Scramble All, Encrypt Small

Markus Jakobsson * Julien P. Stern ** Moti Yung ***

Abstract. In this paper, we propose a new design tool for "block encryption", allowing the en/decryption of arbitrarily long messages, but performing en/decryption on only a single block (e.g., 128 bit block), where the rest of the message is only processed by a good scrambling function (e.g., one based on an ideal hash function). The design can be a component in constructing various schemes where the above properties gives an advantage. A quite natural use of our scheme is for *remotely keyed encryption*. We actually solve an open problem (at least in the relaxed ideal hash model and where hosts are allowed to add randomness and integrity checks, thus giving a length increasing function); namely, we show the existence of a secure remotely keyed encryption scheme which performs only *one* interaction with the smart-card device.

1 Introduction

We provide a basic design allowing encryption and decryption to be performed by combining a high quality scrambling technique with a strong encryption/decryption mechanism. In particular the method applies to cooperation between an untrusted (potentially exposed) but computationally potent device and a trusted (highly secure) but computationally weak device. The method achieves an appropriate balance of computation and trust by employing a scrambling mechanism, which is modeled as an ideal length preserving one-way function which can be built from a regular ideal (random oracle like) hash function.

While following the usual Feistel structure for block cipher design, we show that most of the work can be done by the (publicly known and invertible) scrambling, requiring only a small portion to be performed by encryption. This situation allows a design which can be useful when the encryption is more expensive than the scrambling, or when it is otherwise limited. This is the case when the encryption is done by a small device (e.g. a smart card) which is slower than a general host's software (which has more computational power, but is less secure – thus encryption keys must never be exposed to the host). We call the design "scramble all, encrypt small."

A most natural application of our protocol is for "remotely keyed encryption", which we motivate below. Yet, our protocol is of independent interest

* Information Sciences Research Center, Bell Labs, Murray Hill, New Jersey 07974 `www.bell-labs.com/user/markusj`

** UCL Crypto Group, Batiment Maxwell, 3 Place du levant, B-1348 Louvain-la-Neuve, Belgium (`stern@dice.ucl.ac.be`), and Laboratoire de Recherche en Informatique, Université de Paris-Sud, Batiment 490, F-91405 Orsay Cedex, France (`stern@lri.fr`).

*** CertCo Inc. New York. (`moti@cs.columbia.edu, moti@certco.com`).

L. Knudsen (Ed.): FSE'99, LNCS 1636, pp. 95–111, 1999.
© Springer-Verlag Berlin Heidelberg 1999

and can be used in other settings, for example to speed up certain encryption processes. In particular, our construction nicely applies in the context of traitor tracing [CFN94, Pfi96, NP98]. In traitor tracing, the data is encrypted with a random key, which is itself encrypted so that only users with legitimate keys can decrypt it (so the keys can be traced). Instead of encrypting the data with a random key, we can simply scramble it and encrypt a single block in the same way the random key is encrypted.

The need for remotely keyed encryption was nicely motivated by Blaze. Let us recall the motivation. Today's relatively open environment of hosts (e.g., Internet or Intranet servers) leads to a rather paradoxical situation in terms of security architecture: The hosts which are the gateways to the outside world (or internal users) are frequently required to perform cryptographic tasks, simply because they often use encryption to communicate at a low level, and as such, are the most vulnerable to various attacks (or even to exposure by simple implementation flaws). Given a cryptographic application or service, what is often neglected is its embedding environment and the implementation. It is argued that given that general software and the operating system running the software may be susceptible to viruses, Trojan horses, and other related attacks, there is a substantial risk of attackers momentarily gaining control of networked computers and the information stored by these. Therefore, secret keys should not be kept or used in this environment whenever possible. On the other hand, smart cards and other computational platforms with reduced risks of attacks often have a very limited computational ability and low storage capabilities. Consequently, it is desirable to divide the computation between an untrusted but powerful device (which we will call the slave) and a trusted but weak device (the master).

This problem, which in its encryption incarnation is known as *remotely keyed encryption schemes* (RKES) was proposed in [Bla96] with no model or proof and with certain subtle problems. Several solutions have already been suggested to solve this problem [Bla96, Luc97, BFN98]. If we momentarily disregard the security issues, the common aspect of these three schemes is that they ask a smartcard to generate a "temporary" key, which depends on the message, and which is used to encrypt the largest part of the message. This generates a "binding" between the message and the device via the encryption. They later ask to hide this key. A formal model and secure solution for this (based on pseudorandom functions) was given in [BFN98]; their solution requires two accesses to the smartcard per operation.

While our system provides the same functionality, the use of the "scramble all, encrypt small" notion is different: in our scheme, the host does *not* perform any encryption. It simply scrambles the message in a publicly available (and invertible) way (after adding randomness and integrity check), and then deprives an adversary of the ability to invert the scrambling by letting the smartcard encrypt just a small part of the scrambled message. Also, in our scheme, there is a *single* access to the trusted smart card which performs a *single* decryption/encryption operation before replying. This is implied by the fact that the host does not use a temporary key for encryption and does not encrypt at all,

but rather the host employs scrambling of high quality. In some sense the earlier constructions can be viewed as an unbalanced Feistel (Luby-Rackoff type) encryption. Namely, pseudorandom permutation based on pseudorandom functions, where the pseudorandom functions are on the smart card. This seems to require two applications of the pseudorandom function. This implies (only intuitively) the need for two calls to the card. Thus, to get the result with "a single call" to the smart card we rely on the ideal (random oracle) hash model. It is an open question whether this stronger assumption model is necessary for achieving the "single call" result.

