Using Timed-Release Cryptography to Mitigate The Preservation Risk of Embargo Periods

Rabia Haq
Department of Computer Science
Old Dominion University
Norfolk, VA, 23529
rhaq@cs.odu.edu

Michael L. Nelson
Department of Computer Science
Old Dominion University
Norfolk, VA, 23529
mln@cs.odu.edu

ABSTRACT

Due to temporary access restrictions, embargoed data cannot be refreshed to unlimited parties during the embargo time interval. A solution to mitigate the risk of data loss has been developed that uses a data dissemination framework, the Timed-Locked Embargo Framework (TLEF), that allows data refreshing of encrypted instances of embargoed content in an open, unrestricted scholarly community. TLEF exploits implementations of existing technologies to "time-lock" data using timed-release cryptology so that TLEF can be deployed as digital resources encoded in a complex object format suitable for metadata harvesting. The framework successfully demonstrates dynamic record identification, time-lock puzzle encryption, encapsulation and dissemination as XML documents. We implement TLEF and provide a quantitative analysis of its successful data harvest of time-locked embargoed data with minimum time overhead without compromising data security and integrity.

Categories and Subject Descriptors

H.3.7 [**Information Storage and Retrieval**]: [Digital Libraries]

General Terms

Algorithms, Measurement, Design, Performance

Keywords

Repositories, Time Lock, Timed Release, Cryptography

1. INTRODUCTION

The traditional subscription-based journal model provides information at a cost, making it difficult to afford, especially by researchers in developing countries. This access model has served as a catalyst for the Open Access (OA) movement, which is aimed at providing access to full-text journal articles online toll-free. Advocates of OA argue the subscription-based journal access system hinders free and open flow of ideas and information, while proponents of the traditional system argue that it governs and manages the flow of information, and ensures the maintenance of standard and structure of this information [16]. A balance needs to be achieved in order to ensure continuance of flow of scholarly information and coverage of publication costs by integrating the advantages contained in both these access models.

Embargoed access (i.e., Romeo color "yellow" [1]) to academic material is a hybrid of the restricted, traditional access model and the toll-free, Open Access model. It is a temporary restriction imposed by the publisher on the full-text availability of the latest issues of a journal for a certain time-period, for example, two years, while the economic value of the journal is extracted. Individuals and institutions would have to subscribe to the journal in order to access the latest issues, while the previous issues are available to digital repositories and individuals without subscription cost. This access model has been formulated and adopted by various journals in order to cover publication costs while supporting easy information access [18], a factor that was lacking in the OA cost-recovery model. This allows the publishers to generate revenue from their subscription business during the embargo period as well as include the journal articles in the aggregated databases to improve research accessibility to the scholarly community.

No single digital preservation strategy is appropriate for all data types [12]. The fundamental idea of digital preservation is to not rely on a single method of digital preservation, but to utilize various strategies, such as data refreshing, migration and emulation to ensure data longevity and integrity. Data refreshing is the copying of bits to different systems that are distributed to various heterogeneous locations so that, if one copy is destroyed by accidental or malicious means, other copies can be accessed to recover the local copy of the content. This method of digital preservation can be applied to preserve content that is freely accessible online, but data that has temporarily restricted, embargoed access cannot be preserved by this method.

During the embargo period, users and researchers that have not subscribed to the embargoed journal are unable to access this restricted information. Therefore, there is a limit to the number of people who can refresh this embargoed information, placing it at a risk during this embargo time interval, which we term the "Preservation Risk Interval". We mitigate this risk of data loss associated with embargoed information by suggesting a data dissemination model

JCDL'09, June 15–19, 2009, Austin, Texas, USA.

that allows data refreshing of embargoed content in an open, unrestricted scholarly community. This is in contrast with trusted repository models such as CLOCKSS[1] and Poritco[2]. The Timed-Locked Embargo Framework (TLEF) has been developed by exploiting implementations of existing technologies to "time-lock" and encrypt data using Timed-Release Cryptology [15] so that it can be deployed as digital resources encoded in the MPEG-21 Digital Item Description Language (DIDL) complex object format [3] to harvesters interested in harvesting a local copy of content by utilizing The Open Archives Initiative Protocol for Metadata Harvesting (OAI-PMH) [10].

2. BACKGROUND

2.1 Timed-Release Crypto

Conventional key-generation algorithms, such as the RSA encryption algorithm [14, 8], have encryption keys that are a composite of two large prime numbers, p and q, of approximately equal size

$$n = pq \tag{1}$$

where n is used to produce both the public and private keys. It consists of modular arithmetic used to calculate the factors of large numbers whose complexity is dependent on the size of the numbers used. The complexity of the puzzle generated is just the complexity of the factoring problem. If brute force were to be applied to break the key, estimated computation time can be reduced by k, by dedicating k computers to compute the product in parallel. For timed release crypto, a non-parallelizable key encryption system is proposed,

$$t = TS \tag{2}$$

is calculated, T being the amount of time in seconds for which the puzzle is to be time-locked and S represents the computational power of the machine. A random key can be generated using a conventional cryptosystem, where the key may consist of enough bits to be considered sufficiently complicated, and cannot be easily examined and broken. Calculating inputs n (from equation 1), and a random a, where $1 < a < n$, the encryption key can be encrypted as

$$Key\ encryption = Key + a^{2^t}\ (mod\ n). \tag{3}$$

The output of the time-lock puzzle is thus (n, a, t, encrypted key, encrypted resource). Input variables, such as p and q that were used during the computation of the puzzle, are destroyed.

Without p and q, searching for the key to solve a puzzle that is encrypted using this method is impractical, rendering brute force methods of breaking the encryption key non-parallelizable. The most efficient method of retrieving the encryption key would be to calculate the variable

$$b = a^{2^t}\ (mod\ n) \tag{4}$$

initially used during key encryption. This can be achieved by sequentially performing t squarings on the value a. This requires a dedicated computer performing continuous, sequential computation for approximately t time units to decrypt the key and thus break the encryption. This encryption method can be applied with various, carefully selected values of time unit t to encapsulate the resource for a predetermined amount of time, which is equivalent to the resource embargo period, ensuring that the resource is not released until the desired computation time t has elapsed.

Another existing approach to timed-release encryption is the use of a third party intervention, or Trusted Agents (TA). The use of TAs does not require usage of the receiver's computation resources. A time-server(s) intercedes as the negotiator between the content source and the client who is trying to access the content. During content encryption, a pair of private keys is produced. The client receives an encrypted instance of the content, as well as one of the private keys, later used for identity verification. The other key from the pair, required for successful decryption and accessibility of content, is received from the time-server only after the required embargo period has elapsed [6, 4]. Although this approach is possible within TLEF, we do not require the use of TAs.

