A Note on the Security Models for Key-Dependent-Message Encryption

Alexander W. Dent

Qualcomm Research

Abstract. In this paper we examine the relationship between the security models for key-dependent-message encryption proposed by Backes *et al.* [1] and Camenisch *et al.* [3]. We show that when the two notions are equivalent for certain logical classes of function families when the number of keys ℓ in the system is logarithmically small.

1 Introduction

Standard security definitions for public-key encryption, including non-malleability [4] and ciphertext indistinguishability [6], 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 messages are computed as polynomial-time functions of the public-key alone). However, there are practical situations in which an encryption scheme may be used to encrypt data which depends directly upon its private key. For example, a disk encryption scheme may encrypt a hard-drive which contains its decryption key, a security protocol may need to encrypt its own decryption key in order to achieve the desired level of functionality, or the key generation algorithm of a fully homomorphic encryption schemes may require the publication of an encryption of the decryption key to achieve the desired level of security [5].

A breakthrough paper by Boneh *et al.* [2] proved the security of an ElGamal-based scheme in a 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 a CCA security model through the use of a Naor-Yung-style technique [8].

Another approach was taken by Backes *et al.* [1] who provided a more detailed security model and proved the security of the OAEP padding scheme using the random oracle methodology. The security model of Backes *et al.* is significantly more complex than the security model used by Boneh *et al.* and Camenisch *et al.* It uses a "grey-box" computation model that is similar to the black-box computation model proposed for groups by Maurer [7] and the white-box computational model essentially proposed by Paillier and Vergnaud [9]. 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 not under the direct control of the attacker, but to which the attacker can request access to individual data elements. 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. The authors therefore propose a decision system which allows the attacker to determine if a data element is known or not.

The Camenisch *et al.* (CCS) model [3] essentially only models a situation in which the encryption scheme is used in a naïve manner – i.e. when the encryption scheme is used to encrypt a function of the key and that ciphertext is released directly to the attacker. The Backes *et al.* (BDU) model [1] covers more complex situations in which ciphertexts and private keys can be manipulated in (almost) arbitrary ways before being released to an attacker. These situations can occur when an encryption scheme is used in a larger protocol or in a computer system. For example, a disk encryption scheme which is used to encrypt an already encrypted disk image containing the private decryption key of the encryption scheme.

In a traditional (non-key-dependent-message) security model, the two notions coincide; however, more care is required in key-dependent-message security models. Our results will show that the two models are equivalent in certain circumstances:

Theorem (Informal) *The CCS and BDU security models are equivalent in a systems that only contain $\ell \in O(\log k)$ private keys (where k is a security parameter).*

The proof of this theorem follows the obvious strategy of guessing which private keys will be compromised by the attacker, but the formalisation of this intuition is technically difficult.

So while we may conjecture that the two models are equivalent in the majority of practical situations, a formal proof of the equivalence of the two models seems to be difficult and remains an open problem. Indeed, even a formal proof of equivalence between the Backes *et al.* model and an extended version of the Camenisch *et al.* model which allows private keys to be adaptively corrupted seems challenging (for reasons we shall discuss in Section 4). The Camenisch *et al.* model is attractive due to its simplicity and familiarity, but KDM encryption scheme designers should note that, until equivalence of these two models is proven, the Backes *et al.* model should be preferred for generic KDM schemes.

2 Key-Dependent Message Encryption

2.1 Preliminaries

A public-key encryption scheme Π 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 $\mathcal{E}$ is a PPT algorithm that 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 $\mathcal{D}$ is deterministic polynomial-time algorithm that 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 outout of the decryption 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 $(pk, sk) \xleftarrow{\$} \mathcal{G}(1^k)$.

Lastly, we let $[1, \ell]$ denote the integer set $\{1, \ldots, \ell\}$.

2.2 Function Families

We express the functions that can be applied to different elements in our schemes as a set $\mathcal{F}$ of Boolean circuits with arbitrary output size. This has two useful advantages: (1) Boolean circuits are length regular in the sense that the description of a circuit implicitly describes the input and output sizes for that circuit and (2) the description of a Boolean circuit has to be at least as long as its input and output sizes. This prevents an attacker from deriving exponentially long strings by repeated application of (e.g.) a function whose output size is twice its input size.

We allow the function family to contain randomised circuits in the sense that a circuit f which should consume a random string $r \in \{0,1\}^n$ is represented as a family of circuits $f_r \in \mathcal{F}$

with the randomness $r \in \{0,1\}^n$ hard-wired into the circuit. Thus, evaluation of a randomised circuit f is equivalent to evaluation of a randomly chosen circuit f_r. Somewhat surprisingly, under reasonable assumptions about the function family, the hardwired and non-hardwired function families are equivalent[1]. Two function families have been considered in the previous works: Backes *et al.* consider the set of all polynomial-size circuits [1] and Boneh *et al.* consider the set of group element multiplications (which can be expressed as polynomial-size circuits).

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 exist in $\mathcal{F}$.
- The identity functions $f_j(x_1, \ldots, x_\ell) = x_j$ exist in $\mathcal{F}$.
- $\mathcal{F}$ is closed with respect to fixed inputs, i.e. if $f \in \mathcal{F}$ takes n inputs, then $\mathcal{F}$ also contains

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

  for all $1 \leq \alpha \leq n$ and appropriately sized $\beta \in \{0,1\}^*$.
- $\mathcal{F}$ 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 $\mathcal{F}$ also contains

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

  for all $1 \leq \alpha \leq n$ and f_2 with appropriately-sized output.

2.3 The Camenisch *et al.* Security Model

The security model proposed by Boneh *et al.* is simple, elegant and familiar [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 function family $\mathcal{F}$. The security model 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 $\texttt{encrypt}_2$ oracle that takes as input a public key index j and circuits $f_0, f_1 \in \mathcal{F}$ with the same output size. The oracle computes $m \leftarrow f_b(sk_1, \ldots, sk_\ell)$ and returns $\mathcal{E}(pk_j, m)$.
- A $\texttt{decrypt}$ oracle takes as input a ciphertext C and an index j, and returns $\mathcal{D}(sk_j, C)$. The attacker is forbidden from submitting a $\texttt{decrypt}(j, C)$ query if C was returned by any $\texttt{encrypt}_2(j, f_0, f_1)$ query.

