Securing RSA-KEM via the AES

J. Jonsson[1] and M.J.B. Robshaw[2]

[1] Department of Mathematics, KTH
SE-100 44 Stockholm, Sweden
jakobj@math.kth.se

[2] Information Security Group
Royal Holloway, University of London
Egham, Surrey, TW20 0EX, U.K.
m.robshaw@rhul.ac.uk

Abstract. RSA-KEM is a popular key encapsulation mechanism that combines the RSA trapdoor permutation with a key derivation function (KDF). Often the details of the KDF are viewed as orthogonal to the RSA-KEM construction and the RSA-KEM proof of security models the KDF as a random oracle. In this paper we present an AES-based KDF that has been explicitly designed so that we can appeal to currently held views on the ideal behaviour of the AES when proving the security of RSA-KEM. Thus, assuming that encryption with the AES provides a permutation of 128-bit input blocks that is chosen uniformily at random for each key k, the security of RSA-KEM against chosen-ciphertext attacks can be related to the hardness of inverting RSA.

Keywords: RSA-KEM, AES, key derivation function.

1 Introduction

The RSA [16] public key cryptosystem has been used for more than twenty years and, during that time, a good understanding of how we might best use the basic encryption primitive has evolved [3, 17, 18]. One recent addition to the literature is the *RSA Key Encapsulation Method (RSA-KEM)* due to Shoup [18]; see [2, 8, 11, 20] for similar constructions. Two attractive features of RSA-KEM are its natural simplicity and its excellent security properties. Very loosely, we can summarise the encapsulation process in the following way:

1. Generate an input w (of appropriate size) at random.
2. Encrypt w using RSA for transport to the recipient.
3. Generate keying material $y = \mathrm{KDF}(w)$ for use in the subsequent symmetric-based session encryption.

It is clear that the intended recipient can recover w from the received ciphertext and then generate y so that both sender and receiver can agree on the same symmetric key. When the underlying key derivation function (KDF) is modelled as a random oracle or a black box, the security of RSA-KEM (in a chosen-ciphertext attack model) can be provably related to the hardness of inverting the RSA primitive.

In this paper we consider the role of the KDF. The properties of the KDF are such that a hash function is often used to build the KDF and there are many dedicated and thoroughly suitable designs. However, since we are likely to appeal to the AES [12] for any subsequent symmetric-based session encryption, it might be preferable to build our KDF out of the AES rather than support an additional algorithm. Furthermore, it might be desirable to have a design based on the AES which would provide some immunity from continued cryptanalysis of current hash function proposals [4, 19].

Of course, it is well-known that a hash function can be built out of a block cipher [5, 9] and, at first sight, it appears that one of these constructions might suffice. However, our work is further motivated by the following goal. Instead of modelling the KDF as a random oracle, we would like to provide an explicit KDF construction that allows us to demonstrate the security of RSA-KEM based upon reasonable assumptions about the underlying block cipher (i.e. the AES). Thus our goal is to obtain a security proof for RSA-KEM under the assumption that the block cipher used in our KDF construction acts as an ideal family of random permutations indexed by the choice of key. Such an assumption on the block cipher is often referred to as the *Shannon*, *ideal cipher*, or *black-box* model and it is used widely (see for example Black et al. [5]).

Now our goal is not difficult to achieve for a block cipher with a sufficiently large block length (say at least twice the desired security level in bits). However, we would particularly like to use the AES, and the only block length permitted for the AES is 128 bits (even though the original cipher Rijndael [6] offered more flexibility in this regard). This is a problem since typical approaches for a block cipher-based KDF appear to be at the mercy of birthday attacks; the security level is bound by only half the block length (i.e., 64 bits in the case of a 128-bit block cipher). Since the standardized block ciphers at our disposal have a block length of either 64 or 128 bits, the security level attained using such mechanisms might not be viewed as adequate. While these birthday attacks may not immediately break the security of the full scheme RSA-KEM, they do seem to make it difficult to achieve a sufficiently-tight security proof.

So the goal of our work has been to achieve the level of security offered by conventional constructions that use a 256-bit block cipher, but to do so via a construction built around a 128-bit block cipher. More generally, in the ideal cipher model and using our construction built around a block cipher with block size k_b, an adversary making q oracle queries should not be able to exploit any weakness with a probability better than

$$\frac{c \cdot q^2}{2^{2k_b}} \tag{1}$$

for some reasonably small constant c. This is approximately as hard as finding collisions for an ideal hash function with output $2k_b$ bits. Our trick in accomplishing this with a 128-bit block cipher is to use an encryption key that is twice the length of the input block; i.e. to use a 128-bit block cipher with a 256-bit key. Thus, our specific construction is valid for the AES and all AES finalists, as well as a range of block ciphers that use 64- and 128-bit block lengths[3].

2 Notation and Specification

Establishing some of the machinery that we need in our construction might initially appear to be somewhat complicated. However the description of the scheme itself is straightforward and can be found in Section 2.2.

2.1 Pre-Requisites

Our particular key derivation function KDF_E is defined in terms of any block cipher E with the property that the key length is at least twice the block length. Let E be a block cipher with block length k_b bits and key length at least $2k_b$ bits. We will assume that the key length is exactly $2k_b$; in the case of a longer key only the first $2k_b$ bits will be used and the other bits will be fixed to some prescribed value. Moreover, we will assume that k_b is a multiple of 8. For each integer $k > 0$ let $\{0,1\}^k$ denote the set of bit-strings of length k. For integers $j \geq 0$ and $k > 0$ with $j < 2^k$, let $(j)_k$ be the k-bit big-endian representation of j (e.g., $(13)_6 = \texttt{001101}$). The concatenation of two bit-strings X and Y will be denoted $X \| Y$. For two bit-strings r_1 and r_2 of the same length, $r_1 \oplus r_2$ denotes the bitwise exclusive-or of r_1 and r_2. In situations where a bit-string r of length k and an integer $j < 2^k$ are combined, the expression $r \oplus j$ denotes the sum $r \oplus (j)_k$. We also use the following notational shorthand. For an integer m and a bit-string $s = v_0 \| v_1$ consisting of 2 blocks v_0 and v_1, each of length k, we set $s \uplus m = (v_0 \oplus m) \| (v_1 \oplus m)$.

