Global Information Assurance Certification Paper

Using DomainKeys Identified Mail (DKIM) to Protect your Email Reputation

GIAC (GCIH) Gold Certification

Author: Chris Murphy, chrismrph0@gmail.com
Advisor: Johannes B. Ullrich, Ph.D.

Accepted:

August 19th 2013

Abstract

SPAM and Phishing emails commonly spoof the sender domain, both as a way to evade spam filters and to engender the recipient's trust. Several email authentication methods allow organizations to prove emails purporting to originate from their domain actually did so. One such authentication framework, Domain Keys Identified Mail (DKIM), provides a relatively easy way for organizations to use cryptographic signatures to protect their e-mail reputations with only small changes to their existing email and Domain Name System infrastructures. However, in order to use DKIM successfully and effectively, organizations must fully understand its purpose, function and limitations and then take steps to manage and protect the encryption keys.

1. Introduction

Domain Keys Identified Mail (DKIM) was developed as a successor to the DomainKeys framework originally created by Yahoo! in 2004 (Yahoo! Inc., 2004). DKIM provides a specification that organizations can use to digitally sign their outgoing email using public key cryptography so recipients can verify that an email message truly originated from the sender's domain and that selected headers and message contents were not changed after being sent.

DKIM is designed to be simple to implement; it uses the existing, widely-deployed SMTP and DNS protocols. Though it uses public key cryptography, it offers a simple way to manage keys without the need for certificates or certificate authorities (Crocker, Hansen, & Kucherawy, 2011).

DKIM answers two important questions: 1) did this message actually originate from the domain listed in the From header field, and 2) was the message modified while traversing other mail servers?

Operation is straightforward. One popular implementation called OpenDKIM functions as a plug-in mail filter (or milter) to the Sendmail and Postfix mail transfer agents. After installation, an RSA key pair is created. The public key is published in the signer's DNS zone as a TXT resource record. After the message headers and bodies of new email are hashed and signed with the private key, the hash values and signature are added to the message headers. Recipients can choose to implement DKIM themselves to verify signatures on incoming email. Verifiers read the DKIM message header field, acquire the sender's public key from DNS, verify the signature, compute hash values of their own and compare them to the digitally signed values. After this verification is successful, the recipient is assured the message originated from the advertised sender domain and the content was not modified while in transit.

As mentioned, DKIM solves the problem of sender address spoofing. It does not solve the larger problems of SPAM or Phishing (Crocker, Hansen, & Kucherawy, 2011). Yet, since SPAM and Phishing e-mails routinely use fake sender domains, email recipients who have implemented DKIM can use the the validity of a DKIM signature to enhance their SPAM filtering mechanisms.

Chris Murphy, chrismrph0@gmail.com

DKIM is explicitly not designed to assess the reputation of the sender domain (Crocker, Hansen, & Kucherawy, 2011). So, a recipient could verify that a DKIM signed email from *somebody@2BuyCheapSoftwarez.biz* actually originated from that domain. However, the verifier must use other tools and methods to decide if the domain is trustworthy and the email is worth receiving.

Finally, the security of a DKIM implementation depends upon the integrity and protection of the private key to sign email messages (Crocker, Hansen, & Kucherawy, 2011). If this key is stolen, replaced or otherwise acquired, an attacker could spoof messages that would authenticate the source as the target's domain.

2. Overview

The following overview shows a simple, high-level example of DKIM in action. Both the sender (signer) and recipient (verifier) organizations have implemented DKIM.

