Security Notions for Disk Encryption

Kristian Gjøsteen

Department of Telematics,
Norwegian University of Science and Technology, 7491 Trondheim, Norway
kristian.gjosteen@item.ntnu.no

Abstract. We define security goals and attack models for disk encryption, and prove several results for the resulting security notions, as well as some relationships. We give concrete constructions for every security notion along with security proofs. We briefly discuss the security of some implementations and standards for disk encryption.

1 Introduction

It is quite common for confidential and important data to be written to some storage medium (laptop computers, memory sticks, optical or magnetic media, networked storage systems, etc.), but where the physical integrity of the storage medium cannot be guaranteed. The obvious solution is to use cryptography.

While it is possible to include encryption and integrity into application programs, this is often infeasible, either because proper security must be designed into applications, not added as an afterthought, or because the software cannot be modified. The creation of temporary files as well as working copies leave traces on the physical disk, which means that conventional file encryption programs are of little use. Even when such solutions are possible, they will often be difficult to use correctly.

One popular solution is *disk encryption*, where all data is encrypted by the operating system before it is written to the storage medium. This is an attractive solution, because it is potentially very easy to use (being almost transparent to the user), and existing applications can be used unmodified.

We shall consider four different attack scenarios for disk encryption.

Theft. The simplest situation is when the storage medium is stolen, where the goal will be confidentiality. The thief should not be able to read the confidential information stored.

Passive monitoring. There are situations where the adversary is able to monitor the data being read and written to and from the storage medium, but he is not able to modify the data. One example could be electromagnetic radiation leaking from the cables between the user device and the storage medium. A more plausible example is a storage device connected to a network, where the attacker can read network traffic, but is unable or unwilling to modify traffic. A third example is read-only storage media used for transport.

Theft with recovery. A slightly more complicated situation is where the storage medium is first stolen, the theft is discovered, and the medium is subsequently recovered. Obviously, confidentiality remains a goal, but in addition, the storage medium may have been tampered with. We need to be confident in its integrity.

Active attack. The most difficult situation is when the adversary has surreptitious read and write access to the storage medium while it is in use (or between sessions). The most extreme example is out-sourced storage accessed via a network, where the adversary either controls the network, or is in complete control of the storage medium (we could say that the adversary is the storage medium).

We define precisely what a disk encryptor is in Sect. 2, and look briefly at two practical implementations of disk encryptors, as well as the work of a standards group. In Sect. 3 we define attack models and security goals (combinations of which form security notions). In Sect. 4 we discuss relationships between various security notions, as well as some general results about what is required to reach various security notions.

In Sect. 5, we give several constructions for disk encryptors, meeting every security notion discussed in Sect. 3. We also discuss the security of the implementations and the standard discussed in Sect. 2. The proofs for the results in Sect. 4 and 5 are somewhat technical, and are included in Appendix A.

2 Disk Encryptor

2.1 Definitions

We first define what a sector-based storage medium is. Let S be a set of possible sector values (typically $S = \{0,1\}^l$ for some l). A *storage medium* for n sectors is an interactive deterministic algorithm. It accepts as input write or read requests, and keeps a list of pairs from $\{0,1,\dots,n-1\} \times S$. The list is initially empty.

The write request is a pair $(i,s) \in \{0,1,\dots,n-1\} \times S$ ("store s at index i"). The storage medium stores the pair (i,s) in a list, discarding any previously stored pair (i,s'), and replies with the special symbol $\top$.

The read request is a number $i \in \{0,1,\dots,n-1\}$ ("read from index i"). If the storage medium has a pair (i,s) in its list, it outputs s. Otherwise, it outputs some (fixed) value from S.

Let $x = (x_1, x_2, \dots, x_r) \in S^r$, and let I be a subset of $\{0,1,\dots,n-1\}$ of cardinality r. Order the elements of I and denote the jth element by i_j. *Reading (writing) x according to I* means to read (write) x_j from (to) sector i_j.

A *disk encryptor* $\mathcal{D}$ for N plaintext sectors with values in S is an interactive algorithm, accepting input from a user and giving input to a storage medium[1].

The disk encryptor accepts as its first input a key from a set K and possibly a state. As part of its initialization, it may issue read and write requests to the

[1] It is possible (and desirable) to allow the disk encryptor and the storage medium to have different sector sizes. To simplify the presentation, we keep sector sizes equal.

storage medium. After initialization, the disk encryptor accepts as input read and write requests.

The disk encryptor may keep a state between read and write requests. We shall assume that this state is publicly known at all times, but that no adversary may modify it. A disk encryptor that does not keep a state is *stateless*.

The write request is a pair $(i, s) \in \{0, 1, \dots, N-1\} \times S$. The processing of the write request is probabilistic, and may result in read requests as well as write requests to the storage medium. All the read requests must be completed before any write request occurs. After all the read requests are complete, but before the first write request has been issued, the disk encryptor may stop processing and reply with the special symbol $\perp$ (signifying encryption error). Otherwise, the disk encryptor issues its write requests to the storage medium and replies with the special symbol $\top$ (signifying no error) when processing is complete.

The read request is a number $i \in \{0, 1, \dots, N-1\}$. The processing of the read request is deterministic, and will only result in read requests to the storage medium. When the processing is complete, the disk encryptor replies either with a sector value from S, or with the special symbol $\perp$, signaling decryption error. The disk encryptor's state should not change as a result of a read request.

We require that the indexes of the read and write requests issued to the storage medium should only depend on the index of the input request, not on the key or the state.

The disk encryptor must guarantee that, in normal operation, if (i, s) was the last write request issued for i, then the read request i will return s. If no write request (i, s') for any s' has been issued, the disk encryptor may respond arbitrarily to the read request i.

To provide secure storage of N sectors with values from S, the disk encryptor requires $n \geq N$ sectors of storage medium. The ratio n/N is the *expansion ratio* of the disk encryptor.

We impose one more requirement on a disk encryptor: Any read and write request to the storage medium should result in at most a constant times $\log N$ reads and writes to the storage medium.

The sectors read and written through the user interface of a disk encryptor are the *plaintext sectors*. The sectors on the storage medium are the *ciphertext sectors*.

2.2 Existing Implementations and Standards

The following presentation includes three concrete examples of implementations or standards. We shall analyze these systems in Sect. 5.

LoopAES [11] is a disk encryptor for Linux-based computer operating systems. Its stated aim is to provide confidentiality, but not integrity. It encrypts sectors using a block cipher in CBC mode, and has three modes of operation, one using a single key and two different modes using multiple keys.

The documentation is somewhat unclear, but a (possibly deprecated) single key mode seems to use the sector index as initialization vector for CBC mode.

One multiple-key mode apparently uses a pseudo-random function family to derive the initialization vector from the sector index. We discuss these variants in Sect. 5.2. We quote from the purpose of the Security in Storage Working Group (SISWG) [5]:

> This standard provides a standard architecture for media security and enabling components. Present non-standard, insecure encrypted storage methodologies are augmented, and users will be able to create higher-assurance, standard, interoperable solutions.