In our specification of KDF_E we will appeal to a function δ that "tweaks" the most significant two bits of a string in the following way. Given a bit-string r of length k_b, write $r = (a)_2 \| r'$ (clearly $a \in \{0,1,2,3\}$ and $r' \in \{0,1\}^{k_b-2}$) and define $\delta(r) = \delta((a)_2 \| r') = ((a+1) \bmod 4)_2 \| r'$. The effect of δ is summarized in the following table:

r	$00\|r'$	$01\|r'$	$10\|r'$	$11\|r'$
$\delta(r)$	$01\|r'$	$10\|r'$	$11\|r'$	$00\|r'$

2.2 Definition of KDF_E

Formally, we define KDF_E as $\mathrm{KDF}_E(w, L)$ with two input arguments w and L. The first argument w is the secret input, while the second argument L is an optional label to be associated with the key. Let Valid be the set of valid input pairs (w, L) to KDF_E. To process a pair $(w, L) \in \mathsf{Valid}$, we need to apply a deterministic *encoding function* β to (w, L) to give an input string of an appropriate form (i.e., a sequence of blocks, each of bit length k_b). We also need to generate

[3] Though the security level for 64-bit block lengths is unlikely to be appropriate.

an initial value of bit length k_b from (w, L) using a deterministic *IV generator* $\tau : \mathsf{Valid} \rightarrow \{0,1\}^{k_b}$.

The output from the encoding function β is a string $R = (r_1, \ldots, r_n)$ of blocks r_i, each of bit length k_b. We assume that there is an upper bound $n_{\max}$ on the maximum number of blocks in an output $(r_1, \ldots, r_n) = \beta(w, L)$ with $(w, L) \in \mathsf{Valid}$. We require that it be computationally straightforward to recover w and L in an unambiguous and unique manner from $\beta(w, L)$ and the initial value $t_0 = \tau(w, L)$. Our recommended encoding function $\beta(w, L)$ is specified as $\beta(w, L) = w \| L \| 0^{k_1} \| (l(L))_{64}$; k_1 is the minimum value such that the bit length of $\beta(w, L)$ becomes a multiple of k_b.

The initial value $t_0 = \tau(w, L)$ should contain the length in octets of w (even in applications where the length is fixed) along with a "KDF Mode" indicator.

The output from KDF_E will be a sequence $U = (u_1, \ldots, u_\lambda)$ of blocks u_i, each of bit length k_b. We fix the number λ of blocks to be a constant. For a shorter output, we just truncate U to the desired number of bits.

The specification of $\mathrm{KDF}_E(w, L)$ now follows and consists of two stages. This process is illustrated in Figure 1 provided in Appendix A.

1. Apply the encoding rule to give $\beta(w, L) = (r_1, \ldots, r_n)$ and set the initial value $t_0 = \tau(w, L)$.
2. Extend t_0 to a *padded* initial value $s_0 = t_0 || t_0$ of length $2k_b$.
3. Process the blocks $r_1, \ldots, r_n$ as follows with i running from 1 to n:

$$\begin{aligned} t_{i,0} &= E_{s_{i-1}}(r_i) \oplus r_i \ ; \\ t_{i,1} &= E_{s_{i-1}}(\delta(r_i)) \oplus r_i \ ; \\ s_i &= t_{i,0} \| t_{i,1} \ . \end{aligned} \tag{2}$$

4. Generate λ blocks of output from s_n as follows:

$$u_m = E_{s_n \uplus m}(m) \quad (1 \leq m \leq \lambda) \tag{3}$$

(the k_b-bit representation of m is encrypted). The output is the string

$$U = u_1 \| u_2 \| \ldots \| u_\lambda \ ,$$

which can be truncated to a smaller number of bits if desired.

2.3 Properties of KDF_E

Our construction has some similarity to mechanisms for providing double block-length hash functions out of a block cipher [9]. Schemes such as MDC-2 [10] were designed to give a collision-resistant hash function when using the block cipher DES [13] (with its short block and key sizes as well as unusual complementation and weak key properties) as a building block. While the underlying motivation—to gain a level of security greater than the block size initially allows—is common to both applications, our KDF construction differs in many important ways, not least in how the chaining variables are specified and used.

To gauge the performance of our proposal, we observe that to produce λ blocks of output from n blocks of input, KDF_E requires $2n+\lambda$ applications of E with $n+\lambda$ different keys. The computation can be carried out on $p \geq 2$ parallel processors, each applying the block cipher at most $n + \lceil \lambda/p \rceil$ times. The last λ applications of E are fully parallelizable, whereas the first $2n$ applications are inherently serial with only two computations performed in parallel.

We note that while the AES is a fast cipher, the rate of encryption [7] for the AES with a 256-bit key (which is what we require in our construction) is comparable to the hashing rate of SHA-256 [14] (the NIST hash function that offers a similar level of security to that offered in our construction). Since two invocations of the AES are required at each step of the first stage of KDF_E, we would expect our construction to compare reasonably well to one based on a standardized hash function. Further, since the AES has a particularly lightweight key schedule, even though there is considerable re-keying, we would not expect the overhead to be too significant. Of course, it should also be stressed that if KDF_E is used as a component within RSA-KEM, then the RSA operation is already likely to be a dominating factor (particularly the RSA private key operation) in an application.

2.4 Design Rationale

An overall goal has been to design KDF_E in a manner that puts minimal constraints on the encoding method β; the security of KDF_E should not rely on how inputs are encoded as long as β is reversible. Here we give our rationale behind other aspects to the design of KDF_E.

THE FIRST STAGE IN KDF_E.

The purpose of the first stage of the algorithm is to translate the input into a secret s_n in a collision-resistant manner. Specifically, it should be hard to find two distinct inputs (w, L), (w', L') such that the corresponding outputs s_n, $s'_{n'}$ from the first stage are equal. This is to provide a high level of assurance that different sets of keys are used in the second stage of the algorithm for different inputs. Note that it is easy to find inputs such that the outputs are related in a prescribed manner. Specifically, if we replace the last block r_n in the first stage with $\delta(r_n)$, then the rightmost k_b bits of the new output key coincide with the rightmost k_b bits of the old output key, except that the two leftmost positions in each block may differ. Yet in the ideal cipher model, such a correlation cannot be exploited in a useful manner by an adversary.

In round i of the first stage, the same key s_{i-1} is used for both encryptions. This introduces an effect that we may actually benefit from. Namely, we can control the behaviour of the output key s_i in such a way that collisions with padded initial values are impossible. Indeed, s_i cannot be equal to a padded initial value since this would imply that $E_{s_{i-1}}(r_i) = E_{s_{i-1}}(\delta(r_i))$, which is impossible. The same is true for $s_n \uplus m$; if $E_{s_{n-1}}(r_n) \oplus r_n \oplus m = E_{s_{n-1}}(\delta(r_n)) \oplus r_n \oplus m$, then we would again have $E_{s_{n-1}}(r_n) = E_{s_{n-1}}(\delta(r_n))$, which is impossible. If such collisions had been possible then it would have been difficult to achieve our desired security bound (1) without putting undesirable restrictions on the size of the set of possible initial values.

