Space-Efficient Block Storage Integrity

Alina Oprea* Michael K. Reiter† Ke Yang‡

Abstract

We present new methods to provide block-level integrity in encrypted storage systems, i.e., so that a client will detect the modification of data blocks by an untrusted storage server. We present cryptographic definitions for this setting, and develop solutions that change neither the block size nor the number of sectors accessed, an important consideration for modern storage systems. In order to achieve this, a trusted client component maintains state with which it can authenticate blocks returned by the storage server, and we explore techniques for minimizing the size of this state. We demonstrate a scheme that provably implements basic block integrity (informally, that any block accepted was previously written), that exhibits a tradeoff between the level of security and the additional client's storage overhead, and that in empirical evaluations requires an average of only 0.01 bytes per 1024-byte block. We extend this to a scheme that implements integrity resistant to replay attacks (informally, that any block accepted was the last block written to that address) using only 1.82 bytes per block, on average, in our one-month long empirical tests.

1. Introduction

Modern network attached storage (NAS) and storage area network (SAN) architectures provide remote block-level data storage services for clients, essentially providing the same interface as a local disk would to the client file system. Particularly in the case of a SAN, this service is often owned and managed by an organization other than the client's, and it may additionally store other client organizations' data using the same physical resources. In such an environment, it is prudent for each client to treat the storage service as untrusted, and to take measures before transmitting blocks to the storage service to protect the privacy and integrity of these blocks.

For this purpose, two years ago the IEEE Security In Storage Working Group (SISW) [15] announced a call for algorithms for block level encryption. Among the requirements was that the new encryption algorithm be length preserving, so that block boundaries do not shift or need to be adjusted as a result of encryption. This call led to the design of length-preserving encryption algorithms (e.g., [13, 14]) that are now being considered for standardization.

In this paper, we address the storage integrity problem in this context. Due to the length-preserving requirements for cryptographic operations on blocks, it is not possible to add information to each block (e.g., a MAC) in order to detect its modification, a fact explicitly noted in the SISW requirements. Moreover, due to the performance demands of I/O intensive applications, it would be undesirable to put these MACs in separate blocks also stored at the service, which would require the retrieval of two blocks (one of data, one of MACs) on the critical path of client read operations. Therefore, here we consider a strategy in which a trusted client component—presumably the same one that holds keys for encrypting blocks before their transmission to the storage service, and for decrypting blocks upon their retrieval—holds this integrity information. Among our primary goals is to minimize the size of this integrity information, since for a client with large storage needs, retaining, e.g., a MAC per block would itself require significant storage and resulting overheads.

In this context, we present new, storage-efficient constructions for two definitions of storage integrity. One of the definitions, based on a similar definition for authenticated encryption [4], formally expresses the notion that if the client returns a block B in response to a read request for address a, then the client previously wrote B to a. The second definition is stronger by providing defense against "replay attacks"; informally, it expresses the notion that if the client returns a block B in response to a read request for address a, then B is the content most recently written to address a.

Our constructions are novel in exploiting the fact that

*Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA; alina@cs.cmu.edu

†Department of Electrical & Computer Engineering, Department of Computer Science, and CyLab, Carnegie Mellon University, Pittsburgh, PA, USA; reiter@cmu.edu

‡Google Inc. Mountain View, CA, USA; yangke@google.com

distributions of block contents and of block access patterns are not random in practice, and by doing so they minimize the storage required at the client. We confirm through a month-long empirical evaluation in a Linux environment that we accomplish this goal. For example, our scheme satisfying our weaker notion of integrity achieves client storage overhead of less than 0.01 bytes per block on average (compared to 16-20 bytes per block for the scheme in which a hash or MAC is stored for each block), assuming a block size of 1024 bytes. Our scheme defending against replay attacks is more expensive in terms of storage, but still cheaper than hashing each block: it requires 1.82 bytes per block on average for 1024-byte blocks.

The rest of the paper is structured as follows. We discuss related work in authenticated encryption and storage security in Section 2. We review necessary definitions in Section 3. We define our system model in Section 4. Our new integrity definitions are given in Section 5 and our constructions are described and evaluated in Sections 6 and 7, respectively.

2. Related Work

Encryption algorithms for secure storage have received much attention in recent years, leading to the development of *tweakable* block ciphers [21]. In addition to a key and a plaintext, a tweakable block cipher has as input a *tweak*, for variability. In the secure storage model, a tweak might be the address of a disk block or the block identifier. This notion has been extended to that of tweakable enciphering schemes [13] that operate on larger plaintexts (e.g., 512 or 1024 bytes), and a new *CMC* encryption mode, about twice as expensive as CBC, has been designed for this purpose. Recently, a parallelizable tweakable enciphering scheme was proposed with similar serial efficiency as CMC [14]. These schemes, being length-preserving, provide good solutions to disk block encryption and are currently being considered for standardization by the storage community.

Adopting one of these tweakable encryption schemes for confidentiality, our goal is to augment it to provide efficient integrity for the storage scenario. Therefore, there are two main orthogonal fields related to our work: authenticated encryption and storage security.

Authenticated encryption (e.g., [4, 17, 19]) is a primitive that provides privacy and message authenticity at the same time. That is, in addition to providing some notion of encryption scheme privacy (e.g., [2]), authenticated encryption ensures either integrity of plaintexts or integrity of ciphertexts. The traditional approach for constructing authenticated encryption is by generic composition, i.e., the combination of a secure encryption scheme and an unforgeable message authentication code (MAC). However, Bellare and Namprempre [4] analyze the security of the composition and provide proofs that some of the widely believed secure compositions are actually insecure. Krawczyk [19] proves that the generic composition method used in the Secure Socket Layer (SSL) protocol is insecure, but the particular standard implementation is secure with respect to both privacy and integrity. The authenticated encryption in SSH is also insecure, as demonstrated by Bellare et al. [3]. There, a new definition for integrity is given, that protects against replay and out-of-order delivery attacks; Kohno et al. [18] also supply such definitions. While we also define integrity against replay attacks, our definitions are particularly tailored to the storage scenario, and are thus different from the network case.

A different approach to obtain integrity is to add redundancy to plaintexts. Bellare and Rogaway [5] and Ann and Bellare [1] give necessary and sufficient conditions for the redundancy code such that the composition of the encryption scheme and the redundancy code provide integrity. Our constructions exploit a similar principle in a different way, leveraging the redundancy inherent in typical disk writes to achieve integrity rather than adding redundancy to do so (which is not permitted in the storage case).

In the area of storage system security, to our knowledge all systems that verify the authenticity of retrieved blocks store block data integrity information on the server, in contrast to the scenario we study here. For example, TCFS [8], ECFS [6] (both extensions of CFS [7]), NASD [11] and SNAD [23] each store a hash or a keyed hash for each block, which increases either the size of each block or the number of blocks that must be written per write operation. Cepheus [9] and SUNDR [20] keep for each file the root of a Merkle hash tree with leaves the hashes of the corresponding data blocks. Sirius [12] stores a digital signature for each file. In such systems, there is a tradeoff between the amount of server-side storage of integrity information and the access time to read and write files: e.g., if the root of a Merkle hash tree is the only information stored at the server, then each read and write involves $O(\log n)$ block accesses, with n the number of blocks in the file. Similarly, Sirius retrieves all blocks in a file in order to check the file's digital signature (and hence the authenticity of any block). In our schemes, we take a different approach, in which we do not increase server-side storage or block accesses, and strive to minimize client-side storage to the extent possible.