They restrict their attention to disk encryptors with expansion rate 1. As we shall see, Theorem 5 will severely limit the security level achievable.

One of their proposed methods for disk sector encryption uses tweakable, sector-wide block ciphers. We discuss this construction in Sect. 5.3.

The disk encryptor called GEOM Based Disk Encryption (GBDE) [7] is part of the computer operating system FreeBSD.

The main idea is that every sector is encrypted using a block cipher in CBC mode. A constant initialization vector is used together with a one-time key. The one-time key is encrypted and written to a different sector using a block cipher in CBC mode. The key for this encryption is derived from a master key, sector index and a salt using a pseudo-random function family.

We discuss scheme in Sect. 5.5, along with other alternatives.

3 Security Goals and Attack Models

As usual, we model an attack as a game played between an adversary and a simulator. At the start of the game, the simulator initializes a disk encryptor instance with a randomly chosen key. The adversary is given access to the disk encryptor and the storage medium through the simulator.

In practice, much of the information written to the disk encryptor will either be influenced, known or guessable by the adversary. This means that a chosen ciphertext attack is unrealistic. To simplify modelling, we allow the adversary to write arbitrary data of his choice to the disk encryptor.

The adversary should also have read access to any data written by himself (when relevant). The intuition is that most users would not hesitate in giving the adversary access to data supplied by the adversary.

The simulator must bar read access to any data it has written to the disk encryptor, to keep the adversary from achieving his goals trivially.

The four scenarios in the introduction give us the following attack models[2], which describe the attack conditions.

Non-adaptive chosen plaintext attack (naCPA). The adversary gets write access to the disk encryptor. When he is finished writing, he is given read access to the storage medium for the duration of the game. (This corresponds to the theft scenario.)

[2] Technically, many more attack models are possible, but we consider only those that seem interesting.

Chosen plaintext attack (CPA). The adversary is given read and write access to the disk encryptor. Everything read from and written to the storage medium is simultaneously copied to the adversary. (This corresponds to the passive monitoring scenario.)

Non-adaptive chosen ciphertext attack (naCCA). The adversary is given write access to the disk encryptor. When he is finished writing, he is given read and write access to the storage medium. When he is finished with the storage medium, he is given read access to the disk encryptor. (This corresponds to the theft with recovery scenario.)

Chosen ciphertext attack (CCA). The adversary is given read and write access to the disk encryptor, and the simulator uses the adversary as storage medium. (This corresponds to the active attack scenario.)

naCPA is the weakest attack. naCCA and CPA are not comparable. CCA is the strongest attack.

We shall now describe a series of security goals for disk encryption. Throughout, we consider a disk encryptor $\mathcal{D}$ providing N plaintext sectors with values from a set S.

The classic security goal is confidentiality. Following the standard notion of semantic security, an adversary that has partial information about the plaintext sectors must not be able to deduce anything new by studying the storage medium. We note that this is equivalent to the notion of indistinguishability, where an adversary must distinguish between encryptions of two messages he has chosen.

Definition 1. *An adversary A against* semantic security *works as follows: First the adversary specifies a probability space X over S^r, an index set $I \subseteq \{0, 1, \ldots, N-1\}$ of cardinality r, and a function $f : S^r \to \{0,1\}$ such that* $\Pr[f(x) = 0 \mid x \overset{r}{\leftarrow} X] = \Pr[f(x) = 1 \mid x \overset{r}{\leftarrow} X] = 1/2$*. The simulator samples x from X and writes x to the disk encryptor according to I. The adversary then outputs a bit $b \in \{0,1\}$.*

Let E be the event that $f(x) = b$. The adversary's advantage is

$$\mathrm{Adv}_A^{\mathcal{D},\mathrm{IND},\cdot} = |\Pr[E] - 1/2|.$$

Remark 1. The adversary is assumed to output a description of the probability space X such that the simulator can sample from X.

Remark 2. For all of the attack games described in this section, we say that an adversary requires time at most t if the described game requires at most time t to complete, counting the work done by the adversary *and* the simulator.

Another important security goal is that of non-malleability, where the adversary should not be able to change the ciphertext sectors to cause any meaningful change in the plaintext sectors.

Definition 2. *An adversary A against* non-malleability *works as follows: First the adversary specifies a probability space X over S^r, and an index set $I \subseteq \{0, 1, \dots, N-1\}$ of cardinality r. The simulator samples $x = (x_1, x_2, \dots, x_r)$ from X and writes x to the disk encryptor according to I.*

The simulator will deny any read requests from the sectors in I. The adversary may choose to write to any sector in I. When the ith sector in I is written to, we remove its index from I, decrease r by one, and replace X by the conditional probability space where the ith coordinate is fixed to the value x_i.

When the adversary terminates, he outputs a relation R on $S^r \times S^r$. The simulator reads $x' = (x'_1, x'_2, \dots, x'_r)$ from the disk encryptor according to I. Now we replace X with the conditional probability space where the ith coordinate is fixed to the value x_i if $x_i = x'_i$, and sample x'' from X.

Let E_0 be the event that there was no decryption error when reading x', and $x \, R \, x'$. Let E_1 be the event that there was no decryption error when reading x', and $x'' \, R \, x'$. The adversary's success rate is

$$\mathrm{Succ}_A^{\mathcal{D},\mathrm{NM},\cdot} = |\Pr[E_0] - \Pr[E_1]|.$$

Remark 3. The adversary is assumed to output a description of the probability space X such that the simulator can sample from X and the conditional probability spaces.

Remark 4. If the adversary cannot change the ciphertext sectors, x will be equal to x'', and the adversary has zero sucess rate. Therefore, non-malleability is only relevant for the chosen ciphertext attacks.

In public key cryptography, the adversary is free to construct ciphertexts using the public key. In this way, adversaries can defeat non-malleability without ever tampering with a ciphertext. For private key cryptography, the standard definition [8] allows ciphertexts output by an encryption oracle in the final answer. This mirrors the public key case.

For disk encryption, the equivalent notion would be to allow the adversary to write to the sectors in I without changing the probability space X. We disallow this, thereby separating the goals of indistinguishability and non-malleability.

The following two goals are slightly different. The first (weaker) goal says that the adversary should not be able to cause the encrypted data to change in any way (even randomly). The second (stronger) goal says that any change the adversary makes to the storage medium will result in a decryption error.

Definition 3. *An adversary A against* plaintext integrity *works as follows: The simulator keeps a private copy of anything written to the disk encryptor. When data is read from the disk encryptor, it is compared with the private copy.*

Let E be the event that data successfully read from the disk encryptor is different from the private copy. The adversary's success rate is

$$\mathrm{Succ}_A^{\mathcal{D},\mathrm{PTXT},\cdot} = \Pr[E].$$

Definition 4. *An adversary A against* ciphertext integrity *works as follows: The simulator keeps a private copy of anything the disk encryptor writes to the storage medium. Whenever the disk encryptor reads from the storage medium, it is compared with the private copy.*

Let E be the event that something the disk encryptor reads from the storage medium is different from the private copy, but the disk encryptor does not signal an error. The adversary's success rate is

$$\mathrm{Succ}_A^{\mathcal{D},\mathrm{CTXT},\cdot} = \Pr[E].$$

Remark 5. We note that an adversary that can replace a sector value with random data has no success rate against non-malleability. Random changes may still represent a problem for certain applications, who will require the stronger notion of plaintext integrity.

