On the Equivalence of Two Models for Key-Dependent-Message Encryption

Alexander W. Dent

Information Security Group,
Royal Holloway, University of London
a.dent@rhul.ac.uk

Abstract. In this paper, we show that the security models for key-dependent-message encryption proposed by Backes *et al.* [1] and Camenisch *et al.* [3] are equivalent for certain classes of function families.

1 Introduction

Standard security definitions for public-key encryption, including ciphertext indistinguishability [5] and non-malleability [4], work on the assumption that an attacker is attempting to determine information about a message that is computationally independent of the private key (in the sense that the messages are computed as polynomial-time functions of the public-key alone). However, there are practical situations in which a public-key encryption scheme may be used to encrypt data which depends directly upon the private key of the encryption scheme. For example, a disk encryption scheme may encrypt a hard-drive which contains the private key of the encryption scheme which is being used to encrypt the hard-drive.

There has been some recent research on the subject of key-dependent message encryption. Most notably, a breakthrough paper by Boneh *et al.* [2] proved the security of an ElGamal-based scheme in an KDM-CPA security model in which the challenge message can be a function of the private key. This was achieved by storing the private key in a suitable encoding, rather than by altering the fundamental mechanics of the encryption scheme. Camenisch *et al.* [3] extended the Boneh *et al.* approach to the KDM-CCA2 security model through the use of Naor-Yung-style techniques [7]. Another approach was taken by Backes *et al.* [1] which provided a more detailed and applicable security model, and proved the security of the OAEP padding scheme in the random oracle model.

The security model of Backes *et al.* significantly extends the simple security model used by Boneh *et al.* and Camenisch *et al.* by proposing a "grey-box" computation model — in contrast to the black-box computation model proposed for groups by Maurer [6] or the white-box computational model essentially proposed by Paillier and Vergnaud [8]. This model allows the attacker to force a user to perform arbitrary computations using any combination of public keys, private keys, or arbitrary values chosen by the attacker. These values are stored in a data structure which is initially not accessible by the attacker, but to which the attacker can request access to data elements of his choice. The data structure therefore has hidden and open elements, and some elements which are technically "hidden" may be known to the attacker as they have been computed in a predictable manner from known inputs. The authors therefore propose a decision system which allows the attacker to determine if a data element is known or not.

The difference between the Camenisch *et al.* (CCS) model [3] and the Backes *et al.* (BDU) model [1] is essentially the difference between a confidential message delivery system and a confidential computational system. The CCS model represents the semantic security of a message that is sent using an encryption scheme, whereas the BDU model represents the semantic security of a system which may perform calculations on an internal memory structure. In other words, the BDU model encapsulates the statement that no attacker can deduce any information about the internal memory structure of a system except that which may be trivially deduced by revealing that information. In a traditional (non-key-dependent-message) encryption scheme, these two security notions coincide; however, more care is required in a key-dependent message system.

In this paper we demonstrate that the CCS and BDU security models are equivalent as long the attacker can request the encryption of a set of functions of the secret keys which satisfies certain properties, most notably that this set of functions include the encryption and decryption algorithms.

2 Key-Dependent Message Encryption

2.1 Preliminaries

A public-key encryption scheme is a triple of PPT algorithms $(\mathcal{G}, \mathcal{E}, \mathcal{D})$. The key generation algorithm $\mathcal{G}$ takes as input a security parameter 1^k and returns a public/private key pair (pk, sk). The public key defines a message space $\mathcal{M}$ and a ciphertext space $\mathcal{C}$. The encryption algorithm $\mathcal{E}$ takes as input a public key pk and a message $m \in \mathcal{M}$, and returns a ciphertext $C \in \mathcal{C}$. The decryption algorithm $\mathcal{D}$ takes as input a private key sk and a ciphertext $C \in \mathcal{C}$, and outputs a message $m \in \mathcal{M}$ or an error symbol. Since our application will involve computational systems, rather than communication systems, we have to define the "error symbol" output more formally, as it may be subject to calculations. We assume that we embed the semantic meaning of a message m into the message space $\mathcal{M} \subseteq \{0,1\}^*$ and define $\perp \subseteq \{0,1\}^*$ to be a distinct subset of "error outputs". This divorces the link between the semantic meaning of the message and the input to the encryption algorithm. For example, we could send a message m by encrypting the string $1\|m \in \mathcal{M}$ and define the error set as $\perp = \{0^n : n \in \mathbb{N}\}$.