Riedel et al. [25] provide a framework to evaluate existing storage systems from both the security and performance perspective. We refer the reader to this paper for

an extensive comparison of the security properties of the storage systems considered.

3. Preliminaries

3.1. Tweakable Enciphering Schemes

In this section, we review the definitions and security notions for tweakable enciphering schemes [13]. An enciphering scheme is a strong, length-preserving pseudorandom permutation. A tweakable enciphering scheme is a function of a tweak and has the property that for a fixed tweak, it is an enciphering scheme. More formally, a tweakable enciphering scheme is a function $E : \mathcal{K} \times \mathcal{T} \times \mathcal{M} \rightarrow \mathcal{M}$, where $\mathcal{K}$ is the key space, $\mathcal{T}$ is the tweak set, $\mathcal{M}$ is the plaintext space (strings of length l bits), and for every $K \in \mathcal{K}, \mathcal{T} \in \mathcal{T}$, $E(K, T, \cdot) = E_K^T(\cdot)$ is a length preserving permutation. The inverse of the enciphering scheme E is the enciphering scheme $D : \mathcal{K} \times \mathcal{T} \times \mathcal{M} \rightarrow \mathcal{M}$, where $X = D_K^T(Y)$ if and only if $E_K^T(X) = Y$.

We define $\mathsf{Perm}(\mathcal{M})$ the set of all permutations $\pi : \mathcal{M} \rightarrow \mathcal{M}$ and $\mathsf{Perm}^{\mathcal{T}}(\mathcal{M})$ the set of all functions $\pi : \mathcal{T} \times \mathcal{M} \rightarrow \mathcal{M}$ such that $\pi(t) \in \mathsf{Perm}(\mathcal{M})$ for any $t \in \mathcal{T}$. For a function $\pi : \mathcal{T} \times \mathcal{M} \rightarrow \mathcal{M}$, we define $\pi^{-1} : \mathcal{T} \times \mathcal{M} \rightarrow \mathcal{M}$ such that $\pi^{-1}(T, y) = x$ if and only if $\pi(T, x) = y$.

Definition 3.1 *Let $E : \mathcal{K} \times \mathcal{T} \times \mathcal{M} \rightarrow \mathcal{M}$ be a tweakable enciphering scheme and $\mathcal{A}$ an adversary. $\mathcal{A}$ has access to oracles $E_K(\cdot, \cdot)$ and $D_K(\cdot, \cdot)$ that take as input a tweak and a plaintext, and a tweak and a ciphertext, respectively. The PRP-advantage of adversary $\mathcal{A}$ is defined as:*

$$\mathsf{Adv}_E^{\mathsf{prp}}(\mathcal{A}) = \Pr[K \xleftarrow{R} \mathcal{K} : \mathcal{A}^{E_K, D_K} = 1] - \Pr[\pi \xleftarrow{R} \mathsf{Perm}^{\mathcal{T}}(\mathcal{M}) : \mathcal{A}^{\pi, \pi^{-1}} = 1]$$

In the rest of the paper, we denote by $\mathsf{Adv}_E^{\mathsf{prp}}(q_1, q_2)$ the maximum over all polynomial time adversaries $\mathcal{A}$, that make q_1 queries to E_K and q_2 queries to D_K, of $\mathsf{Adv}_E^{\mathsf{prp}}(\mathcal{A})$. We omit other resources, such as time, from the advantages. The definition of PRP-security is a natural extension of the strong pseudorandom permutation definition from [24].

3.2. Second Preimage Resistant Hash Functions

Let $h : \mathcal{M} \rightarrow \{0,1\}^s$ be an unkeyed hash function. Intuitively, second preimage resistantance requires that given a message $m \in \mathcal{M}$, it is hard to find a collision, i.e., $m' \neq m$ such that $h(m') = h(m)$. More formally:

Definition 3.2 *For an adversary algorithm $\mathcal{A}$, we define the advantage of $\mathcal{A}$ in breaking the second preimage resistance of hash function h as:*

$$\mathsf{Adv}_h^{\mathsf{spr}}(\mathcal{A}) = \Pr[m \xleftarrow{R} \mathcal{M}, m' \leftarrow \mathcal{A}(m) : (m \neq m') \wedge h(m') = h(m)]$$

$\mathsf{Adv}_h^{\mathsf{spr}}$ denotes the maximum advantage $\mathsf{Adv}_h^{\mathsf{spr}}(\mathcal{A})$ for all polynomial-time adversaries $\mathcal{A}$.

4. System Model

We consider a limited storage client that keeps its data on an untrusted storage device, denoted by "storage server". The data is partitioned into fixed-length sectors or blocks. The client can perform two basic operations: read a block from a physical address (or *block identifier*) and write a block to a certain address on the server.

In our model, the server can behave maliciously, by mounting attacks against the confidentiality and integrity of the client's data. We assume that the server is available, i.e., it responds to client's read and write queries. However, no guarantees are given about the correctness of its replies.

For data confidentiality, we assume that blocks are encrypted by clients using a tweakable enciphering scheme, in which tweaks are functions of block identifiers. This ensures that the encryptions of two different blocks with identical content are different.

The client is responsible for protecting its data integrity from malicious server behavior by keeping additional integrity information. Our goal is to design schemes that minimize the client storage and provide provable integrity.

We define a storage scheme to be a tuple of algorithms $S = (\mathsf{INIT}, E, D, \mathsf{WRITE}, \mathsf{READ}, \mathsf{VER})$ where E is a tweakable enciphering scheme, D is its inverse, and where:

1. The initialization algorithm $\mathsf{INIT}()$ outputs a secret key K for the client for the encryption scheme E;
2. The write algorithm $\mathsf{WRITE}(K, m, bid)$ takes as input the secret key generated by the INIT algorithm, block content m and block identifier bid. The client first encrypts the block m with a tweak T derived from bid under the enciphering scheme E and then sends the resulting ciphertext, $c = E_K^T(m)$ and bid to the server.
3. When performing a $\mathsf{READ}(K, bid)$ operation, the client gets from the server the ciphertext c of block bid which should be the last ciphertext written by the client with that particular block identifier. The client decrypts c with tweak T generated from bid and outputs the corresponding plaintext $m = D_K^T(c)$. We denote the read operation by $m \leftarrow \mathsf{READ}(K, bid)$.
4. The verification algorithm $\mathsf{VER}(m, bid)$ is given block content m and block identifier bid. It checks m's integrity, and outputs 1 if it is valid, and 0, otherwise. Note that VER is not a keyed function.

$\mathsf{Exp}^{\mathsf{int\text{-}st}}_{S,\mathcal{A}_1}()$:
$K \leftarrow \mathsf{INIT}()$;
$\mathcal{A}_1$ adaptively queries $E_K(\cdot,\cdot)$ and $D_K(\cdot,\cdot)$, and replies to client's queries.
If $\mathcal{A}_1$ replies to a $\mathsf{READ}(K, bid)$ client query with ciphertext c such that, if $m = D^T_K(c)$ with T generated from bid, then:
1. $\mathsf{VER}(m, bid)$ returns 1
2. c was never sent by the client in a $\mathsf{WRITE}(K, \cdot, bid)$ query
3. $\mathcal{A}_1$ did not query $E_K(T, m)$

then return 1, else return 0.