The Second Stage in KDF_E.

In the second stage we use different keys to derive the blocks u_m and our approach has some similarity to the counter mode of operation for a block cipher [15]. If we were to use a single key, then we would see a small bias in the output due to the non-existence of collisions $E_s(r) = E_s(r')$. This would result in a violation of (1). Indeed, in applications where plenty of output is desirable, the security bound would be weak enough to be a concern in practice.

To minimize the probability of reusing a key, we derive the m^{th} key from s_n by adding a simple counter m to s_n. In this manner, if

$$s_n \uplus m = s'_{n'} \uplus m' \tag{4}$$

for some $m, m' \in \{0, \ldots, \lambda\}$, then

$$s_n = s'_{n'} \uplus (m' \oplus m) = s'_{n'} \uplus m''$$

for some $m'' \in \{0, \ldots, \hat{\lambda}\}$, where

$$\hat{\lambda} = \begin{cases} 0 & \text{when } \lambda = 0; \\ 2^{\lfloor \log_2 \lambda \rfloor + 1} - 1 & \text{otherwise.} \end{cases} \tag{5}$$

Consequently, while there are $(1+\lambda)^2$ pairs $(s_n \uplus m, s'_{n'} \uplus m')$ to be considered in (4), there are only $1 + \hat{\lambda}$ values on $s'_{n'}$ for each s_n that give a collision in (4). In particular, the constant c in our security bound (1) will turn out to be proportional to λ rather than to λ^2.

The keys in the second stage are obtained from s_n by adding the same counter value to each of the two blocks in s_n. This is to ensure that the derived blocks do not collide with padded initial values. The counter starts at 1 to avoid undesirable collisions between keys used in the first stage and keys used in the second stage. For instance, there may be two inputs for which, in the first case, s_n is used as a key in some round $n+1$ of the first stage, while in the second case, the same s_n is used in the second stage[4]. It seems worth taking the precaution to provide this separation.

The use of the function δ.

The function δ is chosen so that $\delta(\delta(r)) \neq r$. Otherwise a function δ' (e.g. one that maps r to $r \oplus d$ for some d) would suffer from the property that if $E_{s_{i-1}}(r_i) \oplus r_i = E_{s_{i-1}}(\delta'(r_i)) \oplus \delta'(r_i)$, then r_i and $\delta'(r_i)$ both yield the same intermediate s_i in (2). This would result in a violation of our bound (1) and a modified security bound would contain a term of the form $c \cdot q/2^{k_b}$.

We have chosen δ to be simple, modifying only two bits of the input. As well as having little impact on efficiency this facilitates the security analysis. To see this, consider the *order* of an element r defined as the smallest integer j such

[4] Essentially the set of possible sequences $(r_1, \ldots, r_n)$ is not necessarily "prefix-free".

that $r = \delta^j(r)$, where δ^j is shorthand for δ applied j times. Clearly the order of any element r with respect to δ is only four. While it seems that any order larger than two would result in a security bound of the form we require (1), analysis could be harder. The reason is as follows.

We say that an input pair (s, r) to the block cipher E is *of relevance* in the i^{th} round of the first stage if $(s, r) = (s_{i-1}, r_i)$ or $(s, r) = (s_{i-1}, \delta(r_i))$. This means that the pair (s, r) is related to the two pairs $(s, \delta(r))$ and $(s, \delta^{-1}(r))$ in an obvious manner. Similarly, $(s, \delta(r))$ is related to $(s, \delta^2(r))$, $(s, \delta^2(r))$ is related to $(s, \delta^3(r))$, and so on and so forth. Consequently, if the order of r were large, then we would have a long chain of related input pairs. This would make it hard to analyse dependencies between pairs of inputs in the first stage of KDF_E, which we require in the context of RSA-KEM. Thus the main benefit of δ as we have defined it is to ensure that the corresponding chain of pairs is short; for the given choice of δ, any set of the form $\{(s, r), (s, \delta(r)), (s, \delta^2(r)), (s, \delta^3(r))\}$ has the property that each pair in the set is only related to other pairs in the set and not to any pairs outside the set. This property makes it easier to obtain stream-lined security proofs.

2.5 Some Related Applications

While an AES-based key derivation function (KDF) for use within RSA-KEM is the focus of our work, we have actually designed something more flexible. Very simple variants and extensions of our KDF design could be used as a *mask generating function* MGF_E, a block-cipher based hash function construction, and as a block-cipher based message authentication code. However, these may compare unfavourably with other, more established, mechanisms [9].

For instance, it is easy to modify $\text{KDF}_E(w, L)$ for use as a hash function and we define the hash function $\text{Hash}_E(M)$ as

$$\text{Hash}_E(M) = \text{KDF}_E(M, \phi) \ ,$$

where ϕ is the empty string. However, we need to make a few minor changes. First, we change the encoding function and set

$$\beta_{\text{HASH}}(M) = \beta(M, \phi) = M \| 0^{k_1} \| (l(M))_{64} \ .$$

Here, $l(M)$ is the length in bits (or bytes) of the message M and k_1 is the minimum value such that the bit length of $\beta_{\text{HASH}}(M)$ becomes a multiple of k_b. Second, we fix the initial value t_0, which should contain a "Hash Mode" indicator, and we set $\lambda = 2$, which would give a hash function with a 256-bit output.

$\text{KDF}_E(w, L)$ can also be used as the basis for a message authentication code. Let the first argument w be the secret key and let the second argument L be the message M to be authenticated (possibly a concatenation of the message and other data). We can define the message authentication code $\text{MAC}_E(w, M)$ as

$$\text{MAC}_E(w, M) = \text{KDF}_E(w, M) \ ,$$

using the same encoding function β as in key derivation mode;

$$\beta_{\mathrm{MAC}}(w, M) = w \| M \| 0^{k_1} \| (l(M))_{64} \ .$$

The initial value t_0 should be fixed and include the length in octets of w (even in applications where the length is fixed) and also a "MAC Mode" indicator. A typical parameter choice would be $\lambda = 1$ or 2 (the latter if collision-resistance is desired). Of course, another possibility would be to define a message authentication code as HMAC [1] with Hash_E as the underlying hash function.

3 $\mathbf{KDF}_{\boldsymbol{E}}$ Within RSA-KEM (and $\boldsymbol{f}$-KEM)

KDF_E is intended for use as an AES-based key derivation function within RSA-KEM [2, 18, 20]. However, to make the discussion as general as possible, we consider an arbitrary trapdoor permutation $f : X_f \to X_f$; see below for a formal treatment of trapdoor permutations. We briefly discuss even more general encryption schemes at the end of Section 4. Let

$$\mathrm{KDF} : X_f \times \mathcal{L} \to \{0, 1\}^*$$

be a key derivation function, where $\mathcal{L}$ is a set of *labels* and $\{0, 1\}^*$ is the set of all finite bit-strings. Then f-KEM is defined as follows, where the input to f-KEM is a label $L \in \mathcal{L}$.