4 General Results

We follow the concrete security approach of [2]. The main technique is to use an attacker against a notion X to construct an attacker against a notion Y. Since security is the absence of attackers, logic then dictates that security notion Y implies security notion X. To separate notions X and Y, we use one disk encryptor to create a second disk encryption. First we show that the latter does not satisfy notion Y. Second, if the first satisfies notion X, then the second also satisfies notion X. This means that security notion X does not imply security notion Y.

An adversary against non-malleability must be a successful adversary against plaintext integrity. Also, any adversary against plaintext integrity must be an adversary against ciphertext integrity. But the converse is not true: Plaintext integrity does not imply ciphertext integrity, since the ciphertext may contain information that can be changed without affecting the decryption.

Theorem 1. *Let $\mathcal{D}$ be a disk encryptor. There exists a disk encryptor $\mathcal{D}'$ and a non-adaptive chosen ciphertext adversary A against ciphertext integrity such that $\mathrm{Succ}_A^{\mathcal{D}',\mathrm{CTXT},\mathrm{naCCA}} = 1$. Further, for any adversary A' against plaintext integrity for $\mathcal{D}'$, there exists an adversary A'' against plaintext integrity for $\mathcal{D}$ such that*

$$\mathrm{Succ}_{A''}^{\mathcal{D},\mathrm{PTXT},\cdot} = \mathrm{Succ}_{A'}^{\mathcal{D}',\mathrm{PTXT},\cdot}.$$

A uses trivial time, and A'' requires as much time as A'.

The following theorem says that plaintext integrity does not imply semantic security, because the ciphertext may contain a copy of the plaintext.

Theorem 2. *Let $\mathcal{D}$ be a disk encryptor. There exists a disk encryptor $\mathcal{D}'$ and a non-adaptive chosen plaintext adversary A against semantic security for $\mathcal{D}'$ with $\mathrm{Adv}_A^{\mathcal{D}',\mathrm{IND},\mathrm{naCPA}} = 1/2$. Further, for any adversary A' against plaintext integrity for $\mathcal{D}'$, there exists an adversary A'' against plaintext integrity for $\mathcal{D}$ such that*

$$\mathrm{Succ}_{A''}^{\mathcal{D},\mathrm{PTXT},\cdot} = \mathrm{Succ}_{A'}^{\mathcal{D}',\mathrm{PTXT},\cdot}.$$

A uses trivial time, and A'' has essentially the same time requirements as A'.

The most interesting result in this section is the following: Semantic security against (non-adaptive) chosen ciphertext attacks follows from semantic security against (non-adaptive) chosen plaintext attacks and ciphertext integrity against (non-adaptive) chosen ciphertext attacks. The idea is that if the adversary changes the ciphertext without causing decryption errors, he is a successful adversary against ciphertext integrity. If not, he reduces to a simple chosen plaintext adversary. (This mirrors results for symmetric cryptosystems in [9].)

Theorem 3. *Let $\mathcal{D}$ be a disk encryptor, and let A be a (non-adaptive) chosen ciphertext adversary against semantic security. Then there exists a (non-adaptive) chosen ciphertext adversary A' against ciphertext integrity, and a (non-adaptive) chosen plaintext adversary A'' against semantic security such that*

$$\mathrm{Adv}_{A}^{\mathcal{D},\mathrm{IND},\mathrm{(na)CCA}} \leq \mathrm{Succ}_{A'}^{\mathcal{D},\mathrm{CTXT},\mathrm{(na)CCA}} + \mathrm{Adv}_{A''}^{\mathcal{D},\mathrm{IND},\mathrm{(na)CPA}}.$$

A' and A'' has the same time requirements as A, except for a trivial amount of processing for every disk encryptor read and write.

Next, we note that ciphertext integrity and semantic security can be handled independently. This simplifies design and analysis of disk encryptors.

We show this by *composing* disk encryptors. If $\mathcal{D}_1$ and $\mathcal{D}_2$ are two disk encryptors with keys from K_1 and K_2, respectively, $\mathcal{D}_1 \circ \mathcal{D}_2$ is a disk encryptor that works as follows: It takes keys from $K_1 \times K_2$ (the keys are independent). Read and write requests to the composition are forwarded to $\mathcal{D}_1$. Any storage read and write requests made by $\mathcal{D}_1$ are forwarded to $\mathcal{D}_2$. If $\mathcal{D}_2$ responds with $\perp$ to any request, processing is halted and $\perp$ is output. Otherwise, $\mathcal{D}_1$ is allowed to complete its processing, and its result is output.

Since the keys for $\mathcal{D}_1$ and $\mathcal{D}_2$ are independent, the following theorem holds.

Theorem 4. *Let $\mathcal{D}_1$ and $\mathcal{D}_2$ be two disk encryptors. For any (non-adaptive) chosen plaintext adversary A_1 and (non-adaptive) chosen ciphertext adversary A_2 against, respectively, semantic security and ciphertext integrity for $\mathcal{D}_1 \circ \mathcal{D}_2$, there exists a (non-adaptive) chosen plaintext adversary A'_1 against $\mathcal{D}_1$ such that*

$$\mathrm{Adv}_{A'_1}^{\mathcal{D}_1,\mathrm{IND},\mathrm{(na)CPA}} = \mathrm{Adv}_{A_1}^{\mathcal{D}_1\circ\mathcal{D}_2,\mathrm{IND},\mathrm{(na)CPA}},$$

as well as a (non-adaptive) chosen ciphertext adversary A'_2 against $\mathcal{D}_2$ such that

$$\mathrm{Succ}_{A'_2}^{\mathcal{D}_2,\mathrm{CTXT},\mathrm{(na)CCA}} = \mathrm{Succ}_{A_2}^{\mathcal{D}_1\circ\mathcal{D}_2,\mathrm{IND},\mathrm{(na)CCA}}.$$

A'_1 and A'_2 have the same time requirements as A_1 and A_2, respectively.

If no redundancy is added to the stored data, the best security that can be achieved is semantic security and non-malleability against a non-adaptive chosen ciphertext attack.

Theorem 5. *Suppose a disk encryptor $\mathcal{D}$ has expansion ratio* 1*. Then there exists a chosen plaintext adversary against semantic security, a chosen ciphertext adversary against non-malleability, and a non-adaptive chosen ciphertext adversary against plaintext integrity, all with advantage* $1/2$ *or success rate* 1*, and trivial time requirements.*

The final result shows that to achieve security against chosen ciphertext attacks, the disk encryptor must keep a state. The attack uses what is commonly known as rollback or replay attacks. These trivially compromise integrity, and also allow attacks on semantic security.