The attacker's advantage is defined to be

$$Adv^{\text{ccs}}_{\mathcal{A}}(k) = \Pr[b' = b] - 1/2\,.$$

The running time of $\mathcal{A}$ is assumed to be the time taken for $\mathcal{A}$ to execute plus the time taken to evaluate any circuit f which was queried to the encryption oracle. An encryption scheme is KDM-CCS-CCA secure if every PPT attacker $\mathcal{A}$ has advantage bounded by a negligible function.

In a later section of this paper, we will have to specify which it means to say that a function $f \in \mathcal{F}$ depends upon an input sk_j. We assume that the description of the circuit f includes a description of the inputs on which it depends. Two circuits that compute identical outputs may not be identical if their description states that they depend on different inputs – i.e. the circuit computing $f(sk_1, sk_2) = sk_1$ is different from the circuit computing $f(sk_1) = sk_1$. Two circuits are identical if and only if they have the same description in $\mathcal{F}$.

[1] An attacker with access to the hardwired randomness circuit family can simulate the randomised circuit by choosing the hardwired circuit at random and "forgetting" the randomness in the circuit. An attacker with access to a function family containing both hardwired-randomness and non-hardwired-randomness versions of the same circuits can still simply choose to use hardwired-randomness circuits).

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 logically distinct memory structure to which that attacker does not have direct read or write access. This structure is reminiscent of the abstract computation model given by Maurer [7] and which is implicitly used by Paillier and Vergnaud [9].

The attacker interacts with a remote memory array. 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 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 $f \in \mathcal{F}_c$. 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 $f : (\{0,1\}^*)^n \rightarrow \{0,1\}^*$, then the value $f(M[i_1], \ldots, M[i_n])$ is computed and returned to the attacker.

The Backes *et al.* security model for key-dependent message encryption (KDM-BDU-CCA)[2] has a similar structure. The security game between a PPT attacker $\mathcal{A}$ and a challenger runs as follows: (1) the challenger generates a bit $b \xleftarrow{\$} \{0,1\}$ 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 locations $M[1]$ to $M[\ell]$. The attacker can interact with the memory array via the following oracles:

- 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 an indices i_0, i_1, and sets the next available memory location to be equal to $M[i_b]$. In order to use this oracle, we require that $|M[i_0]| = |M[i_1]|$.
- An `eval` oracle, which takes as input a circuit $f \in \mathcal{F}$ 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 assume that the run-time of $\mathcal{A}$ includes the time taken for the challenger to evaluate the circuits f used in `eval` queries. 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 $\mathcal{E}$, $\mathcal{D}$ and $f \in \mathcal{F}$. This means that it is always possible to detect whether the attacker makes a `challenge` query on memory locations with $|M[i_0]| \neq |M[i_1]|$. We will assume that this event never occurs (since an execution can be aborted if it is detected).

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 the next available memory location o" to "the attacker makes an $o \leftarrow \texttt{oracle}(i_1, \ldots, i_n)$ 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):

Rule 1: For any $\texttt{reveal}(i)$ query, we have $i \in knowset$.

Rule 2: For any $o \leftarrow \texttt{encrypt}(i, j)$ query with $\{j, o\} \subseteq knowset$, we have $i \in knowset$.

Rule 3: For any $o \leftarrow \texttt{decrypt}(i, j)$ query with $o \in knowset$, we have $i \in knowset$.

Rule 4: For any $o_1 \leftarrow \texttt{encrypt}(i_1, j)$ and $o_2 \leftarrow \texttt{decrypt}(i_2, j)$ queries with $M[o_1] = M[i_2]$ and $o_2 \in knowset$, we have $i_1 \in knowset$.

[2] In their original paper, Backes *et al.* termed this the adKDM model. We use the term BDU model in order to allow the reader to more quickly differentiate it from the CCS model.

Rule 5: For any $o \leftarrow \texttt{challenge}(i_0, i_1)$ query with $o \in \mathit{knowset}$, we have $i_0, i_1 \in \mathit{knowset}$.[3]

Rule 6: For any $o \leftarrow \texttt{eval}(f, i_1, \ldots, i_n)$ query with $o \in \mathit{knowset}$, we have $\{i_1, \ldots, i_n\} \subseteq \mathit{knowset}$.

For an attacker $\mathcal{A}$, we define $\mathsf{Invalid}$ to be the event that $\mathcal{A}$ made an $o \leftarrow \texttt{challenge}(i_0, i_1)$ query with $o \in \mathit{knowset}$. The attacker's advantage is defined to be

$$Adv^{\text{BDU}}_{\mathcal{A}}(k) = \Pr[b' = b \wedge \neg \mathsf{Invalid}] - 1/2$$

An encryption scheme is KDM-BDU-CCA secure if every PPT attacker $\mathcal{A}$ has advantage bounded by a negligible function.

3 The Relationship Between CCS Security and BDU Security

3.1 Intuition and Proof Sketch

It is clear that a scheme which is KDM-BDU-CCA secure is KDM-CCS-CCA secure since all operations that can be computed in the CCS model can also be computed in the BDU model. We show that a scheme which is KDM-CCS-CCA secure is KDM-BDU-CCA secure if the challenger can determine the private keys which are compromised during the execution. In particular, if ℓ is small (i.e. $\ell \in O(\log k)$) then challenger can guess the set of compromised keys with non-negligible probability and the two notions are equivalent.

Our proof will involve a CCS attacker $\mathcal{B}$ simulating the environment for a BDU attacker $\mathcal{A}$. The simulation works roughly as follows:

- The simulator will guess the set of private keys that will be corrupted during the simulation – i.e. the simulator will guess the set $J = [1, \ell] \cap \mathit{knowset}$. The simulator will guess the set correctly with probability $1/2^{\ell}$ which is non-negligible since $\ell \in O(\log k)$. The simulator will generate its own public/private keys for all $j \in J$ and will use the public keys provided by its challenger for all $j \in [1, \ell] \setminus J$.
- The simulator will simulate a memory array with which $\mathcal{A}$ will interact. It will simulate the contents of the memory array in one of two ways – if the value of the memory element can be explicitly computed using information known to the simulator then it will be stored as explicit value, otherwise it will be stored as a function $f \in \mathcal{F}$ with the unknown private keys as input variables. If the simulator guesses J correctly, then $\mathcal{A}$ will never `reveal` a memory location containing a function.
- The simulator will actually compute two elements for each memory location i —$M[i, 0]$ and $M[i, 1]$— where $M[i, b]$ contains the representation of a value that might be held in memory element $M[i]$ if the hidden challenge bit is b. We require that if b is the challenge bit, then $M[i, b]$ contains correct representations. The simulation may hold incorrect values in $M[i, 1 - b]$. If $\mathcal{A}$'s execution is such that $\mathsf{Invalid}$ does not occur, then $\mathcal{A}$ will never `reveal` a memory location containing two different values, i.e. where $M[i, 0] \neq M[i, 1]$.

