Secure Storage with Deduplication

John Best
IBM Research

Wayne Hineman
IBM Research

Steven Hetzler*
Facebook

Guerney Hunt
IBM Research

Charanjit S. Jutla
IBM Research

Abstract

We describe a new secure storage scheme that facilitates *deduplication*. The scheme is also proved secure in the universal-composability model. It is a single server scheme, and the basic scheme does not prevent against off-line dictionary attacks if the server is compromised. However, if a global secret key is shared amongst users of the organization, and this key is never stored at the server, we also get protection against off-line dictionary attacks even if the server is compromised. The UC security model for deduplication is based on an earlier work of Liu, Asokan and Pinkas, Proc. CCS 2015. The scheme obtains additional optimization by employing the XTS-AES mode of encryption in the public random permutation model.

Another upshot of the analysis is that one can first MAC and then encrypt using XTS mode and attain authenticated encryption, avoiding the pitfalls cautioned against by Hugo Krawczyk, in the work "How Secure is SSL?", CRYPTO 2001.

1 Introduction

In this work, we describe a new secure storage scheme that facilitates *deduplication*. The scheme is also proved secure in the universal-composability (UC) model [3]. It is a single server scheme, and the basic scheme does not prevent against off-line dictionary attacks if the server is compromised. However, if a global secret key is shared amongst users of the organization, and this key is never stored at the server, we also get protection against off-line dictionary attacks even if the server is compromised.

The scheme achieves additional storage efficiency by using XTS-mode of encryption, which is a restriction of IAPM [6] to just message-secrecy. A not-so-efficient version of the basic scheme could be described as follows: A client hashes the plaintext file using a random oracle (such as SHA-2) to obtain an AES key k. It then encrypts the file using this key k. As this is deterministic encryption, it leads to easy deduplication. This key k can also be stored at the server, encrypted under a personal (AES) secret key of the user. The encryption of the file could use IAPM to get (deterministic) authenticated-encryption. However, in our scheme the additional IAPM authentication-tag need not be computed (nor stored), as authentication can be obtained from the key k itself. Instead of using k, which was obtained by invoking the random oracle on the plaintext file, as an AES key, we use it as the whitening key for the XTS scheme. The AES-key for the XTS scheme can be set to a

*This research was done while the author was at IBM Research.

global *public random key* (see e.g. [11, 8], where it is defined as IAPM mode in the public random permutation model). On decryption of the ciphertext file, the resulting plaintext is again hashed and checked against k for authentication. We prove that this suffices for authentication. The public random permutation model, however, does not entirely match the security bounds achievable as by usual private random permutations, in particular because the adversary has access to the underlying random permutation as an oracle and can query it in an offline fashion. However, since the size of the files are limited (to say no more than 2^{32} 128-bit blocks), and each file uses a fresh random whitening key, the possibility of an offline attack is limited to a probability of success of at most 2^{-96}.

In a more advanced scheme, the random oracle is replaced by a pseudo-random function (say, HMAC), and the HMAC key is chosen randomly and known only to the members of the organization. The XTS public random key can also be kept secret from everyone except members of the organization.

The UC security definitions [3] of secure storage with deduplication are inspired by password-authenticated key-exchange (PAKE [2]) definitions [4], and in particular from the work [12]. Since there is a single server, and ciphertexts and encrypted keys are stored at the server, our definitions are akin to asymmetric PAKE definitions [5]. We prove that our scheme realizes the UC ideal functionality.

Finally, another upshot of the analysis is that one can first MAC and then encrypt using XTS mode and attain authenticated encryption, avoiding the pitfalls cautioned against by Hugo Krawczyk [10]

2 Introduction to Ideal Functionalities

Cryptographic protocols are inherently multi-party protocols where the parties jointly compute some function. While the issue of liveness of protocols falls in the realm of distributed computing, we are interested here in issues related to privacy (secrecy) of the parties, as well as the related issue of authentication.

These multi-party privacy constraints of a protocol are best **defined** by hypothetically employing an *ideal trusted third party*. In the simplest form of this definitional paradigm, all parties hand their respective inputs to the ideal trusted third party, which then computes the function on these inputs as desired by the protocol being defined, and then hands portions of the output to the individual parties, the portions restricted to what is desired by the protocol.

Note that in the above we assumed that the parties have a secure and persistent tunnel to the ideal third party, but this being just a definition this assumption is not a concern. In a more general setting, this definitional paradigm also allows the ideal third party to leak certain information to an **Adversary** (without loss of generality, we always assume that there is *only* one monolithic adversary). This maybe a necessary part of the definition as otherwise there may not be any implementation possible in the real world (i.e. a distributed implementation without the ideal third party). In a further generalization, the ideal third party may also take directives and/or inputs from the Adversary.

Finally, to allow a **compositional** definition, we assume that all parties (except the ideal third party, but including the Adversary) are driven by an all encompassing entity called the **Environment**. In other words, the inputs of the parties (possibly related) are actually provided

by the Environment so as to cover all possible scenarios. This notion of Environment is *not necessary* to understand ideal functionalities, but it will become important when we discuss compositional security.

Ideal Functionality $\mathcal{F}_{\text{DDUP}}$

Participants: An arbitrary set of users $\{P_i\}_{i \in N}$ and adversary $\mathcal{S}$ interact with functionality $\mathcal{F}_{\text{DDUP}}$ (or $\mathcal{F}$).

Initiate: The functionality initializes variable INDEX to zero. It also initializes a dictionary D of plaintext-indexset pairs to empty. This dictionary is keyed by a plaintext-message, and its value is a (non-empty) set of indices. It also initializes a dictionary L to empty. This dictionary is keyed by an index and its value is a pair consisting of a userid and a plaintext message. It also initializes the set A of pending adversarial queries to empty. This maintains a list of messages that the adversary may have preemptively queried.