Theorem 6. *Let $\mathcal{D}$ be a stateless disk encryptor. Then a chosen ciphertext adversary A against semantic security exists such that*

$$\mathrm{Adv}_A^{\mathcal{D},\mathrm{IND},\mathrm{CCA}} = 1/2.$$

A has trivial time requirements.

5 Concrete Constructions

We give several constructions for meeting the various security notions described in Sect. 3. The strategy in this section is to show that an adversary against the security would imply an adversary against a building block. If we have faith in the building blocks, faith in the construction follows.

Let l and m be integers larger than zero. Throughout this section, the set of sector values S will be the set of bit strings of length lm, $S = \{0,1\}^{lm}$. N will denote the number of plaintext sectors provided by the disk encryptor, and n will be the number of ciphertext sectors required. Let $l_0 = \lceil \log_2 m \rceil$ and $l_1 = \lceil \log_2 N \rceil$.

When convenient, we shall consider integers as bit strings, and vice versa, in the usual manner.

5.1 Building Blocks

We are interested in indistinguishable subsets of function families. So let $\bar{F}$ be a function family, and let F be a subset of $\bar{F}$. A *distinguisher A for F* plays the following game with a simulator: First, the simulator samples a function either from F or from $\bar{F}$. A is allowed to query the function (and its inverse, if relevant) at up to q points. Then A outputs 0 or 1.

Let E be the event that A outputs 0 when the simulator sampled from $\bar{F}$, or 1 when the simulator sampled from F. Then A's distinguishing advantage is

$$\mathrm{Adv}_A^{F,q} = |\Pr[E] - 1/2|.$$

Pseudo-Random Function Families. Let $\mathrm{Map}(S, S')$ denote the set of all functions from the set S to the set S'. We are interested in finding subsets of $\mathrm{Map}(S, S')$ where the functions are easy to evaluate, but it difficult it is to distinguish random elements of the subset from random elements of $\mathrm{Map}(S, S')$.

Definition 5. *Let S, S' and K be sets. A* pseudo-random function family (PRF) $\varPhi$ from S to S' indexed by K *is a subset $\varPhi = \{f_k : S \to S' \mid k \in K\}$ of* $\mathrm{Map}(S, S')$*, along with a deterministic algorithm that on input of $k \in K$ and $s \in S$ computes $f_k(s)$.*

We denote an adversary's distinguishing advantage by $\mathrm{Adv}_A^{\mathrm{PRF},\varPhi,q}$.

Typical examples of interesting pseudo-random function families are message authentication codes, such as HMAC [1] and OMAC [6].

Block Ciphers. Let $\mathrm{Perm}(S)$ denote the set of all permutations on the set S. We are interested in finding subsets of $\mathrm{Perm}(S)$ where the permutations are easy to evaluate, but it is difficult to distinguish random elements of the subset from random elements of $\mathrm{Perm}(S)$.

Definition 6. *Let S, K be sets. A* pseudo-random permutation family (PRP) Π on S indexed by K *is a subset $\Pi = \{f_k \mid k \in K\}$ of* $\mathrm{Perm}(S)$*, along with two deterministic algorithms that on input of $k \in K$ and $s \in S$ computes $f_k(s)$ and $f_k^{-1}(s)$, respectively.*

We denote an adversary's distinguishing advantage by $\mathrm{Adv}_A^{\mathrm{PRP},\Pi,q}$.

Typical examples of pseudo-random permutation families are block ciphers such as AES [3].

We note that any pseudo-random permutation family Π on S can be used as a pseudo-random function family from S to S, and it is easy to show that for any PRF-distinguisher A, there exists a PRP-distinguisher A' such that

$$\mathrm{Adv}_A^{\mathrm{PRF},\Pi,q} \leq \mathrm{Adv}_{A'}^{\mathrm{PRP},\Pi,q} + q^2/|S|.$$

Tweakable Block Ciphers. Let S and T be sets. A *tweakable permutation on S tweaked by T* is a function $f : T \to \mathrm{Perm}(S)$. When convenient, we abuse notation and denote the action of $f(t)$ on s by $f(t, s)$, considering f as a function $f : T \times S \to S$. Let $\mathrm{Perm}_T(S)$ denote the set of tweakable permutations on S.

Definition 7. *Let S, T and K be sets. A* tweakable pseudo-random permutation family $\tilde{\Pi}$ on S indexed by K and tweaked by T *is a subset $\tilde{\Pi} = \{f_k \mid k \in K\}$ of* $\mathrm{Perm}_T(S)$*, along with two deterministic algorithms that on input of $k \in K$, $t \in T$ and $s \in S$ computes $(f_k(t))(s)$ and $(f_k(t)^{-1})(s)$, respectively.*

We denote an adversary's distinguishing advantage by $\mathrm{Adv}_A^{\mathrm{TPRP},\tilde{\Pi},q}$.

We refer to [4,10] for further background on tweakable permutations and concrete constructions. We restrict ourselves to noting that there are practical constructions based on block ciphers.

5.2 Semantic Security Against Non-adaptive Chosen Plaintext Attack

It is fairly easy to see that a block cipher used in Electronic Code Book (ECB) mode does not provide semantic security against a non-adaptive chosen plaintext attack. We outline two simple constructions that provide semantic security.

Our first construction is also our simplest construction. It is based on the well-known counter mode construction. Let Φ be a pseudo-random function family from $\{0,1\}^l$ to $\{0,1\}^l$ indexed by K.

Define the function $r : \mathrm{Map}(\{0,1\}^l, \{0,1\}^l) \times \{0,1,2,\ldots,N-1\} \to S$ to be function that takes (f, i) to the concatenation of the value of $f(i2^{l_0} + j)$ for $0 \le j < m$, that is,

$$(f, i) \mapsto f(i2^{l_0} + 0)||f(i2^{l_0} + 1)||\ldots||f(i2^{l_0} + m - 1).$$

(Remember that $m \le 2^{l_0}$.)

The disk encryptor $\mathcal{D}_1(\Phi)$ takes keys from K, and $n = N$. Suppose the disk encryptor is initialized with the key $k \in K$. Given the write request (i, s), the disk encryptor issues the write request $(i, s \oplus r(f_k, i))$ to the storage medium. Given the read request i, the disk encryptor reads s' from the ith sector of the storage medium and outputs $s' \oplus r(f_k, i)$.

Theorem 7. *Let $\mathcal{D}_1(\Phi)$ be as above, providing N sectors of storage, and let A be a non-adaptive chosen plaintext adversary against semantic security. Then there exists a distinguisher A' for Φ such that*

$$\mathrm{Adv}_A^{\mathcal{D},\mathrm{IND},\mathrm{naCPA}} = 2\mathrm{Adv}_{A'}^{\mathrm{PRF},\Phi,Nm}.$$

We note that block ciphers are good candidates for efficient pseudo-random function families.

The next construction is based on Cipher Block Chaining mode. CBC mode requires an unpredictable initialization vector, so using the sector index does not provide security. We have two easy options: either use a pseudo-random function family to derive the IV from the sector index (the approach taken by one variant of LoopAES described in Sect. 2.2), or simply run the sector index through the block cipher and use that as an initialization vector.