2.1. Signing

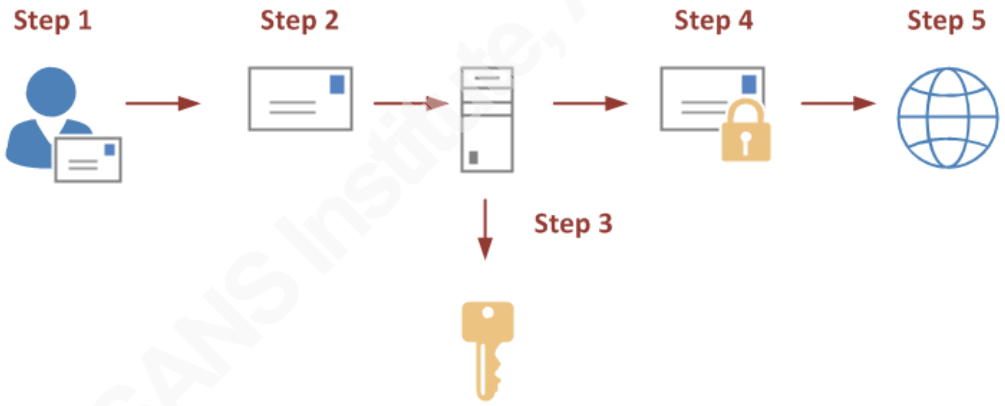

Figure 1: DKIM signing steps

In Step 1, a user creates a mail message using his mail client. He sends the message to his configured SMTP server in Step 2. In Step 3, the SMTP software passes the message off to a DKIM mail filter for post-processing. DKIM queries a mapping table to find the private key associated with the sender domain. (Organizations with many subdomains may likewise use many keys.) The filter formats the message according to the canonicalization rules enabled in its configuration file. It computes hash values for selected header fields and the message body and inserts these in a newly created DKIM-Signature header. In Step 4, using the private key it

retrieved from the local file system in Step 3, it signs the hashes and inserts the signature in a remaining field. In Steps 5, the now signed message is delivered as usual.

2.2. Verification

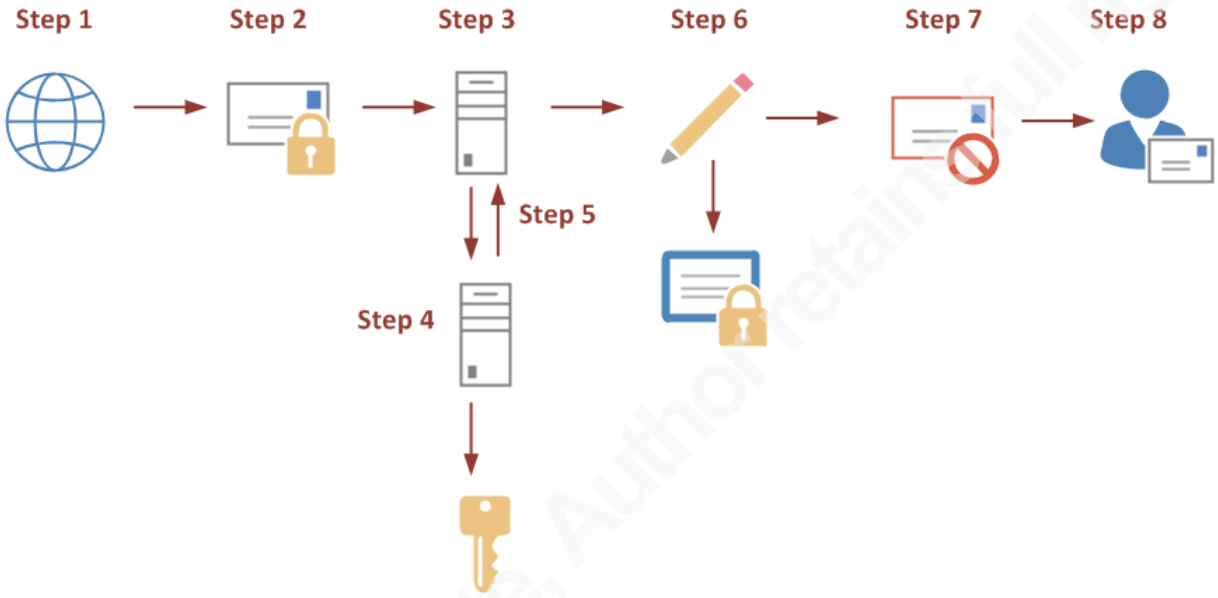

Figure 2: DKIM verification steps

Steps 1 and 2 see the signed message arrive at the recipient's mail server. In Step 3, the SMTP software passes the message off to a configured DKIM mail filter for post-processing. DKIM parses the selector and domain names from the DKIM-Signature header. It uses this information to query the sender's DNS to retrieve the public key in Steps 4 and 5. According to the canonicalization mode referenced in the DKIM-Signature header, it formats the message, computes the hash values and compares them to those in the DKIM-Signature header. If they match, it verifies the signature using the public key it retrieved in Step 5. It writes the results of the verification to the Authentication-Results header field in Step 6. In Step 7, SPAM filtering software can then use the DKIM result as input in determining the message's disposition. Finally, in Step 8, the message is delivered to the recipient.