$\mathsf{Exp}^{\mathsf{int\text{-}st\text{-}rep}}_{S,\mathcal{A}_2}()$:
$K \leftarrow \mathsf{INIT}()$;
$\mathcal{A}_2$ adaptively queries $E_K(\cdot,\cdot)$ and $D_K(\cdot,\cdot)$ and replies to client's queries.
If $\mathcal{A}_2$ replies to a $\mathsf{READ}(K, bid)$ client query with ciphertext c such that, if $m = D^T_K(c)$ with T generated from bid, then:
1. $\mathsf{VER}(m, bid)$ returns 1
2. c was not sent by the client in the **most recent** $\mathsf{WRITE}(K, \cdot, bid)$ query
3. $\mathcal{A}_2$ did not query $E_K(T, m)$

then return 1, else return 0.

Figure 1. Experiments for Defining Storage Integrity

5. Notions of Integrity for Storage Schemes

In defining integrity for storage schemes, we consider polynomial time adversaries $\mathcal{A}_1$ and $\mathcal{A}_2$ with access to two oracles: an enciphering oracle $E_K(\cdot,\cdot)$ and a deciphering oracle $D_K(\cdot,\cdot)$. The enciphering oracle returns the ciphertext corresponding to a tweak and a block. The deciphering oracle returns the plaintext corresponding to a tweak and a ciphertext. Adversaries $\mathcal{A}_1$ and $\mathcal{A}_2$ play the server's role in our model. They also accept READ and WRITE queries from clients.

Intuitively, an adversary for a storage scheme wins if it tricks the client into accepting a block that he never wrote at a particular address. This is the first notion of integrity that we define, and it is a straightforward generalization of the notions of integrity for symmetric encryption schemes from [4]. A replay attack is one in which the server returns an old version of a block (not the last block written by the client at the block address), and the client accepts it as valid. Our second notion of integrity incorporates defense against replay attack, being stronger than the first one.

To formalize our intuition, we define the two experiments from Figure 1.

We define the advantages of the adversaries in attacking the integrity of the scheme as:
$\mathsf{Adv}^{\mathsf{int\text{-}st}}_S(\mathcal{A}_1) = \Pr[\mathsf{Exp}^{\mathsf{int\text{-}st}}_{S,\mathcal{A}_1}() = 1]$
$\mathsf{Adv}^{\mathsf{int\text{-}st\text{-}rep}}_S(\mathcal{A}_2) = \Pr[\mathsf{Exp}^{\mathsf{int\text{-}st\text{-}rep}}_{S,\mathcal{A}_2}() = 1]$
The two notions of integrity require different correctness properties:

1. int-st: If the client performs $\mathsf{WRITE}(K, m, bid)$, then block m is accepted as valid, i.e., $\mathsf{VER}(m, bid) = 1$.

2. int-st-rep: If $\mathsf{WRITE}(K, m, bid)$ is the most recent write operation to block bid, then block content m is accepted as valid, i.e., $\mathsf{VER}(m, bid) = 1$.

6. Constructions of Storage Schemes

In this section, we first describe a very simple int-st-rep secure construction, which is similar to constructions from [8, 20]. We include this basic scheme here to compare its client-side storage and performance to the more sophisticated schemes described next. In the second part of this section, we give a new, space-efficient int-st construction.

6.1. int-st-rep Simple Construction

The construction we give here is very simple: for each block written at a particular address, the client computes and stores the block identifier and a hash of the block. For a given address, the stored hash corresponds to the last written block, thus preventing the adversary in succeeding with a replay attack. The amount of additional storage kept by the client is linear in the number of blocks written to the server, i.e., 20 bytes per block if a cryptographically secure hash function such as SHA-1 is used plus the block identifiers (2 or 4 bytes, depending on the implementation).

In order to fully specify the scheme, we need a tweakable enciphering scheme E and a second preimage resistant hash function on the plaintext space $\mathcal{M}$ of E, $h : \mathcal{M} \rightarrow \{0,1\}^s$. The client keeps a list L of pairs (tweak, block hash), that is initialized to the empty set. The scheme $S_1 = (\mathsf{INIT}, E, D, \mathsf{WRITE}, \mathsf{READ}, \mathsf{VER})$ is detailed in Figure 2. The proof of the following proposition is given in Appendix A.

Proposition 6.1 *If h is a second preimage resistant hash function, then the storage scheme S_1 is* int-st-rep *secure:* $\mathsf{Adv}^{\mathsf{int\text{-}st\text{-}rep}}_{S_1}(q_1, q_2) \leq \mathsf{Adv}^{\mathsf{spr}}_h$

6.2. New Efficient int-st Construction

We design a new, storage-efficient construction to obtain int-st integrity. Our construction is based on two observations. The first one is that blocks written to disk

INIT()	WRITE(K, m, bid)	READ(K, bid)	VER(m, bid)
$K \xleftarrow{R} \mathcal{K}$	remove $(bid, *)$ from L	receive c from server	if $(bid, h(m)) \in L$
$L \leftarrow \emptyset$	insert $(bid, h(m))$ into L	$T \leftarrow bid$	output 1
	$T \leftarrow bid$	output $m = D_K^T(c)$	else
	send $(bid, c = E_K^T(m))$ to server		output 0

Figure 2. Scheme S_1

do not look random in practice; in fact they have very low entropy. And, secondly, if an adversary tries to modify ciphertexts encrypted with a tweakable enciphering scheme, the resulting plaintext looks random with very high probability. The second property derives immediately from the prp-security of a tweakable enciphering scheme defined in Section 3.1.

In our construction, we need a statistical test IsRand that can distinguish uniformly random blocks from non-random ones. More explicitly, $\texttt{IsRand}(M), M \in \mathcal{M}$ returns 1 with high probability if M is a uniformly random block in $\mathcal{M}$, and 0 otherwise. Of course, the statistical test is not perfect. It is characterized by the false negative rate α, defined as the probability that a uniformly random element is considered not random by the test. In designing such a statistical test, the goal is to have very small false negative rate. We will discuss more in Section 6.2.2 about a particular instantiation for IsRand.

The idea of our new construction is very intuitive: before encrypting a block M, the client computes $\texttt{IsRand}(M)$. If this returns 1, then the client keeps a hash of that block for authenticity. Otherwise, the client stores nothing, as the low entropy of the block will be used to verify its integrity upon return. The block is then encrypted with the tweak equal to the block identifier and sent over to the server. When reading a ciphertext from an address, the client first decrypts it to obtain a plaintext M and then computes $\texttt{IsRand}(M)$. If $\texttt{IsRand}(M) = 1$ and its hash is not stored in the hash list, then the client knows that the server has tampered with the ciphertext. Otherwise, the block is authentic. The new construction S_2 is detailed in Figure 3.

6.2.1 The Integrity of the Construction

We give the following theorem that guarantees the int-st integrity of S_2, whose proof is deferred to Appendix B.