Read-Write:

On receiving an input $(\texttt{store}, P_i, M)$ from P_i,

1. increment the variable INDEX,
2. check if M is in the dictionary D. If so, insert INDEX into the set associated with M in D. If not, then create a new entry in dictionary D with key M and indexset $\{\text{INDEX}\}$.
3. Add the key-value pair $\langle \text{INDEX}, (P_i, M) \rangle$ to the dictionary L, and *return* INDEX to the user P_i.
4. Check if M is in the set A. If so, *output* P_i, M *and* the indexset associated with M in D to the adversary $\mathcal{S}$. If not, then just output P_i and the set of indices associated with M in D to the adversary $\mathcal{S}$.

On receiving an input $(\texttt{read}, P_i, j)$ from P_i, check if there is entry $\langle j, (P_i, M^*) \rangle$ in dictionary L. If so, then return M^* to P_i, else return $\perp$ to P_i.

Adversarial Action:

On receiving an input $(\texttt{corrupt-message}, j)$ from adversary $\mathcal{S}$, replace any entry with key j (and userid P_i) in L by $\langle j, (P_i, \perp) \rangle$.

On receiving an input $(\texttt{move-message}, j_1, j_2)$ from adversary $\mathcal{S}$, if entries with keys j_1 and j_2 in L, say, $\langle j_1, (P_i, M_1^*) \rangle$ and $\langle j_2, (P_i, M_2^*) \rangle$, have the same userid P_i, then replace the entry corresponding to key j_2 in L by $\langle j_2, (P_i, M_1^*) \rangle$, remove j_2 from the indexset corresponding to key M_2^* in D, and insert j_2 into the indexset corresponding to key M_1^* in D. If the userids P_{i1} and P_{i2} are not the same in entries corresponding to j_1 and j_2 respectively in L then replace the entry corresponding to j_2 in L by $\langle j_2, (P_{i2}, \perp) \rangle$.

On receiving an input $(\texttt{test-message}, M)$ from adversary $\mathcal{S}$, check if M is in the dictionary D. If so, return the index-set associated with M to $\mathcal{S}$. Else, insert M into the set A.

Figure 1: A functionality for Secure Storage with Deduplication

2.1 Introduction to Simulation Based Security Definition

While the previous section only described a way to define the desired goal of a protocol using an ideal third party, one can also define security of a **real-world protocol** by relating it to this

ideal functionality. Note that the ideal functionality has precisely characterized what output each legitimate party gets, and what is leaked to the Adversary. It also characterizes directives that an Adversary unavoidably controls, e.g. whether a message should be delivered to a counter-party or not. Thus, if we can show that the real world protocol has the same characteristics, we have managed to prove desired security constraints of the real protocol.

A real world protocol between some parties is said to **realize** an ideal functionality, if for every adversary in the the real world protocol, there is an adversary in the ideal world protocol (i.e. involving the ideal trusted third party representing the ideal functionality) such that the Environment interacting with the parties and the two adversaries (the real world and the ideal world) cannot distinguish between the two scenarios[1]. Thus, in this definition the Environment gets the same amount of information in the real and the ideal world. Since the ideal world precisely defined what the Environment gets (actually what the Adversary gets, which it reports back to the Environment), one concludes that also in the real world protocol the Environment (and hence the adversary) gets only the precisely defined information.

More rigorous definitions can be found in [3].

Such *realization proofs* are done by a **simulation** argument. Given a real world protocol, and an Adversary $\mathcal{A}$ in the real world, the proof constructs a Simulator $\mathcal{S}$ that simulates the real world protocol to $\mathcal{A}$. To do this simulation, $\mathcal{S}$ is assumed to reside in the ideal world, and hence has access to the Ideal Functionality (as an ideal world Adversary $\mathcal{S}$). If such a simulation is possible, then we have managed to construct for each adversary $\mathcal{A}$ in the real world an adversary $\mathcal{A}'$ (which is obtained by juxtaposing $\mathcal{S}$ and $\mathcal{A}$) in the ideal world, such that the Environment can not distinguish the two scenarios, and hence, by definition, security follows.

2.2 Ideal Functionality for Secure Storage with Dedup

The ideal functionality for secure storage with depduplication is given in Fig 1.

3 Real World Scheme for Secure Storage with Deduplication

We will assume a secure keystore functionality $\mathcal{F}_{\text{KS}}$, which for each user P_i returns an AES key (which was initialized to a randomly chosen AES key). We will also assume a random oracle functionality $\mathcal{F}_{\text{RO}}$.

The real word scheme will be given in the hybrid-$\mathcal{F}_{\text{RO,KS}}$ model. The scheme will maintain a storage consisting of a few dictionaries. The real-world adversary $\mathcal{A}$ will have full read-write access to this storage.

The storage will maintain two dictionaries, D^*, L^*. The dictionary D^* will have a unique counter value ℓ attached to each entry as a dictionary key, the counter initialized to zero, and incremented every time a new entry is added to D^*. The value associated to ℓ in the dictionary is a ciphertext c.

[1]The main idea is the generalization of the concept of zero-knowledge. A protocol is said to be zero-knowledge if for every adversary in the real world that gains some private knowledge there exists another adversary which can gain the same knowledge from an ideal execution of the protocol. From this, one can then conclude that since the ideal execution leaks (almost) zero information which is meant to be private, then the adversary in the real world also gets zero information. Thus, the real protocol is secure in the sense that all information which is meant to be private remains private.

The dictionary L^* is keyed by a (unique) index, and the value is a triple consisting of a userid P_j, a counter value ℓ, and a cryptographic-tag τ.

As mentioned above, the scheme maintains a counter that is used as a key in dictionary D^*. The scheme also initializes INDEX to zero,, and uses this as index values.

We will also use a XTS-like encryption mode but with public random permutation,instead of private random permutation (see Section 5 for details). The security of such a scheme is proved in [11, 8] (see Section 5.3). The scheme is initialized by choosing a fresh random AES key k. The key k is made public by giving it to the adversary $\mathcal{A}$. The mode derives its security from a secret h being used as a whitening-key (also known as a tweak key). The encryption phase of the mode takes as input the public key k, a message M and an IV and returns a ciphertext $c = \text{XTS-ENC}(k, \mathsf{h}, M)$. The decryption phase takes as input the public key k, a ciphertext c and an IV and returns a plaintext $M = \text{XTS-DEC}(k, \mathsf{h}, c)$.