Since the attacker will never reveal memory locations containing functions or ambiguous values, the attacker will only see correct, consistent values that would have been stored in a real execution of the BDU security model. Hence, the attacker behaves as it would in a real execution of the security model. A major technical component of the proof is proving that this simulation is correct.

However, even after we prove that the simulation is correct, we are still faced with a problem. Since the simulator stores some memory elements as functions, it may not be able to compute *knowset* accurately, as it may not be able to determine if a memory element $M[i]$ contains a value equal to the output of an encryption oracle query if the value stored at $M[i]$ is represented by a function. Hence, the simulator will not be able to apply Rule 4 of Section 2.4 consistently. The

[3] Strictly speaking, this rule is not required to define the security model, but it is useful for the security proofs later in this paper.

result is that our simulator will *never* abort if it guesses J correctly, but will only *sometimes* abort if it guesses J incorrectly. If the probability that the simulator aborts is somehow dependent on the challenge bit b, then the BDU attacker $\mathcal{B}$ may have negligible advantage even though the CCS attacker in the simulation has non-negligible advantage – see Appendix A for an example.

We solve this problem using the artificial abort technique of Waters [10]. The attacker computes the number n of different sets $J \subseteq [1, \ell]$ which would not cause the simulator to abort and artificially aborts the simulator with probability $(n-1)/n$. This means that the attacker always aborts with probability $1 - 1/2^\ell$ regardless of the value of b and so our simulation cannot be biased. Unlike the Waters proof, since $\ell \in O(\log k)$, we do not have to estimate the number of sets J for which the simulator would abort as we can calculate it directly in polynomial time.

The full theorem statement is:

Theorem 1. *Suppose $\mathcal{F}$ is a BDU compliant function family containing $\mathcal{E}$ and $\mathcal{D}$. If Π is KDM-CCS-CCA secure and $\ell \in O(\log k)$, then Π is KDM-BDU-CCA secure.*

3.2 The Simulator

Let $\Pi = (\mathcal{G}, \mathcal{E}, \mathcal{D})$ and suppose $\mathcal{A}$ is a PPT BDU attacker with advantage ε. We develop a CCS attacker $\mathcal{B}$ with advantage at least $\varepsilon/2^\ell$. The main component of this attacker is the simulated memory array. Each memory array element $M[i]$ is split into two parts, $M[i, 0]$ and $M[i, 1]$, where $M[i, b]$ contains representations of the value that might be computed in the BDU memory array in the case that the challenge bit is b. Technically, if the challenge bit used by the CCS $\texttt{encrypt}_2$ oracle is b, then we require that $M[i, b]$ holds accurate representations of the BDU memory array $M[i]$ that would be computed if the BDU challenge bit was also b.

Each memory element $M[i, b]$ which will hold either an exact value or a function $f \in \mathcal{F}$ where $f(sk_1, \ldots, sk_\ell)$ is the value that should be held in that memory element.

The CCS attacker $\mathcal{B}(pk_1, \ldots, pk_\ell)$ is given in Figure 1 and the oracle simulations are given in Figure 2.

1. $\mathcal{B}$ picks a random subset $J \subseteq [1, \ell]$ and sets $\bar{J} = [1, \ell] \setminus J$. The set J is $\mathcal{B}$'s guess as to which private key values $\mathcal{A}$ will explicitly or implicity revealed – i.e. $J = [1, \ell] \cap \mathit{knowset}$.
2. For each $j \in J$, $\mathcal{B}$ computes $(pk_j, sk_j) \xleftarrow{\$} \mathcal{G}(1^k)$ and sets $M[j, 0] = M[j, 1] = sk_j$. These memory locations are labelled "value".
3. For each $j \in \bar{J}$, $\mathcal{B}$ sets $M[j, 0] = M[j, 1] = f_j$ where $f_j \in \mathcal{F}$ 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. These memory locations are labelled "function".
4. $\mathcal{B}$ runs $b' \xleftarrow{\$} \mathcal{A}(pk_1, \ldots, pk_\ell)$. $\mathcal{B}$ answers oracle queries as in Figure 2.
5. $\mathcal{B}$ computes the number n of sets $J \subseteq [1, \ell]$ which would not have caused the above simulation to abort using the algorithm in Figure 3.
6. $\mathcal{B}$ outputs b' with probability $1/n$ and a random value $b' \xleftarrow{\$} \{0, 1\}$ otherwise. We let $\mathsf{Artifical}$ denote the event that $\mathcal{B}$ outputs the random bit.

Fig. 1. The CCS Attacker $\mathcal{B}$ derived from a BDU Attacker $\mathcal{A}$

Claim 1.1 *$\mathcal{B}$ is a valid CCS attacker – i.e. $\mathcal{B}$ does not make an invalid $\texttt{decrypt}$ query.*

The only way that $\mathcal{B}$ could make a $\texttt{decrypt}$ query is if $\mathcal{A}$ makes a $o \leftarrow \texttt{decrypt}(i, j)$ query where $j \in \bar{J}$ and $M[i, b]$ was equal to a value previously returned by $\mathcal{B}$'s $\texttt{encrypt}_2$ oracle using pk_j. However, if that is the case, then $\mathcal{B}$ will detect this violation and set $M[i, b]$ to be equal to the value that is input to the $\texttt{encrypt}_2$ oracle. Hence, $\mathcal{B}$ does not make an illegal query. $\diamond$

Claim 1.2 *Up to the point that it might halt and output $\perp$, $\mathcal{B}$ correctly simulates all of $\mathcal{A}$'s oracles.*