2.2 OAI-PMH Resource Harvesting

In prior work, we have extended the OAI-PMH framework to allow for resource harvesting by using complex object metadata formats to include the resource (e.g., PDF, JPEG) either by reference (URI) or by value (base64 encoded) [19]. The Apache `mod_oai` module has been designed to implement OAI-PMH resource harvesting for all files and URIs on a webserver where the baseURL is the hostname with "/modoai" appended [17]. It implements all of the OAI-PMH selective harvesting features such as the *from-until* parameters and *set* parameters. It supports the *oai_dc* metadata prefix that returns technical metadata about the valid URIs. The *oai_didl* metadata prefix is used to return metadata and the resource itself and is based on a profile of the MPEG-21 Digital Item Declaration Language [2].

The MPEG-21 DID Abstract Model is complex, but offers the flexibility to accurately describe and encompass a digital object and related metadata. The *resource* entity is the actual, identifiable datastream, either digital or non-digital, such as a picture, a video clip, a text document, or a painting in a museum. A *component* entity is used to group together *resources* and related information about these *resources* wrapped in a *descriptor/statement* entity construct. A DID *item* is the point of entry for information pertaining to one *resource*. It groups together one or more *descriptor/statement* constructs. *Items* may contain other *items*, each representing one instance of the *resource*. Figure 1 is an abstracted view of a resource expressed in MPEG-21 DIDL.

3. THE PRESERVATION RISK INTERVAL

Emerging preservation systems are drifting towards open architectures where various distributed systems may be networked together to trigger notifications and preservation techniques, such as replication, migration and emulation. This innovative approach signifies that preservation of scholarly material via replication is no longer limited to supporting institutions, but is diffusing to a much broader and generic community of interest. This requires a more robust content dissemination system that can be expanded to include dissemination of embargoed content.

[1] `www.clockss.org`
[2] `www.portico.org`

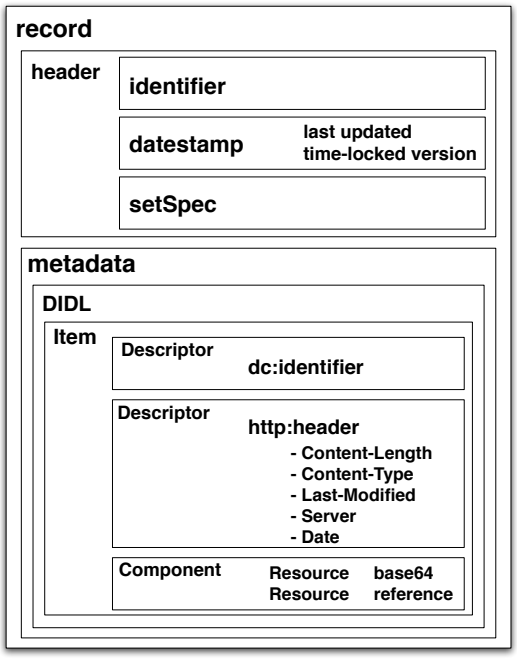

Figure 1: Structure of a resource expressed in MPEG-21 DIDL Abstract Model.

i	time unit
$R(i)$	number of records in repository at time i
R_{update}	record updates to repository per time unit
$R(0)$	initial number of records in repository at $i = 0$
$H(i)$	number of records at harvester at time i
H_{update}	harvester update frequency per time unit. $H_{update} = 0.5$ means data harvest every 2 units
$H(0)$	initial number of records at harvester at $i = 0$
$publisher_{start}$	publisher-imposed global datestamp after which all records published in the are repository embargoed
$embargo_{start}$	creation date for embargoed record
$embargo_{end}$	end date for record embargo period, after time period $embargo_{length}$ has elapsed
$embargo_{length}$	embargo time period for each record
$embargo_{decrement}$	number of times the record time-lock is decreased (i.e., total number of record updates)
r_x	record with embargo period = x, with x = 0 as an unlocked record

Table 1: Variable definitions.

$a_3\ b_3\ c_3\ d_3\ e_3$	$i = 0$	time $i = 0$ with $R(0) = 5$ records
$a_2\ b_2\ c_2\ d_2\ e_2\ f_3$	$i = 1$	5 records updated, 1 new record published
$a_1\ b_1\ c_1\ d_1\ e_1\ f_2\ g_3$	$i = 2$	6 records updated, 1 new record published
$a_0\ b_0\ c_0\ d_0\ e_0\ f_1\ g_2\ h_3$	$i = 3$	unlocked records at $i = embargo_{length} = 3$
$\vdots$		
$a_0\ b_0\ c_0 \ldots p_0\ q_0\ r_1\ s_2\ t_3$	$i = 15$	$R_{update} \times embargo_{length}$ = 3 records locked $\forall i > embargo_{length}$

Table 2: Behavior of an active repository with embargoed records, with $R(0)$=5 records, $embargo_{length}$=3 months and R_{update}=1 record.

Consider a repository of journal articles where the older issues are open access but the latest issues are embargoed. The records contained in the latest issue need to be identified and time-locked before data dissemination. The embargo period of the records would begin at the time of article publication. The repository provides a predetermined number of instance updates of each record with successively weaker time-locks. Thus, every record update is a weaker time-locked encryption that requires less time to break and consequently access the data. The last instance update would be an unlocked version of the record at the end of the embargo time period.

Time-locking embargoed records in an institutional repository for the purpose of digital preservation during data dissemination introduces an overhead. This is the time required to unlock the records in order to effectively access a decrypted version of the records. The establishment of nomenclature, as described in Table 1, is thus imperative for assessing the amount of time required to effectively unlock and access the time-locked records held in the repository.

Let us assume the existence of a repository R that consists of a set of records such as $R = \{a_w, b_x ... k_z\}$, where the subscript for each record instance is the remaining embargo time period for that record. This repository R contains an initial number of $R(0)$ records at time $i = 0$. The records are initially time-locked for a total embargo period $embargo_{length}$, with a predetermined number of $embargo_{decrement}$ embargoed record updates. The embargo period begins at the time of record publication, which is at datestamp $embargo_{start}$. This repository is updated at every time unit i when the existing records in the repository are updated with a new embargo period time-lock. All records in the repository will be updated with a new embargo period with every repository update. The repository also grows linearly, by R_{update} records with every update.

For example, consider a scenario where repository R has: $R(0) = 5$, $R_{update} = 1$ (one new record is published with every repository update), total embargo period for each record $embargo_{length} = 3$ months, and $embargo_{decrement} = 3$ (the number of times the record time-lock is decreased, before an unlocked instance is published). Table 2 describes the number of records and their embargo period with respect to time.