For technical reasons we require that the encryption scheme is "length regular" in the sense that (a) the length of the private key is defined by k, (b) the length of the output of the encryption algorithm is defined by the length of the message and the public-key pk, and (c) the length of the output of the decryption algorithm is defined by the length of the ciphertext and the public key. This latter condition should hold even if the output of the decryption algorithm is $\perp$. This formulation slightly alters the correctness requirement, we now require that is is possible to extract m from $D(sk, C)$ if $C \xleftarrow{\$} \mathcal{E}(pk, m)$ and $(pk, sk) \xleftarrow{\$} \mathcal{G}(1^k)$.

2.2 Function Families

We will prove the security of an encryption scheme with respect to a function family $\mathcal{F}$. Despite the fact that we will refer to the elements of the sets as "functions" we actually assume that the elements are PPT algorithms. It is the set of functions f for which the attacker can obtain the encryption of $f(sk_1, \ldots, sk_\ell)$. We have to make some assumptions about the function family in order to prove any security results. We will also assume that the functions are "length regular" in the sense that if $f \in \mathcal{F}$ then $|f(a_1, \ldots, a_n)|$ is defined by f and $(|a_1|, \ldots, |a_n|)$.

Two function families have been considered in the previous works: Backes *et al.* consider the set of all (length-regular) polynomial-time circuits [1] and Boneh *et al.* consider a set consisting of group element multiplications (which is length-regular). If we do consider the set of all polynomial-circuits, then $\mathcal{F}$ contains the circuits $\mathcal{D}(sk_j, \cdot)$ for all possible values of j and yet we do not assume that the attacker can compute these functions. The method by which functions within the family are specified is clearly as important as the function family itself: if a circuit can only be specified by describing that circuit in terms of gates then security may be proven, but if it is possible to specify a function as "the polynomial-time circuit that inverts $\mathcal{E}(pk_i, \cdot)$" then any encryption scheme is trivially insecure in the BDU model. In general, we will assume that the function family $\mathcal{F}$ implicitly includes a "sensible" method of describing it members.

Since the Backes *et al.* model allows for composition of functions, it is sensible to assume that the function family is closed under composition. We define a function family $\mathcal{F}$ to be *BDU compliant* if:

- The constant functions can be efficiently found in $\mathcal{F}$.
- The identity functions $f_j(x_1, \ldots, x_\ell) = x_j$ can be efficiently found in $\mathcal{F}$.

– $\mathcal{F}$ is closed with respect to fixed inputs, i.e. if $f \in \mathcal{F}$ takes n inputs, then for all $1 \leq \alpha \leq n$ and $\beta \in \{0,1\}^*$ the function

$$f_{\alpha,\beta}(a_1, \ldots, a_{n-1}) = f(a_1, \ldots, a_{\alpha-1}, \beta, a_\alpha, \ldots, a_{n-1})$$

can be efficiently found $\mathcal{F}$ (given knowledge of f and β).
– $\mathcal{F}$ is closed with respect to fixed randomness, i.e. if $f \in F$ makes use of a random tape R then the function f_R with the random tape R "hard-wired" into f can be efficiently found in $\mathcal{F}$.
– $\mathcal{F}$ is closed with respect to composition, i.e. if $f_1, f_2 \in \mathcal{F}$, where f_1 takes $n_1 + 1$-inputs and f_2 takes n_2 inputs, then for all $1 \leq \alpha \leq n$ the function

$$f_\alpha(a_1, \ldots, a_{n_1}, b_1, \ldots, b_{n_2}) = f_1(a_1, \ldots, a_{\alpha-1}, f_2(b_1, \ldots, b_{n_2}), a_\alpha, \ldots, a_{n_1})$$

can be efficient found in $\mathcal{F}$ (given knowledge of f_1 and f_2).

2.3 The Boneh *et al.* Security Model

The security model proposed by Boneh *et al.* is very simple and elegant [2]. The version we present here is a slight variant of the version given by Camenisch *et al.* [3]. A scheme is proven secure relative to a class of PPT algorithms $\mathcal{F}$. The security notion involves a PPT attacker $\mathcal{A}$ playing the following game against a challenger: (1) the challenger generates a bit $b \xleftarrow{\$} \{0,1\}$ and a series of ℓ key-pairs $((pk_1, sk_1), \ldots, (pk_\ell, sk_\ell)) \xleftarrow{\$} \mathcal{G}^\ell(1^k)$; (2) the attacker outputs $b' \xleftarrow{\$} \mathcal{A}(pk_1, \ldots, pk_\ell)$. During its execution, the attacker $\mathcal{A}$ can access two oracles:

