known only by the attacker.*
4. *The output of C' agrees with the public specifications of the output of C . At the same time, it contains published bits (of the user's secret key) which are easily derivable by the attacker (the output can be generated during key-generation or during system operation like message sending).*
5. *Furthermore, the output of C and C' are polynomially indistinguishable to everyone except the attacker.*
6. *After the discovery of the specifics of the setup algorithm and after discovering its presence in the implementation (e.g., reverse-engineering of hardware tamper-proof device), users (except the attacker) cannot determine past (or future) keys.*

The kleptographic attack presented in this paper is what we call a “quasi-setup”. The following is a formal definition of a quasi-setup:

Definition 2. *A QUASI-SETUP is a black-box cryptosystem C (inaccessible to its regular user but may be attacked by reverse engineering). It has secret specifications and we require that:*

1. *The size of the input key, input plaintext, and output ciphertext is publicly known (and perhaps other extraneous and possibly erroneous information is given); key may be loaded by the user.*
2. *If C is a symmetric cipher, then, for efficiency, C does not use public-key operations in real-time (to avoid being recognized by time measurement). C pre-computes using the designer's secret public key E (and possibly other functions as well), contained within C .*
3. *The designer's private key D is not contained within C and is known only by the designer.*
4. *The output of C is secure based on security of an underlying block cipher and other tools. At the same time it contains published bits (of the user's secret key) which can be derived in poly-time by the attacker (designer), assuming the attacker has a sufficient amount of known-plaintext.*
5. *The fact that C leaks key bits is undetectable to everyone except the designer (attacker) and a successful reverse-engineer.*
6. *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.*

2.1 Related Work

In [YY96] a setup attack was given on RSA key generation devices. It was shown how an RSA key generation device could be designed to output a public and private key pair, such that the upper order bits of the public key can be used by the designer to compute the corresponding private key. More generally, they proved that any cryptosystem that contains a subliminal channel contains a SETUP

version. In [YY97a] it was shown how one of the exponents in a Diffie-Hellman key exchange can be securely and subliminally leaked to the designer over the course of two (wlog) consecutive key exchanges. This setup attack was the first setup attack that did not make use of explicit subliminal channels but rather generated channels for leakage due to repeated executions. In [YY97b], setup attacks were given for the ElGamal public key cryptosystem, the ElGamal Digital Signature algorithm, the Digital Signature Algorithm, Schnorr, and other systems. The attacks on the signature schemes leak the private signing key over the course of two (wlog) consecutive signatures. All the above attacks used randomness employed by the cryptosystem.

Rijmen and Preneel [RP97] gave a much more ambitious direction which is different from ours. They suggested the construction of the first example trapdoor ciphers, demonstrating that even an open design has to be justified or pseudo-randomly generated to avoid potential spoofings. (How secure is their design and if strong trapdoors exist at all, are still open). Indeed, the potential existence of trapdoor ciphers already points at a difficulty with secret design. We will show that with Monkey, the attack is assured, and also the attack can be based on minimal knowledge (known bit attack), and the length of a key recovery attack can be much shorter than the specific trapdoor ciphers based attack in [RP97]; (these are of course advantages of attacks on hidden designs which are easier in nature).

An approach to “generating trust” in a secret cipher design was attempted by producing a report of an inspection team in [BD93] in the context of SkipJack (more about that report, see [R94]). The subtleties presented here may potentially cast some extra doubts on a secret way to assure trust by a known team (since one does not know how much the team knows and what information was made available to it). However, we must mention that, by the same token, we cannot have any concrete complaint against the specific report above.

Finally, let us mention that in the context of the recent NIST initiative to design the next generation of block cipher standard (AES), the work here reinforces the notion of public scrutiny of suggested standards (to avoid various trapdoors which are indeed possible).

3 Monkey Implementation

The Monkey secret symmetric cipher takes an 80 bit symmetric key as input, in addition to 64 bits of plaintext. It outputs a 64 bit ciphertext block. Monkey uses for pre-computations the SMALL1 public key cipher to be described. We assume that Monkey is a secret cipher. This is of course nonsensical, since we are publishing it now. What we mean is that our suggestion is an instance of a methodology of design and we can assume that a cipher like it (a variant) may be kept secret and implemented. We assume that it is tamper-resistant so that it is hard to get. Methodologically what is important is this last fact (of being hard to get, i.e. black-box to the user) and not the exact physical assumption about tamper resistance (we are fully aware of recently discovered weaknesses

of certain such claimed designs while we are aware of other designs which were not reversed-engineered so far).