3. Canonicalization

Canonicalization is a critical concept in DKIM. Since verifiers have to compute a hash of the headers and message body that precisely matches those included by the signers, both parties

have to ensure they are starting the process with precisely the same content. The canonicalization rules ensure that both the signer and verifier are computing hashes on the same formatted text. If the signer formats the message body one way before computing the hash, and the verifier formats the message body differently, the signature verification will fail and invalidate the benefit of using DKIM in the first place.

Mail servers sometimes modify email messages that pass through their systems (Eland Systems, 2009) and even insignificant changes can invalidate a DKIM signature. Some signers may not care if something incidental is modified. Others may want strict integrity of the message. DKIM offers options to support both requirements using its canonicalization algorithms. (Crocker, Hansen, & Kucherawy, 2011)

3.1. Simple Canonicalization of the Headers

When simple canonicalization is chosen for the headers, the formatting of the message is mostly left intact. Letter case and all whitespace are preserved. This is the most secure choice but also the most sensitive to any changes an intermediate mail server might introduce after the signature is created. So, there is a greater chance of a verification failure (Crocker, Hansen, & Kucherawy, 2011).

3.2. Relaxed Canonicalization of the Headers

The relaxed algorithm (Crocker, Hansen, & Kucherawy, 2011) is more flexible so the signature verification might survive small changes introduced by an intermediate mail server. However, the content still can't be modified significantly without breaking the signature.

All header field names (such as Content-Type) are converted to lower-case (content-type). Any header values that have been separated by a Carriage Return Line Feed (CRLF) must be joined or "unfolded" (Resnick, 2008) and any whitespace at the end of the line is removed. Any instances of more than one adjacent whitespace characters have to be condensed to one space only. Spaces before or after the colons after the header fields have to be removed. So, "From : Joe" would change to "from:joe".

3.3. Simple Canonicalization of the Message Body

Nothing changes when simple formatting of the body is used other than the addition of a single CRLF at the end of the message (Crocker, Hansen, & Kucherawy, 2011). This happens even if the message is blank.

3.4. Relaxed Canonicalization of the Message Body

Whitespace at the end of a line is left as-is. Any collection of more than one adjacent whitespace characters have to be condensed to one space only. (Crocker, Hansen, & Kucherawy, 2011) A CRLF is added at the end of the message body.

3.5. Examples

The following two examples were canonicalized by feeding a saved email message with headers into Jason Long's dkimverify.pl script (Comprehensive Perl Archive Network).

```
perl dkimverify.pl -debug-canonicalization=/home/joe/rr-msgdump <
/home/joe/relaxed-relaxed-example.txt
```

The first shows the default simple/simple canonicalization rules after application to the message headers and message body. No changes at all are allowed to the headers or message body after signing.

```
Message-ID: <518B912D.1060507@bright-armor.net>
Date: Thu, 09 May 2013 08:06:05 -0400
From: j <joe@bright-armor.net>
MIME-Version: 1.0
To: chrismrph0@gmail.com
Subject: DKIM simple example
Content-Type: text/plain; charset=ISO-8859-1; format=flowed
Content-Transfer-Encoding: 7bit
Hi,

This is the first line.

This is the second line with a tab character    here.

  This  is the      last line with extra whitespace          .

Bye,

Joe
DKIM-Signature: v=1; a=rsa-sha256; c=simple/simple; d=bright-armor.net;
        s=one; t=1368101167;
        bh=jH3/5Z8zc0hMKudu6iMjxsSa9mx6uKAZstBCazaqgIc=;
        h=Message-ID:Date:From:MIME-Version:To:Subject:Content-Type:
         Content-Transfer-Encoding;
        b=
```

The yellow highlighting shows that all whitespace is preserved, including the spaces after the colons in the headers. The carriage return and line feeds (CRLF) after each line in the DKIM-signature header are preserved.

The next example shows relaxed/relaxed canonicalization rules.

```
message-id:<518BCE6E.9070304@bright-armor.net>
date:Thu, 09 May 2013 12:27:26 -0400
from:j <joe@bright-armor.net>
mime-version:1.0
to:chrismrph0@gmail.com
subject:DKIM relaxed-relaxed example
content-type:text/plain; charset=ISO-8859-1; format=flowed
content-transfer-encoding:7bit
Hi,

This is the first line.

This is the second line with a tab character here.

 This is the last line with extra whitespace .

Bye,

Joe
dkim-signature:v=1; a=rsa-sha256; c=relaxed/relaxed; d=bright-armor.net;
s=one; t=1368116848; bh=/LiUYUL3hHykvdT2f1VXwnfYBQPx7WpULLnzVHpYimA=;
h=Message-ID:Date:From:MIME-Version:To:Subject:Content-Type: Content-
Transfer-Encoding; b=
```

Whitespace after the colons in the header has been removed. The CRLFs in the DKIM-Signature header have also been removed. In the message body, a tab has been removed and each collection of whitespace greater than two spaces is reduced to one space only.