Now we describe the real-world read-write operations. It is described in Fig 2.

Real-World Scheme DDUP

On receiving an input $(\texttt{store}, P_i, M)$ from P_i,

1. Increment the variable INDEX,
2. let $\mathsf{h} = \mathcal{F}_{\text{RO}}(M)$. Obtain $c = \text{XTS-ENC}(k, \mathsf{h}, M)$. Search for c in D^*. If there is already an entry, record the counter value ℓ. Else, increment counter, to obtain ℓ, and create a new entry in D^* with this counter value ℓ, and set the value associated with this entry as c.
3. Fetch the key k_i associated with P_i from $\mathcal{F}_{\text{KS}}$. AES-Encrypt h with AES key k_i to obtain τ. Insert in dictionary L^*, an entry with key INDEX and value a triple consisting of (P_i, ℓ, τ).
4. Return to P_i the index value INDEX.

On receiving an input $(\texttt{read}, P_i, j)$ from P_i,

1. Search the key j in the dictionary L^*. If there is no entry, return $\perp$. If there is an entry, say (P_{i^*}, ℓ, τ), then if $i^* \neq i$, return $\perp$. If the userid matches, i.e. $i^* = i$, then fetch key k_i for P_i using $\mathcal{F}_{\text{KS}}$.
2. Next obtain h by AES-decrypting τ with key k_i.
3. Search for key ℓ in D^*, and let the corresponding value be c^*. Obtain $M = \text{XTS-DEC}(k, \mathsf{h}, c)$.
4. Using the Random oracle, obtain $\mathsf{h}^* = \mathcal{F}_{\text{RO}}(M)$.
5. If $\mathsf{h}* \neq \mathsf{h}$ return $\perp$. Otherwise, return M.

Figure 2: A scheme realizing $\mathcal{F}_{\text{DDUP}}$ in hybrid-$\mathcal{F}_{\text{KS,RO}}$ model

4 Authenticated Encryption in Public Random Permutation Model

We give definitions of authenticated encryption schemes in a public random permutation model. Let Coins be the set of infinite binary strings. Let $\mathcal{K} \subseteq \{0,1\}^*$ be the key space, and $\mathcal{D}$ be a

distribution on the key space.

Definition A (2-oracle, probabilistic, symmetric, stateless) *authenticated-encryption* scheme, with block size n, key space $\mathcal{K}$, and distribution $\mathcal{D}$, consists of the following:

- **initialization:** All parties exchange information over private lines to establish a private key $k \in \mathcal{K}$. All parties store k in their respective private memories.
- **message sending with integrity:** Let E and D be efficient 2-oracle algorithms, with E taking as input a key k (in $\mathcal{K}$), COINS (in Coins), and a plaintext binary string and outputting a binary string, and D taking as input a key k and a ciphertext binary string and outputting either $\perp$ or a binary string. The two oracles take n-bits as input and produce n-bits as output.

In addition E and D have the property that if oracles $\mathcal{O}_1$ and $\mathcal{O}_2$ implement inverse functions of each other, then for all $k \in \mathcal{K}$, for all COINS and P,

$$D^{\mathcal{O}_1,\mathcal{O}_2}(k, (E^{\mathcal{O}_1,\mathcal{O}_2}(k, \text{COINS}, P)) = P$$

We will usually drop the random argument to E as well, and just think of E as a probabilistic algorithm. The security of such a scheme is given by the following two definitions, the first defining confidentiality under chosen plaintext attacks, and the second defining message integrity. In the security definitions, we will count the length of plaintext inputs in terms of n-bit blocks. Thus, a plaintext input of length m bits will be considered to have length $\lceil m/n \rceil$ blocks.

Definition (*Chosen-Plaintext Attack Security[1]*)

For any $n > 0$, consider a 3-oracle probabilistic adversary A. Consider an authenticated-encryption scheme with key-space $\mathcal{K}$, key distribution $\mathcal{D}$ and 2-oracle algorithms E and D. For any n-bit permutation π, let $\text{REAL}_{\mathbf{k}}^{\pi}$ be the oracle that on input P returns $E^{\pi,\pi^{-1}}(\mathbf{k}, P)$, and $\text{IDEAL}_{\mathbf{k}}^{\pi}$ be the oracle that on input P returns $E^{\pi,\pi^{-1}}(\mathbf{k}, 0^{|P|})$. The IND-CPA advantage Adv_A of the adversary A in the *public random permutation model* is given by

$$|\Pr[\mathbf{k} \leftarrow \mathcal{D}; A^{\pi,\pi^{-1},\text{REAL}_{\mathbf{k}}^{\pi}} = 1] - \Pr[\mathbf{k} \leftarrow \mathcal{D}; A^{\pi,\pi^{-1},\text{IDEAL}_{\mathbf{k}}^{\pi}} = 1]|$$

where the probabilities are over choice of π as a random permutation on n-bits, and choice of k according to $\mathcal{D}$, other randomness used by E, and the probabilistic choices of A.

An authenticated-encryption scheme with block size n is said to be (t, $q1$, $q2$, m, ϵ)-secure against chosen plaintext attack in the *public random permutation model* if for any adversary A as above which runs in time at most t and asks at most $q1$ queries to π and π^{-1}, and at most $q2$ queries to the third oracle (these totaling at most m blocks), its advantage Adv_A is at most ϵ.

Definition (*Message Integrity):* Consider an adaptive 3-oracle (probabilistic) adversary A running in two stages. Adversary A has access to oracles $\mathcal{O}_1$, $\mathcal{O}_2$ and an encryption oracle $E^{\mathcal{O}_1,\mathcal{O}_2}(k, \cdot)$. In the first stage (*find*) A asks r queries of the encryption oracle. Let the oracle replies be $C^1, ..., C^r$. Subsequently in the second stage, A produces a cipher-text C', different from each C^i, $i \in [1..r]$. The adversary's success probability is given by

$$\text{Succ}_A \stackrel{\triangle}{=} \Pr[D^{\pi,\pi^{-1}}(k, C') \neq \perp]$$

where the probability is over choice of $\mathcal{O}_1$ as a random permutation on n-bits (and $\mathcal{O}_2$ as its inverse), and choice of k according to $\mathcal{D}$, other randomness used by E, and the probabilistic choices of A.