Theorem 6.2 *If E is a PRP-secure tweakable enciphering scheme, h is a second preimage resistant hash function and α (the false negative rate of* IsRand*) is small, then S_2 is* int-st *secure:*

$$\mathsf{Adv}_{S_2}^{\text{int-st}}(q_1, q_2) \leq \mathsf{Adv}_E^{\text{prp}}(q_1, q_2) + \mathsf{Adv}_h^{\text{spr}} + \frac{(q_2+1)\alpha 2^l}{2^l - q_1}$$

Write M as $M = M_1 M_2 \dots M_n$ with $M_i \in \{1, 2, \dots, b\}$
Compute $p_i =$ the frequency of symbol i in M, $i = 1, \dots, b$
Compute $H = -\sum_{i=1}^{b} p_i \log_2(p_i)$
If $H < \tau$, then return 0
Else return 1

Figure 4. $\texttt{IsRand}_{b,\tau}(M)$

6.2.2 The Entropy Statistical Test

In this section we give an example of a statistical test IsRand that can distinguish between random and non-random blocks. $\texttt{IsRand}(M), M \in \mathcal{M}$ returns 1 with high probability if M is a uniformly random block in $\mathcal{M}$, and 0 otherwise. Consider a block M divided into n parts of fixed length $M = M_1 M_2 \dots M_n$ with $M_i \in \{1, 2, \dots, b\}$. For example, a 1024-byte block could be either divided into 1024 8-bit parts (for $b = 256$), or alternatively into 2048 4-bit parts (for $b = 16$). The empirical entropy of M is defined as $H = -\sum_{i=1}^{b} p_i \log_2(p_i)$, where p_i is the frequency of symbol i in the sequence $M_1, \dots, M_n$.

If we fix τ a threshold, depending on n and b (we will discuss later how to choose τ), then the entropy test parameterized by b and τ is defined in Figure 4. In the following, we denote $\texttt{IsRand}_{8,\tau}(\cdot)$ by the 8-bit entropy test and $\texttt{IsRand}_{4,\tau}(\cdot)$ by the 4-bit entropy test.

Analysis. We give an analytical bound for the false negative rate, as a function of n, b and τ.

Theorem 6.3 *For a given threshold τ, if we denote δ the solution of equation (1):*

$$\tau = -(1-\delta)\log_2\left(\frac{1-\delta}{b}\right) \quad (1)$$

then the false negative rate α of $\texttt{IsRand}_{b,\tau}(\cdot)$ *satisfies:*

$$\alpha \leq b e^{-\frac{n}{2b}\delta^2} + b\left(\frac{1}{e}\right)^{\frac{n}{b}}\left(\frac{4e}{b}\right)^{\frac{n}{4}} \quad (2)$$

The proof of the theorem is detailed in Appendix C.

Numerical Interpretation. We performed an analysis for both the 8-bit and 4-bit entropy tests. We determined the threshold τ experimentally: we generated 100,000

INIT()	WRITE(K, m, bid):	READ(K, bid):	VER(m, bid)
$K \xleftarrow{R} \mathcal{K}$ $L \leftarrow \emptyset$	remove $(bid, *)$ from L if $\texttt{IsRand}(m) = 1$ insert $(bid, h(m))$ into L $T \leftarrow bid$ send $(bid, c = E_K^T(m))$ to server	receive c from server $T \leftarrow bid$ output $m = D_K^T(c)$	if $\texttt{IsRand}(m) = 0$ output 1 else if $(bid, h(m)) \in L$ output 1 else output 0

Figure 3. Scheme S_2

b	τ	δ	n	α
256	7.7	0.05	2^{24}	e^{-80}
256	7.7	0.05	2^{25}	e^{-160}
16	2.53	0.5	10000	e^{-80}
16	2.53	0.5	20000	e^{-160}

Figure 5. Relations Among the Parameters of the Entropy Test

uniformly random 1024-byte blocks and computed, for each, its entropy. For the 8-bit test, the range of the entropy was 7.73-7.86 and for the 4-bit test, 2.55-2.64. We picked τ smaller than the minimum entropy of all the random blocks generated. This way, we ensure that uniformly random blocks have the entropy greater than τ with high probability.

Having set τ, we determine δ from (1). We also set the false negative rate desired and from the bound (2), we compute n, the number of parts that we need to ensure this false negative rate. The results are in Figure 5. The 4-bit entropy test performs better, in the sense that it requires smaller n for getting the same α. The results demonstrate, that, in theory, we could get false negative rates as low as needed, at the expense of increasing the block length and of modifying the parameter b of the entropy test.

6.3. Performance Evaluation on Disk Traces

In order to project the behavior of S_1 and S_2 in practice, we collected approximately 200 MB of block disk traces from a SuSe Linux environment. They represent a sampling of disk activity of one of the authors during one month. The applications used most frequently were: Netscape browser and e-mail client, G++ compiler, Java compiler from Sun, XMMS for playing audio, image viewer GIMP, text editor Kedit, LATEX compiler, Acrobat and GV viewers, and VNC server. The reasons we collected traces only from one computer are two-fold: first, getting the traces proved to be cumbersome, as it required non-trivial kernel modification and the installation of a new drive to store them. Secondly, collecting disk traces from users has strong implications for users' privacy.

The block sizes used by the disk interface were 1024 and 4096 bytes. For our experiments, we further divided the blocks into different sizes: 4096, 2048, 1024, 512, 256, 128, 64 and 32 bytes to test the applicability of the integrity schemes proposed in this paper. We performed three different types of experiments:

Entropy Plots We first plot (Figures 6 and 7) the 8-bit entropy of 1024-byte random blocks, compared to 1024-byte trace blocks. The entropy of random blocks is highly concentrated, all the values being between 7.73 and 7.86. In contrast, the entropy of trace blocks is largely varying, between 0.01 and 7.99. Only less than 2 percent of trace blocks have an entropy larger than 7.73, the minimum entropy of a random block.

Storage Plots For each of the eight block sizes, we performed experiments with the 8-bit and 4-bit entropy tests. In Figure 8, we plot the average (i.e, per block) client storage for each of these tests, when we instantiate the collision resistant hash function with SHA-1, whose output length is 20 bytes. For scheme S_1 (MAC scheme), the client keeps 20 bytes per block. For the 8-bit and 4-bit entropy schemes, the average storage decreases as a function of block length. This is intuitive in the sense that, as a data block gets larger, its entropy value has less chance of approaching that of a random block.

For the 4-bit entropy test we can obtain a better theoretical bound for the false negative rate than for the 8-bit test. But, in practice, the average storage that the client keeps for the 4-bit test is larger for block sizes less than 256. Fortunately, both tests perform very well for large values of the block size (e.g., 1024), which are very common in practice. The 4-bit entropy test performs actually very well for 4096-byte blocks, both theoretically and experimentally: theoretically, the false negative rate α is around $e^{-64} \approx 2^{-90}$ and the client storage is 0.0094 bytes per block, on average.