1. Generate an element $w \in X_f$ uniformly at random.
2. Compute $y = f(w)$.
3. Compute $U = \mathrm{KDF}(w, L)$.
4. Output y, the *ciphertext*, and U, the *derived secret*.

In Section 4 we will analyse f-KEM in the special case that the underlying KDF is KDF_E.

For a security parameter k, let $\mathcal{F}_k$ be a finite family of pairs (f, f^{-1}) with the property that f is a permutation with inverse f^{-1}; f takes as input an element x in a set $X = X_f$ and returns an element y in the same set X. We assume that the running time of each of f and f^{-1} is polynomial in k. Let $\mathcal{G}$ be a probabilistic polynomial-time (PPT) algorithm that on input 1^k (i.e., k uniformly random bits) outputs a pair $(f, f^{-1}) \in \mathcal{F}_k$. $\mathcal{G}$ is a *trapdoor permutation generator*. An f-*inverter* $\mathcal{I}$ is an algorithm that on input (f, y) tries to compute $f^{-1}(y)$ for a random $y \in X$. $\mathcal{I}$ has success probability $\epsilon = \epsilon(k)$ and running time $T = T(k)$ if

$$\Pr\left((f, f^{-1}) \leftarrow \mathcal{G}(1^k), y \stackrel{\mathsf{R}}{\leftarrow} X_f : \mathcal{I}(f, y) = f^{-1}(y)\right) \geq \epsilon$$

and the running time for $\mathcal{I}$ is at most T. In words, $\mathcal{I}$ should be able to compute $f^{-1}(y)$ with probability ϵ within time T, where (f, f^{-1}) is derived via the trapdoor permutation generator and y is random. $\mathcal{I}$ solves the f *problem*.

$\mathcal{F}_k$ is a *trapdoor permutation family* with respect to (ϵ, T) if there is no f-inverter with success probability ϵ within running time T. The individual permutation f is referred to as a *trapdoor permutation*.

4 Security Analysis

In this section we prove the security of f-KEM based on KDF_E.

With the random oracle assumption on KDF it is straightforward to prove that f-KEM based on KDF is secure against a chosen-ciphertext adversary if f is a secure trapdoor permutation; see Shoup [18] for details. The purpose of this section is to analyse f-KEM when KDF_E (see Section 3) is used as the underlying KDF. Our goal is to show that the security of f-KEM can be related to the hardness of inverting f if the block cipher E is modelled as an indexed family of random permutations.

The attack model against f-KEM is defined as follows and aligns with the security model for key encapsulation schemes defined in Shoup [18]. The adversary is given free access to a *decryption oracle* that on input (y, L) decrypts y and outputs the corresponding secret $U = \mathrm{KDF}_E(f^{-1}(y), L)$. This means that we consider the family of adaptive chosen-ciphertext attacks (typically referred to as CCA2). The adversary also has free access to an E-oracle and a D-oracle simulating encryption and decryption with the block cipher E.

The task for the adversary is to distinguish a secret U_0 corresponding to a certain *challenge ciphertext* (y^*, L^*) from a random string. To make the challenge nontrivial, we do not allow the adversary to query the challenge ciphertext (y^*, L^*) at the decryption oracle after the challenge ciphertext has been published. However, there are no other restrictions on decryption queries; the adversary may well include either of y^* and L^* in a decryption query as long as the query does not include both.

The attack experiment runs as follows. First, the adversary is given a trapdoor permutation f generated at random. The adversary is allowed to send queries to her oracles during the entire attack and they may be chosen in an adaptive manner depending on responses to previous queries. At any time of the attack — but only once — the adversary sends a label L^* to a *challenge generator*. The challenge generator applies the f-KEM operation, producing a ciphertext y^* and a secret output U_0. In addition, the generator selects a uniformly random string U_1 and flips a fair coin b. The generator returns y^* and U_b; thus the response depends on b.

At the end, the adversary outputs a bit b'. The *distinguishing advantage* ϵ of the adversary is defined as

$$\epsilon = \Pr(b' = b) - \Pr(b' \neq b) = 2\Pr(b' = b) - 1$$

where the probability is computed over all possible trapdoor permutations. The adversary is referred to as an *IND-CCA2* adversary. The main result now follows.

Theorem 1. *Let $\mathcal{A}$ be an IND-CCA2 adversary against f-KEM based on* KDF_E *making q_E queries to the E- and D-oracles and q_f queries to the decryption oracle (including one query to the challenge generator). Let*

$$q = q_E + (n_{\max} + \lambda) \cdot q_f,$$

where $n_{\max}$ is defined in Section 2.1. Assume that $q \leq 2^{k_b}/24$. Moreover, assume that the distinguishing advantage of $\mathcal{A}$ is ϵ' and that the running time is bounded by T'. Then, viewing the block cipher E in the ideal cipher model, there is an f-inverter $\mathcal{I}$ with success probability ϵ and running time T such that

$$\epsilon = \epsilon' - \frac{18(\hat{\lambda}+1)\cdot q^2}{2^{2k_b}} - \frac{q_f}{|X_f|} \tag{6}$$

with $\hat{\lambda}$ defined in (5) and

$$T = T' + O(q \cdot T_f) + O(\lambda \cdot q_f) \ , \tag{7}$$

where T_f is the time needed to compute f on a given input.

The proof of Theorem 1 is given in Appendix B. Here we comment on the security bounds in Theorem 1.

First, consider the difference $\epsilon' - \epsilon$ in success probabilities for the adversary and the inverter. For typical applications, λ will be quite small, say at most 100; this would give $100k_b$ bits of (symmetric) key material as output. Assuming that $\hat{\lambda} = 2^7 - 1$ and $q = 2^{k_b - \mu} \leq T'$ for some μ, the significant term in (6) is equal to

$$\frac{18 \cdot 2^7}{2^{2\mu}} < \frac{2^{12}}{2^{2\mu}} = 2^{12-2\mu} \ .$$

Defining a success probability of an algorithm to be "negligible" if the time-success ratio (time/probability) of the algorithm is at least 2^{k_b}, we may conclude that $\epsilon' - \epsilon$ is "negligible" as long as q is at most 2^{k_b-12}; the running time of the adversary is assumed to be at least q.