As the repository begins with an initial number of records $R(0)$, which have been time-locked for time period $embargo_{length}$, the total amount of time spent on unlocking the records would differ with the time elapsed (i). If the elapsed time is less than the time-lock period $embargo_{length}$, then the repository would not yet contain any unlocked records. The amount of time required to unlock the records would then be the sum of the embargo period of the initial $R(0)$ number of records and the records added to R since $i = 0$ would be ($\forall embargo_{length} - i > 0$):

$$\mathrm{R}(0)(\mathrm{embargo}_{length} - i) + \sum_{k=0}^{i-1} \mathrm{R}_{update}(embargo_{length} - k). \quad (5)$$

If the elapsed time is more than the embargo period, the

Repository		Time	Harvester
$a_3\ b_3\ c_3$	$\longleftarrow$	$i=0$	$a_3\ b_3\ c_3$
$a_2\ b_2\ c_2\ d_3$		$i=1$	$a_3\ b_3\ c_3$
$a_1\ b_1\ c_1\ d_2\ e_3$	$\longleftarrow$	$i=2$	$a_1\ b_1\ c_1 d_2\ e_3$
$a_0\ b_0\ c_0\ d_1\ e_2\ f_3$		$i=3$	$a_1\ b_1\ c_1 d_2\ e_3$
$a_0\ b_0\ c_0\ d_0\ e_1\ f_2\ g_3$	$\longleftarrow$	$i=4$	$a_0\ b_0\ c_0\ d_0\ e_1\ f_2\ g_3$
$a_0\ b_0\ c_0\ d_0\ e_0\ f_1\ g_2\ h_3$		$i=5$	$a_0\ b_0\ c_0\ d_0\ e_1\ f_2\ g_3$
$\emptyset$	$\longleftarrow$	$i=6$	$a_0\ b_0\ c_0\ d_0\ e_1\ f_2\ g_3$
$\emptyset$		$i=7$	$a_0\ b_0\ c_0\ d_0\ e_1\ f_2\ g_3$
$\emptyset$	$\longleftarrow$	$i=8$	$a_0\ b_0\ c_0\ d_0\ e_1\ f_2\ g_3$

Table 3: Repository with $R(0) = 3$ records, $embargo_{length} = 3$ and $R_{update} = 1$. Harvester with $H_{update} = 0.5$. Repository dies at $i = 6$.

initial $R(0)$ records would be unlocked, requiring computation on the time-locked records added to the repository since time i. An embargo period less than or equal to zero refers to an unlocked record. Thus, the computation time required to access these records would be ($\forall embargo_{length} - i > 0$):

$$\sum_{k=0}^{embargo_{length}-1} R_{update}(embargo_{length} - k). \qquad (6)$$

Therefore, at time $i = 15$, the computation time required to access all the records available in the repository, since $3 - 15 < 0$, $= r_1 + s_2 + t_3$, there will be $embargo_{length}$= 3 time-locked records, embargoed for time period 1, 2 and 3 respectively, which will require $1 + 2 + 3 = 6$ months of dedicated computation to be accessed.

Table 3 shows a repository with $R(0) = 3$, $embargo_{length} = 3$ and $R_{update} = 1$. The repository dies at $i = 6$ and does not come back online. A harvester with $H_{update} = 0.5$ has been harvesting the repository ("$\longleftarrow$" indicates a harvesting operation). The harvester has unlocked versions of records a, b, c, and once it determines the repository is not coming back online it can begin to break the encryption on records e, f, g, with only record h permanently lost. If the repository had disseminated only unembargoed records, then e, f, g would have also been lost to the harvester.

The repository example discussed here can be further analyzed to formulate an embargoed record identification method. The active repository contains records with publisher-imposed embargo time period of $embargo_{length} = 3$ months, and the repository mandates $embargo_{decrement} = 3$ updates of each embargoed record, then the embargoed records are updated with a weaker timelock every $\frac{embargo_{length}}{embargo_{decrement}} = 1$ month. This results in a total of three instance updates. The embargo period inception of each embargoed record is from the publication, or creation date, which is $embargo_{start}$. Therefore, the end of the embargo period, $embargo_{end}$ for a record, can be easily projected by adding the embargo period $embargo_{length}$ to the datestamp $embargo_{start}$. A record under embargo can be identified by comparing the datestamp at the time of data dissemination with the publisher-imposed, global $publisher_{start}$ datestamp and the projected $embargo_{end}$ datestamp. If the record is identified as under embargo, then the remaining time of the embargo period of that record is calculated. A predetermined number of $embargo_{decrement}$ updates of the record (three in this example) with decreasing time-lock are provided. The remaining embargo period of the record is also used to calculate the number of updates of the record (e.g., second of three updates). This determines the complexity of the record encryption. Therefore, a decrease in the remaining embargo period results in the creation of a weaker encryption of the record that would require less computation time to break the encryption. Once the required embargo period has elapsed, an unencrypted instance of the record is created during content dissemination. For instance, if a record r_3 is published on Wed, 19 Sep 2007 09:58:19 GMT, the next update would be available after one month has elapsed, on Fri, 19 Oct 2007 09:58:19 GMT. An unlocked instance of the record would be available after the embargo period has elapsed, which would end on Wed, 19 Dec 2007 09:58:19 GMT. Algorithm 1 illustrates the embargo record identification process based on datestamp comparison of the record and the start of the embargo time period.

Algorithm 1 The Embargoed Record Identification process invoked during content dissemination

```
1: for all records do
2:    if (embargo_start > publisher_start) and (embargo_start < embargo_end) then
3:       calculate remaining embargo_length for the record
4:       calculate record instance update
5:       calculate encryption complexity
6:       calculate new record datestamp
7:    end if
8: end for
```

4. IMPLEMENTATION IN MOD_OAI

In response to a GetRecord or ListRecords `mod_oai` HTTP request with metadata prefix oai_didl, the embargoed identification, encryption and encapsulation components incorporated in the Apache module are invoked. Algorithm 2 describes the sequence and interaction of these components.

Algorithm 2 Interaction of various record data and metadata creation components

```
1: if (Request Verb = ListRecords) or (Request Verb = GetRecord) then
2:    Records Index Creation
3:    for all records do
4:       Preservation Metadata Creation
5:       Metadata Encapsulation in MPEG-21 DIDL data item
6:       if record is under Embargo then
7:          Dynamic Embargoed Record Identification
8:          Record Datestamp modification
9:          Dynamic Embargoed Record Encryption
10:         Dynamic Embargoed Record Encapsulation
11:      end if
12:      if Include Resource by-value then
13:         Resource Encapsulation in MPEG-21 DIDL component
14:         if record is under Embargo then
15:            Resource MD5 hash Encapsulation in MPEG-21 DIDL data descriptor
16:            Puzzle Parameters Encapsulation in MPEG-21 DIDL data descriptor
17:         end if
18:      else
19:         Include Resource reference in MPEG-21 DIDL component
20:      end if
21:   end for
22: end if
```

4.1 Dynamic Embargoed Record Identification

The general solution for embargoed record identification has been further developed to handle dynamic record iden-

tification. Algorithm 3 has been incorporated in `mod_oai` to identify records that are under embargo.