Since the host does not use encryption, the method gives a way to encrypt arbitrarily long messages (with strong notion of security), while encrypting only one block (of size, say 128/ 256 bits). We also allow the host to add randomness to the encryption (this can be viewed as an added random IV) so we can formalize our mechanism and security w.r.t. an overall randomized encryptions (which eases the difficulty of pseudorandom ones as in the earlier works). The host also adds an integrity check to prevent message modifications and other attacks.

We validate our design by giving a proof of security assuming the scrambling is an ideal hash and the small encryption is an ideal block cipher (random permutation).

From an engineering perspective, the *Bear* and *Lion* designs [AB96] have taught how to encrypt an arbitrary long block given fast stream ciphers. This is suitable for applications with large messages. While we have the same applications in mind, we teach how to minimize the encryption mechanism to a small block of size 128 bits, say. This minimization may be due to configuration and performance constraints. Thus, our design where only small portions are encrypted (rather than the entire large block which here is only scrambled without a key) may be called "*Cub*" as opposed to "Bear" and "Lion". Note that we do not claim that there is a universal performance advantage of Cub over the earlier designs which applies to all working environments. On the contrary, we understand very well that stream cipher encryption may be fast comparing to a good scrambling mechanisms. What is more interesting is the minimization of encryption itself and the confinement of the entire cryptographic operation.

Finally, we remark that since the design deals with the constraints of having a slow encryption whose use is minimized, it may fit well when the encryption employed is obviously much slower than available good scrambling techniques based on cryptographic hash functions. If the available encryption is based on public-key, the relative performance resembles the gap of "remotely keyed encryption".

Remark on presentation: While we present a general two-process method, its implementation as a "remotely keyed" mechanism is the prime example and will serve as the working example (since it is concrete and it further adds environmental constraints).

Outline: Section 2 discuss previous work on the subject. Section 3 introduces our model, and section 4 presents the basic tools we will use. Section 5 explains our new solution. Validation of the design via a security proof is given in section 6

and experimental results are outlined in section 7. Finally section 8 presents open problems and concludes the work.

2 Related work

There are many issues in block cipher design (for extensive review see the appropriate sections in [Sch96, MOV97]). Feistel introduced the fundamental notion of rounds where a one-way function of half the "current message" is exor-ed into the other half. Provable constructions based on one-way functions that act globally on half the block (analogous to "large s-boxes") were first investigated by Luby and Rackoff [LR88]; their functions were based, in fact, on pseudorandom functions. Many other investigations followed which included simplifications of the analysis and the proof of security and variations/ simplifications of the round functions (see [Mau92, Luc96, NR97]).

In particular, employing stream-ciphers for fast block message encryption in the style of Luby Rackoff was presented in [AB96]. Their design like the one in this work is suitable for block cipher operations for large blocks, which may be typical in software oriented mechanisms. Adding public scrambling via hashing prior to encryption was considered in the past. A prime example is the scrambling using ideal hash and pseudorandom generator in [BR94]. Another example is in a work (coauthored by the third author) [IBM-pa.]. However, none of these works put forth the notion of global scrambling combined with local (small block) encryption as a possible provable design. Related work which we have learned about recently, is in [MPR97, MPRZ98]. They do suggest a design where scrambling via hashing is done prior to partial encryption as in our case, but they do not give a security validation proof. They suggested their design to standardization committees, and our validation proof may support their effort. Their work does not have the context or the consideration for remotely keyed encryption; rather they suggest it in the context of public-key encryption. The goal of the current work is to suggest minimized encryption which is validated and can be used systematically whenever needed, possible or useful. We characterize scenarios (alternatives) in working environments where we get performance advantage.

Blaze's remotely key encryption protocol [Bla96] was based on the idea of letting the host send the card one block which depends on the whole message. This block would be encrypted with the smart card secret key, and would also serve as a "seed" for the creation of a "temporary" secret key, that will be used by the host in order to encrypt the rest of the message. However, as Lucks [Luc97] pointed out, Blaze's scheme had problems in that in allowed an adversary to forge a new valid plaintext/ciphertext pair after several interactions with the card. Lucks suggested an alternative model and protocol, which in turn, was attacked by Blaze, Feigenbaum and Naor [BFN98] who further demonstrated the subtleties of this problem. They showed that the encryption key used for the largest part of the message is deterministically derived from the two first blocks of the message, hence an adversary who takes control of the host will

be able to decrypt a large set of messages with only one query. They derived a careful formal model and a scheme based on pseudorandom functions. Very recently, Lucks [Luc99] further extended their security model and suggested a faster scheme. In contrast, we will allow randomized encryption and we will rely on ideal (random oracle) hash functions.