3.1 Some tools

In [SOMS] a fast key exchange algorithm was given that uses elliptic curves. In their scheme the curve E is a set of points (x, y) with x and y lying in the field $F_{2^{155}}$. They implement Diffie-Hellman over this curve using a publicly defined point P on E . This scheme appears to be secure as long as the Index Calculus method cannot be extended to elliptic curves.

It is a trivial matter to define a public key encryption algorithm based on Diffie-Hellman over E . Suppose Alice wishes to encrypt the message m where m is 80 bits in size. Alice wishes to send this message to Bob, whose private key is x , where x is in the range $[2, \text{order}(E)-2]$. Bob's corresponding public key is the point $y = xP$. To send the encryption of m , Alice chooses a random integer r in the range $[2, \text{order}(E)-2]$ and computes kP by iterating the addition of P using the "double and add" scheme. Alice then computes $z = ry$. Alice can then use some or all of the shared secret string $H(z)$ to encipher the value m to get the value c . Here H is a suitable hash function. Note that c need only be as large as m . Alice then sends (rP, c) to Bob. Decryption is straightforward. Note that Alice sends Bob 310 bits corresponding to rP plus 80 bits that constitute c . Hence, Alice sends Bob 390 bits of information. Note that the ciphertext size is smaller than what is possible using RSA with a 512 bit modulus. In this paper, we define this to be the *SMALL1* public key cryptosystem. Hence, *SMALL1* uses a random parameter r that is at least 2 and at most $\text{order}(E)-2$, and takes a public key y as input. *SMALL1* takes 80 bits of input data m_a and produces a 390 bit ciphertext c_a . The operation of *SMALL1* is denoted by $c_a = \text{SMALL1}(r, y, m_a)$. Let *SMALL1'* denote decryption. Hence, $m_a = \text{SMALL1}'(x, c_a)$.

3.2 System Setup

The designer chooses a keyed pseudorandom function F_{63} that takes as input a large seed (key) s and a 63 bit input x , and produces a 63 bit output y . For an explanation on how to construct pseudorandom functions, see [GGM86]. The operation of F_{63} is denoted by $y = F_{63}(s, x)$. The designer also chooses two seeds, s_1 and s_2 uniformly at random (from the seeds of the given length). The designer chooses a private key x randomly for use in *SMALL1*, and computes the corresponding public key y . The designer puts (F_{63}, s_1, s_2, y) in the black-box device and keeps x private.

3.3 System Operation: Encryption

Let K denote the 80 bit key of the user. The user wishes to use the black-box device to encrypt the 64 bit plaintext message m , to get the corresponding 64 bit ciphertext c . The operation of Monkey is denoted by $c = \text{Monkey}(K, m)$. The

device contains a secret symmetric block cipher called *CIPHER1*. *CIPHER1* takes a 63 bit symmetric key k_b and a 63 bit plaintext m_b . *CIPHER1* produces a 63 bit ciphertext c_b . We denote the operation of *CIPHER1* by $c_b = CIPHER1(k_b, m_b)$. The corresponding decryption operation is *CIPHER1'*. Hence, $m_b = CIPHER1'(k_b, c_b)$. Let H denote a cryptographically secure hash function (e.g., a collision intractable function, or a pseudo-random function) that maps $\{0, 1\}^{80}$ to $\{0, 1\}^{63}$. G is a random function that maps $\{0, 1\}^{80}$ to a value in the interval $[2, \text{order}(E)-2]$. The following is the computation of $\text{Monkey}(K, m)$ to get c :

1. m_b is set to the lower order 63 bits of m
2. $k_b = H(K)$
3. $c_b = CIPHER1(k_b, m_b)$
4. the lower order 63 bits of c is set to c_b .
5. $r = G(K)$
6. $c_a = SMALL1(r, y, K)$
7. $i = F_{63}(s_1, c_b) \bmod 390$
8. b is set to the i th bit of c_a .
9. $z = F_{63}(s_2, c_b) \bmod 2$
10. p is set to the most significant bit of m
11. the most significant bit of c is set to be $b \text{ XOR } z \text{ XOR } p$