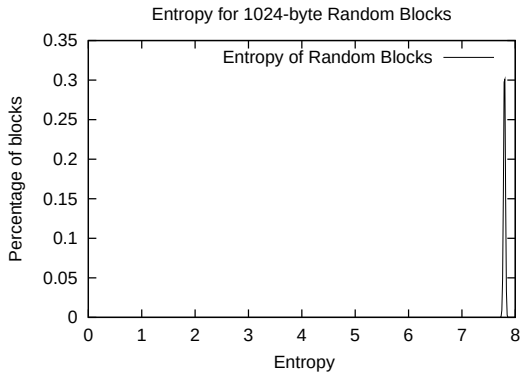

Figure 6. Entropy of 1024-byte Random Blocks

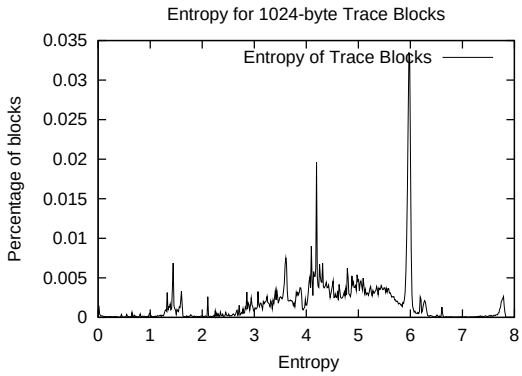

Figure 7. Entropy of 1024-byte Trace Blocks

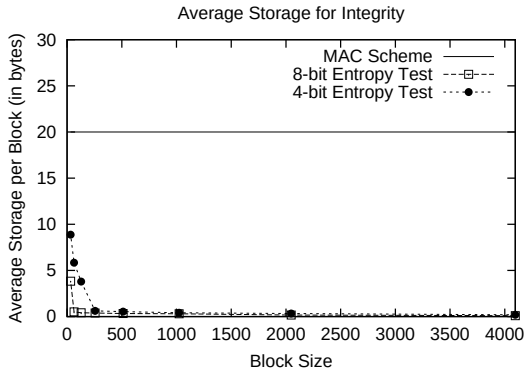

Figure 8. Average Client Storage for Integrity

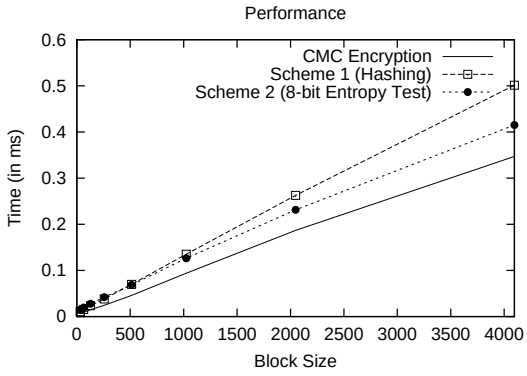

Figure 9. Performance Time for Different Storage Schemes

Performance Plots Lastly, but very importantly, we measured the average time to encrypt the collected blocks using the CMC-AES tweakable enciphering mode described in [13]. We also implemented the two schemes for integrity and compared their overhead with that of the CMC encryption.

The results in Figure 9 show that the overhead due to hashing is 44% more than encryption alone, while the overhead for the entropy test is 19% for 4096-byte blocks. For S_1, we have used the hash function SHA-1. Our entropy test is about twice as fast as SHA-1. As expected, the measured times rise linearly with the block size.

From the experiments, the advantages and disadvantages of the two integrity schemes are clear. Scheme S_1 provides integrity against the replay attack, at the expense of high storage cost on the client and increased client computation time. On the other hand, our second scheme S_2 is very efficient in both computation time and additional storage on the client. This comes at the expense of guaranteeing only a weaker notion of integrity, namely one that permits replays (and leaves them to be dealt with at a higher layer).

7. Is There A More Efficient Solution for Preventing Replay Attacks?

We have analyzed two schemes, one that keeps a hash for each block and defends against the replay attack; and the second one more efficient that only satisfies a weaker notion of integrity. The natural question that comes into mind is whether there are other schemes that prevent replay attacks more efficiently than storing a hash for each block. In this section, we answer this question affirmatively.

First we give a simple example to demonstrate that scheme S_2 is vulnerable to replay attacks. Consider a scenario in which the client writes two messages m_1 and m_2 to block bid in that order, both messages having low

INIT()	WRITE(K, m, bid):	READ(K, bid):	VER(m, bid)

$K \xleftarrow{R} \mathcal{K}$
$L_R \leftarrow \emptyset$
$L_C \leftarrow \emptyset$
$F(bid) \leftarrow 0, \forall bid$
	if $F(bid) = 1$
	  if $(bid, w) \in L_C$
	  remove (bid, w) from L_C
	  $w \leftarrow w + 1$
	  else
	  $w \leftarrow 2$
	  insert (bid, w) into L_C
	else
	  $w \leftarrow 1$
	  $F(bid) \leftarrow 1$
	$T \leftarrow bid \| w$
	if IsRand$(m) = 1$
	  insert $(bid, h(m))$ into L_R
	send $(bid, c = E_K^T(m))$ to server
		if $(bid, w) \in L_C$
		  $T \leftarrow bid \| w$
		else
		  if $F(bid) = 1$
		  $T \leftarrow bid \| 1$
		  else
		  $T \leftarrow bid \| 0$
		receive c from server
		output $m = D_K^T(c)$
			if IsRand$(m) = 0$
			  output 1
			else
			  if $(bid, h(m)) \in L_R$
			  output 1
			  else
			  output 0

Figure 10. Scheme S_3

entropy. Later, when the client performs a read on bid, imagine that the server replies with m_1 instead of the legitimate more recent written m_2. Because m_1 has low entropy (i.e., IsRand$(m_1) = 0$), the client accepts it as valid. This demonstrates that the server succeeds in a replay attack.

The solution that we propose here stems from the observation that the block access distribution in practice is not uniformly random, in fact it follows a Zipf-like distribution. More specifically, there are few blocks that are written more than once, with the majority of the blocks being written just once. If all the blocks were written only once, then scheme S_2 would suffice to defend against replays, as well. If the blocks were written uniformly, then scheme S_1 could be used. The solution we give here is a hybrid scheme, that combines the previous two constructions.

Briefly, the solution is to keep a counter for each block that is written more than once. The counter denotes the number of writes to a particular block. We also keep a flag (one bit for each block) that is initially set to 0 for all blocks and becomes 1 when the block is written first. We make the observation that we do not need to store counters for blocks that are written once or not written at all, as the counter could be inferred in these cases from the flags. We then compute the tweak as a function of the block identifier and the counter, so that if a block is written more than once, it is encrypted every time with a different tweak. After computing the tweak as indicated, the scheme proceeds as in S_2: at each WRITE operation, if the message has high entropy, then its hash is stored in a list L_R. A message is considered valid if either it has low entropy or its hash is stored in L_R. The intuition for the correctness of this scheme is that decryptions of the same ciphertext using the same key, but different tweaks, are independent. Thus, if the server replies with an older version of an encrypted block, the client uses a different tweak for decrypting it than the one with which it was originally encrypted. Then, the chances that it still yields a low-entropy plaintext are small.

We denote by L_R the associative array of (block identifiers, block hashes) pairs for random-looking blocks, by L_C the associative array of (block identifiers, counter) pairs and by $F(bid)$ the flags for each block identifier. The detailed scheme, S_3 is given in Figure 10.

Security of S_3 The proof for the integrity of S_3 is similar to that of the integrity of S_2. Here we just state the theorem that relates the security of S_3 to the prp-security of E, the collision resistance of the hash function and the false negative rate of the entropy test.