Our work makes sure that missing a small piece of the scrambled data (via encryption) while keeping the rest available makes it hard to recover the message. This bears some relationship to Rivest's notion of All-Or-Nothing encryption [Riv97]. Informally, given a symmetric cipher, a mode of encryption is defined to be *strongly non-separable* when it encrypts s blocks of plaintext into t blocks of ciphertext ($t \geq s$) which are such that an attacker cannot get any information on the plaintext without decrypting *all* the ciphertext blocks. In order to obtain strongly non-separable encryption, Rivest suggests to perform an "all-or-nothing" transform on the plaintext, followed by a regular encryption of the result. There is an obvious parallel between our scrambling step and the all-or-nothing transform. As a matter of fact, our scrambling step possesses a slightly more general property than the strong non-separability. It has the property that no information can be gained on the pre-image of a scrambling as soon as *any* k-bits are missing (in the design we fix which k bits to hide). The two preprocessing steps are interchangeable: Rivest's all-or-nothing transform could be followed by a single encryption, and our scrambling yields an all-or-nothing encryption mode. However, the motivations of the two notions are very different. Our goal is to design a scheme which minimizes encryption with a given key, while Rivest's goal is to make brute-force decryption more difficult to an adversary. As a final note, we should point out that our techniques is much more efficient than the one proposed in [Riv97], notably because we do not use encryption during the preprocessing step.

3 Model and Definitions

We will present a two-stage model: scrambling and encryption. As noted above, the presentation follows the remotely keyed encryption model.

3.1 Model

Our scheme involves two connected devices: a computationally potent device, let us call this device the *slave* device, and a computationally weak device, which we denote the *master* device.

We assume a limited and low bandwidth communication channel between the master and the slave. We trust the master device to follow the protocols and to be tamper resistant. We only trust the slave device to *be able to* perform the operations it is asked to. On the other hand, we do not trust it for being intrusion resistant: we assume that an adversary may take full control of it for some time period. During this preliminary period the adversary may in particular obtain

any information that the slave has and also interact with the master in any way he likes.

We would like to construct schemes that allow the slave device to perform encryption (and decryption) on large messages at high speed, with the help of private information owned by the master device. In the full generality, we may consider that the master can own keys corresponding to any kind of encryption (symmetric, public-key, probabilistic or non-probabilistic), and that the encryption obtained at the end of the protocol can be of any of the previous types. However, for public-key we may employ the master only for decryption, and with symmetric encryption the type of queries and attacks is larger (since "encryption queries" are meaningful). We consider as our example model of choice a master which employs an ideal cipher (e.g., a (pseudo)-random permutation), and a slave which performs sampling of a randomized IV, and scrambling via a public ideal (random oracle) hash.

The requirements on the schemes are as follow:

Balanced computation. The slave should perform the largest possible part of the computation, the master should perform the lowest. Their respective parts of the computation can be proportional to their computing rate (thus we balance the time spent by each component). Other choices of resource balancing are possible– e.g., limit the slow component to a "constant usage" and vary the fast component as a function of the message size.

Low communications. The number of interactions between the slave and the master should be low, and each of these interactions should need only a small amount of communication. Ideally, there should be only one interaction per protocol.

Security. Intuitively and informally, we require that after having taken control of the slave and making a number of queries (bounded by some polynomial) to the master, and then losing the control of the slave, an adversary will not have any advantage in distinguishing subsequent plaintext/ciphertext pairs from random pairs. Variations on the security requirements are possible: e.g., the adversary may choose the plaintext for which a distinguishing challenge is required. Other challenges than distinguishability are possible as well. Here, we will consider two attacks. (Though the above talks about "polynomial time" and "advantages" in general terms, we will actually compute actual probabilities of successful attacks).

Of course, the interaction and the encryption blocks are still large enough in the size of the security parameter. Namely, the security of the protocol is assured sub-exponentially in the size of that block (e.g., 128 or 256 bit size).

3.2 Definitions

Configuration A *probabilistic remotely key encryption scheme* (PRKES), consists of two protocols, one for encryption, the other for decryption, both executed

on two communicating devices. These devices are (1) a probabilistic machine (machine with a truly random generator) called a "master" and (2) a machine called a "slave". (The slave can also be probabilistic in some designs. In fact, herein we model the slave as an ideal cipher, namely a random permutation). Given an input to the slave, it interacts with the master and produces an output. The input includes an indicator for performing either encryption or decryption, possibly a size parameter, and the argument (resp. cleartext or ciphertext).

Background for attacks Let us review attacks on the protocol.

The polynomial-time attacker $\mathcal{A}$ (we will use concrete assumptions regarding the "polynomial-time power") has a **challenge** phase, and one or more **probing** (tampering) phases. Typically the probing enables the adversary to activate the device up to some bound (some polynomial in the key length). The probing is a preliminary step prior to the challenge (or a prior step followed by additional probing with certain restriction after the challenge has been issued).

In the *challenge phase*, a certain goal is required from $\mathcal{A}$:

Distinguishability challenge: $\mathcal{A}$ is presented with a challenge pair c_1, c_2, which is either a plaintext/ciphertext pair or a random pair from the same distribution. (Below we specify one such distribution).