– An `encrypt` oracle that takes as input a public key index j and two function specifications $(f_0, i_{1,0}, \ldots, i_{n_0,0})$ and $(f_1, i_{1,1}, \ldots, i_{n_1,1})$ where $f_0, f_1 \in \mathcal{F}$, f_b takes n_b inputs and

$$|f_0(sk_{i_{1,0}}, \ldots, sk_{i_{n_0,0}})| = |f_1(sk_{i_{1,1}}, \ldots, sk_{i_{n_1,1}})|\,.$$

The oracle computes $m \leftarrow f_b(sk_{i_{1,b}}, \ldots, sk_{i_{n_b,b}})$ and returns $\mathcal{E}(pk_j, m)$.
– A `decrypt` oracle takes as input a ciphertext $C \in \mathcal{C}$ and an index j, and returns $\mathcal{D}(sk_j, C)$. The attacker is forbidden from submitting a ciphertext C for decryption under private key sk_j if C was returned by the encryption oracle for some message encrypted under pk_j.

The attacker's advantage is defined to be:

$$\begin{aligned} Adv^{\text{ccs}}_{\mathcal{A}}(k) &= |\Pr[b' = 1 \,|\, b = 1] - \Pr[b' = 1 \,|\, b = 0]| \\ &= 2 \cdot |\Pr[b' = b] - 1/2|\,. \end{aligned}$$

An encryption scheme is KDM-CCS-CCA2 secure if every PPT attacker $\mathcal{A}$ has negligible advantage in winning the above game. The authors note that this security model is only interesting if $\mathcal{F}$ contains at least all constant functions and the identity function.

2.4 The Backes *et al.* Security Model

Backes *et al.* [1] consider a more complex security model in which encryption (and other operations) are performed in a distinct memory structure to which that attacker does not have direct read or write access. This structure is essentially identical to the abstract computation model given by Maurer [6] and which is implicitly used by Paillier and Vergnaud [8] in their results about the feasibility of proving the security of discrete-logarithm-based signature schemes.

The attacker interacts with a memory array to which it does not have direct read or write access. We refer to the contents of the i-th memory location as $M[i]$. The array is initially populated with ℓ data items, defined by the security problem/model under consideration, in the memory locations $1, 2, \ldots, \ell$. The attacker has the ability to perform computations from the class $\mathcal{F}_c$ on the contents of these memory locations, via a request of the form $\texttt{eval}(f, i_1, \ldots, i_n)$ where $f \in \mathcal{F}_c$ and $f : (\{0,1\}^*)^n \to \{0,1\}^*$. The value $f(M[i_1], \ldots, M[i_n])$ is computed and the value stored in

the next free memory location. The attacker also has the ability to gain information about the contents of the memory array via a class of functions $\mathcal{F}_r$. If the attacker makes a request of the form $\texttt{ret}(f, i_1, \ldots, i_n)$ where $f \in \mathcal{F}_r$ and $f : (\{0,1\}^*)^n \to \{0,1\}^*$, then the value $f(M[i_1], \ldots, M[i_n])$ is computed and returned to the attacker.

This security model is used by Maurer [6] to prove results about the generic hardness of certain group-based computational problems and by Paillier and Vergnaud [8] to prove results about the difficulty in proving the security of discrete-logarithm-based signature schemes. Maurer's black-box group model can be recovered by insisting that all memory items are elements of a group G, that the set of computation functions $\mathcal{F}_c$ contains only maps from group elements to group elements, and that the set of reveal functions $\mathcal{F}_r$ contains only predicates on group elements. Paillier and Vergnaud's white-box model can be recovered by insisting that all memory items are elements of a group G, that the set of computation functions $\mathcal{F}_c$ contain only functions which output group elements, and that the set of reveal functions $\mathcal{F}_r$ is the set $\{\texttt{reveal}\}$, where the reveal function is defined as $\texttt{reveal}(M[i]) = M[i]$. In such a situation, we will change the notation $\texttt{ret}(\texttt{reveal}, i)$ to $\texttt{reveal}(i)$.