Next, consider the running time of the adversary in terms of the inverter. A close examination of the proof of Theorem 1 yields that the term $O(q \cdot T_f)$ in (7) is approximately $4T_f \cdot q \leq 4T_f \cdot T'$. Namely, for each application of the E-oracle simulation, the inverter applies f up to four times. As a consequence, T'/T is approximately $1/(4T_f)$. We may ignore the rightmost term $O(\lambda \cdot q_f)$ in (7) as q_f is typically bounded by a fairly small value such as 2^{48}. Note that the factor T_f is not due to the specific KDF_E construction but rather it is a generic factor that is also present when the entire key derivation function is modelled as a random oracle; see Shoup [18]. Hence, only the factor 4 is actually related to the specifics of KDF_E. To conclude, we lose approximately two bits of tightness with respect to running time when replacing the random oracle with KDF_E.

Remark. While we only consider trapdoor permutations, we conjecture that the proof might extend to general deterministic public-key encryption algorithms [8].

5 Conclusion

In this paper we have introduced and analysed a new key derivation function KDF_E. Defined in terms of a block cipher E, KDF_E has been specifically designed as an AES-based key derivation function for use within the key encapsulation mechanism RSA-KEM [18]. However the KDF_E construction could also be used as the basis for a mask generating function, a hash function, or a message authentication code. While the KDF_E construction might be somewhat unusual, there is considerable value in considering designs that allow us to demonstrate the security of RSA-KEM under reasonable assumptions on the behaviour of AES rather than the black-box behaviour of some *ad-hoc* construction. We leave the definition of alternative proposals as a matter for further research.

References

1. M. Bellare, R. Canetti, and H. Krawczyk. Keying hash functions for message authentication. In N. Koblitz, editor, *Advances in Cryptology – Crypto '96*, LNCS 1109, pp. 1–15, Springer-Verlag, 1996.
2. M. Bellare and P. Rogaway. Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. *Proceedings of the First Annual Conference on Computer and Communications Security.* ACM, 1993.
3. M. Bellare and P. Rogaway. Optimal Asymmetric encryption - How to Encrypt with RSA. In A. De Santis, editor, *Advances in Cryptology – Eurocrypt'94*, LNCS 950, pp. 92–111, Springer-Verlag, 1995.
4. E. Biham and R. Chen. Near-collisions in SHA-0. In M. Franklin, editor, *Advances in Cryptology – Crypto '04*, LNCS 3152, pp. 290–305, Springer-Verlag, 2004.
5. J. Black, P. Rogaway, and T. Shrimpton. Block-Box Analysis of the Block-Cipher-Based Hash-Function Constructions from PGV. In M. Yung, editor, *Advances in Cryptology – Crypto '02*, LNCS 2442, pp. 320–335, Springer-Verlag, 2002.
6. J. Daemen and V. Rijmen. AES Proposal: Rijndael. Version 2. 1999.
7. W. Dai. Performance figures. Available via `www.eskimo.com/~weidai/`.
8. A. Dent. A Designer's Guide to KEMs. In K. Paterson, editor, 9^{th} *IMA Conference on Coding and Cryptography*, LNCS 2898, 133–151, Springer-Verlag, 2004.
9. A. Menezes, P. van Oorschot, and S. Vanstone. Handbook of Applied Cryptography. CRC Press. 1997.
10. C.H. Meyer and M. Schilling. Secure program load with manipulation detection code. In Proceedings of SECURICOM '88, pp. 111–130, 1998.
11. T. Okamoto and D. Pointcheval. REACT: Rapid Enhanced-security Asymmetric Cryptosystem Transform. In D. Naccache, editor, *Topics in Cryptology – CT-RSA 2001*, LNCS 2020, pp. 159–175. Springer-Verlag, 2001.
12. National Institute of Standards and Technology. FIPS 196: The Advanced Encryption Standard. October, 2001. Available via `csrc.nist.gov`.
13. National Institute of Standards and Technology. FIPS 46-2: The Data Encryption Standard. December, 1993. Available via `www.itl.nist.gov/fipspubs/`.
14. National Institute of Standards and Technology. FIPS 180-2: The Secure Hash Standard. August, 2002. Available via `csrc.nist.gov`.
15. National Institute of Standards and Technology. Special Publication SP-800-38A: Recommondation for Block Cipher Modes of Operation – Methods and Techniques. December, 2001. Available via `csrc.nist.gov`.
16. R. Rivest, A. Shamir, and L. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM*, 21 (2), 120–126, February 1978.
17. RSA Laboratories. PKCS #1 v2.1: RSA Cryptography Standard. June 14, 2002. Available via `www.rsasecurity.com`.

18. V. Shoup. *A Proposal for an ISO Standard for Public Key Encryption*. Preprint, December 2001. Available via eprint.iacr.org/2001/112.
19. X. Wang, D. Feng, X. Lai, and H. Yu. Collisions for hash functions MD4, MD5, Haval-128 and RIPEMD. Available via http://eprint.iacr.org/2004/199.
20. Y. Zheng and J. Seberry. Practical Approaches to Attaining Security Against Adaptively Chosen Ciphertext Attacks. In E.F. Brickell, editor, *Advances in Cryptology – Crypto '92*, LNCS 740,pp. 292-304. Springer-Verlag, 1992.