It is also possible to combine the canonicalization rules to produce hybrid formatting. You can specify relaxed/simple or simple/relaxed. In both cases, the formatting rules are different for the message headers and the message bodies.

4. Selectors and Keys

The selector is an arbitrary string associated with a key. Its chief purpose is to allow an organization using multiple keys to manage them more easily (Crocker, Hansen, & Kucherawy, 2011). An organization might maintain one key for its top-level domain and then create another key for a marketing subdomain, for example.

Judging from a sampling of received email, most DKIM users also incorporate the date the key was created in the selector name, perhaps to aid in periodic replacement.

DKIM keys can removed if they are no longer necessary simply by deleting the associated TXT record from DNS. Keys can also be revoked. Revoking the key expresses a more formal message to verifiers (Crocker, Hansen, & Kucherawy, 2011). The selector is left published in DNS as a TXT record but the actual public key data after the *p*= tag is deleted. This leaves a selector with a null key value so any verification attempts will fail permanently. If a key is replaced, the retiring selector should be left in place for a reasonable period of time to ensure that any signed mail that may have not been verified yet will have a chance to access the old key (Crocker, Hansen, & Kucherawy, 2011).

RFC 6376 cautions against re-using the same selector names since organizations won't be able to troubleshoot verification failures: did the verifier pull an old key from a DNS cache or is someone actually tampering with your message?

DKIM uses an RSA private key for signing and a public key for verification. As such, implementers can use the openssl libraries to create the key pair (Eland Systems, 2009). Before publishing the public key in DNS, has to be manually formatted to incorporate any optional fields.

Alternately, OpenDKIM includes the opendkim-genkey utility which will create the key pair and add the options you specify, resulting in a public key that is pre-formatted to publish in DNS.

Like the DKIM-Signature header, the value of the selector TXT record published in DNS contains tags that can communicate important information to verifiers. *v*= should be DKIM1. *p*= contains the public key, and this is the only tag that is absolutely required. *t*=*y* means that the organization is testing DKIM. Many organizations are misusing this tag by setting it and never turning it off (Santos, 2009).

5. DKIM's Use of DNS

DKIM leverages DNS to store the public key. This arrangement is beneficial because the DNS infrastructure is already ubiquitous. Any organization with an Internet presence already

uses it. DNS also provides basic protection for the public key since each organization, or its authorized delegate, should have exclusive administrative control over the zone records.

DKIM public keys are stored as TXT type resource records in the zone file for the domain with which they are associated. The DKIM record creates a fake subdomain called _domainkey (Crocker, Hansen, & Kucherawy, 2011). The full record is comprised of the *selector name* + "." + *_domainkey* + *domain* like this example: one._domainkey.bright-armor.net.

When verifying email messages, DKIM retrieves the public key from DNS using the selector specified in the DKIM-Signature header. The dig command can be used to retrieve a key manually.

```
p@stark:~$ dig -t TXT one._domainkey.bright-armor.net +short
"v=DKIM1\; g=*\; k=rsa\;
p=MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDaBwAH38KrQ15mSZqYpAB9P/nQEOfPOWmX
RQFec0DIytTLKBwgVOs4iW4yyQq6hI3/QUD5D+U+bTTZUuznBCEPURxYkpasgyS/xrEAHgFiSk
AyKG2d0L2TDOyon88WpG9LcQor6eqT/ovXJpDS9Sk4NCSGdua0NHXuf1fmmwqSiQIDAQAB"
```