The Backes *et al.* security model for key-dependent message encryption (KDM-BDU-CCA2) is based on the Maurer model of computation. The basic form of the security game between a PPT attacker $\mathcal{A}$ and a challenger is as follows: (1) the challenger generates a bit b and ℓ key-pairs $((pk_1, sk_1), \ldots, (pk_\ell, sk_\ell)) \xleftarrow{\$} \mathcal{G}^\ell(1^k)$; (2) the attacker generates a bit $b' \xleftarrow{\$} \mathcal{A}(pk_1, \ldots, pk_\ell)$. During its execution, the attacker has access to an abstract memory array which has been preloaded with the private keys $sk_1, \ldots, sk_\ell$ in the memory location 1 to ℓ. The attacker can interact with the memory array via the following commands:

- An `encrypt` oracle, which takes as input indices i and j, and places $\mathcal{E}(pk_j, M[i])$ into the next available memory location.
- A `decrypt` oracle, which takes as input indices i and j, and places $\mathcal{D}(sk_j, M[i])$ into the next available memory location.
- A `challenge` oracle, which takes as input two indices i_0, i_1 such that $|M[i_0]| = |M[i_1]|$, and places $M[i_b]$ into the next available memory location.
- An `eval` oracle, which takes as input a polynomial-sized circuit f which takes n arguments, and n indices $(i_1, \ldots, i_n)$, and places $f(M[i_1], \ldots, M[i_n])$ into the next available memory location.
- A `reveal` oracle, which takes as input an index i, and returns $M[i]$ to the attacker.

We use the convention that the run-time of $\mathcal{A}$ includes the time taken for the challenger to compute functions f used in `eval` queries. Hence, if we say that $\mathcal{A}$ is polynomial-time, then we mean that there exists a polynomial which bounds the run-time of $\mathcal{A}$ and all the functions f which $\mathcal{A}$ causes to be evaluated by the `eval` oracle. Note that it is always possible to compute $|M[i]|$ without knowing $M[i]$ using the fact that we can predict the length of the output of $\mathcal{E}$ and f for all $f \in \mathcal{F}$.

For notational convenience, we abbreviate the statement "the attacker makes a query to `oracle` on indices $(i_1, \ldots, i_n)$ and stores the result in memory location o" to "the attacker makes a $o \leftarrow \texttt{oracle}(i_1, \ldots, i_n)$ query". The challenger determines the knowledge that the attacker has of the memory array as the smallest subset $knowset \subseteq \mathbb{N}$ which respects the following rules (noting that if $j \in knowset$ for $1 \leq j \leq \ell$ then the attacker knows sk_j)[1]:

- If the attacker made a $\texttt{reveal}(i)$ query, then $i \in knowset$.
- Suppose the attacker made a $o \leftarrow \texttt{encrypt}(i, j)$ query. If $\{j, o\} \subseteq knowset$ then $i \in knowset$.
- Suppose the attacker made a $o \leftarrow \texttt{decrypt}(i, j)$ query. If $o \in knowset$ then $i \in knowset$.
- Suppose the attacker made $o_1 \leftarrow \texttt{encrypt}(i_1, j)$ and $o_2 \leftarrow \texttt{decrypt}(i_2, j)$ queries. If $M[o_1] = M[i_2]$ and $o_2 \in knowset$ then $i_1 \in knowset$.
- Suppose the attacker made an $o \leftarrow \texttt{eval}(f, i_1, \ldots, i_n)$ query. If $o \in knowset$, then $\{i_1, \ldots, i_n\} \subseteq knowset$.

[1] The original definition included a rule that stated that if $o \leftarrow \texttt{challenge}(i_0, i_1)$ and $o \in knowset$ then $i_0, i_1 \in knowset$. However, this is unnecessary as we exclude attackers for which $o \in knowset$.

The attacker's advantage is defined to be

$$\begin{aligned} Adv^{\text{BDU}}_{\mathcal{A}}(k) &= |\Pr[b' = 1 \mid b = 1] - \Pr[b' = 1 \mid b = 0]| \\ &= 2 \cdot |\Pr[b' = b] - 1/2| \,. \end{aligned}$$

An encryption scheme is KDM-BDU-CCA2 secure if every PPT attacker $\mathcal{A}$ (subject to the restriction that every $o \leftarrow \texttt{challenge}(i_0, i_1)$ query has $o \notin \mathit{knowset}$) has negligible advantage in winning the above game.

3 Relation Between Security Notions

We relate the CCS and BDU notions of security via the following two theorems:

Theorem 1. *Let $\mathcal{F}$ be a function family for which all constant functions can be efficiently found. If there exists an attacker against the KDM-CCS-CCA2 security of an encryption scheme $(\mathcal{G}, \mathcal{E}, \mathcal{D})$ which runs in time t and has advantage ϵ, then there exists an attacker against the KDM-BDU-CCA2 security which runs in time approximately t and has advantage ϵ.*