A Pictorial Representation of $\mathbf{KDF}_{\boldsymbol{E}}$

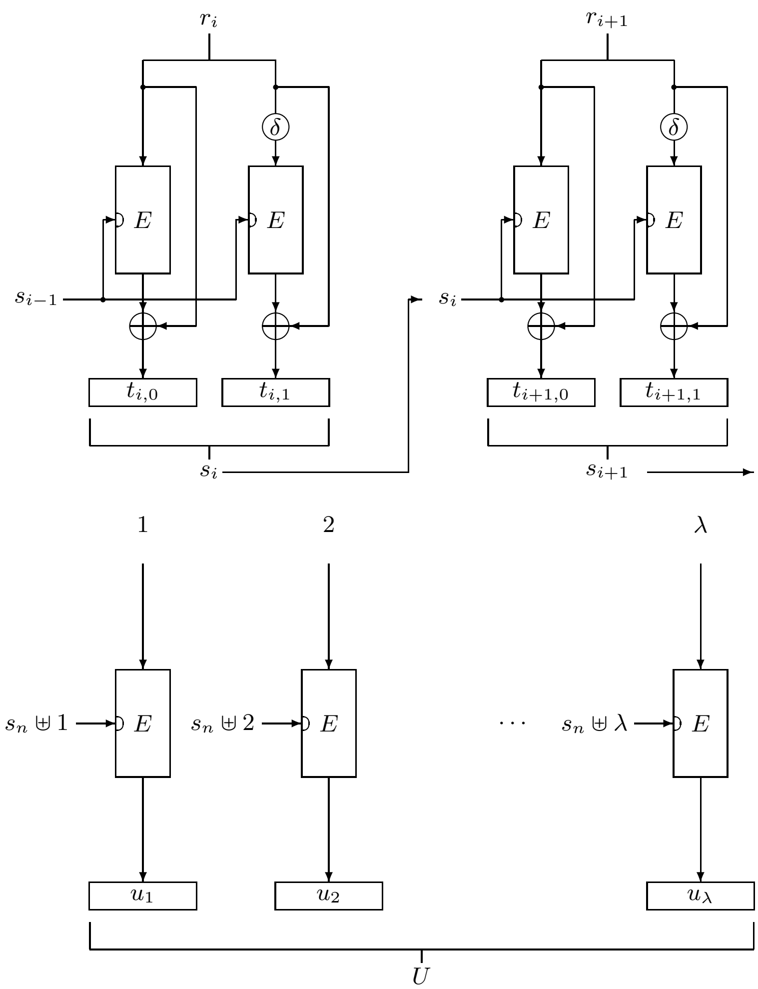

Fig. 1. The two stages of KDF_E; E represents a k_b-bit block cipher with a $2k_b$-bit key.

B Proof of Theorem 1

Theorem 1. Let $\mathcal{A}$ be an IND-CCA2 adversary against f-KEM based on KDF_E making q_E queries to the E- and D-oracles and q_f queries to the decryption oracle (including one query to the challenge generator). Let

$$q = q_E + (n_{\max} + \lambda) \cdot q_f ,$$

where $n_{\max}$ is defined in Section 2.1. Assume that $q \leq 2^{k_b}/24$. Moreover, assume that the distinguishing advantage of $\mathcal{A}$ is ϵ' and that the running time is bounded by T'. Then, viewing the block cipher E in the ideal cipher model, there is an f-inverter $\mathcal{I}$ with success probability ϵ and running time T such that

$$\epsilon = \epsilon' - \frac{18(\hat{\lambda} + 1) \cdot q^2}{2^{2k_b}} - \frac{q_f}{|X_f|}$$

with $\hat{\lambda}$ defined in (5) and

$$T = T' + O(q \cdot T_f) + O(\lambda \cdot q_f) \ ,$$

where T_f is the time needed to compute f on a given input.

Let $\mathcal{A}$ be the adversary. We will define an inverter $\mathcal{I}$ in terms of $\mathcal{A}$ where $\mathcal{I}$ stores information on several lists:

1. f-list: Entries of the form (y, w) (with $y = f(w)$), sorted in alphabetic order with respect to y. Refer to an entry starting with y as a y-entry.
2. KEM-list: Entries of the form (y, L, U) where L, $w = f^{-1}(y)$, and U satisfy $\mathrm{KDF}_E(w, L) = U$. The entries are sorted alphabetically with respect to y and then L. Refer to an entry starting with y as a y-entry.
3. History list: Entries of the form $(s_i; (s_0, r_1, \ldots, r_i))$, sorted with respect to s_i where s_i is derived from s_0 and $r_1, \ldots, r_i$ via $i < n_{\max}$ rounds of (2). Refer to an entry starting with s_i as an s_i-entry.
4. E- and D-list: Entries of the form $(s, (r_1, v_1), \ldots, (r_d, v_d))$ where $v_i = E_s(r_i)$ sorted with respect to s. Within each entry pairs are sorted with respect to r_i on the E-list and with respect to v_i on the D-list. Refer to an entry starting with s as an s-entry. Since the E- and D-lists are essentially the same, we suppress the D-list. Whenever $\mathcal{I}$ requires an output value v, it is implicitly assumed that $\mathcal{I}$ looks on the D-list rather than on the E-list.

Let S_0 be the set of possible padded initial values s_0. Introduce additional sets S_1 and S_2 as follows. S_1 is the set of elements s such that there is an s-entry on the history list, whereas S_2 is the set of elements queried to the E- and D-oracles (by either $\mathcal{A}$ or $\mathcal{I}$) that are *not* contained in S_0 or S_1. At the beginning of the experiment, all lists and all sets except S_0 are empty.

Suppose that $\mathcal{A}$ sends a decryption query (y, L). Then $\mathcal{I}$ proceeds as follows.

F1 If (y, L) is on KEM-list, output the corresponding U and exit.

F2 If no (y, L) is found on KEM-list, check if there is some y-entry on f-list to examine whether $w = f^{-1}(y)$ is known. If w is known, simulate the encryption oracle as specified below to compute $U = \mathrm{KDF}_E(w, L)$, output U, and exit.
F3 In the case that w is unknown, generate a string U as the concatenation of λ uniformly random blocks of length k_b, add (y, L, U) to KEM-list, output U, and exit.

At some point during the attack, the adversary $\mathcal{A}$ requests a challenge ciphertext, providing as input a label L^*. $\mathcal{I}$ proceeds as follows; y^* is the value that he wants to invert.

C1 If (y^*, L^*) is a previous decryption query, output Error and exit.
C2 Generate uniformly random strings U_0 and U_1 of length λk_b. Add (y^*, L^*, U_0) to KEM-list, flip coin b, output (y^*, U_b, U_{1-b}), and exit.

Suppose that $\mathcal{A}$ sends an E-query (s, r). Say that $E_s(r) = v$ is *consistent* if there is no conflict between this assignment and the pairs (r_i, v_i) within the s-entry on E-list. $\mathcal{I}$ proceeds as follows.

E1 If $v = E_s(r)$ is already known, output v and exit.
E2 If $s \notin S_0 \cup S_1$, generate a uniformly random v such that the assignment $v = E_s(r)$ is consistent. Add the pair (r, v) to the s-entry on E-list (introduce the entry if necessary), output v, and exit.
E3 If $s \in S_0 \cup S_1$, for each $j \in \{0, 1, 2, 3\}$ generate a uniformly random string v_j such that the four assignments $v_j = E_s(\delta^j(r))$ are consistent. Add the pairs $(\delta^j(r), v_j)$ to the s-entry on E-list (introduce the entry if necessary).
E4 For $0 \le j \le 3$, let $s^j = (v_j \oplus \delta^j(r)) \| (v_{(j+1) \bmod 4} \oplus \delta^j(r))$. The simulation fails if any of the $4(1 + \lambda)$ elements in