An authenticated-encryption scheme with block size n is $(t, q1, q2, m, \epsilon)$-secure for message integrity in the *public random permutation model* if for any 3-oracle adversary A running in time at most t and making at most $q1$ queries to $\mathcal{O}_1$ and $\mathcal{O}_2$ and at most $q2$ queries to the encryption oracle (these totaling m blocks), its success probability is at most ϵ.

5 IAPM (and XTS) in Random Permutation Model

5.1 Preliminaries

Definition 1 (**Keyed Hash Functions**): For any finite set H, an H-keyed (m, n)-hash function $\mathcal{H}$ has signature $\mathcal{H} : H \times \{0,1\}^m \to \{0,1\}^n$.

Definition 2 (ϵ-**XOR-Universal Keyed Hash Function**) [9] For any finite set H, an H-keyed (m, n)-hash function $\mathcal{H}$ is called an ϵ-XOR-Universal keyed hash function, if for every m-bit value M, and every n-bit value c, $\Pr_h[\mathcal{H}(h, M) = c] \leq \epsilon$, and further if for every pair of distinct m-bit values $M1$ and $M2$, and every n-bit value c, $\Pr_h[\mathcal{H}(h, M1) \oplus \mathcal{H}(h, M2) = c] \leq \epsilon$, where the probabilities are over choosing h uniformly from H.

An example 2^{-n}-XOR-universal keyed $(2n, n)$ hash function is built using Galois field GF(2^n), where $\mathcal{H}(\mathsf{h}, (a, b)) = \mathsf{h} * a + b$.

Definition 3 (**min-entropy**) For a random variable X defined on $\{0,1\}^n$, its min-entropy is the minimum over all n-bit strings x of $\log(1/\Pr_X[X = x])$.

While in this work we are mostly focused on XTS, which does not provide message authentication, but since it is built from the authenticated-encryption mode IAPM, we will define the schemes using IAPM. The XTS scheme just ignores the MAC-tag computation.

We will prove our results for more general (abstract) IAPM-like schemes, but to serve as a background we briefly review the definition of IAPM from [6, 7]. In the following, the operator "+" will stand for integer addition, and "$\oplus$" for n-bit exclusive-or. Since with wide permutations on n bits, the "MAC" tag produced by the permutation may need to be truncated, the authentication check in decryption is defined slightly differently (as in OCB [13] and [11]). In the following, when using n-bit permutations, we will refer to n-bit strings as a *block*.

Definition 4 Given a permutation f from n bits to n bits, an H-keyed $(2n, n)$-hash-function g, where H is the set of all ν-bit strings ($\nu \leq n$), the (deterministic) function E-IAPM$_{f,g}$: $H \times \{0,1\}^n \times (\{0,1\}^n)^* \to (\{0,1\}^n)^+$ is defined as follows:

- Let the input to E-IAPM$_{f,g}$ be $h \in H$, an n-bit (block) IV, and an m block string P ($= P_1, P_2, ..., P_m$).
- Define $C_0 = IV$, and checksum $= 0 \oplus \bigoplus_{j=1}^{m} P_j$.
- Define for $j = 1$ to m:
  $C_j = g(h, \langle IV, j \rangle) \oplus f(P_j \oplus g(h, \langle IV, j \rangle))$.

- $C_{m+1} = g(h, \langle IV, 0\rangle) \oplus f(\text{checksum} \oplus g(h, \langle IV, m+1\rangle))$.
- The output of the function E-IAPM$_{f,g}$ is the $m+2$ block string $C_0, C_1, ..., C_{m+1}$. The last block can be truncated to the required "MAC" tag-length, say μ bits.

Definition 5 With the same parameters as above, the function D-IAPM$_{f,g}$: $H \times (\{0,1\}^n)^+ \rightarrow (\{0,1\}^n)^* \cup \{\perp\}$ is defined as follows:

- Let the input to D-IAPM$_{f,g}$ be an $h \in H$, an $((m+1)n+\mu)$-bit string C, which is divided into $(m+1)$ blocks IV$, C_1, ..., C_m$ and a tag T of μ bits.
- Define for $j = 1$ to m:
  $P_j = g(h, \langle IV, j\rangle) \oplus f^{-1}(C_j \oplus g(h, \langle IV, j\rangle))$.
- $T^* = g(h, \langle IV, 0\rangle) \oplus f(\bigoplus_{j=1}^{m} P_j \oplus g(h, \langle IV, m+1\rangle))$.
- if $(\text{trunc}_\mu(T^*) \neq T)$ return $\perp$, otherwise the output of D-IAPM$_{f,g}$ is the m block string $P_1, ..., P_m$.

5.2 Public Random Permutation Model

If g is an efficiently computable function, the above two functions E-IAPM and D-IAPM can be computed efficiently given oracle access to f and f^{-1}. It is important to make this characterization as we intend to instantiate f and f^{-1} by public permutations. Further, the definition of an (authenticated) encryption scheme requires specifying the distribution from which the keys are sampled. We may assume a benign setting where the ν-bit key h above is chosen uniformly from H.

Definition 6 (**IAPM with uniform keys in public RP model**) Authenticated Encryption scheme IAPM-uniform(g, ν, μ) with block size n, and oracle f and f^{-1} is given by a key space $\mathcal{K}$ that is the set of ν-bit strings, and a distribution $\mathcal{D}$ on keys that is the uniform distribution on $\mathcal{K}$. Moreover, the encryption and decryption algorithms under key k are given by E-IAPM$_g^{f,f^{-1}}$ $(k, \cdot, \cdot)$, and D-IAPM$_g^{f,f^{-1}}$ $(k, \cdot)$ resp.