- If $\mathcal{A}$ makes a $\texttt{reveal}(i)$ query and $M[i]$ contains a single value, i.e. $M[i,0] = M[i,1]$ and $M[i,0]$ is a value, then $\mathcal{B}$ returns $M[i,0]$. Otherwise, $\mathcal{B}$ outputs a random value $b' \xleftarrow{\$} \{0,1\}$ and halts. We let Abort denote the event that $\mathcal{B}$ halts in this manner.
- If $\mathcal{A}$ makes an $o \leftarrow \texttt{eval}(f, i_1, \ldots, i_n)$ query then for each $b \in \{0,1\}$:
  - If $M[i_1,b], \ldots, M[i_n,b]$ are memory elements which are values, then $\mathcal{B}$ computes the (deterministic) function $f(M[i_1,b], \ldots, M[i_n,b])$ and stores this result in $M[o,b]$. The memory location $M[o,b]$ is declared to be a value.
  - If not all $M[i_1,b], \ldots, M[i_n,b]$ are values, then $\mathcal{B}$ builds a function f^* in the variables $(sk_1, \ldots, sk_\ell)$ to represent the output. $\mathcal{B}$ initially set $f^*(\cdot) = f(\cdot)$, then for each input index i_α:
    - If $M[i_\alpha,b]$ is a value, then $\mathcal{B}$ sets f^* to be the function obtained by fixing the α-th input of f^* to be $M[i_\alpha,b]$. 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 $M[i_\alpha,b]$ holds the function $f_{i_\alpha}(sk_1, \ldots, sk_\ell)$. $\mathcal{B}$ sets f^* to 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 memory element $M[o,b]$ stores the function f^* and declares the element to be function.
- If $\mathcal{A}$ makes an $o \leftarrow \texttt{challenge}(i_0, i_1)$ query then $\mathcal{B}$ sets $M[o,b] = M[i_b,b]$ for $b \in \{0,1\}$. $M[o,b]$ is declared to be of the same type as $M[i_b,b]$ (either a value or a function). 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 $o \leftarrow \texttt{encrypt}(i,j)$ query for $j \in J$ then:
  - If $M[i,0] = M[i,1]$ (either as functions or values):
    - If $M[i,0]$ is a value, then $\mathcal{B}$ computes $C = \mathcal{E}(pk_j, M[i,0])$, sets $M[o,0] = M[o,1] = C$, and declares this memory location to be a value.
    - If $M[i,0]$ is a function f then $\mathcal{B}$ generates randomness for the encryption algorithm, stores the (deterministic) function $\mathcal{E}(pk_j, f(\cdot))$ at $M[o,0]$ and $M[o,1]$, and declares this memory location to be a function. This new function is in $\mathcal{F}$ by the "composition closure" property of $\mathcal{F}$ since $\mathcal{E} \in \mathcal{F}$.
  - If $M[i,0] \neq M[i,1]$ then for each $b \in \{0,1\}$:
    - If $M[i,b]$ is a value, then $\mathcal{B}$ sets $M[o,b] = \mathcal{E}(pk_j, M[i,b])$ and declares this memory locations to be a value.
    - If $M[i,b]$ is a function f then $\mathcal{B}$ generates a random value for the encryption algorithm, stores the (deterministic) function $\mathcal{E}(pk_j, f(\cdot))$ at $M[o,b]$, and declares this memory location to be a function. This new function is in $\mathcal{F}$ by the "composition closure" property of $\mathcal{F}$ since $\mathcal{E} \in \mathcal{F}$.
- If $\mathcal{A}$ makes an $o \leftarrow \texttt{encrypt}(i,j)$ query for $j \in \bar{J}$ then $\mathcal{B}$ makes an $\texttt{encrypt}_2$ query using the functions $M[i,0]$ and $M[i,1]$. (If either $M[i,0]$ or $M[i,1]$ is a value then the corresponding constant function is passed instead.) $\mathcal{B}$ sets $M[o,0]$ and $M[o,1]$ to be equal to the received ciphertext C and declares these memory locations to be values.
- If $\mathcal{A}$ makes an $o \leftarrow \texttt{decrypt}(i,j)$ query for $j \in J$, then for each $b \in \{0,1\}$:
  - If $M[i,b]$ is a value, then $\mathcal{B}$ computes $M[o,b] \leftarrow \mathcal{D}(sk_j, M[i,b])$ and declares this memory location to be a value.
  - If $M[i,b]$ is a function, then $\mathcal{B}$ associates the function $\mathcal{D}(sk_j, f(\cdot))$ with $M[o,b]$ and declares this memory location to be a function. This function is in $\mathcal{F}$ due to the "composition closure" property of $\mathcal{F}$ since $\mathcal{D} \in \mathcal{F}$.
- If $\mathcal{A}$ makes an $o \leftarrow \texttt{decrypt}(i,j)$ query where $j \in \bar{J}$, then for each $b \in \{0,1\}$:
  - If $M[i,b]$ is a value that has not been returned by an $\texttt{encrypt}_2$ query on public key index j, then $\mathcal{B}$ queries the $\texttt{decrypt}$ oracle on $M[i,b]$ with public key index j, stores the result in $M[o,b]$, and declares this memory location to be a value.
  - If $M[i,b]$ is a value that has been returned by an $\texttt{encrypt}_2$ query on public key index j, then there must have been a query $o' \leftarrow \texttt{encrypt}(i',j)$ query which caused the value $M[i,b]$ to be placed into $M[o',b]$. $\mathcal{B}$ sets $M[o,b]$ to be equal to $M[i',b]$ and declares this memory location to be of the same type as $M[i',b]$ (either value or function).
  - If $M[i,b]$ is a function, then $\mathcal{B}$ associates the function $\mathcal{D}(f_j(\cdot), f(\cdot))$ with $M[o,b]$ where $f_j \in \mathcal{F}$ is the function $f_j(sk_1, \ldots, sk_\ell) = sk_j$ and f is the function held at $M[i,b]$. $\mathcal{B}$ declares this memory location to be a function. This function is in $\mathcal{F}$ due to the "composition closure" property of $\mathcal{F}$ and as $\mathcal{D} \in \mathcal{F}$.

Fig. 2. The Oracle Simulation Algorithms for the CCS Attacker $\mathcal{B}$

In most cases this is easily seen by inspection of the oracles, but we highlight a few of the more complex cases:

- A $o \leftarrow \texttt{challenge}(i_0, i_1)$ oracle query should place either $M[i_0]$ or $M[i_1]$ into $M[o]$. If the challenge bit $b = 0$, then $M[o]$ should be the value of $M[i_0]$ that would be computed if $b = 0$, i.e. $M[o, 0] = M[i_0, 0]$. Similarly, $M[o, 1] = M[i_1, 1]$.
- A $o \leftarrow \texttt{decrypt}(i, j)$ oracle query should decrypt $M[i]$ and place it in $M[o]$ using sk_j. This is clear, except perhaps in the case that $j \in \bar{J}$ and $M[i, b]$ contains a ciphertext that has been output by $\mathcal{B}$'s $\texttt{encrypt}_2$ oracle. $\mathcal{B}$ only makes an $\texttt{encrypt}_2$ query as the result of an $o' \leftarrow \texttt{encrypt}(i', j)$ query. The decryption of the ciphertext at $M[o, b]$ is equal to the input to the message at $M[i', b]$. Hence, $M[o, b] = M[i', b]$. $\diamond$

The attacker attempts to guess $[1, \ell] \cap \mathit{knowset}$. However, *knowset* may take different values depending on the hidden bit b. We define $\mathit{knowset}_b$ to be the set that would have been computed if $\mathcal{A}$ had been allowed to execute completely in an instance of the BDU game with the public/private keys as in the simulation, the same randomness as in the simulation, and the challenge bit equal to b.