$$\{s^j \uplus m : 0 \le j \le 3, 0 \le m \le \lambda\} \tag{8}$$

are contained in $S_1 \cup S_2$ or collide with each other. Let $\mathsf{E4\text{-}Err}$ be the event that this failure occurs at some point during the attack.
E5 If $s \in S_1$, there is a (unique) s-entry $(s = s_i; (s_0, r_1, \ldots, r_i))$ on the history list. If $s \in S_0$, consider the "empty" entry $(s; (s, -))$ and let $i = 0$ and $s_0 = s$. If $i + 1 < n_{\max}$, then add the entries $(s^j, (s_0, r_1, \ldots, r_i, \delta^j(r)))$ to the history list.
E6 For each $j \in \{0, 1, 2, 3\}$, check whether s_0 and $(r_1, \ldots, r_i, \delta^j(r))$ correspond to a valid input (w, L) to KDF_E (meaning that s_0 is the padded initial value corresponding to (w, L) and $\beta(w, L) = (r_1, \ldots, r_i, \delta^j(r))$). If this is the case:
1. Compute $y = f(w)$ and add (y, w) to f-list (if not present).
2. If there is an entry (y, L, U) on KEM-list, then $\mathrm{KDF}_E(w, L)$ has already been defined (implicitly) as $U = u_1 \| u_2 \| \ldots \| u_\lambda$. If this is the case, for $1 \le m \le \lambda$ assign

$$E_{s^j \uplus m}(m) = u_m \ . \tag{9}$$

For each m introduce an $(s^j \uplus m)$-entry on E-list, add (m, u_m) to the $(s^j \uplus m)$-entry, and remove (y, L, U) from KEM-list.

E7 Output $v_0 = E_s(r)$ and exit.

The s-entry in step E5 being unique follows from the fact that E5 is the only step where we add new entries to the history list; if some of the added keys s^j were already there (thus already contained in the set S_1), the error E4-Err would have occurred in step E4. The assignments in (9) are trivially consistent; arriving at step E6 means that no error occurred in step E4, which implies that $s^j \uplus m$ has never been used as a key before.

Now consider a D-query (s, v) by $\mathcal{A}$. $\mathcal{I}$ proceeds as follows.

D1 If $r = D_s(v)$ is already known, output r and exit.
D2 Generate a random string r such that the assignment $r = D_s(v)$ is consistent and add (r, v) to the s-entry on E-list (introduce the entry if necessary).
D3 Check whether $s \in S_0 \cup S_1$. If this is *not* the case, output r, and exit.
D4 Proceed with steps E3-E6 in the simulation above with s and r, keeping in mind in step E3 that $E_s(r)$ has already been defined as v.
D5 Output $r = D_s(v)$ and exit.

We need to analyse what could go wrong in this simulation. First, we have a possible error in step C1, but this error occurs only if the adversary picks y^* in one of her decryption queries preceding the challenge ciphertext query; denote this event as C1-Err. Since the adversary has no prior information about y^*, $\Pr(\text{C1-Err}) \le q_f/|X_f|$. Note that this value is an extremely small value if f is RSA with key size at least 1024 bits.

The remaining source of error is related to how $\mathcal{I}$ simulates the E- and D-oracles. Besides the event E4-Err in step E4, we also have the potential error that the uniformly random strings generated in steps C2 and F3 may not be consistent with other values. To analyse the probability of this error, we introduce an auxiliary algorithm $\mathcal{J}$ that can compute inverses of f. To make $\mathcal{J}$ indistinguishable from $\mathcal{I}$ for the adversary, we let $\mathcal{J}$ do exactly what $\mathcal{I}$ does during the whole experiment until the adversary exits. At the very end, we add a *checking phase*, where $\mathcal{J}$ proceeds with each entry (y, L, U) on KEM-list, computes $w = f^{-1}(y)$, and simulates the encryption oracle as specified above on all inputs necessary to compute $\mathrm{KDF}_E(w, L)$ (keeping in mind as specified in step E6 that the end result should be U).

Now $\mathcal{J}$, and hence $\mathcal{I}$, will provide a perfect simulation unless an error occurs in step C1 or the error E4-Err occurs in step E4, either during the original experiment or during the additional checking phase. Namely, as long as all responses are consistent and chosen uniformly at random, there is no way for the adversary to distinguish the two simulations.

Before we can estimate the probability of the error E4-Err, we need to count the number of keys s for which $\mathcal{J}$ ever provides some assignment $v = E_s(r)$. Now, while $\mathcal{J}$ simulates the E-oracle on some inputs not queried by $\mathcal{A}$, the underlying key is always part of some other explicit or implicit query from $\mathcal{A}$. This implies that the total number of keys is at most $q = q_E + (n_{\max} + \lambda) \cdot q_f$; the latter term $(n_{\max} + \lambda) \cdot q_f$ estimates the total number of keys used when responding to the decryption queries and the challenge ciphertext query. In particular, the size of S_2 is at most q, as is the number of applications of each of the steps E1-E7.

We also need an upper bound on the total number of assignments $v = E_s(r)$ for any fixed key s. Such a bound is given by $4q$. Namely, each E- and D-query results in at most four assignments, whereas each decryption query and the challenge ciphertext query results in at most $4n_{\max}$ assignments in the first stage and at most λ assignments in the second stage. For the last claim, note that step E6.2 is applied at most once for each entry on KEM-list; the number of entries on this list is bounded by q_f.

In step E3, there are four assignments $v_j = E_s(\delta^j(r))$. We refer to the set $Q = \{(s, \delta^j(r)) : 0 \leq j \leq 3\}$ as a *4-set* and to a pair of the form $\{(s, r), (s, \delta(r))\}$ as a *window*. Within a 4-set Q there are four windows $\{(s, \delta^j(r)), (s, \delta^{j+1}(r))\}$ for $0 \leq j \leq 3$.

To estimate $\Pr(\mathsf{E4\text{-}Err})$, consider a set of assignments to be made in step E3 (steps D2 and E3 in case of a decryption query) corresponding to a 4-set Q and let s be the underlying key. Since $s \in S_0 \cup S_1$ and since it is impossible for a key once in S_2 to end up in S_1 (this would result in an error in step E4), each previous application of the key s must have been an assignment of values for a full 4-set (as opposed to an assignment of a single value as would have been the case if $s \in S_2$). As a consequence, the four values $E_s(\delta^i(r))$ all remain to be assigned.

First, assume that the underlying query was not a decryption query; we did not arrive at step E3 from step D4. For $i \in \{0, 1, 2, 3\}$, the adversary cannot predict the two values $v_j = E_s(\delta^j(r))$ and $v_{j+1} = E_s(\delta^{j+1}(r))$, and hence not the value s^j, with probability better than

$$\frac{1}{(2^{k_b} - 4q)^2} < \frac{1}{2^{2k_b}} \cdot \frac{1}{1 - 8q/2^{k_b}} \leq \frac{1}{2^{2k_b}} \cdot \frac{1}{1 - 1/3} = \frac{3}{2^{2k_b+1}} =: \rho \ . \tag{10}$$

The value $4q$ in the denominator is the upper bound derived above on the number of previous queries with the key s; each such query corresponds to a value $E_s(r')$ that must be different from all $E_s(\delta^i(r))$. The second inequality follows from the assumption $q \leq 2^{k_b}/24$.