Proof Suppose there exists an attacker $\mathcal{A}$ against the KDM-CCS-CCA2 security of an encryption scheme $(\mathcal{G}, \mathcal{E}, \mathcal{D})$. We construct an attacker $\mathcal{A}'(pk_1, \ldots, pk_\ell)$ against the KDM-BDU-CCA2 security of $(\mathcal{G}, \mathcal{E}, \mathcal{D})$ as follows:

1. Initialise a counter $ctr \leftarrow \ell + 1$. This tracks the next available memory element.
2. Run $b' \xleftarrow{\$} \mathcal{A}(pk_1, pk_\ell)$.
   - If $\mathcal{A}$ makes an `encrypt` query for public key index j, and function specifications $(f_0, i_{1,0}, \ldots, i_{n_0,0})$ and $(f_1, i_{1,1}, \ldots, i_{n_1,1})$, then $\mathcal{A}'$ makes:
     - A $ctr \leftarrow \texttt{eval}(f_0, (i_{1,0}, \ldots, i_{n_0,0}))$ query and increments ctr.
     - A $ctr \leftarrow \texttt{eval}(f_1, (i_{1,1}, \ldots, i_{n_1,1}))$ query and increments ctr.
     - A $ctr \leftarrow \texttt{challenge}(ctr - 1, ctr - 2)$ query and increments ctr.
     - A $ctr \leftarrow \texttt{encrypt}(ctr - 1, j)$ query and increments ctr.
     - A $\texttt{reveal}(ctr - 1)$ query and returns the resulting ciphertext $\mathcal{A}$.
   - If $\mathcal{A}$ makes a `decrypt` query on a ciphertext C and public key index j, then $\mathcal{A}'$ makes:
     - A $ctr \leftarrow \texttt{eval}(f_C)$ query and increments ctr, where f_C is the constant function which always returns C.
     - A $ctr \leftarrow \texttt{decrypt}(ctr - 1, j)$ query and increments ctr.
     - A $\texttt{reveal}(ctr - 1)$ query and returns the resulting message to $\mathcal{A}$.
3. Output b'.

It is easy to see that $\mathcal{A}'$ implements the oracles to which $\mathcal{A}$ has access and so $\mathcal{A}'$ outputs 1 if and only if $\mathcal{A}$ outputs 1 *and* $\mathcal{A}'$ is a legal attacker in the BDU model. In order to prove this, we must demonstrate that the address of the output of the challenge oracle is never in *knowset*. This is easy to see as (a) the attacker never reveals any private keys sk_j and (b) that the attacker may not query the decryption oracle on any ciphertext returned by the encryption oracle. □

The second case is more complex.

Theorem 2. *Suppose $\mathcal{F}$ is a BDU-compliant function family containing $\mathcal{E}$ and $\mathcal{D}$. If there exists an attacker against the KDM-BDU-CCA security which runs in time t and has advantage ϵ, then there exists an attacker against the KDM-CCS-CCA security which runs in time approximately t and has advantage at least $\epsilon/2^\ell$.*

Proof Suppose there exists an attacker $\mathcal{A}$ against the KDM-BDU-CPA security of an encryption scheme $(\mathcal{G}, \mathcal{E}, \mathcal{D})$. We assume that for all $f \in \mathcal{F}$ it is possible to determine the maximal possible size of randomness that f takes as input and that this value is polynomially bounded. This always holds since the run-time of $\mathcal{A}$ includes the time taken to run evaluate the functions f that $\mathcal{A}$ causes

to be invoked and is polynomially-bounded. We will write $f(x; R)$ to denote that the function f is being run on the input x using the randomness R.

We construct an attacker $\mathcal{A}'(pk_1, \ldots, pk_\ell)$ against the KDM-CCS-CPA security of $(\mathcal{G}, \mathcal{E}, \mathcal{D})$ that uses a simulated enhanced memory array, in which each element in the array will store up to two items and each item is either a value or a function in $(sk_1, \ldots, sk_\ell)$. If the memory array contains two values at location i then these are referred to as $M[i, 0]$ and $M[i, 1]$ (with the conventional that $M[i, 0] = M[i, 1]$ if the memory location only stores one value). The memory elements $M[i, b]$ will hold (representations of) the data values that would be computed if the bit used by the `challenge` oracle was b. The algorithm runs as follows:

1. Pick a random subset $J \subseteq \{1, 2, \ldots, \ell\}$ and set $\bar{J} = \{1, \ldots, \ell\} \setminus J$. The set J is the guess $\mathcal{A}'$ as to which secret key values $\mathcal{A}$ will explicitly or implicity reveal.
2. For each $j \in J$, compute $(pk_j, sk_j) \xleftarrow{\$} \mathcal{G}(1^k)$ and set $M[j, 0] = M[j, 1] = sk_j$. This process over-writes the previous values of pk_j supplied as part of the input. These memory locations are labelled "exact".
3. For each $j \in \bar{J}$, set $M[j, 0] = M[j, 1] = f_j$ where f_j is the function that takes ℓ inputs and outputs the j-th input. These memory locations can be now be thought of as holding functions in $(sk_1, \ldots, sk_\ell)$ where the j-th function outputs sk_j for each $j \in \bar{J}$. These values are in memory locations are labelled "function".
4. Run $b' \xleftarrow{\$} \mathcal{A}(pk_1, \ldots, pk_\ell)$.
   - If $\mathcal{A}$ makes a `reveal`(j) query and $M[j]$ contains a single, exact value, then return $M[j]$. Otherwise, $\mathcal{A}'$ outputs $\perp$ and aborts.
   - If $\mathcal{A}$ makes an $o \leftarrow$ `eval`$(f, (i_1, \ldots, i_n))$ query then for each $b \in \{0, 1\}$:
     - If $M[i_1, b], \ldots, M[i_n, b]$ are memory elements which are exact, then compute the function $f(M[i_1, b], \ldots, M[i_n, b])$ and store this result in $M[o, b]$. The memory location $M[o, b]$ is declared to be exact.
     - If not all $M[i_1, b], \ldots, M[i_n, b]$ are exact, then we build a function f^* in the variables $(sk_1, \ldots, sk_\ell)$ to represent the output. We pick a suitably sized random string R for f and initially set $f^*(\cdot) = f(\cdot; R)$, then for each input index i_α:
       * If $M[i_\alpha, b]$ is an exact memory element, then set f^* to be the function obtained by fixing the α-th input of f^* to be the constant value $M[i_\alpha, b]$. This function is in $\mathcal{F}$ by the "fixed-value closure" property of $\mathcal{F}$.
       * If $i_\alpha \in \bar{J}$, then no change to f^* is made. I.e. f^* takes sk_{i_α} as input in this position.
       * If $i_\alpha > \ell$ and $M[i_\alpha, b]$ is a function memory element, then $M[i_\alpha, b]$ must be associated with a function $f_{i_\alpha}(sk_1, \ldots, sk_\ell)$. Let f^* be the function which is the composition of f^* with f_{i_α} in the α-th input position. This function is in $\mathcal{F}$ by the "composition closure" property of $\mathcal{F}$.

     The function f^* is associated with memory element $M[o, b]$ and this value is declared to be function.
   - If $\mathcal{A}$ makes an $o \leftarrow$ `challenge`(i_0, i_1) query then $\mathcal{A}'$ sets $M[o, 0] = M[i_0, 0]$ and $M[o, 1] = M[i_1, 1]$. This step is critical as it ensures that the memory location $M[o, b]$ holds the correct value which would be computed if the challenge bit is b.
   - If $\mathcal{A}$ makes an $o \leftarrow$ `encrypt`(i, j) query for $j \in J$ then for each $b \in \{0, 1\}$:
     - If $M[i, b]$ contains an exact value then $\mathcal{A}'$ computes $M[o, b] = \mathcal{E}(pk_j, M[i, b])$ and declares this memory locations to be exact.
     - If $M[i, b]$ contains a function f then $\mathcal{A}'$ generates a random value R for the encryption algorithm, stores the function $\mathcal{E}(pk_j, f(\cdot); R)$ at $M[o, b]$ and declare this memory location to be a function. This new function is in $\mathcal{F}$ since $\mathcal{E} \in \mathcal{F}$ and by the "composition closure" property of $\mathcal{F}$.
   - If $\mathcal{A}$ makes an $o \leftarrow$ `encrypt`(i, j) query for $j \in \bar{J}$ then $\mathcal{A}'$ makes an `encrypt` query using the functions $M[i, 0]$ and $M[i, 1]$. (If either $M[i, 0]$ or $M[i, 1]$ is an exact value then the corresponding constant function is passed instead.) $\mathcal{A}'$ sets $M[o, 0]$ and $M[o, 1]$ to be equal to the received ciphertext C. $\mathcal{A}'$ declares these memory locations to be exact.
   - If $\mathcal{A}$ makes an $o \leftarrow$ `decrypt`(i, j) where $M[i, b]$ is an exact value, then

- If $j \in J$ then $\mathcal{A}'$ computes $M[o,b] \leftarrow \mathcal{D}(sk_j, M[i,b])$. This memory location is declared to be exact.
- If $j \in \bar{J}$ then $\mathcal{A}'$ queries its decryption oracle on $M[i,b]$ and stores the result in $M[o,b]$. This memory location is declared to be exact.