Definition 7 (**Zero-IV IAPM**) An IAPM scheme is called a zero-IV scheme if IV is always set to zero. Thus, $C_0 = 0$ for all ciphertexts, and g function is computed with IV set to zero. As a consequence, the encryption function does not need the IV input.

In this work, we will be using Zero-IV (IAPM and) XTS. Essentially, only the block number is used in the XOR-universal g function. A concrete instantiation of g is then $g(h, i) = h * i$ where both h and i are treated as elements of a finite field, preferably Galois field GF(2^n), and the multiplication is in the field.

5.3 Theorems for IAPM in Public Random Permuation Model

The following result is proved in [11, 8] IAPM-uniform(g, ν, μ) is secure for message integrity in the public random permutation model, with

$$\text{Succ}_A \leq 2^{-\mu} + (m^2 + 3v) \cdot \epsilon + q * 2^{-n} + (2 * q * m + m(m+1)) * \epsilon$$

where A makes at most z queries to the encryption oracle, these totaling at most m blocks, and A makes at most q queries to the public random permutation. Here, $\nu = n$, and hence ϵ is 2^{-n} (from ϵ-XOR-universal g). Recall μ is the size of the authenticaion tag. Here v is the number of blocks in the challenge ciphertext.

If authentication is not required, the step in IAPM that computes the authentication tag can be removed, and we get the XTS scheme. The above bound also holds for message secrecy, with no $2^{-\mu}$ term. To be precise,

$$\mathrm{Adv}_A \leq (q + 2 * q * m + 2 * m(m+1)) * 2^{-n},$$

where A makes at most z queries to the encryption oracle, these totaling at most m blocks, and A makes at most q queries to the public random permutation.

Note that the q queries to the public random permutation should be considered as offline attack, i.e. these can be performed by the Adversary regardless of how many actual block m are encrypted, and even before any blocks are encrypted using the whitening key h. Hence, m should be kept as low as possible, preferably less than 2^{32}.

6 Proof of Security

We start by describing the simulator $\mathcal{S}$ which interacts with the ideal functionality $\mathcal{F}_{\mathrm{DDUP}}$ and presents a view to the adversary$\mathcal{A}$ (and the environment), which we will later show (in Section 6.2) is indistinguishable to the view of the adversary (and the environment) in the real world scheme.

6.1 Simulator

Since we are presenting the real world scheme in the hybrid-$\mathcal{F}_{\mathrm{RO,KS}}$ model, the simulator $\mathcal{S}$ will internally mimic (i.e. simulate) the random oracle and the keystore functionalities.

It will also present a storage to the Adversary (with full read-write capability for the Adversary), consisting of two dictionaries D^{**} and L^{**} which simulate the real world dictionaries D^* and L^*.

This is how $\mathcal{S}$ behaves.

- **Initialize:** The Simulator initializes the random oracle to an empty table. It chooses a fresh random AES key for each user P_i and maintains it in a table keystore. It initialzes D^{**} to an empty dictionary, and L^{**} to an empty ditionary as well. It initializes ℓ to zero, and $*$ to zero.

- **Read-Write:**

  Store: When the ideal functionaity receives a store command, it outputs to $\mathcal{S}$, either (a) P_i, M and indexset associated with it, or (b) just P_i and an indexset. The former happens when the $\mathcal{S}$ had earlier called "test-message" with M. In any case, the indexset is either singleton or a larger set. In the former case (i.e. singleton case), the Simulator creates a new random string c^* (of length of the ciphertexts) and increments ℓ. and stores in D^{**} a new entry with key ℓ and value c^*. It also inserts in L^{**} an entry with the key the singleton value in the indexset, and value a triple (P_i, ℓ, τ^*), where tau^* is AES encryption of h^* under AES-key of P_i (obtained from its simulated keystore), and h^* is a fresh random n -bit string.

If the indexset is not a singleton, it searches L^{**} for entries where some entry INDEX_0 of the just received indexset is a key, and the value associated with it is (P_{i0}, ℓ_0, τ_0). Then for all (actually, just one) values INDEX in the just received indexset which do not already have an entry in L^{**}, it creates a new entry with key INDEX and value the triple (P_i, ℓ_0, τ^*), where τ^* is obtained by AES-encrypting h_0 with key of P_i, where h_0 is obtained by AES-decrypting τ_0 with key for P_{i0}.

In case (a) *and if* the indexset just received is a singleton, recall the $\mathcal{S}$ had already called "test-message" with M. As we will see below, this happens whenever the real world adversary calls the random oracle with M, and RO simulation by $\mathcal{S}$ had returned an $\mathsf{h}(M)$, which it maintains in its RO simulation table. So, in this case the τ is obtained by AES encrypting this $\mathsf{h}(M)$ with the AES key of P_i. Rest of the simulation is same as above.

Read: $\mathcal{S}$ is not even notified by the ideal functionality during a read operation, so no action is needed.

- **Actions by $\mathcal{A}$:** These consist of $\mathcal{A}$ either calling the random roacle $\mathcal{F}_{\text{RO}}$ or $\mathcal{A}$ modifying L^* and/or D^* in the real world. We show how $\mathcal{S}$ simulates these behaviors using L^{**} and D^{**} instead, and its simulation of the random oracle.

  RO call: If $\mathcal{A}$ calls the random oracle with a value M then S checks its table to see if M was ever called before, in which case it returns the stored random value. Else, it chooses a fresh random n bit value and stores it associated with M and returns the new value to $\mathcal{A}$. The simualtor $\mathcal{S}$ *also calls* $\mathcal{F}_{\text{DDUP}}$ with $(\texttt{test-message}, M)$.

  Modify L^* or D^*: Whatever modifications $\mathcal{A}$ performs on the real-world L^* and D^*, the simulator does the same modifications to L^{**} and D^{**}. For example, if $\mathcal{A}$ moves an entry from one record to another, $\mathcal{S}$ does the same. In this case, $\mathcal{S}$ also calls $\mathcal{F}_{\text{DDUP}}$ with `move-message`. If $\mathcal{A}$ modifies with a brand new value, that same brand new value is used on L^{**} and/or D^{**}. In this case $\mathcal{S}$ calls $\mathcal{F}_{\text{DDUP}}$ with `corrupt-message`.