Next, assume that we arrived at step E3 from step D4. Thus the first of the four queries in step E3 is a new D-query (s, v_0). As in (10), the adversary cannot predict any two of the four values

$$r = D_s(v_0), v_1 = E_s(\delta(r)), v_2 = E_s(\delta^2(r)), v_3 = E_s(\delta^3(r))$$

with probability better than ρ, not even if the two other values were revealed. In particular, since the adversary needs at least two of these values to determine any s^j, no s^j can be determined with probability better than ρ.

We may now easily compute a bound on the probability that we have a collision between some element in the set (8) and some element in S_2; refer to this event as $\mathsf{E4\text{-}Err2}$. Specifically, for any j and m, $s^j \uplus m$ collides with any fixed element in S_2 with probability at most ρ. Since there are a total of at most

$4(\lambda+1)\cdot q$ values in (8) to be considered during the entire experiment and since $|S_2| \leq q$, we have that

$$\Pr(\textsf{E4-Err2}) \leq \rho \cdot 4(\lambda+1)\cdot q^2 = \frac{3\cdot 4(\lambda+1)\cdot q^2}{2^{2k_b+1}} \leq \frac{6(\hat{\lambda}+1)\cdot q^2}{2^{2k_b}} \;; \tag{11}$$

$\hat{\lambda}$ was defined in (5).

Next, consider the probability of a collision either between two of the elements in the set (8) or between one of these elements and some element in the set S_1; refer to this event as E4-Err1. In such an eventuality, we have distinct (s, r) and (s', r') such that

$$\begin{cases} E_s(r) \oplus r = E_{s'}(r') \oplus r' \oplus m \\ E_s(\delta(r)) \oplus r = E_{s'}(\delta(r')) \oplus r' \oplus m \end{cases} \tag{12}$$

for some integer m (with $0 \leq m \leq \hat{\lambda}$). Now, each element in S_1 corresponds to a 4-set generated in a previous application of steps E3 and E4. This means that we are effectively looking for collisions of the kind (12) with both keys in $S_0 \cup S_1$ and we have up to q different 4-sets among which we want to find a collision.

For any fixed previously known window W, and for each of the four windows within the 4-set Q under consideration, the probability that the two windows satisfy (12) is at most $(\hat{\lambda}+1)\cdot\rho$; use (10) and the fact that there are $\hat{\lambda}+1$ possibilities for m. At the end of the experiment, there are a total of at most $4^2 q(q-1)/2 = 8q(q-1)$ pairs of known windows from *different* 4-sets. This implies that the probability that (12) holds for some pair of this kind is bounded by

$$(\hat{\lambda}+1)\cdot\rho\cdot 8q(q-1)\;. \tag{13}$$

Next, we turn our attention to windows within the same 4-set. Let W_i be the window $\{(s, \delta^i(r)), (s, \delta^{i+1}(r))\}$. For $0 \leq i \leq 3$, the pair (W_i, W_{i+1}) (indices computed modulo 4) cannot satisfy (12). Namely, if

$$\begin{cases} v_i \oplus \delta^i(r) = v_{i+1} \oplus \delta^{i+1}(r) \oplus m \\ v_{i+1} \oplus \delta^i(r) = v_{i+2} \oplus \delta^{i+1}(r) \oplus m\;, \end{cases}$$

then $v_i = v_{i+2}$, which is impossible. The remaining two cases (W_0, W_2) and (W_1, W_3) both result in the same system of equations

$$\begin{cases} v_2 = v_0 \oplus c \oplus m \\ v_3 = v_1 \oplus c \oplus m\;; \end{cases}$$

$c = r \oplus \delta^2(r) = \delta(r) \oplus \delta^3(r) = 100\ldots0$. By (10), the left-hand side cannot be predicted with probability better than ρ even if the right-hand side (i.e., v_0 and v_1) is known. As a consequence, since there are at most q known 4-sets and since m can be chosen in $\hat{\lambda}+1$ ways, the probability that (12) holds for some pair of known windows from the same 4-set is bounded by

$$(\hat{\lambda}+1)\cdot\rho\cdot q\;. \tag{14}$$

Combining (13) and (14), we obtain that

$$\begin{aligned}\Pr(\mathsf{E4\text{-}Err1}) &\leq (\hat{\lambda}+1)\cdot\rho\cdot 8q(q-1) + (\hat{\lambda}+1)\cdot\rho\cdot q\\ &< 8(\hat{\lambda}+1)\cdot\rho\cdot q^2 = \frac{12(\hat{\lambda}+1)\cdot q^2}{2^{2k_b}}\ . \end{aligned} \tag{15}$$

Summing (11) and (15), we conclude that

$$\begin{aligned}\Pr(\mathsf{E4\text{-}Err}) &\leq \Pr(\mathsf{E4\text{-}Err1}) + \Pr(\mathsf{E4\text{-}Err2})\\ &\leq \frac{12(\hat{\lambda}+1)\cdot q^2}{2^{2k_b}} + \frac{6(\hat{\lambda}+1)\cdot q^2}{2^{2k_b}} = \frac{18(\hat{\lambda}+1)\cdot q^2}{2^{2k_b}}\ . \end{aligned} \tag{16}$$

Now return to the original inverter $\mathcal{I}$. Assume that $\mathcal{A}$ is able to guess the bit b with advantage ϵ' in a perfect simulation model. In the model provided by $\mathcal{I}$, the advantage of $\mathcal{A}$ is at least

$$\epsilon = \epsilon' - \Pr(\mathsf{E4\text{-}Err}) - \Pr(\mathsf{C1\text{-}Err}) \geq \epsilon' - \frac{18(\hat{\lambda}+1)\cdot q^2}{2^{2k_b}} - \frac{q_f}{|X_f|}\ ,$$

the subtracted terms bounding the probability that $\mathcal{J}$ fails, in which case $\mathcal{I}$ does not necessarily provide a perfect simulation. To demonstrate that ϵ is at least the success probability of $\mathcal{I}$, note that the only situation where the interactions between $\mathcal{I}$ and $\mathcal{A}$ depend on b is in step E6 when the underlying values y and L coincide with y^* and L^*. Namely, this is the only place where U_0 is used in a way distinguishable from U_1. However, if $\mathcal{I}$ obtains y^* in step E6, then by construction $\mathcal{I}$ obtains it from $w^* = f^{-1}(y^*)$; hence $\mathcal{I}$ wins.