– If $\mathcal{A}$ makes an $o \leftarrow \texttt{decrypt}(i,j)$ where $M[i,b]$ is a function, then there is a function $f(sk_1, \ldots, sk_\ell)$ associated with $M[i,b]$ and
- If $j \in J$ then $\mathcal{A}'$ associates the function $\mathcal{D}(sk_j, f(\cdot))$ with $M[i,b]$. This function is in $\mathcal{F}$ due to the fixed inputs, the fact that $\mathcal{D} \in \mathcal{F}$ and composition closure properties of $\mathcal{F}$. This memory location is declared to be a function.
- If $j \in \bar{J}$ then $\mathcal{A}'$ associates the function $\mathcal{D}(f_j(\cdot), f(\cdot))$ with $M[i,b]$ where $f_j(sk_1, \ldots, sk_\ell) = sk_j$ is the function. This function is in $\mathcal{F}$ due to the composition closure properties of $\mathcal{F}$. This memory location is declared to be a function.

Let $J' = \{1, 2, \ldots, \ell\} \cap \mathit{knowset}$ — i.e. the indices of the private keys which are known by the attacker. We claim that if $J = J'$ then $\mathcal{A}'$ is a valid attacker in the CCS security model and $\mathcal{A}'$ correctly simulates the oracles for $\mathcal{A}$ and so $\mathcal{A}'$ outputs 1 if and only if $\mathcal{A}$ outputs 1. Since $J' = J$ with probability $1/2^\ell$ this would mean that $\mathcal{A}'$ has advantage $\epsilon/2^\ell$. It therefore remains to show if $J = J'$ then $\mathcal{A}'$ correctly simulates the oracles for $\mathcal{A}$.

The reveal oracle works correctly as long as it is only called on memory locations that contain single, exact values. $\mathcal{A}'$ will abort if:

– $\mathcal{A}$ makes a $\texttt{reveal}(i)$ oracle query on a memory location $M[i]$ containing two values. This can only occur if there exists a chain of commands which means that $M[i]$ depends on the result of an $o \leftarrow \texttt{challenge}(i_0, i_1)$ query and that this chain cannot contain any `encrypt` commands using public keys pk_j for $j \in \bar{J}$. Hence, if $i \in \mathit{knowset}$ then $o \in \mathit{knowset}$ which is a contradiction as $\mathcal{A}$ is a valid BDU attacker.

– $\mathcal{A}$ makes a $\texttt{reveal}(i)$ oracle query on a memory location $M[i]$ containing a function. This will occur if there exists a chain of commands which means that $M[i]$ depends on some sk_j for $j \in \bar{J}$ and that this chain cannot contain any `encrypt` commands using public keys $pk_{j'}$ for $j' \in \bar{J}$. Hence, if $i \in \mathit{knowset}$ then $j \in \mathit{knowset}$ which is a contradiction as $j \in \bar{J}$.

The evaluation oracle works correctly. It always holds representations of the values that would be computed if $\mathcal{A}$ were really interacting with the BDU game, where these representations are either the exact values or functions of the unknown private key values sk_j for $j \in \bar{J}$. Note that while $\mathcal{A}'$ generates the values R used by probabilistic functions, this value is not directly revealed to $\mathcal{A}$. Hence, the "de-randomisation" of these functions does not affect $\mathcal{A}$'s execution.

The challenge oracle also works correctly. Recall that memory location $M[i,b]$ holds the data value that would be held in location $M[i]$ if the challenge bit was b. Consider a command $o \leftarrow \texttt{challenge}(i_0, i_1)$. If $b = 0$ then $M[o]$ should contain the value of $M[i_0]$ which would have been computed in the game with $b = 0$, i.e. $M[o,0] = M[i_0,0]$. Similar, if $b = 1$ then $M[o]$ should contains the value of $M[i_1]$ that would have been computed in the game with $b = 1$, i.e. $M[o,1] = M[i_1,1]$.

The encrypt oracle works correctly. Again, note that the function $\mathcal{E}(pk_j, f(\cdot); R)$ correctly simulates the view that $\mathcal{A}$ should be given. $\mathcal{A}$ does not know the value R even though it is given to $\mathcal{A}'$. It is interesting to note that only an $\texttt{encrypt}(i,j)$ command for some $j \in \bar{J}$ can map a memory location with two function/exact inputs to a single exact output. This is as this is the only command that "blocks" the backward action in the computation of *knowset*. Hence, the contents of $M[o]$ should be able to be revealed without revealing the contents $M[i]$.