6.2 Indistinguishability

Since in the UC-model [3], the Environment $\mathcal{E}$ drives the parties and the Adversary, and security is defined in terms of indistinguishability of the view of $\mathcal{E}$ in the ideal and the real world, we will now combine the simulator $\mathcal{S}$ and the ideal functionality $\mathcal{F}_{\text{DDUP}}$ into a a single entity responding to $\mathcal{E}$ (and $\mathcal{A}$ which we will just merge with $\mathcal{E}$). we remind the user that even legitimate users P_i receive all their commands from $\mathcal{E}$, and anything that $\mathcal{F}$ outputs to P_i, party P_i reports it back to $\mathcal{E}$. We will denote this as experiment EXPT_0 between the environment $\mathcal{E}$ and the simulator $\mathcal{S}$ (which now merges with $\mathcal{F}_{\text{DDUP}}$. We will describe a sequence of experiments between these two entities, with the last one being identical to the real world. We will show that $\mathcal{E}$ cannot distinguish between the experiments with non-negligible probability.

6.3 Experiment EXPT_0

This is how $\mathcal{S}$ behaves in EXPT_0 (with $\mathcal{F}_{\text{DDUP}}$ merged with it) against $\mathcal{E}$.

- **Initialize:** The Simulator $\mathcal{S}$ initializes the random oracle to an empty table. It chooses a fresh random AES key for each user P_i and maintains it in a table keystore. It initializes D^{**}

to an empty dictionary, and L^{**} to an empty dictionary as well. It initializes ℓ to zero, and $*$ to zero.

- **Read-Write:**

  Store: When the ideal functionality (and thus $\mathcal{S}$) receives a store command, recall $\mathcal{F}_{\text{DDUP}}$ outputs to to $\mathcal{S}$, either (a) P_i, M and indexset associated with it, or (b) just P_i and an indexset. The former happens when the $\mathcal{S}$ had earlier called "test-message" with M. In any case, the indexset is either singleton or a larger set. In the former case (i.e. singleton case), the Simulator creates a new random string c^* (of length of the ciphertexts) and increments ℓ. and stores in D^{**} a new entry with key ℓ and value c^*. It also inserts in L^{**} an entry with the key the singleton value in the indexset, and value a triple (P_i, ℓ, τ^*), where tau^* is AES encryption of h^* under AES-key of P_i (obtained from its simulated keystore), and h^* is a fresh random n -bit string. The *main change now* is that $\mathcal{S}$ also knows the M which P_i requested to store, and hence in its RO-table it puts h^* as a random oracle output of M.

  If the indexset is not a singleton, it searches L^{**} for entries where some entry INDEX_0 of the just received indexset is a key, and the value associated with it is (P_{i0}, ℓ_0, τ_0). Then for all (actually, just one) values INDEX in the just received indexset which do not already have an entry in L^{**}, it creates a new entry with key INDEX and value the triple (P_i, ℓ_0, τ^*), where τ^* is obtained by AES-encrypting h_0 with key of P_i, where h_0 is obtained by AES-decrypting τ_0 with key for P_{i0}.

  In case (a) *and if* the indexset just received is a singleton, recall the $\mathcal{S}$ had already called "test-message" with M. As we will see below, this happens whenever the real world adversary calls the random oracle with M, and RO simulation by $\mathcal{S}$ had returned an $\mathsf{h}(M)$, which it maintains in its RO simulation table. So, in this case the τ is obtained by AES encrypting this $\mathsf{h}(M)$ with the AES key of P_i. Rest of the simulation is same as above.

  It is *important* to note that the dictionaries L and D that $\mathcal{F}_{\text{DDUP}}$ maintained are consistent with the dictionaries L^{**} and D^{**}. In other words, for all entries in L^{**} with key and value (P', ℓ', τ') such that ℓ' is same as some common ℓ^*, there is an entry in D with indexset comprising of these collection of . This indexset has a particular M^* associated with it in D. And in D^{**} there is a single ciphertext c^* associated with this ℓ^*. Thus, in the merged $\mathcal{S}$, c^* and M^* are matched (and also matched with a unique ℓ^* and indexset). L and D are already consistent with each other in the sense that if L has an entry $\langle \text{INDEX}, (P_i, M) \rangle$, then the indexset associated with M in D contains INDEX.

  Read: on receiving $(\texttt{read}, P_i, j)$ from $\mathcal{E}$, the merged $\mathcal{S}$ follows what $\mathcal{F}_{\text{DDUP}}$ does, i.e. the following: check if there is entry $\langle j, (P_i, M^*) \rangle$ in dictionary L. If so, then return M^* to P_i, else return $\perp$ to P_i.

- **Actions by $\mathcal{A}$ (i.e $\mathcal{E}$):** These consist of $\mathcal{E}$ either calling the random oracle $\mathcal{F}_{\text{RO}}$ or $\mathcal{E}$ modifying L^* and/or D^* in the real world. We show how $\mathcal{S}$ simulates these behaviors using L^{**} and D^{**} instead, and its simulation of the random oracle.

  RO call: If $\mathcal{E}$ calls the random oracle with a value M then S checks its table to see if M was ever called before, in which case it returns the stored random value. Else, it chooses a

fresh random n bit value and stores it associated with M and returns the new value to $\mathcal{E}$. Since simulator $\mathcal{S}$ *also calls* $\mathcal{F}_{\text{DDUP}}$ with (`test-message`, M), this has the effect that M is added to set A.