Theorem 7.1 *If E is a PRP-secure tweakable enciphering scheme, h is a second preimage resistant hash function and α (the false negative rate of* IsRand*) is small, then S_3 is* int-st-rep *secure:*

$$\mathsf{Adv}^{\mathsf{int\text{-}st\text{-}rep}}_{S_3}(q_1, q_2) \leq \mathsf{Adv}^{\mathsf{prp}}_{E}(q_1, q_2) + \mathsf{Adv}^{\mathsf{spr}}_{h} + \frac{(q_2+1)\alpha 2^l}{2^l - q_1}$$

Scheme Evaluation A fundamental difference between this scheme and the previous two is that in S_3 the block tweak (or the block identifier) is used to decide whether to keep the hash of the block or not. For this reason, it was not possible to divide a block into sub-blocks and we chose to evaluate this scheme separately. We evaluate this scheme for the one-month long trace that we collected, that has 4096-byte and 1024-byte blocks. In the trace, there were 813,124 distinct block IDs, from which only 113,785 were written more than once. The amount of storage that the client needs to keep for the three schemes is given in Figure 11. As expected, the amount of client storage for S_3 is between

Storage for S_1	Storage for S_2	Storage for S_3
16.262 MB	0.022 MB	0.351 MB

Figure 11. Client Storage for the Three Schemes for One-Month Traces

the storage for S_1 and S_2. Of course, the client storage increases with the lifetime of the system, as more blocks are overwritten. One solution to prevent the indefinite expansion of client state is to periodically change the encryption key, re-encrypt all the data under the new key (perhaps opportunistically), recompute all the integrity information and reset all the block flags.

8. Conclusions

We have given new cryptographic definitions and constructions for block-level storage integrity in a scenario in which storage servers are assumed to be untrusted. In order to authenticate data without changing the block size or the number of sectors accessed, clients need to keep themselves additional integrity information. Our constructions minimize the size of the integrity information, are provably secure, and are storage-efficient as demonstrated by our experimental evaluation.

9. Acknowledgements

We would like to thank the anonymous reviewers for pointing out an efficiency improvement to scheme S_3.

References

[1] J. Ann, M. Bellare. Does Encryption with Redundancy Provide Authenticity? In *Proceedings of Eurocrypt 2001*.

[2] M. Bellare, A. Desai, D. Pointcheval, P. Rogaway. Relations among notions of security for public-key encryption schemes. In *Proceedings of Crypto 1998*, LNCS 1462, 1998.

[3] M. Bellare, T. Kohno, C. Namprempre. Authenticated Encryption in SSH: Provably Fixing the SSH Binary Packet Protocol. In *9th ACM Conference on Computer and Communications Security, 2002*.

[4] M. Bellare, C. Namprempre. Authenticated Encryption: Relations among Notions and Analysis of the Generic Composition Paradigm. In *Proceedings of Asiacrypt 2000*.

[5] M. Bellare, P. Rogaway. Encode-then-Encipher Encryption: How to Exploit Nonces or Redundancy in Plaintexts for Efficient Cryptography. In *Proceedings of Asiacrypt 2000*.

[6] D. Bindel, M. Chew, C. Wells. Extended Cryptographic File System. Unpublished manuscript, December 1999.

[7] M. Blaze. A Cryptographic File System for Unix. In *First ACM Conference on Communications and Computing Security, CCS 1993*.

[8] G. Cattaneo, L. Catuogno, A. Del Sorbo, P. Persiano. The Design and Implementation of a Transparent Cryptographic File System for UNIX. In *USENIX Annual Technical Conference 2001, Freenix Track*.

[9] K. Fu. Group Sharing and Random Access in Cryptographic Storage File Systems. Master's thesis, Massachusetts Institute of Technology, June 1999.

[10] H. Gobioff, G. Gibson, D. Tygar. Security for Network Attached Storage Devices. *CMU SCS Technical Report CMU-CS-97-185*, 1997.

[11] H. Gobioff, D. Nagle, G. Gibson. Integrity and Performance in Network Attached Storage. *CMU SCS Technical Report CMU-CS-98-182*, 1998.

[12] E. Goh, H. Shacham, N. Modadugu, D. Boneh. SiRiUS: Securing Remote Untrusted Storage. In *Proceedings of the Internet Society (ISOC) Network and Distributed Systems Security (NDSS) Symposium 2003*, pages 131-145.

[13] S. Halevi, P. Rogaway. A Tweakable Enciphering Mode. In *Proceedings of Crypto 2003*.

[14] S. Halevi, P. Rogaway. A Parallelizable Enciphering Mode. In *The RSA conference - Cryptographer's track, RSA-CT '04*, LNCS vol. 2964, pages 292-304.

[15] IEEE Security in Storage Working Group. http://siswg.org.

[16] M. Kallahalla, E. Riedel, R. Swaminathan, Q. Wang, K. Fu. Plutus: Scalable Secure File Sharing on Untrusted Storage. In *Proceedings of the Second USENIX Conference on File and Storage Technologies (FAST)*, March 2003.

[17] J. Katz, M. Yung. Unforgeable Encryption and Chosen Ciphertext Secure Modes of Operation. In *Proceedings of FSE 2000*, LNCS 1978, pages 284-299, 2001.

[18] T. Kohno, A. Palacio, J. Black. Building Secure Cryptographic Transforms, or How to Encrypt and MAC. *Cryptology ePrint Archive, Report 2003/177*.

[19] H. Krawczyk. The Order of Encryption and Authentication for Protecting Communications (Or: How Secure is SSL?). In *Proceedings of Crypto 2001*.

[20] J. Li, M. Krohn, D. Mazieres, D. Shasha. Secure Untrusted Data Repository. *Technical Report TR2003-841, NYU Department of Computer Science*, June 2003.

[21] M. Liskov, R. Rivest, D. Wagner. Tweakable Block Ciphers. In *Proceedings of Crypto 2002*.

[22] D. Mazieres, M. Kaminsky, M. Kaashoek, E. Witchel. Separating Key Management From File System Security. In *Proceedings of the 17th ACM Symposium on Operating Systems Principles, SOSP 1999*.

[23] E. Miller, D. Long, W. Freeman, B. Reed. Strong Security for Network-Attached Storage. In *Proceedings of the First USENIX Conference on File and Storage Technologies (FAST)*, 2002.

[24] M. Naor, O. Reingold. On the Construction of Pseudorandom Permutations: Luby-Rackoff Revisited. In *STOC 1997*, pages 189–199, 1997.

[25] E. Riedel, M. Kallahalla, R. Swaminathan. A Framework for Evaluating Storage System Security. In *Proceedings of the First USENIX Conference on File and Storage Technologies (FAST)*, 2002.

A. Proof of Proposition 6.1

Proof: Assume there is an adversary $\mathcal{A}$ with an advantage $\mathsf{Adv}^{\mathsf{int\text{-}st\text{-}rep}}_{S_1}(\mathcal{A})$ of success in attacking the scheme. $\mathcal{A}$ replies to a $\mathsf{READ}(K, bid)$ query with c such that: (1) $\mathsf{VER}(m, bid) = 1$, i.e., $(bid, h(m)) \in L$, with $m = D^T_K(c)$ and $T = bid$; and (2) c was not sent by the client in the last $\mathsf{WRITE}(K, \cdot, bid)$ query. There are two possibilities:

- c was never written with block identifier bid. Since $(bid, h(m)) \in L$, there exists a different query $\mathsf{WRITE}(K, m', bid)$ with $m' \neq m$ and $h(m') = h(m)$. But this contradicts the second preimage resistance of h.
- c was written with block identifier bid, but it was overwritten by a subsequent query $\mathsf{WRITE}(K, m', bid)$. Assume that m' is the last query written with block identifier bid. Then $h(m) = h(m')$ and this again breaks the second preimage resistance of h.