Claim 1.3 *If $\mathcal{B}$ guess the set $J = [1, \ell] \cap \mathit{knowset}_b$ correctly, then $\mathcal{A}$ will never make a $\texttt{reveal}(i)$ query on a memory location that contains a function.*

Suppose $J = [1, \ell] \cap \mathit{knowset}_b$. We prove the claim by induction on the statement "if $i \in \mathit{knowset}_b$, then $M[i, 0]$ and $M[i, 1]$ contain values". This is trivially true for the first ℓ memory locations. Suppose this is true for the first $t - 1$ memory locations and consider the t-th memory location.

The inductive hypothesis is trivial true if $M[t, 0]$ and $M[t, 1]$ contain values or if $t \notin \mathit{knowset}_b$. We therefore concentrate on the case that either $M[t, 0]$ or $M[t, 1]$ contains a function and $t \in \mathit{knowset}_b$. Recall that if $M[o]$ is constructed through a $o \leftarrow \texttt{oracle}(i_1, \ldots, i_n)$ command, then we must have that $i_1, \ldots, i_n < o$. If $M[t, \delta]$ contains a function, then the memory location must have been defined by one of the following commands:

- $t \leftarrow \texttt{eval}(f, i_1, \ldots, i_n)$ with some i_α for which $M[i_\alpha, \delta]$ contains a function. If $t \in \mathit{knowset}_b$, then $i_\alpha \in \mathit{knowset}_b$, which contradicts the inductive hypothesis.
- $t \leftarrow \texttt{challenge}(i_0, i_1)$ with $M[i_\delta, \delta]$ containing a function. If $t \in \mathit{knowset}_b$, then $i_\delta \in \mathit{knowset}_b$, which contradicts the inductive hypothesis.
- $t \leftarrow \texttt{encrypt}(i, j)$ with $j \in J$ and $M[i, \delta]$ contains a function. If $t \in \mathit{knowset}_b$, then $i \in \mathit{knowset}_b$ as $j \in \mathit{knowset}_b$, which contradicts the inductive hypothesis.
- $t \leftarrow \texttt{decrypt}(i, j)$ and $M[t, \delta]$ is computed by composing the function $\mathcal{D}$ with the function at $M[i, \delta]$. If $t \in \mathit{knowset}_b$, then $i \in \mathit{knowset}_b$, which contradicts the inductive hypothesis.
- $t \leftarrow \texttt{decrypt}(i, j)$ with $j \in \bar{J}$ and $M[t, \delta]$ is computed as being equal to a memory location $M[i', \delta]$ that contains a function. This can occur if $M[i, \delta]$ contains a ciphertext that was output by an $o' \leftarrow \texttt{encrypt}(i', j)$ query. If $t \in \mathit{knowset}_b$, then $i' \in \mathit{knowset}_b$, which contradicts the induction hypothesis.

Hence, if $J = [1, \ell] \cap \mathit{knowset}_b$ and $i \in \mathit{knowset}_b$, then we must have that $M[i, 0]$ and $M[i, 1]$ contain values. $\diamond$

Claim 1.4 *Suppose $J = [1, \ell] \cap \mathit{knowset}_b$. If the event* Invalid *does not occur, then for all $i \in \mathit{knowset}_b$ we must have that $M[i, 0] = M[i, 1]$. Most notably, if $\mathcal{A}$ makes a $\texttt{reveal}(i)$ query, then we must have that $M[i, 0] = M[i, 1]$.*

We prove this theorem by induction on the statement "if $i \in \mathit{knowset}_b$ has $M[i, 0] \neq M[i, 1]$ then Invalid occurs". This statement is clearly true when $1 \leq i \leq \ell$ since $M[i, 0] = M[i, 1]$ in this range. Suppose this statement is true for the first $t - 1$ memory locations and consider the t-th memory location. Note that since we assume that $J = [1, \ell] \cap \mathit{knowset}_b$, we have that if $t \in \mathit{knowset}_b$, then both $M[t, 0]$ and $M[t, 1]$ contain values.

If $M[t, 0] = M[t, 1]$ or $t \notin \mathit{knowset}_b$, then the statement trivially holds, so we concentrate on the case where $M[t, 0] \neq M[t, 1]$ and $t \in \mathit{knowset}_b$ query. The contents of the t-th memory location must have been defined through one of the following commands:

- $t \leftarrow \texttt{eval}(f, i_1, \ldots, i_n)$ with some i_α for which $M[i_\alpha, 0] \neq M[i_\alpha, 1]$. If $t \in \mathit{knowset}_b$, then $i_\alpha \in \mathit{knowset}_b$ and so Invalid occurs by the inductive hypothesis (since $i_\alpha < t$).
- $t \leftarrow \texttt{challenge}(i_0, i_1)$. If $t \in \mathit{knowset}_b$, then Invalid occurs by definition.
- $t \leftarrow \texttt{encrypt}(i, j)$ with $j \in \mathit{knowset}_b$ and $M[i, 0] \neq M[i, 1]$. If $t \in \mathit{knowset}_b$, then $i \in \mathit{knowset}_b$, since $j \in \mathit{knowset}_b$. Hence, Invalid occurs by the inductive hypothesis.
- $t \leftarrow \texttt{decrypt}(i, j)$ with $M[i, 0] \neq M[i, 1]$. If $t \in \mathit{knowset}_b$, then $i \in \mathit{knowset}_b$ and so Invalid occurs by the inductive hypothesis.
- $t \leftarrow \texttt{decrypt}(i, j)$ with $M[i, 0] = M[i, 1]$. Since $M[t, 0] \neq M[t, 1]$, we must have that $M[i, 0]$ and $M[i, 1]$ must contain a value equal to the ciphertext at some location $M[o', 0] = M[o', 1]$. This must have been computed using an $o' \leftarrow \texttt{encrypt}(i', j)$ query with $M[i', 0] \neq M[i', 1]$. If $t \in \mathit{knowset}_b$, then $i' \in \mathit{knowset}_b$, and so Invalid occurs by the inductive hypothesis. $\diamond$

This is sufficient to show that the simulator is correct unless it aborts. In other words, if Abort does not occur, then the simulator correctly simulates the environment for $\mathcal{A}$.