Modify L^* or D^*: Whatever modifications $\mathcal{E}$ performs on the real-world L^* and D^*, the simulator does the same modifications to L^{**} and D^{**}. For example, if $\mathcal{E}$ moves an entry from one record to another, $\mathcal{S}$ does the same. In this case, $\mathcal{S}$ also calls $\mathcal{F}_{\text{DDUP}}$ with `move-message`, which has the following effect: On receiving an input (`move-message`, j_1, j_2) from $\mathcal{S}$, if entries with keys j_1 and j_2 in L, say, $\langle j_1, (P_i, M_1^*)\rangle$ and $\langle j_2, (P_i, M_2^*)\rangle$, have the same userid P_i, then replace the entry corresponding to key j_2 in L by $\langle j_2, (P_i, M_1^*)\rangle$, remove j_2 from the indexset corresponding to key M_2^* in D, and insert j_2 into the indexset corresponding to key M_1^* in D. If the userids P_{i1} and P_{i2} are not the same in entries corresponding to j_1 and j_2 respectively in L then replace the entry corresponding to j_2 in L by $\langle j_2, (P_{i2}, \perp)\rangle$.

If $\mathcal{E}$ modifies with a brand new value, that same brand new value is used on L^{**} and/or D^{**}. In this case $\mathcal{S}$ calls $\mathcal{F}_{\text{DDUP}}$ with `corrupt-message` which has the following effect: replace any entry with key j (and userid P_i) in L by $\langle j, (P_i, \perp)\rangle$.

6.4 Experiment EXPT_1

Experiment EXPT_1 is different from EXPT_0 in the way $\mathcal{S}$ generates c^*, which we already showed is matched with a particular M^*. This does not include the case where $\mathcal{E}$ (or $\mathcal{A}$) actually modified a ciphertext in D^{**}, in which case the corresponding entry in L had been changed to $\perp$ (see `corrupt-message`).

Instead of generating c^* at random (as in EXPT_0), the simulator having merged with $\mathcal{F}_{\text{DDUP}}$ knows the M^* associated with it. So, it first calls the random oracle on M^* to get h^* (and note there is already an h^* associated with it in EXPT_0, so it is exactly this h^*). Next it calls the XTS-scheme in the public random permutation model (see section 4 and 5), with whitening key h^* and plaintext M^* to get a c^*.

Proof: We now show that experiments EXPT_0 and EXPT_1 are distinguishable by $\mathcal{E}$ with only negligible probability . First note that while the adversary $\mathcal{E}$ can modify the ciphertexts in the dictionaries, any new ciphertexts lead to corrupt-message invocation, which returns $\perp$ to the user (and hence to $\mathcal{E}$, as the user reports back to the environment). Thus, we are in the chosen-plaintext security model (CPA) for message privacy of encryption mode (in the public random permutation model). Then by results mentioned in section 5.3, the distinguishing probability of $\mathcal{E}$ is at most

$$(q_1 + 2 * q_1 * m + 2 * m(m+1)) * 2^{-n},$$

where m is the number of blocks in M, $\mathcal{E}$ makes at most q_1 queries to the public random permutation. □

6.5 Experiment EXPT_2

While in experiment exp_1, $\mathcal{S}$ marks and returns some entries with $\perp$ based on adversarial actions, in experiment EXPT_2 such markings are removed, and instead will be inferred (when returning to a party on a `read` invocation) just as in the real world scheme. Recall, the following is the behavior

of the real world scheme on a `read` invocation, and EXPT_2 will also do the same (using L^{**} and D^{**} instead of L^* and D^*):

- On receiving an input $(\texttt{read}, P_i, j)$ from P_i,

  1. Search the key j in the dictionary L^{**}. If there is no entry, return $\perp$. If there is an entry, say (P_{i^*}, ℓ, τ), then if $i^* \neq i$, return $\perp$. If the userid matches, i.e. $i^* = i$, then fetch key k_i for P_i using *simulation of* $\mathcal{F}_{\text{KS}}$.
  2. Next obtain h by AES-decrypting τ with key k_i.
  3. Search for key ℓ in D^{**}, and let the corresponding value be c^*. Obtain $M = \text{XTS-DEC}(k, \mathsf{h}, c^*)$.
  4. Using the *simulation of* random oracle, obtain $\mathsf{h}^* = \mathcal{F}_{\text{RO}}(M)$.
  5. If $\mathsf{h}* \neq \mathsf{h}$ return $\perp$. Otherwise, return M.

From description of EXPT_0, we note that $\perp$ is returned in the following cases:

- on receiving $(\texttt{read}, P_i, j)$ from $\mathcal{E}$,

  (i) if there is no entry $\langle j, (P_i, M^*)\rangle$ in dictionary L return $\perp$ to P_i. Note that since L and L^{**} are consistent, this is equivalent to checking: if there is no entry $\langle j, (P_i, \ell, \tau)\rangle$ in L^{**} then return $\perp$.

  (ii) If adversary $\mathcal{E}$ takes a τ_1 from L^{**} entry $\langle j_1, (P_{i_1}, \ell_1, \tau_1)\rangle$ and replaces the τ in entry corresponding to j, say $\langle j, (P_i, \ell, \tau)\rangle$ with τ_1, and if $P_{i_1} \neq P_i$ then return $\perp$.

  (iii) If adversary $\mathcal{E}$ takes a τ_1 from L^{**} entry $\langle j_1, (P_i, \ell_1, \tau_1)\rangle$ and replaces the τ in entry corresponding to j, say $\langle j, (P_i, \ell, \tau)\rangle$ with τ_1, but $\ell_1 \neq \ell$, and entry corresponding to ℓ in D^{**} is untouched, then return $\perp$.

  (iv) If adversary $\mathcal{E}$ takes a τ_1 from L^{**} entry $\langle j_1, (P_i, \ell_1, \tau_1)\rangle$ and replaces the τ in entry corresponding to j , say $\langle j, (P_i, \ell, \tau)\rangle$ with τ_1, but $\ell_1 \neq \ell$, and entry corresponding to ℓ in D^{**} is altered and has a c that is different from the c in the (original, if touched) entry corresponding to ℓ_1, then return $\perp$.

  (v) If adversary $\mathcal{E}$ has replaced τ in entry corresponding to j in L^{**} with a completely new value (i.e. not copied as in above cases) then return $\perp$.

  (vi) If entry corresponding to j in L^{**} is $\langle j, (P_i, \ell, \tau)\rangle$, and adversary has replaced the entry corresponding to ℓ in D^{**} with a new c (i.e. not copied from elsewhere as in above cases) then return $\perp$.