3.4 System Operation: Decryption

The following is the operation of $\text{Monkey}'(K, c)$ which returns m :

1. c_b is set to the lower order 63 bits of c
2. $k_b = H(K)$
3. $m_b = CIPHER1'(k_b, c_b)$
4. the lower order 63 bits of m is set to m_b
5. $r = G(K)$
6. $c_a = SMALL1(r, y, K)$
7. $i = F_{63}(s_1, c_b) \bmod 390$
8. b is set to the i th bit of c_a .
9. $z = F_{63}(s_2, c_b) \bmod 2$
10. p' is set to the most significant bit of c
11. the most significant bit of m is set to be $b \text{ XOR } z \text{ XOR } p'$

3.5 Secret Key Recovery

Suppose that we manage to obtain 390 ciphertexts c_0, c_1, \dots, c_{389} such that we know the least significant bits of the 390 corresponding plaintexts. Suppose further that $F_{63}(s_1, c_j) \bmod 390$ for $0 \leq j \leq 389$ is a permutation on $0, 1, \dots, 389$. The following algorithm computes the ciphertext bit in c_a corresponding to c_j .

1. c_b is set to the lower order 63 bits of c_j

2. $i = F_{63}(s_1, c_b) \bmod 390$
3. $z = F_{63}(s_2, c_b) \bmod 2$
4. p is set to the most significant bit of m_j
5. p' is set to the most significant bit of c_j
6. outputs $p \text{ XOR } z \text{ XOR } p'$

The above algorithm is applied to c_0, c_1, \dots, c_{389} , to recover the ciphertext c_a . We then decrypt c_a using x to recover K . We choose the most significant bit since if the plaintext is ASCII for instance, this bit is known to be zero. In fact, since the block size is 64 bits, if the plaintext is ASCII we can leak using a bandwidth of 8 bits. Another possibility is to compress the plaintext (when possible) and then add high order bits which are known, decryption will decompress.

3.6 Needed Strengthening

The above cipher is a concatenation of two secure ciphers. The problem is the separability of the ciphers, so that messages that differ only at the last bit has a ciphertext which also differs on that bit. (This separability in general may help in attacks like chosen message attacks. This is the case here as messages come in pairs and when they differ on the last bit the ciphertext of one implies that of the other in the pair). This can be overcome by a cascading of ciphers. Namely by a post-processing (pre-processing in decryption) by CIPHER2 which employs (say, at least four) Feistel transformations based on Luby-Rackoff's construction [LR88] with a fixed secret pseudorandom function. This spreads the local difference in the last bit only, uniformly over the resulting ciphertext.

Another aspect that we did cover carefully are the sizes of the various keys (of the pseudorandom functions). If we need larger keys in total we may derive these keys pseudorandomly (using a secret fixed internal seed) from the given key.

4 Security of the Black-Box Monkey Implementation

There are two perspectives with which to analyze the security. Those perspectives are the black-box perspective, and the perspective of an attacker who is able to reverse-engineer the device (hence, no black-box assumption). In this section we consider both of these in turn. We assume that the ciphers and pseudorandom functions used CIPHER1, SMALL1, G , H , and F_{63} are secure.

Now we assume that a given user is unable to reverse-engineer the black-box device. It follows that C is a black-box cryptosystem with private specification from the user's perspective.

We note that a specific "security of a block cipher" is not defined here (of course, such generic definition does not exist!). Our general methodology therefore attempts to preserve within the overall Monkey design, whatever "security notion" CIPHER1 has, based on the strong security properties of the other building blocks (pseudorandom functions and permutations). One may argue

that what we assume are heavy cryptographic tools; but this should not be a problem at this stage of the development of the methodology. Indeed, we leave open the issue of minimal assumptions needed, as well as the efficiency of design of ciphers with Quasi-SETUP. What we claim is:

Claim 1 *Assuming that CIPHER1, CIPHER2, SMALL1, G, H, and F_{63} are secure, then if Monkey is a black-box cryptosystem with private specification, then C constitutes a secure symmetric cipher.*

Proof. c_b constitutes a secure encryption of m_b since CIPHER1 is a secure symmetric cipher. It remains to show that the least significant bit of c constitutes a secure encryption of the least significant bit of m . Since F_{63} is a secret pseudo-random function, and since s_2 is unknown, it follows that $F_{63}(s_2, c_b)$ is random and unknown to the user. Thus $z = F_{63}(s_2, c_b) \bmod 2$ is a random secret bit with respect to the user. Since this bit is exclusive-or'ed (one-time padded) with the least significant bit of m , the least significant bit of c constitutes a secure encryption.