3.3 Aborts and Artificial Aborts

Now we examine the probability that $\mathcal{B}$ aborts or artificially aborts. Figure 3 describes an algorithm which determines how many sets $J \subseteq [1, \ell]$ would cause the simulator to abort. Note that since $\ell \in O(\log k)$ we can in consider each possible $J \subseteq [1, \ell]$ individually in polynomial-time. This avoids the need for the conceptually complex sampling step in Waters proof [10].

1. Set $n = 0$.
2. For each $J \subseteq [1, \ell]$
   (a) Run $b' \leftarrow \mathcal{A}(pk_1, \ldots, pk_\ell)$. If $\mathcal{A}$ makes an oracle query, then the result placed into the memory array is exactly the result that would have been placed into that location in the original simulation (either value or function). However, the memory element is labelled with value or function depending on the set J currently under consideration. The simulator still aborts if a $\texttt{reveal}(i)$ query is made on a memory location with $M[i, 0] \neq M[i, 1]$ or on a memory location that contains a function.
   (b) If the value of J should cause the simulation to abort, then halt $\mathcal{A}$'s execution and re-run with the next value of J.
   (c) If $\mathcal{A}$ terminates without aborting, then increment n by one.
3. Return n.

Fig. 3. The Algorithm to Determine the Subsets $J \subseteq [1, \ell]$ Which Cause $\mathcal{B}$ to Abort

A key component of understanding this algorithm is to realise that the values placed into the memory array are (representations of) correctly distributed values that would be placed into the memory array in the BDU game. It does not matter whether these values are computed by the challenger through oracle queries or by $\mathcal{B}$ through the simulation. Hence, up until the point that Abort occurs, the values in the memory array are correctly distributed. Therefore, we can assume that the same (representations of) values are placed in the memory array regardless of the set J up until the simulator aborts and so we are only required to track whether the simulator would abort given the values computed in the initial simulation.

Claim 1.5 $\Pr[\mathit{Abort} \vee \mathit{Artificial}] = 1 - 1/2^\ell$.

First we show that there exists at least one set J for which the simulator does not abort — i.e. Abort does not happen. Up until the point where Abort occurs, the view that the simulator $\mathcal{B}$ provides to $\mathcal{A}$ is completely consistent with the values that it would receive in the valid BDU security model. Hence, by fixing the public/private key pairs, the randomness of the BDU attacker

$\mathcal{A}$, the randomness used by the encryption algorithms (regardless of whether it is computed by the oracle or the simulator), and the challenge bit b, the value of $knowset_b$ that would be computed in a complete execution of $\mathcal{A}$ is completely determined. Hence, if $J = knowset_b \cap [1, \ell]$, then the simulator will provided consistent values and will not abort.

This means that the artificial abort algorithm in Figure 3 will output a value $n \geq 1$. Therefore,

$$\begin{aligned}
\Pr[\mathsf{Abort} \vee \mathsf{Artificial}] &= \Pr[\mathsf{Abort}] + \Pr[\mathsf{Artificial} \wedge \neg\mathsf{Abort}] \\
&= \Pr[\mathsf{Abort}] + \Pr[\mathsf{Artificial}|\neg\mathsf{Abort}] \cdot \Pr[\neg\mathsf{Abort}] \\
&= (1 - \frac{n}{2^\ell}) + \frac{n-1}{n} \cdot \frac{n}{2^\ell} \\
&= 1 - 1/2^\ell
\end{aligned}$$

Note that this is independent of the value of the challenge bit b. $\diamond$

This allows us to compute the advantage of $\mathcal{B}$ in the CCS model:

$$\begin{aligned}
Adv^{\text{CCS}}_{\mathcal{B}}(k) &= \Pr[\mathcal{B} \text{ outputs } b'] - \frac{1}{2} \\
&= \Pr[\mathcal{B} \text{ outputs } b' \,|\, (\mathsf{Abort} \vee \mathsf{Artificial})] \cdot \Pr[(\mathsf{Abort} \vee \mathsf{Artificial})] + \\
&\qquad \Pr[\mathcal{B} \text{ outputs } b' \,|\, \neg(\mathsf{Abort} \vee \mathsf{Artificial})] \cdot \Pr[\neg(\mathsf{Abort} \vee \mathsf{Artificial})] - \frac{1}{2} \\
&= \Pr[\mathcal{B} \text{ outputs } b' \,|\, (\mathsf{Abort} \vee \mathsf{Artificial})] \cdot (1 - \frac{1}{2^\ell}) + \\
&\qquad \Pr[\mathcal{B} \text{ outputs } b' \,|\, \neg(\mathsf{Abort} \vee \mathsf{Artificial})] \cdot \frac{1}{2^\ell} - \frac{1}{2} \\
&= \Big(1 - \frac{1}{2^\ell}\Big)\Big\{ \Pr[\mathcal{B} \text{ outputs } b' \,|\, (\mathsf{Abort} \vee \mathsf{Artificial})] - \frac{1}{2} \Big\} + \\
&\qquad \frac{1}{2^\ell}\Big\{ \Pr[\mathcal{B} \text{ outputs } b' \,|\, \neg(\mathsf{Abort} \vee \mathsf{Artificial})] - \frac{1}{2} \Big\} \\
&= \Big(1 - \frac{1}{2^\ell}\Big)\Big\{\frac{1}{2} - \frac{1}{2}\Big\} + \frac{1}{2^\ell}\Big\{ \Pr[\mathcal{B} \text{ outputs } b' \,|\, \neg(\mathsf{Abort} \vee \mathsf{Artificial})] - \frac{1}{2} \Big\} \\
&= \frac{1}{2^\ell}\Big\{ \Pr[\mathcal{A} \text{ outputs } b'] - \frac{1}{2} \Big\} \\
&= \frac{1}{2^\ell}\Big\{ \Pr[\mathcal{A} \text{ outputs } b' \wedge \mathsf{Invalid}] + \Pr[\mathcal{A} \text{ outputs } b' \wedge \neg\mathsf{Invalid}] - \frac{1}{2} \Big\} \\
&\geq \frac{1}{2^\ell} Adv^{\text{BDU}}_{\mathcal{A}}(k)
\end{aligned}$$

Hence, if there exists a BDU attacker $\mathcal{A}$ with non-negligible advantage, then there exists a CCS attacker $\mathcal{A}$ with non-negligible advantage. This is sufficient to prove the theorem. $\square$

4 Conclusions and Open Problems

Backes *et al.* definition of security for key-dependent message encryption [1] appears to cover more situations than the Camenisch *et al.* definition of security [3]. However, we have shown in this paper that this is not true in the case that the number of public keys ℓ is small.