This UDP stream from a packet capture exported from Wireshark shows DKIM querying DNS for Google's public key while verifying a signed message from Gmail. The query is represented in red and the response, in blue. The fully-qualified selector name is evident: 20120113._domainkey.gmail.com.

```
I...........20120113
_domainkey.gmail.com.......)........
I...........20120113
_domainkey.gmail.com.............,...k=rsa;
p=MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA1Kd87/UeJjenpabgbFwh+eBCsST
rqmwIYYvywlbhbqoo2DymndFkbjOVIPIldNs/m40KF+yzMn1skyoxcTUGCQs8g3FgD2Ap3ZB5
DekAo5wMmk4wimDO+U8QzI3SD0.7y2+07wlNWwIt8svnxgdxGkVbbhzY8i+RQ9DpSVpPbF7yk
QxtKXkv/ahW3KjViiAH+ghvvIhkx4xYSIc9oSwVmAl5OctMEeWUwg8Istjqz8BZeTWbf41fbN
hte7Y+YqZOwq1Sd0DbvYAD9NOZK9vlfuac0598HY+vtSBczUiKERHv1yRbcaQtZFh5wtiRrN0
4BLUTD21MycBX5jYchHjPY/wIDAQAB
```

6. The DKIM-Signature Header

During the hashing and signing process, DKIM will insert its own header into the email message to carry important information needed by the verifier.

In this example, the DKIM-Signature header uses a series of tags (Crocker, Hansen, & Kucherawy, 2011) which are described below.

```
DKIM-Signature: v=1; a=rsa-sha256; c=relaxed/relaxed; d=bright-armor.net;
       s=one; t=1368500485;
       bh=zY5K2FTMhCDb83qap+bWvIiKgV6sWyxFHMzsi7r5hng=;
       h=Message-ID:Date:From:MIME-Version:To:Subject:Content-Type:
        Content-Transfer-Encoding;
       b=PYPaTwy6z9E9VHkHLedDkVjT8OrgH8z51q04hCOJwm8eGwO3GrIu3uxrP0+s8i++o
        bf9pzkE5/oJdwedFE4GvX1wD2IQJrHEZOCn7IQImz0GIIad87nQSfGIWwhoRNhJyes
        ihEukH0cOJrgx7e/SfFVu5ZCfW4eEJlFIMm1cOcg=
```

The *v=1* tag represents the version. It should always be set to 1 for DKIM.

The *a=rsa-sha256* lists the hashing algorithm used. Obviously, the verifier needs to use the same one in order to successfully authenticate the message.

The *c=relaxed/relaxed* tag identifies the canonicalization algorithm. Both the header and message body use relaxed. The verifier needs to format the same way to successfully authenticate the message.

The *d=bright-armor.net* tag names signing domain that is claiming ownership of the message.

The *s=one* tag is the selector name associated with the key being used.

The *bh=zY5K[…]5hng=;* tag stores the SHA-256 hash value of the message body after it has been canonicalized and then encoded with base64.

The *t=1368500485* tag holds the timestamp of the signature creation in Epoch time format.

The *h=Message-ID:[…]Content-Transfer-Encoding;* tag lists the message header fields that were hashed. The verifier needs to know which ones to include as well.

The *b=PYPa[…]cOcg=* tag stores the RSA signature.

There are other optional tags that can be included in the DKIM-Signature header but only a, b, bh, d, h and s are required.

7. The Signing Process

Before signing an email message, DKIM computes two hash values (Crocker, Hansen, & Kucherawy, 2011). Fundamentally, these hashes provide the verifier the assurance that the message header and body have not been modified after the message left its origin mail server.

First, DKIM computes a hash for the message body after it has been canonicalized. It encodes the value in base64 and then inserts in the *bh*= field.

Second, DKIM computes a hash for 1) the selected header fields after they have been canonicalized and 2) the DKIM-Signature header fields with a blank *b*= field after they have been canonicalized. This makes sense since at this point, the signature has not yet been created.

Signing and verification use either the RSA-SHA1 or RSA-SHA256 digital signature algorithms (Crocker, Hansen, & Kucherawy, 2011). RSA-SHA256 is recommended for signing but DKIM implementations must still support verification of RSA-SHA1.

Signers can choose which headers fields to sign, or even all of them except for the h=DKIM-Signature. RFC 6376 recommends choosing the common headers such as To, Date, Subject, etc. and exclude any that will change, such as Received. Only the From header is absolutely required.

After the hash is computed, it is signed with the RSA private key. That signature is inserted into the *b*= header field value, which was blank until this point.

8. The Verification Process

Verification steps are performed almost exactly the same way. Verifiers of course verify the signature with the public key whereas the signers sign with the private key. Verifiers reverse the process and then simply compare their results to those in the DKIM-Signature headers before issuing one of three determinations (Crocker, Hansen, & Kucherawy, 2011) -- SUCCESS, PERMFAIL or TEMPFAIL -- and adding an Authentication-Results header field (Kucherawy, 2009).