Now, the two secure values give two separable encryptions and can be viewed as a secure block cipher on 63 bits, concatenated to a one-bit strong stream cipher encryption. The further Feistel like transformations in CIPHER2 based on a pseudorandom function which strengthen the design, assure a strong inseparable encryption (due to the “avalanche” properties of pseudorandom functions) and prevents easy chosen message attacks. The overall cipher can be viewed as two ciphers cascaded. The cascade is as secure as the first cipher (or as each of the ciphers for weaker attacks) [MM93,EG85]. QED.

Recall that if the pseudorandom function (its key) is not known then the value of the function at a point cannot be approximated even in a very liberal sense even if the values of the function at polynomially many other points is also given [GGM86]. It is this property of pseudorandom functions that makes this attack secure; practical designs of pseudorandom functions can be based on iterated strong block ciphers with large keys. Only a minimal amount of security at best is sacrificed by using the 63 bit block cipher CIPHER1, as opposed to a true 64 bit block cipher (e.g., it is very easy to modify DES to work on 63 bits). The additional bit encryption of the last bit is a strong cipher as well. Note also that cipher designs that are tunable to each size in small granularity (e.g. as in the NIST's AES specifications) have this disadvantage of being a potential CIPHER1 component in a Monkey design – where the difference (one bit in our case) is the amount of required known bits per message.

Now suppose that an attacker manages to reverse-engineer the device (thus, CIPHER2 can be ignored hereafter). The attacker therefore knows (F_{63}, s_1, s_2, y) and the complete specification of the cipher Monkey. In this case, the reverse-engineer is able to recover at most the least significant bit corresponding to the ciphertexts that are output by the device, as long as a sufficient number of known-plaintext bits are gathered. To see this, note that the reverse-engineer can recover the bit z from the secret key recovery algorithm in the same way

as the designer. Since the reverse-engineer also knows p' and presumably the least significant bit of m , he or she can exclusive-or these three bits together to recover one of the bits of c_a . If the reverse-engineer gets a sufficient number of known plaintexts, the reverse-engineer can reconstruct c_a . When equipped with the value c_a for a given key K (not available in the device at time of reverse engineering i.e. a past or future key), the reverse-engineer can decrypt the least significant plaintext bits of all values encrypted with K . Can the reverse-engineer also recover the bits of K , like the designer? The answer to this is no, as we will claim next.

Claim 2 *Assuming that CIPHER1, SMALL1, G , H , and F_{63} are secure, and that the adversary has a polynomial number of plaintext/ciphertext pairs, the successful reverse-engineer is unable to learn anything about K .*

Proof. Consider the ciphertext values that result from a particular K . Since we assume that CIPHER1 is secure, the reverse-engineer is unable to learn anything about k_b and hence K , from the 63 upper order bits of the ciphertexts alone. Note that the application of the pseudorandom function to c_b to derive the least significant ciphertext bits has the effect of the application of a random oracle to c_b to get the least significant ciphertext bits. Thus, anything that can be deduced about K from all of the bits of the ciphertexts can be deduced from the least significant bits alone. So, it remains to consider what can be deduced from the least significant bits of the ciphertexts alone. Since m , c , and (F_{63}, s_1, s_2) are already known to the reverse-engineer, the reverse-engineer knows c_a . It remains to show that nothing about K can be learned from c_a . Since we assumed that G is a secure random function, $G(K)$ is a random string to the reverse-engineer. So, since SMALL1 is a secure public key cryptosystem, $\text{SMALL1}(G(K), y, K) = c_a$ is a secure public key encryption of K . QED.

Claim 2 hinges on the fact that F_{63} and G are random functions, and that SMALL1 is a secure public key encryption function. At the same time Claim 1 argues that in the event that the device is never reverse-engineered, and in the event that the designer never abuses his or her power, Monkey *is* a secure symmetric cipher. Thus, in summary, the capability of users with respect to Monkey can be broken down into three different categories:

1. Users who are unable to reverse-engineer the device are unable to learn any plaintext.
2. Users who are able to reverse-engineer the device, when given enough known-plaintext, are able to learn one plaintext bit of every ciphertext.
3. The designer, when given enough known-plaintext, is able to learn all plaintext bits of every ciphertext (since it monopolizes the keys in use).