Since the most obvious difference between the BDU and CCS models is that it is possible to corrupt private keys in the BDU model whereas it is not possible to corrupt private keys in the CCS model, it is very tempting to conjecture that the BDU model is equivalent to a version of the CCS model with private key corruption. Of course, care has to be taken in order to define this notion sensibly without simply re-defining the BDU notion of security. We give the following strawman definition:

Definition 2. *The security definition for CCS security with dynamic corruptions is similar to the existing KDM-CCS-CCA security notion but the attacker is furnished with two extra oracles:*

– *An* $\texttt{encrypt}_1$ *oracle that takes a public key index* j *and a function specifier* $f \in \mathcal{F}$ *as input, and returns* $\mathcal{E}(pk_j, f(sk_1, \ldots, sk_\ell))$.
– *A* `corrupt` *oracle that takes as input a public key index* j *and returns* sk_j.

A public key pk_j *is declared as a* challenge *public key if the attacker makes an* $\texttt{encrypt}_2(j, f_0, f_1)$ *query. We define a set corrupt recursively by two rules:*

– *If* $\mathcal{A}$ *makes a* $\texttt{corrupt}(j)$ *query, then* $j \in corrupt$.
– *If* $\mathcal{A}$ *makes a* $\texttt{encrypt}_1(j, f)$ *query for* $j \in corrupt$ *and the value of* f *depends on* $sk_{j'}$*, in the sense that the circuit* $f \in \mathcal{F}$ *takes bits of* $sk_{j'}$ *as input, then* $j' \in corrupt$.

An attacker is legitimate if it is PPT, generates a corrupt set which does not contain a challenge public key, and does not submit a ciphertext output by either `encrypt` *oracle to the* `decrypt` *oracle. We define an encryption scheme to be KDM-dCCS-CCA secure if every legitimate attacker has negligible advantage. We stress that the attacker can make a* $\texttt{corrupt}(j)$ *query even if it made an* $\texttt{encrypt}_1(j, f)$ *query since this does not reveal any information about* b*, but that an attacker cannot submit a ciphertext output by* $\texttt{encrypt}_1(j, f)$ *to the* `decrypt` *oracle.*

Unfortunately, more subtleties arise when attempting to prove the equivalence between the BDU model and the dCCS model. A critical fact that we made use of in the proof in this paper is that our simulation was able to handle `encrypt` and `decrypt` queries different depending on whether the public/private key index j used by the oracle will end up in *knowset* or not. We would not be able to use this technique in the dCCS model as we will not know in advance whether $j \in knowset$. This become particular troublesome when we consider double encryption – i.e. when we consider an attacker that computes $\mathcal{E}(pk_2, \mathcal{E}(pk_1, f(sk_1, sk_2)))$. A discussion of the difficulty of simulating a memory array against this attacker is given in Appendix B. Preliminary results suggest that these two models may be equivalent for attackers which do not perform double encryptions.

We are forced to conclude that, while the Camenisch *et al.* model provides sufficient security in certain specific circumstances such as in the construction of fully homomorphic encryption schemes, researchers who are attempting to prove the security of an encryption scheme for use in arbitrary circumstances should prove their schemes are secure in the Backes *et al.* model.

Acknowledgements This research was mostly conducted while the author was employed in the Information Security Group at Royal Holloway, University of London, and also partly while visiting the City University of New York. The author would like to gratefully acknowledge the extensive comments of the reviewers of PKC 2012 which were essential in correcting errors in earlier versions of the proof.

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. C. Gentry. *A Fully Homomorphic Encryption Scheme.* PhD thesis, Stanford University, 2009.
6. S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Science*, 28:270–299, 1984.

7. 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.
8. M. Naor and M. Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In *Proc. 22nd ACM Symposium on the Theory of Computing – STOC '90*, pages 427–437. ACM Press, 1990.
9. 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.
10. B. Waters. Efficient identity-based encryption without random oracles. In R. Cramer, editor, *Advances in Cryptology - EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 114–127. Springer-Verlag, 2005.

A Requirement for Artificial Abort

As an example of the need for the artificial abort step, we show in this appendix how the construction of Figure 1 may turn an attacker with non-negligible advantage into an attacker with negligible advantage unless we apply artificial aborts. Consider a BDU attacker $\mathcal{A}$ which takes ℓ public keys as input where $\ell = \log k$. Let f_0 and f_1 be the constant functions that output 0 and 1 respectively. For convenience, we will assume that the encryption scheme is such that the attacker $\mathcal{A}$ can determine the challenge bit b with certainty by observing an encryption of b. $\mathcal{A}$ runs as follows:

1. $M[1] = sk_1$; $M[2] = sk_2$; $M[3] = sk_3$; …; $M[\ell] = sk_\ell$ (by definition).
2. $\mathcal{A}$ makes $M[\ell+1] \leftarrow \texttt{eval}(f_0)$ and $M[\ell+2] \leftarrow \texttt{eval}(f_1)$ queries. In other words, $\mathcal{A}$ uses constant functions to set $M[\ell+1] = 0$ and $M[\ell+2] = 1$.
3. $\mathcal{A}$ makes a $M[\ell+3] \leftarrow \texttt{challenge}(M[\ell+1], M[\ell+2])$ query to place either $M[\ell+1]$ or $M[\ell+2]$ into $M[\ell+3]$ depending on the challenge bit b. This has the effect of setting $M[\ell+3] = b$.
4. $\mathcal{A}$ makes a $M[\ell+4] \leftarrow \texttt{encrypt}(M[\ell+3], pk_\ell)$ to encrypt $M[\ell+3]$ using the public key pk_ℓ. The resulting ciphertext is stored in $M[\ell+4]$.
5. $\mathcal{A}$ makes a $\texttt{reveal}(M[\ell+4])$ query to reveal the encrypted ciphertext stored in $M[\ell+4]$. By assumption, $\mathcal{A}$ can use this ciphertext to recover the challenge bit b.
6. If $b = 1$, then $\mathcal{A}$ makes $\texttt{reveal}(M[i])$ queries for $1 \le i \le \ell - 1$ to reveal the private keys $sk_1, \ldots, sk_{\ell-1}$, then outputs the bit $b' = 1$ and halts.
7. If $b = 0$, then $\mathcal{A}$ outputs $b' = 0$ with probability $\alpha = 1/2 - 1/2^\ell$, otherwise $\mathcal{A}$ outputs $b' = 1$.