Valid pair creation challenge: Another possible type of challenge, yielding another attack, is to ask $\mathcal{A}$ to exhibit one plaintext/ciphertext pair more than the number of probes he performed.

The *probing phase*. As noted above it can come before the challenge but also some limited part of it can be allowed to occur after the challenge.

We consider two probing phases regarding PRKES:

System probing: The attacker gets input-output access to the slave but not to its memory. Namely, he uses the slave as an oracle.

Master probing: The attacker gets full access to the memory of the slave. He can use the slave to interact with the master (input-output access) on queries of its own using the master as an oracle.

Attacks and security against them Let us next describe the entire attack. We consider two types of attack, which differs only by the challenge phase, described below.

An *adaptive chosen-message chosen-ciphertext attack with distinguishability challenge* includes:

– first phase: $\mathcal{A}$ performs "master probing". We assume that it performs up to p_1 pre-challenge probes. We also assume that the slave is *reset* (to a random state) after the intrusion, that is before the second phase. (This insures that no state information on the slave is known after the probing).

- second phase (distinguishability challenge): $\mathcal{A}$ *presents a plaintext* (or plaintexts) *pl* and gets a challenge pair (plaintext, value) whose plaintext part is *pl* and the value part is either a ciphertext of the plaintext or a random value, each case chosen with probability 1/2.

- third phase: $\mathcal{A}$ performs "system probing" where he can ask any oracle query (including further encryptions of the challenge plaintext) *BUT* is not allowed to perform a *decryption* query on the ciphertext of the challenge pair. We allow up to p_2 such post-challenge probes.

Security: We say that a PRKES is *secure* against a distinguishability attack if for a plaintext of its choice, $\mathcal{A}$ cannot distinguish a valid {plaintext/ciphertext} pair from a {plaintext/random-text} pair with probability asymptotically better than 1/2 (for the same chosen plaintext).

An *adaptive chosen-message chosen-ciphertext attack with valid pair creation challenge* includes:

- first phase: $\mathcal{A}$ performs "master probing". We assume that it performs up to p_1 pre-challenge probes. We also assume that the slave is *reset* (to a random state) after the intrusion, that is before the second phase. (This insures that no state information on the slave is known after the probing).

- second phase (valid pair creation challenge): $\mathcal{A}$ is challenged to exhibit p_1+1 valid plaintext/ciphertext pairs.

Security: We say that a PRKES is *secure* against a valid pair creation attack if $\mathcal{A}$ is able to answer the challenge only with an asymptotically small probability (to be computed concretely).

Remark: It should be noted here that our definition is different from the basic definition in [BFN98]. Here the definition follows the one in (adaptive) chosen ciphertext security (See [NY90, RS92, DDN91, BDPR98]). This is due to a difference in the model. In [BFN98], length preserving encryption was considered. As a consequence, their encryption model was deterministic, and thus, their definition required the introduction of an *arbiter* to filter the choice of $\mathcal{A}$ in the second phase (whereas in our case, internal randomization may allow oracle style probing on the challenge). In a recent extended version of their paper [BFN99], a formal model and treatment of length increasing functions was given as well; it formalized an indistinguishability attack. Our indistinguishability attack is of a similar nature.

4 Basic Tools

Ideal hash function We assume the existence of an ideal hash function (a function whose behavior is indistinguishable from a random oracle, see [BR93]). In numerous practical constructions, the validation or indication of the security

of the design was based on such an assumption; we use the assumption in the same manner here. In practice, the ideal hash function may be replaced by a strong hash (such as one based on SHA-1). Then from this hash function, we show how to construct, in a very simple manner, an *ideal length-preserving* one-way function H which – apart from the fact that its output has the same length as its input – has the same properties as an ideal hash function.

Let h be an ideal hash function and l be the size of the hash produced by this function. Let x be a message of size n. We assume, for ease of explanation, that l divides n.

We define H_i as $H_i(x) = h(t||i||h(x))$ (where $||$ denotes the concatenation and t is a tag designated specifically for this usage of the hash function, and which can include a specific number and the length of x).

Then, we can define $H(x)$ as being the concatenation of the $H_i(x)$ so that the size of $H(x)$ matches the size of x:

$$H(x) = H_0||H_1||\ldots||H_{n/(l-1)}$$

If l does not divide n, we simply concatenate with the first bits of an extra $H_{1+(n/(k-1))}$, as to ensure that the sizes match.

It is easy to argue that since each sub-block of H depends on the whole message, that if h is ideal in the random oracle model, H is ideal as well. (In short, this is so since any weakness (say, a bias from randomness or predictability) with H can translate to a weakness on one of the blocks and thus to a weakness with h). We comment that if (unlike the construction above) each block of H is not global (sensitive to all bits) there may be problems. These problems arise either from the definition of ideal hash function and also certain concrete problems as pointed out at in [And95].

While the above construction allows us to build a pseudo-random stream of the same size of the message without relying on other assumptions than ideal-ness of the hash function, we can, for efficiency reasons, replace the above quadratic construction by faster ones. A single regular hashing of the message can be used as a seed for a pseudo-random number generator, which can be, for instance, the PRNG suggested in [AB96]. We can also employ the Panama cipher [DC98] (which combines hashing and stream cipher).