□

Simulation of $E^T_K(m)$:	Simulation of $D^T_K(c)$:
output $c = G(T, m)$	output $m = G^{-1}(T, c)$

Figure 12. Simulation for $\mathcal{A}$

B. Proof of Theorem 6.2

Proof: Assume there is a polynomial-time attacker $\mathcal{A}$ for the int-st integrity of S_2 that makes q_1 queries to the encryption oracle and q_2 queries to the decryption oracle. We construct a distinguisher $\mathcal{D}$ for the PRP-security of E. $\mathcal{D}$ has access to oracles G and G^{-1}, which are either E_K, D_K with $K \xleftarrow{R} \mathcal{K}$ or π, π^{-1} with $\pi \xleftarrow{R} \mathsf{Perm}^{\mathcal{T}}(\mathcal{M})$.

$\mathcal{D}$ has to simulate oracles E_K and D_K for $\mathcal{A}$. The simulation is in Figure 12. $\mathcal{D}$ makes the same number of encryption and decryption queries as $\mathcal{A}$.

If $\mathcal{A}$ succeeds, i.e., replies to a $\mathsf{READ}(K, T)$ query with c such that the corresponding block $m = D^T_K(c)$ is valid ($\mathsf{VER}(m, T) = 1$) and c was not sent in a $\mathsf{WRITE}(K, \cdot, T)$ query or generated in an $E(\cdot, \cdot)$ query, then $\mathcal{D}$ outputs 1. Otherwise, $\mathcal{D}$ outputs 0. Since $T = bid$ we use T instead of bid in the READ and WRITE queries. We express the advantage of distinguisher $\mathcal{D}$ as a function of the advantage of adversary $\mathcal{A}$.

The PRP-advantage of adversary $\mathcal{D}$, from Definition 3.1 is:

$$\begin{aligned}\mathsf{Adv}^{\mathsf{prp}}_E(\mathcal{D}) &= \Pr[K \xleftarrow{R} \mathcal{K}, \mathcal{D}^{E_K, D_K} = 1] - \\ &\quad \Pr[\pi \xleftarrow{R} \mathsf{Perm}^{\mathcal{T}}(\mathcal{M}), \mathcal{D}^{\pi, \pi^{-1}} = 1]\end{aligned}$$

It is immediate that the probability of $\mathcal{D}$ outputting 1 in the case when the oracles G and G^{-1} are E_K, D_K, respectively, is equal to the probability of success of $\mathcal{A}$:

$$\begin{aligned}&\Pr[K \xleftarrow{R} \mathcal{K}, \mathcal{D}^{E_K, D_K} = 1] \\ =\ &\Pr[\mathcal{A} \text{ succeeds}] = \mathsf{Adv}^{\mathsf{int\text{-}st}}_{S_2}(\mathcal{A})\end{aligned}$$

In the case when the oracles G and G^{-1}, are π, π^{-1} with π drawn randomly from $\mathsf{Perm}^{\mathcal{T}}(\mathcal{M})$, we can express the probability of $\mathcal{D}$ outputting 1 as a function of the false negative rate of `IsRand` and the collision-resistance of h:

$$\begin{aligned}&\Pr[\pi \xleftarrow{R} \mathsf{Perm}^{\mathcal{T}}(\mathcal{M}), \mathcal{D}^{\pi, \pi^{-1}} = 1] \\ =\ &\Pr[\mathcal{A} \text{ succeeds} \mid \mathcal{A} \text{ sees random } c_1, \ldots, c_{q_1} \text{ from} \\ &\quad E_K \text{ and random } m_1, \ldots, m_{q_2} \text{ from } D_K]\end{aligned}$$

Making the notation $\mathcal{A} \sim (m, c, T)$ for $\mathcal{A}$ replying with c to a $\mathsf{READ}(K, T)$ query such that $m = G^{-1}(T, c)$, the last probability can be written as:

$$\begin{aligned}
&\Pr[\pi \xleftarrow{R} \mathsf{Perm}^{\mathcal{T}}(\mathcal{M}), \mathcal{D}^{\pi,\pi^{-1}} = 1] \\
= \;& \Pr[\mathcal{A} \sim (m,c,T) : c \text{ was not sent in a} \\
& \quad \mathsf{WRITE}(K,\cdot,T) \text{ query to } \mathcal{A} \\
& \quad \text{AND } \mathsf{VER}(m,T) = 1] \\
= \;& \Pr[\mathcal{A} \sim (m,c,T) : (K,m,T) \text{ was not a query to} \\
& \quad \mathsf{WRITE} \text{ AND } (\mathtt{IsRand}(m) = 0 \\
& \quad\quad \text{OR } (h(m),T) \in L)] \\
\leq \;& \Pr[\mathcal{A} \sim (m,c,T) : (K,m,T) \text{ was not a query to} \\
& \quad \mathsf{WRITE}, \mathtt{IsRand}(m) = 0] + \\
& \Pr[\mathcal{A} \sim (m,c,T) : (K,m,T) \text{ was not a query to} \\
& \quad \mathsf{WRITE}, (h(m),T) \in L]
\end{aligned}$$

Denote the last of these probabilities by p_1 and p_2:

$$\begin{aligned}
p_1 &= \Pr[\mathcal{A} \sim (m,c,T) : (K,m,T) \text{ was not a} \\
&\quad \text{query to } \mathsf{WRITE}, \mathtt{IsRand}(m) = 0] \\
p_2 &= \Pr[\mathcal{A} \sim (m,c,T) : (K,m,T) \text{ was not a} \\
&\quad \text{query to } \mathsf{WRITE}, (h(m),T) \in L]
\end{aligned}$$

We try to upper bound each of these two probabilities. In order to bound p_1, let's compute the probability that $\mathcal{A}$ outputs a ciphertext c for which $\pi^{-1}(T,c)$ is considered not random by the entropy test. $\mathcal{A}$ makes q_1 queries to π, which could not be returned to the client, from the int-st definition of adversary's success. If $\mathcal{A}$ picks a $c \in \mathcal{M}$, then $\mathtt{IsRand}(\pi_K^{-1}(T,c)) = 0$ with probability α, the false negative rate of the entropy test. So, in $\mathcal{M}$, there are $\alpha|\mathcal{M}| = \alpha 2^l$ ciphertexts for which $\mathtt{IsRand}(\pi_K^{-1}(T,c)) = 0$. $\mathcal{A}$ makes q_2 queries to π^{-1}, and he can make one more guess to return to the client if the decryption of none of those resulting plaintexts m satisfies $\mathtt{IsRand}(m) = 0$. Thus:

$$p_1 \leq \frac{(q_2+1)\alpha 2^l}{2^l - q_1}$$

For bounding p_2, if $(h(m),T) \in L$ and (K,m,T) was not a query to WRITE, then there exists $m' \in \mathcal{M}$ such that (K,m',T) was a query to WRITE and $h(m) = h(m')$. Then $p_2 \leq \mathsf{Adv}_h^{\mathsf{spr}}$.