Note that $\mathcal{A}$ never causes the event Invalid to occur; hence, $\mathcal{A}$'s advantage is

$$\begin{aligned} Adv^{\text{BDU}}_{\mathcal{A}}(k) &= \Pr[b = b'] - 1/2 \\ &= \frac{1}{2}\Big\{ \Pr[b' = 1 | b = 1] + \Pr[b' = 0 | b = 0] - 1 \Big\} \\ &= \frac{1}{2}\Big\{ 1 + \alpha - 1 \Big\} \\ &= \frac{1}{2} \cdot \alpha \\ &= \frac{1}{4} - \frac{1}{2^{\ell+1}} \end{aligned}$$

which is non-negligible.

However, if we consider the CCS attacker $\mathcal{B}$ that is constructed from $\mathcal{A}$ using the algorithm in Figure 1 without artificial aborts, then the advantage of this algorithm turns out to be negligible. This is because $\mathcal{B}$ will Abort with higher probability if $b = 0$ than it will if $b = 1$ because of the extra `reveal` queries. To be precise

$$\Pr[\mathsf{Abort} | b = 1] = \frac{1}{2^\ell} \qquad \text{and} \qquad \Pr[\mathsf{Abort} | b = 0] = \frac{1}{2}\,.$$

Therefore, $\mathcal{B}$'s advantage is

$$\begin{aligned}
Adv^{\text{ccs}}_{\mathcal{B}}(k) &= \Pr[b = b'] - 1/2 \\
&= \frac{1}{2}\Big\{ \Pr[b' = 1 | b = 1] + \Pr[b' = 0 | b = 0] - 1 \Big\} \\
&= \frac{1}{2}\Big\{ \Pr[b' = 1 | \mathsf{Abort} \wedge b = 1] \Pr[\mathsf{Abort} | b = 1] + \\
&\qquad \Pr[b' = 1 | \neg\mathsf{Abort} \wedge b = 1] \Pr[\neg\mathsf{Abort} | b = 1] + \\
&\qquad \Pr[b' = 0 | \mathsf{Abort} \wedge b = 0] \Pr[\mathsf{Abort} | b = 0] + \\
&\qquad \Pr[b' = 0 | \neg\mathsf{Abort} \wedge b = 0] \Pr[\neg\mathsf{Abort} | b = 0] - 1 \Big\} \\
&= \frac{1}{2}\Big\{ \frac{1}{2} \cdot \Big(1 - \frac{1}{2^\ell}\Big) + 1 \cdot \frac{1}{2^\ell} + \frac{1}{2} \cdot \frac{1}{2} + \alpha \cdot \frac{1}{2} - 1 \Big\} \\
&= \frac{1}{2}\Big\{ \frac{1}{2}\alpha + \frac{1}{2^{\ell+1}} - \frac{1}{4} \Big\} \\
&= \frac{1}{2}\Big\{ \frac{1}{2}\Big(\frac{1}{2} - \frac{1}{2^\ell}\Big) + \frac{1}{2^{\ell+1}} - \frac{1}{4} \Big\} \\
&= 0
\end{aligned}$$

which is clearly negligible.

B The Problem of Double Encryptions

The double encryption problem is essentially a problem of determinism. Suppose we attempt to construct a dCCS attacker from a BDU attacker using the simulated memory array techniques in Theorem 1. Consider a BDU attacker $\mathcal{A}$ (with $\ell = 4$) that runs as follows:

1. $M[1] = sk_1$; $M[2] = sk_2$; $M[3] = sk_3$; $M[4] = sk_4$ (by definition)
2. $M[5] \leftarrow \texttt{challenge}(M[1], M[2])$
3. $M[6] \leftarrow \texttt{encrypt}(M[5], pk_3)$
4. $M[7] \leftarrow \texttt{encrypt}(M[6], pk_4)$
5. $\texttt{reveal}(M[7])$

It is unclear how to simulate $M[6]$ and $M[7]$. One way for $\mathcal{B}$ to simulate this memory array would be to compute $M[6]$ using an $\texttt{encrypt}_2(pk_3, f_1, f_2)$ query (where f_i is the function that outputs sk_i) and then to compute $M[7]$ as an encryption of $M[6]$. However, suppose $\mathcal{A}$ continues as follows:

6. $\texttt{reveal}(M[3])$

$\mathcal{B}$ would be unable to make a $\texttt{corrupt}(pk_3)$ query to recover sk_3 since $\mathcal{B}$ cannot corrupt a challenge public key. Hence, the simulation seems to have to abort despite the fact that $\mathcal{A}$ hasn't made an invalid query.

Another way for $\mathcal{B}$ to simulate the memory array would be to compute $M[7]$ by making an $\texttt{encrypt}_2(pk_4, \mathcal{E}(pk_3, f_1(\,\cdot\,)), \mathcal{E}(pk_3, f_2(\,\cdot\,)))$ query using the fact that $\mathcal{E} \in \mathcal{F}$. Note that this would involve committing to the randomness used by the encryption circuit when we make the $\texttt{encrypt}_2$ query. However, suppose $\mathcal{A}$ continues as follows:

6. $\texttt{reveal}(M[1])$
7. $\texttt{reveal}(M[2])$
8. $\texttt{reveal}(M[6])$

This would give us all the information that we require to re-compute the (deterministic) functions $\mathcal{E}(pk_3, f_1(sk_1, sk_2))$ and $\mathcal{E}(pk_3, f_2(sk_1, sk_2))$ used to compute the value at $M[6]$. This would leak the value of b used by the challenge oracle.

Of course, $\mathcal{B}$ could compute $M[6]$ using an $\texttt{encrypt}_2(pk_3, f_1(\,\cdot\,), f_2(\,\cdot\,))$ query and $M[7]$ using a separate $\texttt{encrypt}_2(pk_4, \mathcal{E}(pk_3, f_1(\,\cdot\,)), \mathcal{E}(pk_3, f_2(\,\cdot\,)))$ query. In this situation, the simulation is

likely to be inconsistent, because the ciphertext which is encrypted in the computation of $M[7]$ is unlikely to be the same as the ciphertext stored at $M[6]$. We would have to argue that this (inconsistent) representation did not significantly affect $\mathcal{A}$'s advantage and this argument would have to be based on the dCCS security of the encryption scheme. This does not seem to be possible within the dCCS security model framework as the same determinism problems would occur in such a proof.