Encryption function We now discuss the properties that we require for the encryption function used by the master (smartcard). We require the encryption to be at least pseudorandom (a pseudorandom permutation or function). This assures "strong security" in the sense that no partial information is revealed about the input given the output (and vice versa). It protects as in semantic security except for the "block repeat problem" where the encryption of the same block is deterministic. If we add a random IV we get rid of this "issue" as well.

Denote by k_{in} the input size of the cipher, and by k_{out} its output size. For symmetric ciphers, without IV we have $k_{in} = k_{out}$. Indeed this may be a pseudorandom function (such as encrypting a block in an EBC or CBC mode; or we may allow adding IV increasing the input size). $k_{in} = k$ is our security parameter

(namely we will achieve security while allowing polynomial probes in its size) and we assume that the encryption above is invertible with probability $1/(2^{(k\delta)})$ for some constant δ (to an adversary which gets to see a concrete given bound of cleartext ciphertext pairs). When validating our design we will assume the encryption to be an ideal cipher, namely a random permutation (so that the probability of inverting above can be easily related to the number of cleartext ciphertext pairs available to a concrete attacker).

5 The Scheme

Let n be an integer, which represents the size of a message to be encrypted, and let $M \in \{0,1\}^n$ be this message. Also let k be a security parameter. We will denote by x the secret key held by the card, and by $E(\cdot)$ and $D(\cdot)$ the encryption and decryption functions used by the card. Finally, we will denote by H a length preserving hash function, as defined in the previous section.

We first present the encryption scheme:

Encryption

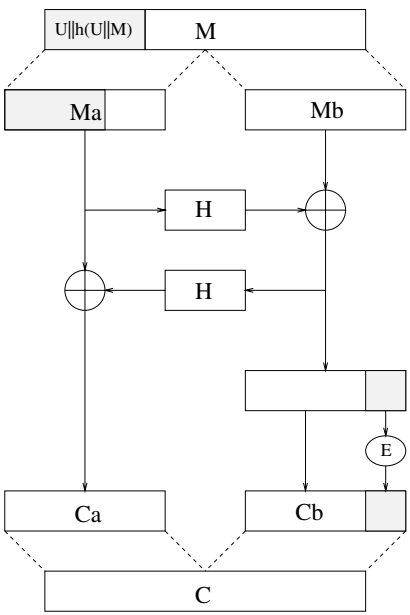

Fig. 1. The Encryption Algorithm

The encryption protocol is sketched in figure 1 and precisely goes as follows:

Slave computation Let M be the plaintext to be encrypted. The slave first chooses an uniformly chosen random number U of size polynomial in the security parameter (say k_{in} bits without loss of generality) and pads M on the left with $R = U||h(M, U)$.

Then, if the resulting string is of odd size, an additional padding of one random bit is performed (this can be dealt with by the definition of the hash function h). Remark: we assume the length of the input is known and fixed, otherwise a byte with the actual length of the randomness can be appended as well; this also enables dealing with encryption of small messages.

Then, the resulting string ($\tilde{M}$) is split in two equal size parts, M_a, M_b ($\tilde{M} = (M_a||M_b)$), and the slave computes:

$$C_b = M_b \oplus H(M_a)$$
$$C_a = M_a \oplus H(C_b)$$

Master computation The slave extracts the k_{in} last bits of C_b (say S) and asks the master to encrypt it. The master computes the encryption T of S ($T = E(S)$) and sends it back. This ciphertext is k_{out} bits long.

Final slave operation The slave finally erases the k_{in} last bits of C_b and replaces them by T.

Decryption

The decryption protocol is as follows:

Master computation Let C be the ciphertext to be decrypted. C is split into two equal parts (assuming the master's encryption is non-expanding), C_a, C_b ($Y = (C_a||C_b)$). The slave extracts the k_{out} last bits of C_b (say T) and asks the master to decrypt this string. The master computes a decryption S of T ($S = D(T)$) and sends it back.

Slave computation The host replaces the k_{out} last bits of C_b by S to obtain $\tilde{C}_b$ ($k_{in} = k_{out}$, if not we simply adapt the lengths and the halving of the intermediate ciphertext). The slave computes:

$$M_a = C_a \oplus H(\tilde{C}_b)$$
$$M_b = \tilde{C}_b \oplus H(M_a)$$

The slave finally recovers the initial random IV at the beginning of M_a and recovers M from the two parts. He checks whether the random IV part is correct (that is if the random IV is of the form $U||h(M, U)$), and if so returns the message M, else it outputs "error".

6 Security

We will next sketch the proof showing that our scheme is secure against an adaptive chosen-message chosen-ciphertext attack. This will validate the design as secure as long as the scrambling function is ideal (i.e., polynomially indistinguishable from a random oracle or intuitively not easily distinguishable from a large random table) and the master's encryption is not easily distinguishable from an ideal encryption. (Based on the ideal hash and encryption we can calculate the attack success by calculating probability of certain detection event performed by the attacker, such as the detection of a valid ciphertext in the challenge).

We will analyze both attacks (challenges): valid pair creation and indistinguishability.