To conclude, we have:

$$\begin{aligned}
&\Pr[\pi \xleftarrow{R} \mathsf{Perm}^{\mathcal{T}}(\mathcal{M}), \mathcal{D}^{\pi,\pi^{-1}} = 1] \\
\leq \;& p_1 + p_2 \leq \frac{(q_2+1)\alpha 2^l}{2^l - q_1} + \mathsf{Adv}_h^{\mathsf{spr}}
\end{aligned}$$

and:

$$\begin{aligned}
\mathsf{Adv}_{S_2}^{\mathsf{int\text{-}st}}(\mathcal{A}) &= \mathsf{Adv}_E^{\mathsf{prp}}(\mathcal{D}) + \\
&\quad \Pr[\pi \xleftarrow{R} \mathsf{Perm}^{\mathcal{T}}(\mathcal{M}), \mathcal{D}^{\pi,\pi^{-1}} = 1] \\
&\leq \mathsf{Adv}_E^{\mathsf{prp}}(\mathcal{D}) + \mathsf{Adv}_h^{\mathsf{spr}} + \frac{(q_2+1)\alpha 2^l}{2^l - q_1}
\end{aligned}$$

The statement of the theorem follows from the last relation. □

C. Proof of Theorem 6.3

In the proof, we are using the following lemma, whose proof is obvious and omitted here.

Lemma C.1 *Assume $A_1, \ldots, A_m$ are some events, not necessarily independent. Then:*

1. $\Pr(A_1 \cup \cdots \cup A_m) \leq \sum_{i=1}^m \Pr(A_i)$
2. $\Pr(A_1 \cap \cdots \cap A_m) \geq 1 - \sum_{i=1}^m (1 - \Pr(A_i))$

□

We also use Chernoff bounds:

Chernoff Bounds Let $X_1, \ldots, X_n$ be independent Poisson trials: $\Pr[X_i = 1] = p_i$, $\Pr[X_i = 0] = 1 - p_i$, $0 < p_i < 1$. Denote $X = \sum_{i=1}^n X_i, \mu = \mathbb{E}(X) = \sum_{i=1}^n p_i$. Then the following bounds hold:

1. For any $\epsilon > 0$, $\Pr[X > (1+\epsilon)\mu] < \left(\frac{e^\epsilon}{(1+\epsilon)^{1+\epsilon}}\right)^\mu$
2. For any $0 < \epsilon \leq 1$, $\Pr[X < (1-\epsilon)\mu] < e^{-\frac{\mu\epsilon^2}{2}}$

□

Now we can proceed to the analysis.

$\alpha = \Pr(H(R = R_1 R_2 \ldots R_n) \leq \tau)$, for $R_1, R_2, \ldots, R_n$ random in $\{1, 2, \ldots, b\}$.

Consider a fixed $i \in \{1, 2, \ldots, b\}$. To each R_j we associate a 0-1 variable X_j with the property: $X_j = 1 \Leftrightarrow R_j = i$. $X_1, \ldots, X_n$ are independent and the mean of each X_j is $\mathbb{E}(X_j) = \frac{1}{b}$.

Define $f_i = \sum_{j=1}^n X_j$, i.e, f_i denotes the number of blocks that have value i. From the independence of $X_1, \ldots, X_n$, it follows that $\mathbb{E}(f_i) = \frac{n}{b}$.

We also define $p_i = \frac{f_i}{n}$. Then the entropy of R is $H = -\sum_{i=1}^b p_i \log_2(p_i)$.

We will use Chernoff bounds for f_i:

For any $\delta \in (0, 1]$:

$$\begin{aligned}
&\Pr(p_i < \tfrac{1}{b}(1-\delta)) = \Pr(f_i < \tfrac{n}{b}(1-\delta)) \\
= \;& \Pr(f_i < \mathbb{E}(f_i)(1-\delta)) < e^{\frac{-\mathbb{E}(f_i)\delta^2}{2}} = e^{-\frac{n}{2b}\delta^2}
\end{aligned}$$

Now, assume that all p_i are less or equal to $\frac{1}{4}$.

The function $x \to -x\log_2(x)$ is monotonically increasing on the interval $(0, \frac{1}{4}]$, so if $p_i \geq \frac{1}{b}(1-\delta)$, then:

$$H = -\sum_{i=1}^b p_i \log_2(p_i) \geq -(1-\delta)\log_2(\tfrac{1-\delta}{b})$$

For a fixed i, from the Chernoff bounds, $p_i \geq \frac{1}{b}(1-\delta)$ with probability at least $1 - e^{-\frac{n}{2b}\delta^2}$. $p_i \geq \frac{1}{b}(1-\delta)$ for all $i = 1, \ldots, b$ with probability at least $1 - be^{-\frac{n}{2b}\delta^2}$, from Lemma C.1.

τ was defined as $\tau = -(1-\delta)\log_2(\frac{1-\delta}{b})$ and this implies $\Pr(H \geq \tau | p_i \leq \frac{1}{4}, i = 1, \ldots, b) \geq 1 - be^{-\frac{n}{2b}\delta^2}$.

Let's bound the probability $\Pr(p_i \leq \frac{1}{4}, i = 1, \ldots, b)$ using Chernoff bounds.

For a fixed $i \in \{1, \ldots, b\}$, let's denote $\epsilon = \frac{b}{4} - 1$. Then:

$$\begin{aligned}
&\Pr(p_i > \tfrac{1}{4}) = \Pr(p_i > \tfrac{1}{b}(1 + \tfrac{b}{4} - 1)) \\
= \;& \Pr(f_i > \tfrac{n}{b}(1 + \tfrac{b}{4} - 1)) < \left(\frac{e^\epsilon}{(1+\epsilon)^{1+\epsilon}}\right)^{\frac{n}{b}} \\
= \;& \left(\tfrac{1}{e}\right)^{\frac{n}{b}} \left(\tfrac{4e}{b}\right)^{\frac{n}{4}}
\end{aligned}$$

Using the lemma, it follows that:

$$\Pr(p_i \leq \tfrac{1}{4}, i = 1, \ldots, b) \geq 1 - b\left(\tfrac{1}{e}\right)^{\frac{n}{b}} \left(\tfrac{4e}{b}\right)^{\frac{n}{4}}$$

Combining the bounds, we get:

$$\begin{aligned} & \Pr(H \geq \tau) \\ = \; & \Pr(H \geq \tau | p_i \leq \tfrac{1}{4}, i = 1, \ldots, b) \cdot \\ & \Pr(p_i \leq \tfrac{1}{4}, i = 1, \ldots, b) \\ + \; & \Pr(H \geq \tau | \exists\, i = 1, \ldots, b \text{ st } p_i \leq \tfrac{1}{4}) \cdot \\ & \Pr(\exists\, i = 1, \ldots, b \text{ st } p_i \leq \tfrac{1}{4}) \\ \geq \; & (1 - be^{-\frac{n}{2b}\delta^2})(1 - b\left(\tfrac{1}{e}\right)^{\frac{n}{b}}\left(\tfrac{4e}{b}\right)^{\frac{n}{4}}) \end{aligned}$$

In conclusion:

$$\begin{aligned} \alpha \; & \leq \; 1 - (1 - be^{-\frac{n}{2b}\delta^2}) \left(1 - b\left(\frac{1}{e}\right)^{\frac{n}{b}}\left(\frac{4e}{b}\right)^{\frac{n}{4}}\right) \\ & \leq \; be^{-\frac{n}{2b}\delta^2} + b\left(\frac{1}{e}\right)^{\frac{n}{b}}\left(\frac{4e}{b}\right)^{\frac{n}{4}}. \end{aligned} \tag{3}$$

□