Algorithm 3 Dynamic Embargoed Record Identification Process incorporated within `mod_oai`

1: $publisher_{start}$ = Global Zulu date when embargo period for each record begins
2: $embargo_{length}$ = 365 {embargo time period for each record in days}
3: $embargo_{decrement}$ = 12 {number of embargoed record updates}
4: $lockStart$ = startDate in unix seconds
5: $lockDuration = embargo_{length}$ in seconds
6: **for all** *records* **do**
7: $currentTime$ = current date in unix seconds {input current date from the system}
8: $fileTime$ = current record modified date,$embargo_{start}$, in unix seconds
9: **if** ($fileTime < lockStart$) or ($fileTime \leq (currentTime - lockDuration)$) **then**
10: the record is not under embargo
11: **else**
12: **if** (($fileTime + lockDuration) > currentTime$) **then**
13: $noOfSecsInInterval = (lockDuration/p)$
14: $elapsedTimeFraction = (currentTime - fileTime/lockDuration)$
15: $intervalNo = elapsedTimeFraction * p$
16: $elapsedLockTime = intervalNo * noOfSecsInInterval$
17: $intervalsLeft = p$ - $intervalNo$
18: $lockTimeLeft = intervalsLeft * noOfSecsInInterval$ {lockTimeLeft is used to linearly interpolate the anticipated computation timeUnit}
{calculate new timestamp of the file according to last interval update}
19: $newTimestamp = elapsedLockTime + fileTime$ {calculating next update}
20: $nextUpdate = (intervalNo + 1) * noOfSecsInInterval$
21: $nextTimestamp = nextUpdate + fileTime$
22: convert $newTimestamp$ and $nextTimestamp$ integer variables from seconds to Zulu time
23: print $intervalNo$ as the current version of the record, out of a total of $embargo_{decrement}$ versions
24: print the $nextTimestamp$ in Zulu time as the next anticipated update timestamp
25: print $lockTimeLeft$ as the anticipated computation time required to unlock the time-locked puzzle
26: **end if**
27: **end if**
28: **end for**

The publisher-imposed start date for the embargo period of each record has been included in the `mod_oai` configuration file as a variable that can be modified and set by the user accordingly. The duration of the embargo period, with a granularity of days, has also been established as a known variable. The number of instance updates, or intervals during these updates, is also a known variable and is included in the configuration file as required input. These variables can be adjusted to modify the effective embargo time period for the records and the number of instance updates desired for each record under embargo. Algorithm 3 and remaining examples of embargoed record identification and encryption are based upon an $embargo_{length}$ = 365 days, with $embargo_{decrement}$ = 12 instance updates for each record. A later version of each record would correspond to a less complex time-lock puzzle that would require less computation time to break the encryption.

Algorithm 3 computes time values in unix seconds to enable mathematical manipulation on time values and then converts the time in seconds back to ISO 8061 time before being included in XML output. After identifying a record as under embargo, the algorithm first calculates the time fraction that has elapsed since the start of the embargo period. This fraction is then converted into an integer value that represents the current number of the instance update. This interval number is then used to determine the effective remaining lock-time, on which the complexity of the time-lock puzzle is dependent.

The remaining lock-time in seconds is the computation time it should take to access the record once it has been time-locked. The timed-release cryptology algorithm requires an integer input value that determines the complexity of the record encryption. Thus, various tests, described during system evaluation in section 5, have been conducted to linearly interpolate and map the remaining lock-time with an appropriate integer value that is used as the input for the embargoed record encryption algorithm.

4.2 Dynamic Embargoed Record Encryption

The LCS35 Time Capsule Crypto-Puzzle [13], created in 1999, was designed to take about 35 years of linear computation to solve. The puzzle takes into consideration Moore's Law [11], and anticipates that computational power will double approximately every two years. The puzzle follows the future trend described by SEMATECH National Technology Roadmap for Semiconductors [7] that predicts an exponential increase in processing speed by a factor of 13 from 1999 until 2012. A further increase in speed by a factor of 5 until 2034 has been estimated and taken into account in the algorithm. In order to ensure the computation time of 35 years, it is assumed that a faster computer will be used every year to perform the required sequential computation to break the LCS35 puzzle.

Algorithm 4 Dynamic Embargoed Record Encryption algorithm incorporated within `mod_oai`

1: $squaringsPerTimeunit$ = 3000
2: $secondsPerTimeunit$ = 1800
3: $squaringsFirstTimeunit = secondsPerTimeunit * squaringsPerTimeunit$
4: $t = 0$
5: **for** $i = 1$ till $i \leq timeUnit$ **do**
6: $t = t + squaringsFirstTimeunit$ {the number of squarings depends on timeUnit. t increases linearly}
7: **end for**
8: $primelength$ = 1024 {generating RSA parameters}
9: $twoPower$ = shift left primelength(1)
10: $prand = 3$
11: $qrand = 5$
12: $p = 5$
13: $q = 5$ {5 has maximal order modulo 2^k (see Knuth)}
14: $p = (p^{prand})$ mod $twoPower$
15: p = get next prime of p
16: $q = (q^{qrand})$ mod $twoPower$
17: q = get next prime of q
18: $n = p * q$
19: $pMinus1 = p - 1$
20: $qMinus1 = q - 1$
21: $phi = pMinus1 * qMinus1$
22: $u = (2^t)$ mod phi {Generating final puzzle value w}
23: $w = (2^u)$ mod n {convert the file in Base256}
24: **while** $buffer$ = read a char from file **do**
25: $c = buffer$ {convert character into ascii equivalent integer value}
26: $secret$ = shift left($secret$)
27: $secret = secret + c$
28: **end while**
29: $z = (secret)$xor(w) {print puzzle parameters}
30: print n and t parameters in a string variable
31: print extra information required to break the encryption
32: Base64_encode(z)
33: print the puzzle parameters and Base64 encoded z in XML document