The decryption oracle is clearly functionally correct; however, there may be some suspicion that the use of $\mathcal{D}$ as a function (with different knowledge rules) may leak information to the attacker which should not be available. This is not correct – recall that we are simply simulating an environment for $\mathcal{A}$. If the environment correctly matches the BDU model then $\mathcal{A}$ will break the scheme with advantage $Adv^{\text{BDU}}_{\mathcal{A}}(k)$.

Since the oracles are correctly simulated, we have the result. □

Corollary 1. *If $\mathcal{F}$ is the set of all polynomial-time circuits, 2^ℓ is polynomially bounded, and Π is KDM-CCS-CCA2 secure, then Π is KDM-BDU-CCA2 secure.*

There are a couple of points to note about the proof. The first is that the factor of 2^ℓ is likely to be optimal for black-box reductions. This is because, unlike the CCS model, the BDU model allows the adversary to corrupt entities and obtain their private keys. These keys may not be used in computing a "challenge" ciphertext which is revealed to the attacker. Therefore, if we are trying to simulate the BDU environment, we must decide which indices $j \in \{1, \ldots, \ell\}$ correspond to keys which can be corrupted and which correspond to keys which can be used to compute challenge ciphertexts. We will guess this correctly with probability $1/2^\ell$. In some sense, this loss is equivalent to the factor of ℓ that would be lost when moving from a multiple-user-with-corruption model to a single-user model for traditional encryption schemes (without key dependent messages).

Secondly, the restriction that $\mathcal{D} \in \mathcal{F}$ may be prohibitive. While this holds for the results in Backes *et al.* paper [1] where the function family $\mathcal{F}$ consists of all polynomial-time circuits, it does not hold for the variants of the Boneh *et al.* scheme [2]. In this case, we may only prove the following simpler case:

Theorem 3. *Suppose $\mathcal{F}$ is a BDU-compliant function family which contains $\mathcal{E}$. If there exists an attacker against the KDM-BDU-CPA security which runs in time t and has advantage ϵ, then there exists an attacker against the KDM-CCS-CPA security which runs in time approximately t and has advantage at least $\epsilon/2^\ell$.*

Acknowledgements

The author would like to thank Georg Fuchsbauer and Dominique Schröder (and the entire ECRYPT II MAYA working group on the design and analysis of primitives and protocols) for interesting discussions on this topic. The work described in this report has in part been supported by the Commission of the European Communities through the ICT program under contract ICT-2007-216676 ECRYPT II. The information in this document is provided as is, and no warranty is given or implied that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability.

References

1. M. Backes, M. Dürmuth, and D. Unruh. OAEP is secure under key-dependent messages. In J. Pieprzyk, editor, *Advances in Cryptology – Asiacrypt 2008*, volume 5350 of *Lecture Notes in Computer Science*, pages 506–523. Springer-Verlag, 2008.
2. D. Boneh, S. Halevi, M. Hamburg, and R. Ostrovsky. Circular-secure encryption from decision Diffie-Hellman. In D. Wagner, editor, *Advances in Cryptology – Crypto 2008*, volume 5157 of *Lecture Notes in Computer Science*, pages 108–125. Springer-Verlag, 2008.
3. J. Camenisch, N. Chandran, and V. Shoup. A public key encryption scheme secure against key dependent chosen plaintext and adaptive chosen ciphertext attacks. In A. Joux, editor, *Advances in Cryptology – Eurocrypt 2009*, volume 5479 of *Lecture Notes in Computer Science*, pages 351–368. Springer-Verlag, 2009.
4. D. Dolev, C. Dwork, and M. Naor. Non-malleable cryptography. In *Proc. 23rd Symposium on the Theory of Computing – STOC 1991*, pages 542–552. ACM, 1991.
5. S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Science*, 28:270–299, 1984.
6. U. Maurer. Abstract models of computation in cryptography. In N. P. Smart, editor, *Coding and Cryptography: 10th IMA International Conference*, volume 2796 of *Lecture Notes in Computer Science*, pages 1–12. Springer-Verlag, 2005.
7. M. Naor and M. Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In *Proc. 22nd Symposium on the Theory of Computing, STOC 1990*, pages 427–437. ACM, 1990.
8. P. Paillier and D. Vergnaud. Discrete-log-based signatures may not be equivalent to discrete log. In B. Roy, editor, *Advances in Cryptology – Asiacrypt 2005*, volume 3788 of *Lecture Notes in Computer Science*, pages 1–20. Springer-Verlag, 2005.