Let us first analyze the master probing phase (which is performed in both attacks). The attacker has taken control over the slave and can query the master without any filtering, both for encryption and decryption. Hence, at the end of the master probing phase, he has potentially gathered a list of p_1 plaintext/ciphertext pairs for the master's inner encryption.

6.1 Against the valid pair creation attack

Informally, the core of the proof is to show that it will be possible to get information on only one valid plaintext/ciphertext pair for the global encryption out of a valid plaintext/ciphertext pair for the master encryption, (except with negligible probability).

With this type of challenge, there is no additional slave probing phase, as we can assume that everything potentially done during the slave probing phase can be done during the master probing phase.

Assume that the attacker $\mathcal{A}$, after performing p_1 queries to the master, is able to produce $p_1 + 1$ valid plaintext/ciphertext pairs for the whole encryption.

Call S the part to be given as input to the master (that is, with the notations of section 5, the k_{in} last bits of $H(R||M_1) \oplus M_2$). Then, we have two cases: either for all $p_1 + 1$ pairs, S is different, or it is the same for at least two of them.

In the first case, $\mathcal{A}$ can be straightforwardly adapted to build an attacker which breaks the master encryption, as it exhibits p_1+1 valid encryption/decryption pairs for the master after only p_1 queries. We assumed the master's encryption can only be inverted with probability $2^{-k_{in}\delta}$ for some small constant δ, hence in the first case, the attacker can fulfill the challenge only with probability $2^{-k_{in}\delta}$ (in the ideal cipher case this probability is easily derived from the number of encryption probes).

In the second case, if two master's inputs are the same, it means that $\mathcal{A}$ was able to find two distinct pairs (U, M), (U', M') such that the two values $H(U||h(M, U)||M_1) \oplus M_2$ and $H(U'||h(M', U')||M_1') \oplus M_2'$ match on their k_{in} last bits.

Let us analyze the probability that such a collision occurs. The above can be rewritten:

$$H(U||h(M,U)||M_1) \oplus H(U'||h(M',U')||M_1') = M_2 \oplus M_2'$$

In order to be able to find such a relation, the hash function H has to be evaluated on its inputs (because of the random oracle model). This means that M_2 and M_2' have to be fixed prior to the evaluation of the left part. So, M_2 and M_2' being fixed, $\mathcal{A}$ was in fact able to find two couples (U, M_1) and (U', M_1') such that $H(U||h(M,U)||M_1) \oplus H(U'||h(M',U')||M_1')$ match on their k_{in} last bits with a given constant.

Now, we consider two sub-cases, whether $h(M,U)$ and $h(M',U')$ are equal or not. If they are not equal, the attack performed in simply a birthday attack on a part of the hash function H, whose probability of success (based on the assumption on H) is $2^{-\frac{k_{in}}{2}}$. If they are equal (which can only happen if M_2 and M_2' match on their k_{in} last bits), then $\mathcal{A}$ was able to find a collision on the hash function, (which occurs only with probability $2^{-k_{in}\gamma}$ for some constant γ given the access limitation of the adversary).

Finally, (under concrete limited access of the adversary to the hash, and encryption boxes, formalized as bound on the success probability), will be able to answer the challenge with probability bounded by:

$$2^{-k_{in}\delta} + 2^{-k_{in}/2} + 2^{-k_{in}\gamma}$$

This is a negligible and controllable concrete probability of successful attack.

6.2 Against the distinguishability attack

Note that, for the distinguishability attack, the attacker can always guess the challenge and due to the challenge being random it has probability 1/2 of being correct. The only other way for the attacker is to probe and perform encryptions and decryptions and match them against the challenge.

In this type of attack, $\mathcal{A}$ first performs p_1 master probes. Then, in the second phase, the attacker has lost control of the slave. He is presented a challenge pair (of which he can choose the plaintext which we call the target plaintext.) His goal is to guess with probability substantially larger than 1/2 whether the challenge pair represents a valid encryption pair (i.e. whether the ciphertext is a correct encryption of the plaintext).

We first consider the security without the third slave probing phase. We will show afterwards that this phase does not help the attacker.

Again, we have two cases: either the k_{out} last bits of the ciphertext match a ciphertext from the master ciphertext list or not. In the first case (which happen with probability $p_1 2^{-k_{out}}$), $\mathcal{A}$ will be able to decrypt and to answer the challenge.

In the second case, in order to be able to answer correctly to the challenge, he as to be able to find out whether the following relations hold:

$$\begin{cases} C_b \oplus M_b = H(M_a) \\ C_a \oplus M_a = H(C_b) \end{cases}$$

However, he has no information about the k_{in} last bits of C_b (but with probability $2^{-k_{in}\delta}$, if he successfully inverts the master's encryption), and he has no information on the k_{in} first bits of M_a (because the slave was reset after the intrusion, and thus he cannot predict the value or the random number chosen by the slave).

So, in order to be able to check with some advantage if the above relations hold, the attacker has to guess the value of H on M_a (to validate the first eq. based on partial availability of C_b) or the value of H on C_b (to validate via the second eq.). This can be done with probability $2^{-k_{in}}$ for each (since only if the right string is guessed it is validated via the random oracle hash).