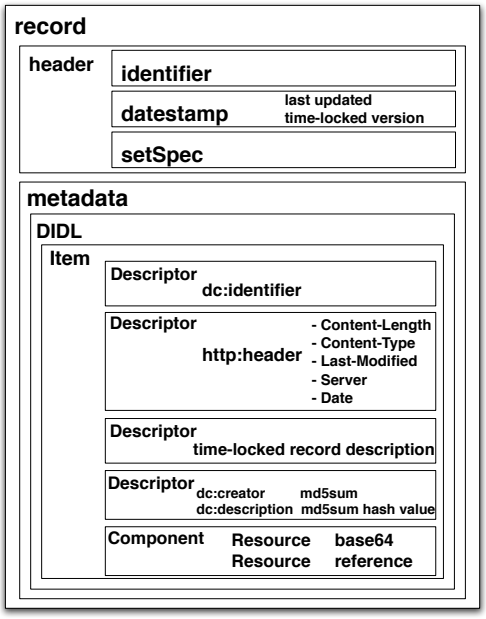

Figure 2: MPEG-21 DIDL structure of an embargoed record

Algorithm 4 has been programmed in the C programming language and incorporated in `mod_oai`. It takes the variable *timeUnit* as input and outputs the time-locked instance of the record along with the puzzle variables n and t required to break the puzzle, along with the instructions on how to break the puzzle. The GMP C library[3] has been used to declare the large numbers produced in the algorithm. The logic of the timed-release cryptographic algorithm has been preserved. It has been amended to accept binary files as input, to permit all files to be accessed and encrypted irrespective of their MIME type. To allow files with various MIME types to be time-locked, it was not possible to append or include the seed value b in the file during data encryption. This implementation does not take Moore's law into account because of the anticipated relatively short embargo periods (e.g., 1-2 years). Instead, near-horizon values are chosen as described in section 5. The MD5 cryptographic hash function has been used to create 32-character hash values of the file to verify the integrity of the file contents upon decryption.

4.3 Dynamic Embargoed Record Encapsulation

The encrypted instance of the embargoed record is encapsulated in MPEG-21 DIDL XML document. This document reflects the appropriate changes in the included metadata to be identified as an encrypted instance of the record. Figure 2 is the resulting DIDL document structure that encapsulates data and metadata pertaining to a record under embargo.

The two occurrences of the last-modified datestamp of the record have been updated in the resulting XML document to reflect the latest embargoed record instance. This datestamp represents the last instance when the record was encrypted with the latest embargo period and has been updated in the DIDL header section, in ISO 8061 [9] date format, and in the http metadata descriptor, in RFC 822 [5] date format. The resource, in either base64 format or referenced by a URI, has been relocked with a weaker time-lock. The MD5 checksum of an unlocked instance of the record has been included in a descriptor in the response to provide sufficient information for integrity verification upon record decryption. It has been encapsulated in a DIDL descriptor element.

An extra descriptor DIDL element is introduced to identify the record as time-locked and encapsulates related information such as the instance of the embargoed record update, the original start of the record embargo, and when an unlocked instance of the record can be anticipated. It also provides the variables and method to break and access the encrypted record, and includes information on the estimated computation hours required to achieve this task.

The `mod_oai` Apache module allows the record resource to be included in the XML document by reference as well as by value. The reference URL pertaining to an encrypted record has been modified to be directed through a script that allows the file accessed to be time-locked before data display. It provides the same information included in the DIDL document required to unlock and access the encrypted record instance. When dereferencing the URI of an embargoed record, the resulting representation displays the resource in decimal values without base64 encoding, which is the original output of the timed-release puzzle. The returned page reiterates the information included in the XML document that is required to decrypt the accessed encrypted record:

> This version of the record is 7 of 12 separate encryptions, each of which is successively easier to break.
> It will take approximately 3650 hours of computation to break this time-lock.
> The next update will be available on 2008-01-16T20:56:15Z.
> Crypto-Puzzle for LCS35 Time Capsule.
> Puzzle parameters (all in decimal): n = 398399 t = 264600000.
> z =
> 313239174518025552773909388461801735302388...
> 893375562056859914777144518879488573607906...
> 742437030171894184996228671834511813009803...
> *(many lines deleted for space)*
>
> To solve the puzzle, first compute $w = 2^{(2^t)} \pmod{n}$.
> Then exclusive-or the result with z.
> (Right-justify the two strings first).
> The result is the secret message (8 bits per character).

5. SELECTING T

The timed-release cryptography algorithm has been designed to take an integer variable *timeUnit* as an input. This *timeUnit* value linearly increases the complexity of the puzzle by increasing the t value in the 2^{2^t} computation performed to break the file time-lock. A puzzle corresponding to a higher t value requires more time to break the encryption. This decryption time can be mapped with $embargo_{length}$, the remaining embargo period of the file that was calculated during dynamic embargoed record identification. Due to the mathematical properties of this cryptography method, the time required to break the time-lock puzzle is directly dependent upon the speed of the processor used to perform the required calculations. An appropriate value of *timeUnit* can be selected to accurately calculate the puzzle complexity and the time required to break and access the encryption in relation to the desired processor speed.

A correlation between the remaining embargo period, the

[3]http://gmplib.org/

t value and $embargo_{length}$ on a particular computer system can be calculated by assimilating the results of the timed-release algorithm. To compute this correlation, the timed-release cryptography algorithm has been executed with increasing values of $timeUnit$ to create puzzles of increasing complexity. These puzzles have then been broken to determine the amount of time required to break the time-lock and subsequently access the encrypted content. A table of increasing $timeUnit$ values and their corresponding decryption time can be created to describe this dependency. This unlock time can then be mapped to $embargo_{length}$, the remaining embargo period for a record, to ensure that the decryption time is no less than $embargo_{length}$.

The linear correlation between $embargo_{length}$, the unlock time required, and tU, the timeUnit value, to be used for record encryption can be described as:

$$f(x) = \frac{embargo_{length}}{tU} \tag{7}$$

where x is the processor speed of the machine utilized for computation. An increase in processor speed x would result in a decrease in $embrgo_{length}$. Puzzles have been created using this timed-release algorithm on various computer systems to establish a linear correlation between $embargo_{length}$ and tU, and to determine and analyze this variant $f(x)$ value differing with each system speed x.

Four x values have been empirically calculated. A Sun Solaris cluster of 31 nodes has been utilized for performance testing during the research. 26 nodes of this cluster have the processing speed of $x = 1.6$ GHz, and five nodes have a processing speed of $x = 1.8$ GHz. Two machines with a computation speed of $x = 1$ GHz and $x = 0.75$ GHz respectively, have also been utilized to compute and compare the time required to break the timed-release cryptography puzzle.