Let Π be a pseudo-random permutation family on $\{0,1\}^l$ indexed by K.

The disk encryptor $\mathcal{D}_2(\Pi)$ takes keys from K, and $n = N$. Let $\mathcal{D}_2(\Pi)$ be initialized with the key $k \in K$. Given the write request (i, s), the value s is split into blocks $s_1, \ldots, s_m \in \{0,1\}^l$. The IV is derived as $c_0 = f_k(i)$. Then c_i is computed as $f_k(c_{i-1} \oplus s_i)$, and the write request $(i, c_1||c_2||\ldots||c_m)$ is issued to the storage medium.

We leave the read operation and the security proof to the interested reader.

Compared to counter mode above, this scheme is more complicated, requires both the encryption and decryption part of the block cipher, and cannot easily be parallellized. CBC-mode does, however, seem to have wider hardware support, though this will probably change in the future. (Some would say that since counter mode is totally insecure against stronger attacks, and CBC-mode could in practice thwart some stronger attacks, there are security advantages to using CBC-mode. However, if one worries about stronger attacks, one should defend against stronger attacks.)

5.3 Non-malleability Against Non-adaptive Chosen Ciphertext Attack

This construction uses a tweakable permutation on the sector level to encrypt the data. Note that the permutation has to be tweakable, otherwise the usual attacks on ECB mode apply.

Let K be a set and $T = \{0, 1, \ldots, N-1\}$. Let $\tilde{\Pi}$ be a tweakable pseudo-random permutation family on S indexed by K and tweakable by T.

The disk encryptor $\mathcal{D}_3(\tilde{\Pi})$ takes keys from K, and $n = N$. Let $\mathcal{D}_3(\tilde{\Pi})$ be initialized with the key $k \in K$. Given the write request (i, s), it issues the write request $(i, f_k(i, s))$ to the storage medium. Given the read request i, it passes it on to the storage medium and gets a value s'. It then returns the value $f_k(i)^{-1}(s)$.

Theorem 8. *Let $\mathcal{D}_3(\tilde{\Pi})$ be as above, and let A be a non-adaptive chosen ciphertext adversary against semantic security (non-malleability). Then there exists distinguisher A' and A'' for $\tilde{\Pi}$ such that*

$$\mathrm{Adv}_A^{\mathcal{D}_3(\tilde{\Pi}),\mathrm{IND},\mathrm{naCCA}} = 2\mathrm{Adv}_{A'}^{\mathrm{TPRP},\tilde{\Pi},N}$$

and

$$\mathrm{Succ}_A^{\mathcal{D}_3(\tilde{\Pi}),\mathrm{NM},\mathrm{naCCA}} \leq 2\mathrm{Adv}_{A''}^{\mathrm{TPRP},\tilde{\Pi},N}.$$

Note that this scheme has expansion rate 1, and by Theorem 5 this is the best we can do with expansion rate 1. Furthermore, this scheme is essentially equivalent to the sector-wide scheme adopted by SISWG (see Sect. 2.2).

5.4 Ciphertext Integrity Against Non-adaptive Chosen Ciphertext Attack

The following construction provides ciphertext integrity using a pseudo-random function family.

Let $T = \{0, 1, \ldots, N-1\}$, and let Φ be a pseudo-random function family from $T \times S$ to $\{0, 1\}^l$ indexed by K.

The disk encryptor $\mathcal{D}_4(\Phi)$ takes keys from K, and $n = 2N$. Let $\mathcal{D}_4(\Phi)$ be initialized with the key $k \in K$. Given the write request (i, s), it issues the write requests $(2i, s)$ and $(2i+1, f_k(i, s))$ to the storage medium (the bit string $f_k(i, s)$ is padded with zeros to get a string of length ml). Given the read request i, it issues the read requests $2i$ and $2i+1$ to the storage medium, getting values s' and s''. If $s'' = f_k(i, s)$ (ignoring zero padding), s' is output, otherwise $\perp$.

Theorem 9. *Let $\mathcal{D}_4(\Phi)$ be as above, and let A be a non-adaptive chosen ciphertext adversary against ciphertext integrity. Then there exists a distinguisher A' for Φ such that*

$$\mathrm{Succ}_A^{\mathcal{D}_4(\Phi),\mathrm{CTXT},\mathrm{naCCA}} \leq \mathrm{Adv}_{A'}^{\mathrm{PRF},\Phi,N} + \frac{N}{2^l}.$$

We note that the zero padding is rather wasteful, but it is required for technical reasons[3]. However, the storage medium can easily arrange to store several checksums in one physical sector, giving an expansion rate of $(m+1)/m$.

[3] If we stored more than one checksum in the same sector, we could change the first checksum, and then read a sector corresponding to an unchanged checksum. The read would return $\top$, but the attacker would win the chosen ciphertext attack game.

We also note that it is easy to construct Φ using for example HMAC [1] or OMAC [6].

By Theorems 3 and 4, we can compose a scheme from Sect. 5.2 with this scheme to achieve semantic security against non-adaptive chosen ciphertext attacks.

5.5 Semantic Security Against Chosen Plaintext Attack

The following construction provides semantic security against a chosen plaintext attack using a pseudo-random function family. It is based on counter mode, but each sector is given its own initialization vector.

Let Φ be a pseudo-random function family from $\{0,1\}^l$ to $\{0,1\}^l$ indexed by K. Let $r : \mathrm{Map}(\{0,1\}^l, \{0,1\}^l) \times \{0,1\}^{l-l_0} \to S$ be the function defined by

$$(f,t) \mapsto f(t2^{l_0}+0)||f(t2^{l_0}+1)||\ldots||f(t2^{l_0}+m-1).$$

(Remember that $m \leq 2^{l_0}$.)

The disk encryptor $\mathcal{D}_5(\Phi)$ takes keys from K, and $n = 2N$. Let $\mathcal{D}_5(\Phi)$ be initialized with the key $k \in K$. Given the write request (i,s), $\mathcal{D}_5(\Phi)$ samples j from $\{0,1\}^{l-l_0}$, then issues the write requests $(2i, j)$ (where j is padded with zeros) and $(2i+1, s \oplus r(f_k, j))$.

Given the read request i, the disk encryptor issues the read requests $2i$ and $2i+1$ to the storage medium, getting values s' and s''. It then outputs $s'' \oplus r(f_k, s')$ (where the zero padding in s' is ignored).

Theorem 10. *Let $\mathcal{D}_5(\Phi)$ be as above, and let A be a chosen plaintext adversary against semantic security that writes at most q sectors to the disk encryptor. Then there exists a distinguisher A' for Φ such that such that*

$$\mathrm{Adv}_A^{\mathcal{D}_5(\Phi),\mathrm{IND},\mathrm{CPA}} \leq \mathrm{Adv}_{A'}^{\mathrm{PRF},\Phi,qm} + q^2/2^{l-l_0}.$$

We note that a similar technique can be used for CBC-mode or with a tweakable block cipher. We also note that the storage medium could arrange to store several initial values in one physical sector, reducing the expansion rate to $(m+1)/m$.