Hence, with only the initial master probing phase, the probability that $\mathcal{A}$ can correctly answer the challenge is $1/2 + 2^{-k_{in}\delta} + p_1 2^{-k_{out}} + 2^{-k_{in}+1}$.

Now, let us analyze what the attacker can do after the last slave probing phase.

Recall, that $\mathcal{A}$ is allowed to perform p_2 slave probes. He is allowed to query the encryption of any message, and the decryption of any message but the ciphertext challenge.

Of course, we assume that the ciphertext challenge has not already been decrypted, that is the k_{out} last bits of the challenge do not appear in any of the p_1 pairs previously obtained.

If he queries for encryption on a different plaintext than the challenge plaintext, he will not get any information (at best, he will obtain a ciphertext which he can decrypt (with the help of the p_1 pairs previously obtained), and will be able to retrieve the random IV of the slave).

If he queries for encryption on the target plaintext, then he will obtain information on the challenge only if he actually obtains the target ciphertext (which happens if the random IV (which is totally under the control of the slave now) is the same as in the challenge, which happens with probability $2^{-k_{in}}$.

If he queries for decryption, the best he can try is to query for a decryption of a ciphertext which matches the challenge on its k_{out} last bits. In this case, he will be able to find the decryption of the master's encryption on these bits if he can guess the random IV of the slave, that is, again, with probability $2^{-k_{in}}$.

Finally, the probability that he can answer the challenge is:

$$1/2 + 2^{-k_{in}\delta} + p_1 2^{-k_{out}} + 2^{-k_{in}+1} + p_2 2^{-k_{in}}$$

This is, again, a controllable probability (due to the actual limitations of the adversary) that can be made as close to 1/2 as we want as we allow probing, by proper choice of k_{in}, k_{out}.

7 Experimental results

We now briefly comments on the efficiency of our scheme within the suggested context of hardware assisted encryption. We implemented it on a personal computer (a PentiumPro(tm) 200). We chose our security parameters (k_{in} and k_{out}) to be 128 bits long. Times for the scrambling part are summarized in the table below.

length (bits)	1024	2048	4096	8192	16384	32768	65536
time (ms)	0.6	1.0	1.8	3.6	7.1	14.2	28.3

These speeds are not at first very impressive, but they have to be compared to a case where the master device (typically a smartcard) performs all the computation alone. Taking the DES as an example, implementations on smartcards theoretically range from 5 ms to 20 ms per block, but are often in the vicinity of 40 ms in order to prevent differential fault analysis [Nac]. Taking 40 ms for two DES blocks as an optimistic average value, we summarize the encryption speed of a smartcard based DES for messages of the same size as above.

length (bits)	1024	2048	4096	8192	65536
time (ms)	320	640	1280	2560	20480

Assuming that the encryption is pipelined, and choosing to encrypt $98304 = 3.2^{16}$ bits blocks (which makes the master and the slave computation workload about equal), we can encrypt with our scheme at a rate of about 2460 kbits/s, which has to be compared to the 3.2 kbits/s rate of the smartcard alone. We comment that in the context of public-key encryption, assuming that a smartcard can perform an 1024 bit RSA encryption in about one second, our design allows the encryption in one second of a 300 kilobyte message with the smart card's public-key.

8 Conclusion

We have presented the notion of "scramble all, encrypt small." We showed how it fits within the "remotely keyed encryption." The principle demonstrated by the notion can be employed elsewhere as a sub-structure, for example if the scrambling is not "ideal" we may iterate the basic overall encryption many times (adding randomness only at the first iteration, or avoiding adding randomness). Also, we may first perform more "Feistel scrambling" rounds. These types of variations and extensions and their impact on security (and its proof) under different assumptions about the scrambling component are interesting open questions. Variations on attacks, challenges and notions of security are also of interest.

9 Acknowledgments

We would like to thank Mike Matyas and Ron Rivest for pointing out related work and for discussions. We also thank Ross Anderson, Joan Daemen, Antoine

Joux, and the anonymous referees on the program committee for their valuable remarks which improved the presentation of the paper.

References

[AB96] Ross Anderson and Eli Biham. Two practical and provably secure block ciphers: BEAR and LION. In Dieter Gollman, editor, *Fast Software Encryption: Third International Workshop*, number 1039 in LNCS, pages 113–120, Cambridge, UK, 1996. Springer-Verlag.

[AB96] W. Aiello and S. Rajagopalan and R. Vankatesan. High Speed Pseudorandom Number Generation With Small Memory. These proceedings.

[And95] Ross Anderson. The classification of hash functions. In *Codes and Ciphers – Cryptography and Coding IV*, pages 83–93, 1995.

[BDPR98] Mihir Bellare and Anand Desai and David Pointcheval and Philip Rogaway. Relations among notions of security for public-key encryption schemes. In Hugo Krawczyk, editor, *Advances in Cryptology—Crypto '98* Number 1462 in Lectures Notes in Computer Science, pages 26–45. Springer-Verlag, 1998.

[BR93] Mihir Bellare and Philip Rogaway. Random oracles are practical: a paradigm for designing efficient protocols. In *1-st ACM Conference on Computer and Communication Security* Nov. 3–5, 1993, pages 62–73. ACM Press.