An identical text file was used by these machines as input to create puzzles of varying complexity by using the timed-release algorithm, starting with the input value $timeUnit = 1$. This timeUnit value is used by the timed-release algorithm, as described in section 4.2, to calculate the t value used for data encryption, which remains unchanged throughout the tests performed. As described in Algorithm 4, tU linearly increases the puzzle t value. Each increment in the tU value increases the t value by a value of 5,400,000. Therefore, tU, the timeUnit value used as input in the algorithm as $t = 5400000tU$.

The created time-lock puzzles of varying complexity are then broken to record the amount of decryption time on the particular system. The accumulated values of effective $timeUnit$ values and unlock time $embargo_{length}$ have been plotted to demonstrate a linear correlation between puzzles with increasing complexity and unlock time $embargo_{length}$. All time values have been recorded in seconds time granularity. Figure 3 is a plot of the datapoints gathered by tests performed on the four classes of machines that demonstrate the linear relationship between $embargo_{length}$ and tU.

From the graph in Figure 3 and the corresponding *f(x)* values of the four x classes of machines in Table 4, it can be discerned that the rate of increase of a particular class of machine can be projected from any one selected class of machine. An increase in the rate of change $f(x)$ for a given class of machine is inversely proportional to the computation speed x of that class.

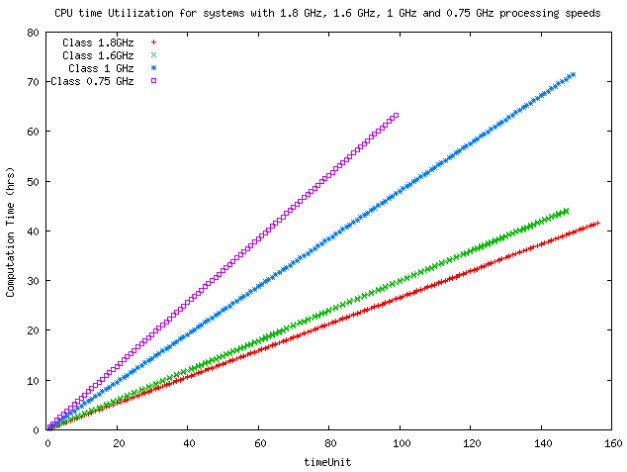

Figure 3: Unlock time of a time-locked puzzle with increasing complexity on four classes of machines

x Class (GHz)	*f(x)* value
0.75	2302.16
1	1727.61
1.6	1079.14
1.8	959.78

Table 4: Corresponding *f(x)* values of the four x classes of machines.

With the known pair of values for $x = 0.75$, the *f(x)* value of a 1GHz machine can be projected using this inverse proportion relation as

$$\frac{0.75}{1} = \frac{f(1)}{2302.16} \tag{8}$$

$$f(1) = 0.75 * 2302.16 \tag{9}$$

$$= 1726.62 \tag{10}$$

The $f(x_k)$ value corresponding to the known x_k computation speed can be projected with:

$$f(x_k) = \frac{f(x_j) * x_j}{x_k}. \tag{11}$$

A calculated *f(x)* value for a particular class of machine that is closer to the actual, calculated *f(x)* value in Table 4 indicates the observance of Moore's Law, whereby concluding that this inverse correlation can be utilized to project the *f(x)* value for various classes of machines.

For instance, if a file is to be locked for two years on a 2.5 GHz machine, an appropriate *f(x)* value corresponding to this processing speed x can be calculated. This *f(x)* value is the rate of change for unlock time with respect to increasing values of time Unit and can be represented as equation 7. Substituting *f(x)* with its calculation in equation 11 gives us the new formula for determining unlock time embargo$_{length}$,

$$embargo_{length} = \frac{f(x_j) * x_j}{x} tU \tag{12}$$

where $f(x_j)$ and x_j are a known *x-f(x)* value pair and x

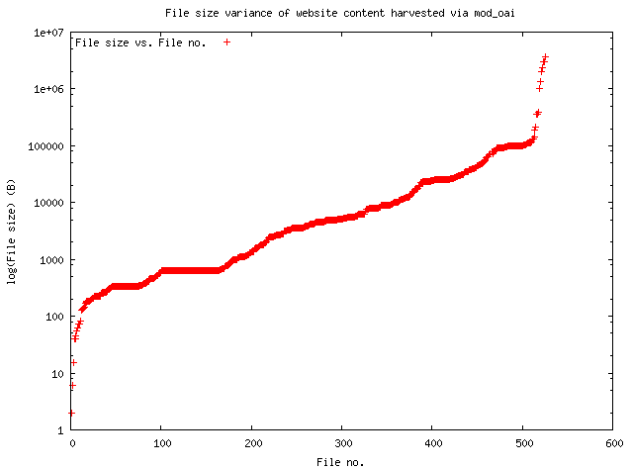

Figure 4: File size variance of website content harvested during `mod_oai` performance testing

is the processing speed for which a rate of change is to be projected. With a known unlock time value, equivalent to $embargo_{length}$, a corresponding $timeUnit$ value to be used as input in the timed-release cryptography algorithm can be calculated as

$$tU = \frac{x}{f(x_j) * x_j} embargo_{length}. \quad (13)$$

Substituting the class 1 value as x_j, and its calculated $f(x_j)$ value would result in

$$tU = \frac{x}{1727.61} embargo_{length} \quad (14)$$

that can be used as a benchmark to project the timeUnit value for the desired class of machine with a known x. This formula can be utilized to calculate tU for known $embargo_{length}$ on the machine selected for data encryption during data dissemination via `mod_oai`. Thus, the tU value required to time-lock a file for $embargo_{length} = 2$ years, which is 63115200 seconds, on machine $x = 2.5$ GHz can be calculated as

$$tU = \frac{2.5}{1727.61} 63115200 \quad (15)$$

$$= 9133. \quad (16)$$

6. EXPERIMENTAL EVALUATION

A modified version of `mod_oai` was tested with a collection of 525 files that total 17.3 MB of data. 63% of the files are text files of various sizes, and the average size of each file is approximately 33 KB. Figure 4 shows the size of each file in relation with the number of files comprising the website. Baseline harvests were done with no time-locks, and then harvests were d one with $embargo_{length}$ of one year. A variable, *modoai_encode_size*, contained in the module's configuration file, determines the maximum allowable file size for inclusion by-value in the resulting XML document. It subsequently controls the size of each resulting XML document during data dissemination. Therefore, differing harvest times of the website with increasing values of *modoai_encode_size* have been collected to analyze the performance of `mod_oai` with increasing quantity of time-locked data in the resulting XML document in response to a `mod_oai` *ListRecords* request.

modoai_encode_size (Bytes)	Responses	None locked (sec)	All locked (sec)
150,000	69	16.2	555
300,000	35	9.5	635
500,000	24	7.6	913
700,000	16	5.4	988
1,000,000	13	4.5	937
5,000,000	6	4.5	3648
10,000,000	6	4.5	10380
15,000,000	6	4.7	10962

Table 5: Wallclock harvest times of website with varied *modoai_encode_size* and embargoed content.