The above scheme aims to achieve that same goal as FreeBSD's GBDE (see Sect. 2.2). Briefly, GBDE works as follows: it encrypts each data sector using a pseudo-random permutation family Π in CBC mode with a fixed IV, but with a random one-time "sector key". The random key is then written to a different sector, encrypted using Π and a "key-key" derived from a "master key" and the sector index using a pseudo-random function family Φ.

We sketch a possible proof that GBDE is semantically secure against chosen plaintext attacks: First we replace the pseudo-random function used to derive the "key-keys" with a random function. Then we replace the pseudo-random permutations used to encrypt the "sector keys" with random permutations. If "sector key" is ever reused, the adversary will never learn any information about the "sector keys". Then we replace the pseudo-random permutations used to encrypt the data sectors with random permutations. After the final replacements, the adversary can have no advantage. The resulting bound will be somewhat weaker than that the above theorem.

While we believe $\mathcal{D}_5$ to be a superior construction to GBDE, the latter has the significant advantage of being available for use today.

5.6 Ciphertext Integrity Against Chosen Ciphertext Attack

By Theorem 6, a disk encryptor must keep a state to achieve security against chosen ciphertext attack. Our goal is to keep the state as small and simple as possible. The idea is to use an m-ary tree of checksums (see Fig. 1). The root of the tree is authenticated using the state, and the state changes with every write.

Let T be the set $\{0, 1, \dots, 2^l - 1\}$, and let $\varPhi$ be a pseudo-random family of functions from $T \times S$ to $\{0,1\}^l$ indexed by K.

Set $N_1 = N$, and define the sequence N_j by $N_j = \lceil N_{j-1}/m \rceil$. Let h be the smallest integer such that $N_h = 1$. Set $n_1 = N$ and define the sequence n_j by $n_{j-1} + N_{j-1}$. For an integer i, we let $i_j = \lfloor i/m^{j-1} \rfloor$.

The disk encryptor $\mathcal{D}_6(\varPhi)$ takes keys from K, and $n = n_{h+1} + 1$. Let $\mathcal{D}_6(\varPhi)$ be initialized with the key k. It sets the state σ to the integer 1.

Denote the value of the sector $n_j + t$, $0 \le t < N_j$, by $s_t^{(j)}$, and the sector $n_{h+1} = n_h + 1$ by $s^{(h+1)}$. The initialization process first zeros $s_i^{(1)}$ for all i. Then it computes the correct checksums $s_{i_j}^{(j)}$ for all i and $1 < j \le h$ using

$$s_{i_j}^{(j)} = f_k(j 2^{l_1} + i_j, s_{mi_j+0}^{(j-1)} || s_{mi_j+1}^{(j-1)} || \dots || s_{\max\{mi_j+m-1, N_j\}}^{(j-1)}). \tag{1}$$

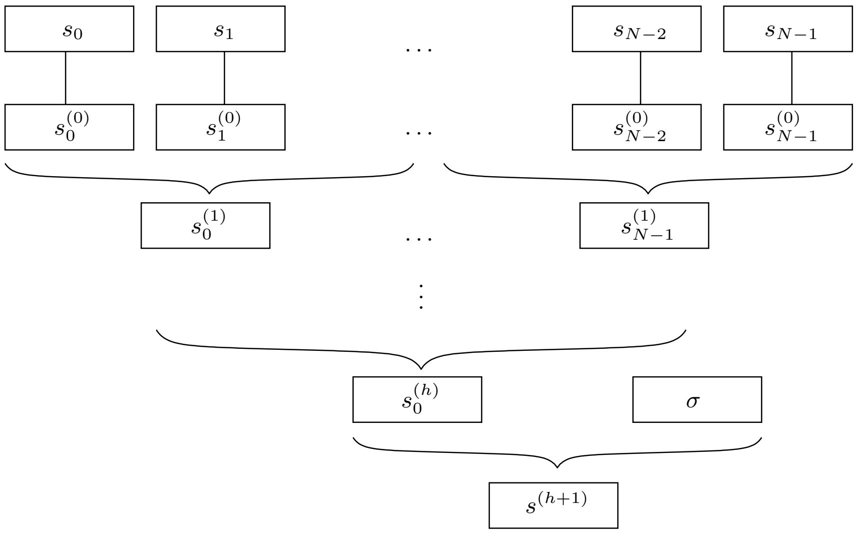

Fig. 1. The MAC tree for $\mathcal{D}_6(\varPhi)$

Note that in the concatenation, the zero padding of each $s_t^{(j)}$ is discarded. Finally, it computes $s^{(h+1)}$ using

$$s^{(h+1)} = f_k(\sigma 2^{2l_1}, s_0^{(h)}). \tag{2}$$

Then the disk encryptor issues the writes $(n_j + i_j, s_{i_j}^{(j)})$ for all $0 \leq i < N$, $1 \leq j \leq h$, and finally the write $(n_{h+1}, s^{(h+1)})$.

Let the write request be (i, s). The disk encryptor reads $s_t^{(j)}$ from the index $n_j + t$, for $i_{j+1}m \leq t < \max\{i_{j+1}m + m - 1, N_j\}$, $1 \leq j \leq h$. Then it reads $s^{(h+1)}$ from the index n_{h+1}.

It verifies that any zero padding remains zero, that for all $1 < j \leq h$, (1) holds, and that (2) holds. If any verification fails, $\perp$ is output and processing terminated.

If all of these checks are correct, the disk encryptor changes $s_i^{(1)}$ to be $f_k(i, s)$, updates every $s_{i_j}^{(j)}$ for $1 < j \leq h$ according to (1), increases σ by 1, and updates $s^{(h+1)}$ according to (2).

Then it issues the write requests (i, s), $(n_j + i_j, s_{i_j}^{(j)})$ for $1 \leq j \leq h$, and $(n_{h+1}, s^{(h+1)})$, and outputs $\top$.

Given the read request i, the disk encryptor reads $s_t^{(j)}$ from index $n_j + t$, for $i_{j+1}m \leq t < \max\{i_{j+1}m + m - 1, N_j\}$, $1 \leq j \leq h$. Then it reads s_i from index i and $s^{(h+1)}$ from index n_{h+1}.

Now the disk encryptor verifies that $s_i^{(1)} = f_k(i, s_i)$, that any zero padding remains zero, that the $s_{i_j}^{(j)}$ satisfy (1) for $1 < j \leq h$, and that $s^{(h+1)}$ satisfies (2). If any verification fails, $\perp$ is output, otherwise s_i is output.

Theorem 11. *Let $\mathcal{D}_6(\Phi)$ and h be as above, and let A be a chosen ciphertext adversary against ciphertext integrity that writes and reads at most q sectors to the disk encryptor. Then there exists a distinguisher A' for Φ such that*

$$\mathrm{Succ}_A^{\mathcal{D}_6(\Phi),\mathrm{CTXT},\mathrm{CCA}} \leq \mathrm{Adv}_{A'}^{\mathrm{PRF},\Phi,qmh} + hq^2/2^l + q/2^l.$$

Again, we note that the storage medium can easily arrange to store several checksums in one physical sector, giving an expansion rate of less than 2. This also reduces the number of extra reads to $h + 1$.