We now prove that these are exactly the cases caught with high probability by steps 1-5 of the real world scheme described above.

Proof: Recall that an uncompromised ciphertext is obtained as follows from a plaintext M: (a) $\mathsf{h} = \mathcal{F}_{\text{RO}}(M)$, (b) $c = \text{XTS}(k, \mathsf{h}, M)$. Recall, k is a random but public key. Moreover, the authentication tag τ is obtained as $\tau = \text{AES}(k_i, \mathsf{h})$, where k_i is the key of P_i. In the following we will let q_0 be the total number of messages stored in $\mathcal{F}_{\text{DDUP}}$. First note that any τ^* which is different from any τ_i ever encrypted by $\mathcal{S}$ using k_i, will lead to an h^* which is uniformly random (modulo not being the same as any h_i encrypted with k_i). Since the number of encryptions under k_i is at most q_0, then h^* takes any particular value x with probability at most $1/(2^n - q_0)$.

- Thus, in cases (ii) and (v) above, we have the following analysis: Step 3 of the real scheme above obtains $M = \text{XTS-Dec}(k, \mathsf{h}^*, c^*)$. Since, h^* is different from the one actually used to obtain c^* (during encryption), we can focus on any block of c^*, say block one, and then from the description of XTS we have that $M_1 = g(\mathsf{h}^*, 1) \oplus \text{AES-inverse}(k, c_1^* \oplus g(\mathsf{h}^*, 1))$, where g is the XOR-universal hash function used in XTS (see Section 5). The probability that $c_1^* \oplus g(\mathsf{h}^*, 1)$ is equal to any block earlier queried of AES-inverse$(k, \cdot)$ is at most $(q_1 + m * q_0) * 1/(2^n - q_0)$. Thus, excluding this probability, the value M_1 will be a fresh random value (excluding, earlier $(q_1 + m * q_0)$ values, as AES is a permutation). Thus, the probability that M was ever used in a call to the random oracle is at most $(q_0 + q_1) * 1/(2^n - (q_1 + m * q_0))$. Thus, the probability that $\mathsf{h} = \mathcal{F}_{\text{RO}}(M)$ is same as h^* is (by union bound) at most

$$(q_1 + m * q_0) * 1/(2^n - q_0) + (q_0 + q_1) * 1/(2^n - (q_1 + m * q_0)) + 1/2^n$$

- Case (iii) is similarly analyzed as cases (ii) and (v) above.

- In cases (iv) and (vi), the h maybe correct, but c^* has been altered. Since, XTS is an encryption scheme, the decryption will necessarily lead to a different plaintext M^* than use to obtain the original c, and the original h (which is still same in cases (iv) and (vi)). The probability that a different M^* on invocation of the random oracle obtains the same h is then at most $1/2^n$. This $\mathsf{h} = \mathsf{h}^*$ with probability at most $1/2^n$.

This completes the proof of indistinguishability of the ideal world and the real world by the environment $\mathcal{E}$, noting that EXPT_2 is same as the real world. The distinguishing probability is at most

$$(q_1 + 2 * q_1 * m + 2 * m(m+1)) * 2^{-n} + (q_1 + m * q_0) * 1/(2^n - q_0) + (q_0 + q_1) * 1/(2^n - (q_1 + m * q_0)) + 1/2^n$$

where q_0 is the number of messages stored, q_1 is the number of AES oracle calls the Adversary performs with the public random key k, and m is the block size of each message. □

References

[1] M. Bellare, A. Desai, E. Jokipii, and P. Rogaway. A concrete security treatment of symmetric encryption. In *38th FOCS*, pages 394–403. IEEE Computer Society Press, Oct. 1997.

[2] S. M. Bellovin and M. Merritt. Augmented encrypted key exchange: A password-based protocol secure against dictionary attacks and password file compromise. In D. E. Denning, R. Pyle, R. Ganesan, R. S. Sandhu, and V. Ashby, editors, *ACM CCS 93*, pages 244–250. ACM Press, Nov. 1993.

[3] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, Oct. 2001.

[4] R. Canetti, S. Halevi, J. Katz, Y. Lindell, and P. D. MacKenzie. Universally composable password-based key exchange. In R. Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 404–421. Springer, Heidelberg, May 2005.

[5] C. Gentry, P. MacKenzie, and Z. Ramzan. A method for making password-based key exchange resilient to server compromise. In C. Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 142–159. Springer, Heidelberg, Aug. 2006.

[6] C. S. Jutla. Encryption modes with almost free message integrity. In B. Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 529–544. Springer, Heidelberg, May 2001.

[7] C. S. Jutla. Encryption modes with almost free message integrity. *Journal of Cryptology*, 21(4):547–578, Oct. 2008.

[8] C. S. Jutla. Authenticated encryption mode IAPM using sha-3's public random permutation. *IACR Cryptol. ePrint Arch.*, 2018:128, 2018.

[9] H. Krawczyk. LFSR-based hashing and authentication. In Y. Desmedt, editor, *CRYPTO'94*, volume 839 of *LNCS*, pages 129–139. Springer, Heidelberg, Aug. 1994.

[10] H. Krawczyk. The order of encryption and authentication for protecting communications (or: How secure is SSL?). In J. Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 310–331. Springer, Heidelberg, Aug. 2001.

[11] K. Kurosawa. Power of a public random permutation and its application to authenticated encryption. *IEEE Transactions on Information Theory*, 56(10):5366–5374, 2010.

[12] J. Liu, N. Asokan, and B. Pinkas. Secure deduplication of encrypted data without additional independent servers. In I. Ray, N. Li, and C. Kruegel, editors, *ACM CCS 2015*, pages 874–885. ACM Press, Oct. 2015.

[13] P. Rogaway, M. Bellare, J. Black, and T. Krovetz. OCB: A block-cipher mode of operation for efficient authenticated encryption. In M. K. Reiter and P. Samarati, editors, *ACM CCS 2001*, pages 196–205. ACM Press, Nov. 2001.