As shown in Table 5 and Figure 5, an increase in the *modoai_encode_size* results in a larger XML document response and fewer number of total XML document responses to a *ListRecords* request. An increase in *modoai_encode_size* results in an increase in the number of files included by-value via base64 data inclusion in the data harvest. With no data under embargo, there is no lock-time overhead, and the increase in the *modoai_encode_size* favors the increase in by-value data inclusion and results in faster data dissemination.

With the entire website content under embargo, an increase in *modoai_encode_size* leads to an increase in the harvest time due to the time overhead required to lock the increasing volume of data to be included in the response as base64 datastream. The entire website content is harvested as a base64 encoded datastream in 3.2 hours, with *modoai_encode_size* set to 10 MB.

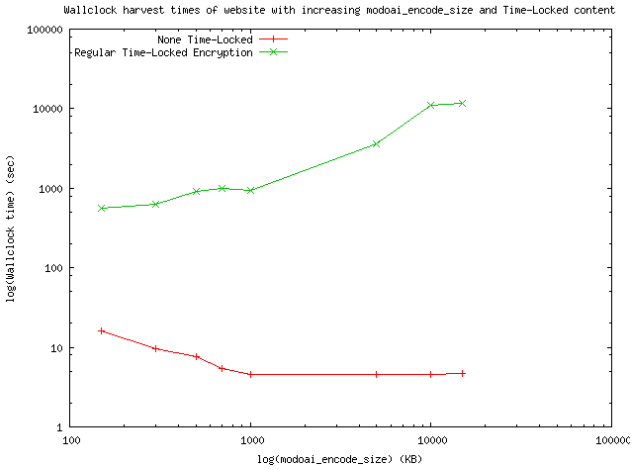

Figure 5: Comparison of harvest times between unlocked and time-locked website content during dynamic data dissemination

An increase in by-value content inclusion during content dissemination leads to an exponential increase in harvest time. The file contents of the website were individually time-locked, with embargo time-period $embargo_{length}$ = one year, to determine that the amount of time required to time-lock

a file is dependent upon the size of the file. Figure 6 is the resulting graph representing the time required to time-lock individual files, in log representation, in relation to their respective file size.

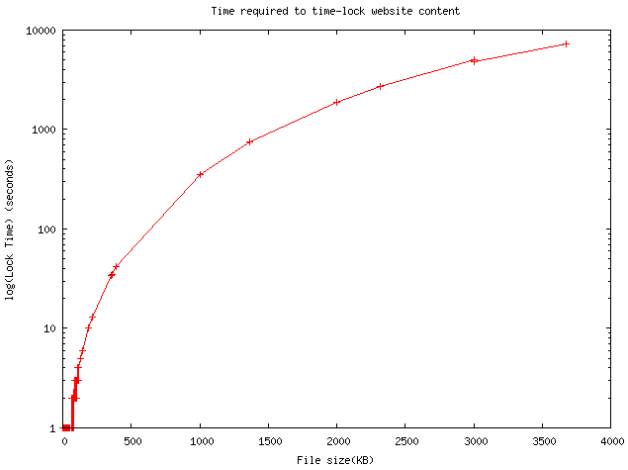

Figure 6: Time required to individually time-lock files contained in the test website

7. CHUNKED ENCRYPTION

Since time-locking was discovered to have a run-time complexity of $O(n^2)$, an optimization was needed to feasibly handle typical file sizes. For example, a file of size 100 KB on a 1.8 GHz machine requires 3 seconds to be time-locked, whereas a 200KB file requires 13 seconds. If the 200KB file is divided into two "chunks" of 100 KB each, the 200 KB file can be time-locked, without parallelism, in six seconds, resulting in a 54% improvement. During the decryption the chunks can be decrypted individually then concatenated to reproduce the original file.

This "chunked" time-lock data model has been included in `mod_oai` to benefit from this observed speedup during during dynamic data dissemination. A chunk size of 10 KB has been selected for implementation in `mod_oai` since we empirically determined it requires about 0.2 CPU seconds for time-lock computation. Any file residing in the website with file size greater than 10 KB that is to be included as an encrypted datastream in the resulting XML document has been divided into 10 KB chunks to achieve faster encryption time. The algorithm still runs in $O(n^2)$ time, but with a more favorable constant. Table 6 and figure 7 shows the results of harvesting with 10KB chunks, as well as the times required to harvest without chunking, and without time-locking. The times are the mean of five separate harvests.

A comparison of the plotted times between regular time-lock encryption and chunked encryption reveals a 70× speedup in harvest times. Even though an exponential increase in harvest time is still observed, the chunked harvest time curve has been pushed to the right for common file sizes, resulting in a slower increase in exponential harvest time. As observed from the graph, with *modoai_encode_size* set to 15 MB, chunked encryption results in a speedup of 70, with the harvest time reduced from 3.2 hours to only 2.6 minutes. This time

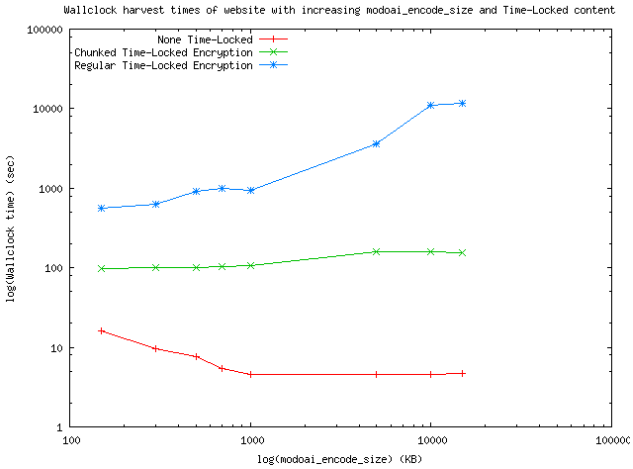

Figure 7: Harvest times of website using no data time-lock, regular time-lock and chunked time-lock encryption during dynamic data dissemination.

modoai_encode_size (Bytes)	Responses	Chunked (sec)	σ (sec)	Speedup
150,000	70	99	7.7	6
300,000	38	101	12.4	6
500,000	24	102	7.9	9
700,000	16	103	6.5	10
1,000,000	13	108	10.7	9
5,000,000	6	161	16.1	23
10,000,000	6	160	16.2	65
15,000,000	6	156	16.9	70

Table 6: Wallclock harvest times of website with using "chunked" time-lock encryption.

penalty for disseminating embargoed content is within the realistic, feasible range of website harvest time.