If this construction is used in conjunction with one from Sect. 5.5, we get semantic security and non-malleability against chosen ciphertext attacks by Theorems 3 and 4.

We also note that it is easy to replace the state σ with something that is easier for a user to remember (or write down), such as a date.

As an example, we compute the numbers for providing one gigabyte of storage (2^{33} bits of storage), using 512 byte sectors (2^{12} bits). Let $l = 7$ and $m = 2^5$. We get that $n_1 = 2^{21}$, $n_2 = 2^{16}$, $n_3 = 2^{11}$, $n_4 = 2^6$, $n_5 = 2$ and $n_6 = 1$. The expansion rate is roughly 1.03. Every read operation requires 8 reads from the storage medium, and every write operations requires 7 reads and 8 writes. Caching can potentially reduce the number of physical reads and writes.

6 Concluding Remarks

This paper attempts to define all useful security notions for disk encryption, and determining what is required to achieve those notions.

One common feature of many disk encryption implementations (SISWG's standards work and FreeBSD's GBDE being notable exceptions) is that the documentation says very little about the cryptographic reasoning behind the system, and there is little in the way of useful security analysis. This paper provides a set of formal security notions as well as constructions, which should provide a sound basis for evaluating disk encryption systems.

While the SISWG does an excellent job, providing solutions that are as good as possible under the circumstances, we believe that the restrictions they have imposed on themselves makes it impossible to reach certain worthwhile security notions. Even though the stronger security notions like semantic security and integrity against chosen ciphertext attacks are costly to achieve, it should be up to the users to balance cost against security, not standards.

Also, certain applications require weaker security than some of the solutions considered by SISWG. One example is encryption of swap space. If the disk encryptor is initialized with a random key when the system starts, all that is required to protect the swap space is semantic security against non-adaptive chosen plaintext attacks. Using FreeBSD's GBDE or SISWG's tweakable block ciphers is simply overkill.

References

1. M. Bellare, R. Canetti, and H. Krawczyk. Keying hash functions for message authentication. In Neal Koblitz, editor, *Proceedings of CRYPTO '96*, volume 1109 of *LNCS*, pages 1–15. Springer-Verlag, 1996.
2. M. Bellare, A. Desai, E. Jokipii, and P. Rogaway. A concrete security treatment of symmetric encryption. In *FOCS 1997*, pages 394–403, 1997.
3. FIPS 197. *Advanced Encryption Standard.* Federal Information Processing Standards Publication. National Technical Information Service, Springfield, Virginia, November 2001.
4. Shai Halevi and Phillip Rogaway. A tweakable enciphering mode. In Dan Boneh, editor, *Proceedings of CRYPTO 2003*, volume 2729 of *LNCS*, pages 482–499. Springer-Verlag, 2003.
5. J. Hughes. Chair of the IEEE security in storage working group, 2004. `http://www.siswg.org/`.
6. Tetsu Iwata and Kaoru Kurosawa. OMAC: One-key CBC MAC. In Thomas Johansson, editor, *Fast Software Encryption*, volume 2887 of *LNCS*, pages 129–153. Springer-Verlag, 2003.
7. Poul-Henning Kamp. GBDE – GEOM based disk encryption. BSDCON '03, 2003. `http://phk.freebsd.dk/pubs/bsdcon-03.gbde.paper.pdf`.
8. J. Katz and M. Yung. Complete characterization of security notions for probabilistic private-key encryption. In *Proceedings of the 32nd Annual Symposium on Theory of Computing*, pages 245–254. ACM, 2000.

9. Hugo Krawczyk. The order of encryption and authentication for protecting communications (or: How secure is SSL?). In Joe Kilian, editor, *Proceedings of CRYPTO 2001*, volume 2139 of *LNCS*, pages 310–331. Springer-Verlag, 2001.
10. Moses Liskov, Ronald L. Rivest, and David Wagner. Tweakable block ciphers. In Moti Yung, editor, *Proceedings of CRYPTO 2002*, volume 2442 of *LNCS*, pages 31–46. Springer-Verlag, 2002.
11. Jari Ruusu. LoopAES, 2005. http://loop-aes.sourceforge.net/.

A Proofs

A.1 General Results

Proof (Theorem 1). $\mathcal{D}'$ uses one extra sector of storage space. Whenever it receives a write request, it gives the write request to $\mathcal{D}$. If that write returns $\top$, then $\mathcal{D}'$ writes a random value to the extra sector. Whenever it receives a read request, it reads the extra sector, then passes the read request to $\mathcal{D}$.

An adversary that changes the extra sector in an arbitrary fashion and then reads any sector from the disk encryptor will win the ciphertext integrity game.

The extra sector can be simulated by any adversary, so any adversary against plaintext integrity for $\mathcal{D}'$ can be used against $\mathcal{D}$ as well. □

Proof (Theorem 2). If $\mathcal{D}$ provides N plaintext sectors using n ciphertext sectors, $\mathcal{D}'$ will provide N plaintext sectors using $n + N$ ciphertext sectors.

The idea is that $\mathcal{D}'$ keeps an unencrypted copy of the plaintext in the extra N sectors. When reading, this unencrypted copy is ignored. The adversary against semantic security is obvious.

In the game that defines plaintext integrity, the adversary is the only one writing data to the disk encryptor. This means that the extra plaintext copy can easily be simulated, and the theorem follows. □

Proof (Theorem 3). We only consider the chosen ciphertext attack. The adversaries derived from the non-adaptive chosen ciphertext adversary are similar, and we leave them to the reader.

Consider first the chosen ciphertext attack game between the adversary and a simulator. Let E be the event that the bit output by the adversary is correct in this game.

Now we change the game as follows: A copy of anything the disk encryptor writes to the storage medium is kept. If the result of any read request made by the disk encryptor differs from this copy, the disk encryptor's output for the operation is ignored and $\perp$ is returned. Let E' be the event that the bit output by the adversary is correct in this modified game.

We have that

$$\mathrm{Adv}_A^{\mathcal{D},\mathrm{IND,CCA}} = |\Pr[E] - 1/2| \leq |\Pr[E] - \Pr[E']| + |\Pr[E'] - 1/2|.$$

As usual in game hopping, we want to bound $|\Pr[E] - \Pr[E']|$ and $|\Pr[E'] - 1/2|$.

The two games proceed identically until $\perp$ is returned in the modified game, but not in the original game. Let F be the event that this happens. Now we observe that F is the event that leads to success against chosen ciphertext attack.

The adversary A' is then simply A, augmented with parts simulating the attack against semantic security. The success rate of A' against ciphertext integrity is $\Pr[F]$, and as usual we have that

$$|\Pr[E] - \Pr[E']| \leq \Pr[F] = \mathrm{Succ}_{A'}^{\mathcal{D},\mathrm{CTXT},\mathrm{CCA}}.$$

The chosen plaintext adversary A'' encapsulates A. It keeps a copy of the storage medium state. Whenever a read or write request for the storage medium is copied to A'', A'' fakes a corresponding read or write request for A.