Given the above two claims, recall also that Monkey can be loaded with external keys. Monkey therefore constitutes a quasi-setup.

Note that this paper provides motivation for having new public key cryptosystems that output very small ciphertexts (such schemes, with yet hard to

analyze security, based on polynomial manipulation have been design, e.g. in [P96]). If a public key cryptosystem exists that outputs ciphertexts that are, say, 200 bits in size, then far fewer known-plaintexts need to be gathered to leak the secret key K securely. (The size of the public key block is related to the number of known messages required).

5 Conclusion

We introduced the notion of a quasi-setup and demonstrated a symmetric cipher (Monkey) which constitutes a quasi-setup. We showed how to design a secret cipher that gives an unfair advantage to the designer, and that is very robust against reverse-engineering. Our results imply that secret symmetric ciphers implemented in black-box settings should only be used if they come from trusted sources and cannot be simply trusted based on extensive statistical testing. It strengthens the need for open cipher design efforts. We did not attempt to hide which “known bit” is required for the attack. It may be the case that in more convoluted ciphers where this needed-bit is not specified, the combination of internal stream cipher operations, pseudorandom operations like S-boxes and Feistel transformations and exponentiation operations, can even hide which bits are needed to be known (and may evade an inspecting team lacking the original design documents).

Finally, efficient and minimal designs and designs that maintain specific properties of block ciphers while enabling a quasi-SETUP attacks are left as open questions.

References

- AK96. R. Anderson and M. Kuhn, Tamper Resistance – a Cautionary Note. In *The 2-d Usenix Workshop on Electronic Commerce*. 1996, pages 1–11.
- BD93. E. F. Brickell, D. E. Denning, S. T. Kent, D. P. Maher, and W. Tuchman SkipJack review: the SkipJack Algorithm, Interim Report, July 28, 1993.
- EG85. S. Even and O. Goldreich, On the Power of Cascade Ciphers. *ACM Tra. on Comp. Systems*, V. 3, 1985, 108–116.
- GGM86. O. Goldreich, S. Goldwasser, and S. Micali, How to Construct Random Functions. *J. of the ACM*, 33(4), pp 210-217, 1986.
- LR88. M. Luby and C. Rackoff, How to Construct Pseudorandom Permutations from Pseudorandom Functions. In *SIAM J. on Computing*, V. 17, 1988, pages 373–386.
- MM93. U. Maurer and J. Massey, Cascade ciphers: the importance of being first. In *J. of Cryptology*, V. 6, 1993, pages 55–61.
- P96. J. Patarin, Hidden Field Equations (HFE) and Isomorphisms of Polynomials (IP): Two New Families of Asymmetric Algorithms, In *Advances in Cryptology—Eurocrypt '96*, pages 33–48, 1996. Springer-Verlag.
- RP97. V. Rijmen and B. Preneel, A Family of Trapdoor Ciphers, *Fast Software Encryption 97*, (Ed. E. Biham).
- R94. M. Roe, How to Reverse Engineer an EES Device, *Fast Software Encryption 94*, Dec. 94, (Ed. B. Preneel), LNCS 1008, Springer-Verlag.

- Sch. B. Schneier. *Applied Cryptography*, 1994. John Wiley and Sons, Inc.
- SOMS. R. Schroepel, H. Orman, S. O'Malley, O. Spatscheck. Fast Key Exchange with Elliptic Curve Systems. In *Advances in Cryptology—CRYPTO '95*, pages 43–56, 1995. Springer-Verlag.
- S94. G. J. Simmons. Subliminal Channels: past and present. *European Tra. on Telecommunications V. 5*, 1994, pages 459–473.
- YY96. A. Young, M. Yung. The Dark Side of Black-Box Cryptography. In *Advances in Cryptology—CRYPTO '96*, pages 89–103, Springer-Verlag.
- YY97a. A. Young, M. Yung. Kleptography: Using Cryptography against Cryptography. In *Advances in Cryptology—Eurocrypt '97*, pages 62–74. Springer-Verlag.
- YY97b. A. Young, M. Yung. The Prevalence of Kleptographic Attacks on Discrete-Log Based Cryptosystems. In *Advances in Cryptology—CRYPTO '97*, Springer-Verlag.