The MPEG21 DIDL document format represented in Figure 2 has been modified to reflect the inclusion of chunked time-lock encryption. In this optimization, the size of each encrypted chunk is set to 10 KB. Every file of size greater than 10 KB has been divided into 10 KB chunks and individually encrypted. Each file chunk has been encapsulated into one *component*. Therefore, each chunked encrypted file, contained in a DIDL *record* entity, contains multiple *components*. Chunked *records* in the exported XML document can be identified by their file size, as well as the number of *components* contained in each *record*.

Each record *component* needs to be identified for accurate reordering, decryption and reassembly of contained data chunks into one file. These *components* have been associated with Identifiers in increasing lexicographical order, which have been encapsulated in an *identifier* entity within each component. A *descriptor* has also been inserted in each *component* to provide additional information regarding the total number of chunks contained in the record to ensure that each isolated component contains sufficient information required for reassembly of chunks into one file. This modified document model still contains the original record identifier and related metadata for record identification.

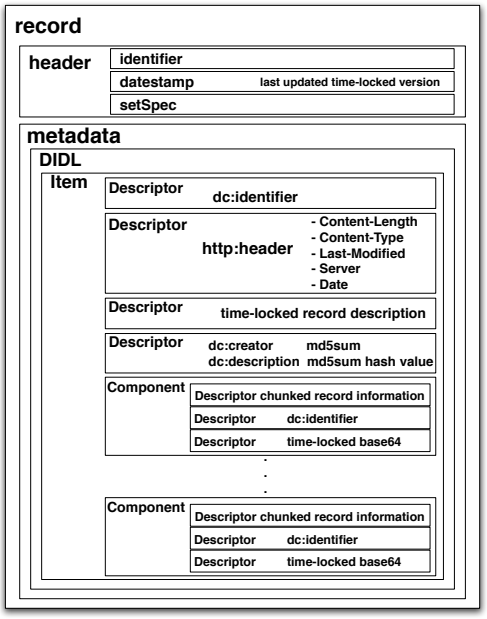

Figure 8: MPEG-21 DIDL Document of a record time-locked using chunked encryption.

8. FUTURE WORK AND CONCLUSIONS

We have introduced the "Preservation Risk Interval" problem associated with embargoed content caused due to limited diffusion of embargoed scholarly material within the digital library community. The Time-Locked Embargo Framework introduces the identification of embargoed content and calculates the required complexity of the time-lock puzzle to be created for that content. During the integration of time-lock puzzles into `mod_oai`, an initial system evaluation revealed that the amount of time required to create time-lock puzzles during dynamic DIDL document creation increases exponentially with the size of the embargoed file. This exponential increase in time overhead has been reduced by splitting the files into 10 KB data chunks for faster encryption and harvest time. The framework has also modified the MPEG-21 DIDL complex object format utilized by `mod_oai` to accurately encapsulate chunked embargoed content and related metadata. With the use of the expanded `mod_oai` module, resources under embargo can be exchanged between a much broader scholarly community for the purpose of digital preservation as well as content diffusion.

Optimum chunk size should be investigated further. 10KB was empirically determined for the class of machines we used, but other values or approaches should be investigated. Alternate approaches could include using parallel machines for time-locking, or "pre-locking" popular content prior to distribution (the current implementation dynamically locks on dissemination, trading time for space by not requiring the overhead of cache maintenance). There could also be hybrid approaches where the content is locked with conventional encryption and the only the keys are time-locked. Furthermore, the time-lock approach could be defined as an HTTP "Content-Encoding" (like "gzip" or "deflate") and be used outside of OAI-PMH.

9. REFERENCES

[1] C. W. Bailey. The Spectrum of E-Journal Access Policies: Open to Restricted Access. http://dlist.sir.arizona.edu/990/01/spectrum.htm, 2005.

[2] J. Bekaert, P. Hochstenbach, and H. Van de Sompel. Using MPEG-21 DIDL to represent complex digital objects in the Los Alamos National Laboratory digital library. *D-Lib Magazine*, 9(11), 2003.

[3] J. Bekaert, E. D. Kooning, and H. Van de Sompel. Representing Digital Assets using MPEG-21 Digital Item Declaration. *International Journal on Digital Libraries*, 6(2):159–173, 2006.

[4] S. S. Chow, V. Roth, and E. G. Rieffel. General Certificateless Encryption and Timed-Release Encryption. Cryptology ePrint Archive, Report 2008/023, 2008. `http://eprint.iacr.org/`.

[5] D. H. Crocker. RFC822 - Standard for the Format of ARPA Internet Text Messages, 1982.

[6] J. A. Garay and M. Jakobsson. Timed release of standard digital signatures. In *Financial Cryptography 02*, pages 168–182. Springer-Verlag, 2002.

[7] S. Harrell, T. Seidel, and B. Fay. The national technology roadmap for semiconductors and sematech future directions. *Microelectron. Eng.*, 30(1-4):11–15, 1996.

[8] J. Jonsson and B. Kaliski. Public-Key Cryptography Standards (PKCS) 1: RSA Cryptography Specifications Version 2.1, 2003. RFC 3447.

[9] G. Klyne and C. Newman. RFC3339 - Date and Time on the Internet: Timestamps, 2002.

[10] C. Lagoze and H. Van de Sompel. The Open Archives Initiative: building a low-barrier interoperability framework. In *Proceedings of JCDL '01*, pages 54–62, 2001.

[11] G. E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8):114–117, 1965.

[12] T. Morrow, N. Beagrie, M. Jones, and J. Chruszcz. A Comparative study of e-Journal Archiving Solutions. Technical report, JISC, 2008.

[13] R. L. Rivest. Description of the LCS35 Time Capsule Crypto-Puzzle, April 1999.

[14] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, 1978.

[15] R. L. Rivest, A. Shamir, and D. A. Wagner. Time-lock puzzles and timed-release crypto. Technical Report 684, Massachusetts Institute of Technology, Cambridge, MA, USA, 1996.

[16] A. Robinson. Open access: the view of a commercial publisher. *Journal of Thrombosis and Haemostatis*, 4(7):1454–1460, 2006.

[17] J. A. Smith and M. L. Nelson. Creating Preservation-Ready Web Resources. *D-Lib Magazine*, 14(1/2), 2008.

[18] J. Tucker and M. S. Hoyle. Understanding embargoes and utilizing other services. *The Serials Librarian*, 45(3):115–117, 2003.

[19] H. Van de Sompel, M. L. Nelson, C. Lagoze, and S. Warner. Resource harvesting within the OAI-PMH framework. *D-Lib Magazine*, 10(12), 2004.