[BR94] Mihir Bellare and Philip Rogaway. Optimal Asymmetric Encryption. In Alfredo De Santis, editor, *Advances in Cryptology—EUROCRYPT 94*, number 950 in LNCS, pages 92–111. Springer-Verlag, 1994.

[Bla96] Matt Blaze. High-bandwidth encryption with low-bandwidth smartcards. In Dieter Gollman, editor, *Fast Software Encryption: Third International Workshop*, number 1039 in Lecture Notes in Computer Science, pages 33–40, Cambridge, UK, 1996. Springer-Verlag.

[BFN98] Matt Blaze, Joan Feigenbaum, and Moni Naor. A formal treatment of remotely keyed encryption. In Kaisa Nyberg, editor, *Advances in Cryptology—EUROCRYPT 98*, number 1403 in LNCS, pages 251–265. Springer-Verlag, 1998.

[BFN99] Matt Blaze, Joan Feigenbaum, and Moni Naor. A formal treatment of remotely keyed encryption. Full version of Eurocrypt 98.

[CFN94] Benny Chor, Amos Fiat, and Moni Naor. Tracing traitors. In Yvo G. Desmedt, editor, *Proc. CRYPTO '95*, pages 257–270. Springer, 1994. Lecture Notes in Computer Science No. 839.

[DC98] J. Daeman and C.S.K.Clapp. Fast hashing and stream encryption with PANAMA. In Serge Vaudenay, editor, *Fast Software Encryption: Fifth International Workshop*, number 1372 in Lecture Notes in Computer Science, pages 60–74, 1998. Springer-Verlag.

[DDN91] Danny Dolev, Cynthia Dwork, and Moni Naor. Non-malleable cryptography (extended abstract). In *Proc. of the Twenty Third Annual ACM Symposium on Theory of Computing*, pages 542–552, 6–8 May 1991.

[IBM-pa.] USA patents: 5,796,830 and 5,815,573 to Johnson et al.

[LR88] Michael Luby and Charles Rackoff. How to construct pseudorandom permutations and pseudorandom functions. *SIAM J. Computing*, 17(2):373–386, April 1988.

[Luc96] Stefan Lucks. Faster Luby-Rackoff Ciphers. In Dieter Gollman, editor, *Fast Software Encryption: Third International Workshop*, number 1039 in Lecture Notes in Computer Science, pages 189–203, 1996. Springer-Verlag.

[Luc97] Stefan Lucks. On the security of remotely keyed encryption. In Eli Biham, editor, *Fast Software Encryption: 4th International Workshop*, number 1267 in LNCS, pages 219–229, 1997. Springer-Verlag.

[Luc99] Stefan Lucks. Accelerated Remotely Keyed Encryption. These proceedings.

[MPR97] Stephen M. Matyas, Mohammad Peyravian, and Allen Roginsky. Encryption of Long Blocks Using a Short-Block Encryption Procedure. IBM Technical Report, TR 29.2236, Research Triangle Park, North Carolina, March 1997.

[MPRZ98] Stephen M. Matyas, Mohammad Peyravian, Allen Roginsky and Nev Zunic. Reversible Data Mixing Procedure for efficient public-key encryption. *Computer and Security* V. 17 N. 3, pages 265–272, 1998.

[Mau92] Ueli Maurer. A simplified and generalized treatment of Luby-Rackoff pseudorandom permutation generators. In R. Rueppel, editor, *EUROCRYPT 92*, number in LNCS, pages 239–255. Springer-Verlag, 1993.

[MOV97] Alfred J. Menezes, Paul C. van Oorschot and Scott A. Vanstone. Handbook of Applied Cryptography, 1997. CRC Press LLC.

[Nac] David Naccache. Personal communication.

[NP98] Moni Naor and Benny Pinkas. Threshold Traitor Tracing. In Hugo Krawczyk, editor, *Proc. CRYPTO 98*, number 1462 in LNCS, pages 502–517. Springer-Verlag, 1998.

[NR97] Moni Naor and Omer Reingold. On the construction of pseudo-random permutations: Luby-Rackoff revisited. In *Proc. 29-th STOC* ACM Press, May 4–6 1997, El-Paso, Texas, pages 189–199.

[NY90] Moni Naor and Moti Yung. Public-key cryptosystems provably secure against chosen ciphertext attack. In *Proc. 22-th STOC*, pages 427–437, 1990. ACM.

[Pfi96] Birgit Pfitzmann. Trials of traced traitors. In Ross Anderson, editor, *Information Hiding*, volume 1174 of *Lecture Notes in Computer Science*, pages 49–64, Springer-Verlag, 1996.

[RS92] C. Rackoff and D.R. Simon. Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack. In J. Feigenbaum, editor, *Proc. CRYPTO 91*, number 576 in LNCS, pages 433–444. Springer-Verlag, 1992.

[Riv97] Ronald L. Rivest. All-or-Nothing Encryption and the Package Transform. In Eli Biham, editor, *Fast Software Encryption: 4th International Workshop*, number 1267 in LNCS, pages 210–218, 1997. Springer-Verlag.

[Sch96] Bruce Schneier. *Applied Cryptography: protocols, algorithms, and source code in C*, John Wiley and Sons Inc., 2-d ed., 1996.