When A issues a read or write request to the disk encryptor, A'' first issues the read queries for the simulator. (It can do this because the unknown key is not involved in determining these reads.) If A's answers correspond to the copy kept by A'', A's request is forwarded to the simulator. Otherwise, $\perp$ is returned.

A'' provides A with the same environment as in the modified game. Hence A's success probability in the modified game is equal to the success probability of A''. Therefore,

$$|\Pr[E'] - 1/2| = \mathrm{Adv}_{A''}^{\mathcal{D},\mathrm{IND},\mathrm{CPA}}.$$

This concludes the proof. □

Proof (Theorem 5). First, we note that any disk encryptor with expansion ratio 1 must be deterministic, and no matter what the storage medium contains, no read operation will result in a decryption error.

The chosen plaintext adversary against semantic security simply picks two distinct values $x^{(0)}$ and $x^{(1)}$ from S^N, and writes $x^{(0)}$ to the disk encryptor. It makes a copy of the storage medium state. Then it outputs the probability space X over S^N such that $\Pr[x = x^{(0)} \mid x \overset{r}{\leftarrow} X] = \Pr[x = x^{(1)} \mid x \overset{r}{\leftarrow} X] = 1/2$ and $f(x^{(b)}) = b$ for $b \in \{0, 1\}$.

After the simulator has written $x^{(b)}$ to the disk encryptor, the adversary compares the contents of the storage medium with its copy. If it matches, 0 is output, otherwise 1 is output. This adversary has advantage 1/2.

The adversary against non-malleability writes both $x^{(0)}$ and $x^{(1)}$ to the disk encryptor, saving the storage medium states. It then outputs the probability space X. When the simulator writes $x^{(b)}$ to the disk encryptor, it replaces the storage medium contents with the one corresponding to $x^{(1-b)}$, and outputs the relation $R = ((x^{(0)}, x^{(1)}), (x^{(1)}, x^{(0)}))$. This adversary has success rate 1.

Finally, the adversary against plaintext integrity writes arbitrary data to the disk encryptor. Then it makes some arbitrary change to the storage medium, and reads back from the disk encryptor. This adversary has success rate 1. □

Proof (Theorem 6). The idea is that since the disk encryptor is stateless, if a state for the storage medium results in a valid read once, that state will always result in a valid read.

The adversary plays the role of the storage medium for the simulator. He chooses (arbitrarily) two different values $x^{(0)}$ and $x^{(1)}$ from S, and outputs the probability space X on S satisfying $\Pr[X = x^{(0)}] = \Pr[X = x^{(1)}] = 1/2$, the set $I = \{0\}$, and a function $f : S \to \{0, 1\}$ such that $f(x^{(j)}) = j$.

The simulator samples b and writes $x^{(b)}$ to the disk encryptor. At this point, reading from the disk encryptor will return $x^{(b)}$.

The adversary now saves the storage medium state. Then he writes $x^{(0)}$ to the disk encryptor. Since he has written to this sector, the simulator will allow him to read it again. The adversary restores the saved storage medium state, and reads $x^{(b)}$ from the disk encryptor. The adversary now knows b. □

A.2 Concrete Constructions

Proof (Theorem 7). We play two games. In the first game, we sample f_k from Φ and run the disk encryptor as specified above. Then we simulate a non-adaptive chosen plaintext attack on $\mathcal{D}_1$ for A. Let E be the event that the bit output by A is correct.

In the second game, we sample f from $\mathrm{Map}(\{0, 1\}^l, \{0, 1\}^l)$ and use it instead of f_k in the disk encryptor. Then we simulate a non-adaptive chosen plaintext attack on $\mathcal{D}_1$ for A, exactly as in the first game. If E' is the event that the bit output by A is correct, we must have that $\Pr[E'] = 1/2$, since the function values $r(f, i)$ will be independent and uniformly random.

Let A' be the distinguisher that computes the function r by requesting function values for the points $i2^{l_0} + j$, $0 \le j < m$. Then it simulates a non-adaptive chosen plaintext attack on $\mathcal{D}_1$ for A. If A guesses correctly, the simulator outputs 0, otherwise 1.

If the function A' tries to distinguish was sampled from Φ, everything proceeds as in the first game. The probability that A' correctly answers 0 is $\Pr[E]$. Otherwise, everything proceeds as in the second game, and the probability that A' correctly answers 1 is $\Pr[E'] = 1/2$.

We get that

$$\begin{aligned}\mathrm{Adv}_{A'}^{\mathrm{PRF},\Phi,Nm} &= |\Pr[E]/2 + \Pr[E']/2 - 1/2| \\ &= \frac{1}{2}|\Pr[E] - 1/2| = \frac{1}{2}\mathrm{Adv}_{A}^{\mathcal{D},\mathrm{IND,naCPA}},\end{aligned}$$

which concludes the proof. □

Proof (Theorem 8). We play two games. The first game is the unmodified attack game. In the second game, instead of sampling f_k from $\tilde{\Pi}$, we sample f from $\mathrm{Perm}_T(S)$ and use it instead of f_k.

In the second game, two different sectors are encrypted with independent permutations, so every ciphertext sector is independent of the plaintext sector, and any change in the storage medium will induce a random change in the corresponding plaintext sector.

Therefore, the adversary cannot have any advantage against semantic security or success rate against non-malleability in the second game. The theorem follows after a few simple calculations. □

Proof (Theorem 9). Again, we play two games: One where a function is sampled from Φ, and one where it is sampled from $\mathrm{Map}(T \times S, \{0,1\}^l)$.

To succeed in the latter game, the adversary A has to find a value s, an index i, as well as the function value $f_k(i,s)$. When f_k is replaced by a random function, then for any $s' \in S$ we have that $f(i,s) = s'$ with probability 2^{-l}. It has at most N chances of getting at least one sector right, giving a success probability of at most $N/2^l$. This concludes the proof. □

Proof (Theorem 10). As usual, we play two games: One where a function is sampled from Φ, and one where it is sampled from $\mathrm{Map}(\{0,1\}^l, \{0,1\}^l)$.

In the latter game, unless the same j is sampled for two different write operations, the adversary has no advantage. The probability that one j is sampled at least twice is at most $q^2/2^{l-l_0}$, which concludes the proof. □

Proof (Theorem 11). As usual, we play two games: One where a function is sampled from Φ, and one where it is sampled from $\mathrm{Map}(T \times S, \{0,1\}^l)$. The first game proceeds exactly as in a real attack.

In the second game, we note that the value of every checksum sector is written using different values from T. So all checksums will be independent. Also, the master checksum denoted by $s^{(h+1)}$ will never be written twice using the same value from T.

Every time the adversary attempts a read after making a change to the checksum tree such that $s_0^{(h)}$ changes, the probability that the read succeeds is $1/2^l$.

The only way the adversary can change the checksum tree without $s_0^{(h)}$ changing, is by finding a checksum collision. This means that the adversary's success rate in the second game is at most $q^2/2^l + q/2^l$. The theorem follows. □