The possible results written to the header include: none, pass, fail, policy, neutral, temperror and permerror. None, pass and fail are obvious. The message was either not signed, it passed verification, or it failed verification, respectively. Temperror can sometimes indicate

DNS problems while retrieving a key. Permerror means that the message could not be verified due to some irrecoverable error.

The policy and neutral results are less intuitive.

Gmail failed this according to a new policy they have implemented to refuse to verify keys weaker than 1024-bits (Takahashi, 2013):

```
Authentication-Results: mx.google.com;
       dkim=policy (weak key) header.i=@bright-armor.net
```

Outlook.com ultimately sent this message to the SPAM folder due to a syntax error:

```
Authentication-Results: hotmail.com; spf=none (sender IP is 54.225.21.1)
smtp.mailfrom=joe@bright-armor.net; dkim=neutral header.d=bright-armor.net;
x-hmca=none header.id=joe@bright-armor.net
```

9. Options for Deployment

For organizations that wish to sign their outgoing email or verify incoming email, the DKIM Organization web site maintains an extensive list of software, hardware and services that support DKIM (DKIM Software and Services Deployment Reports). There are many third party software MTAs and hardware appliances that implement DKIM.

If you run Linux MTA servers on your network, DKIM software implementations are available for the popular choices - Sendmail, Postfix -- and natively included in Exim (Chapter 56 - Support for DKIM). OpenDKIM (OpenDKIM), DKIM-proxy (Mail-DKIM and DKIMproxy) and AMaViS (amavisd-new) represent some of the available choices.

If you run Microsoft Exchange mail servers, you have fewer options. Microsoft Exchange Server has no native support for DKIM, perhaps because Microsoft has instead promoted their Sender-ID framework (Spiezle & Nikolayev, 2006).

Microsoft Exchange Server versions higher than 2007 support custom Transport Agents. Nicholas Piasecki developed custom Transport Agent code to sign outgoing email in Exchange (Piasecki, 2010) which was later adapted by Stefan Profanter into an installable Exchange add-on (Profanter).

The examples in this paper use OpenDKIM version 2.0.1 on Debian Linux Squeeze configured to work with the Postfix MTA (Simon, 2013).

10. Problems and Limitations

As emphasized in RFC 6376, DKIM does not account for the trustworthiness of emails. It only confirms that the sender's domain is genuine. If an attacker compromises an organization's mail infrastructure, or infects client machines authorized to send mail from that domain, those emails will be signed just like authorized email.

This example shows a suspicious e-mail from a yahoo.com address that appeared obviously fraudulent based on its subject ("Please get this handled ASAP.") and message body (a single hypertext link).

According to the message headers, this message was signed by DKIM:

```
DKIM-Signature: v=1; a=rsa-sha256; c=relaxed/relaxed;
d=yahoo.com; s=s1024; t=1365314818;
[…output snipped…]
```

The DKIM signature was verified by the recipient's mail server:

```
Authentication-Results: mx.google.com;
       spf=pass (google.com: best guess record for domain of
<redacted>@yahoo.com
designates x.x.x.x as permitted sender) smtp.mail=<redacted>@yahoo.com;
       dkim=pass header.i=@yahoo.com
```

But analysis with the Wepawet (The Regents of the University of California) tool showed the linked URL was a redirect to an affiliate tracking web site, making this an example of click fraud or click abuse.

It is worth noting that Yahoo! is a provider of millions of free email accounts. In this case, the account was likely compromised. However, this overly simple example illustrates the potential pitfall with using DKIM that organizations must understand. If a malicious email is authorized to send through an organization's mail system, it will be signed with their DKIM key just the same as the legitimate email leaving that organization. Organizations must use additional tools then to judge the reputation of messages. In the example above, the provider's

SPAM filtering software marked this message as SPAM based on other criteria despite the valid DKIM signature and moved it to the SPAM folder.

11. Alternatives to DKIM

Organizations do not have to implement DKIM. They may opt not to use any authentication framework at all to protect their email reputations. Others may want to use a protective framework but hesitate to use DKIM since it requires installing and managing additional software. There are alternatives they can use instead.

11.1. Sender Policy Framework (SPF)

Sender Policy Framework (Mehnle) uses DNS like DKIM, but unlike DKIM, it does not require any software to be installed on mail servers. Mail admins can publish a DNS TXT resource record to specify a sending policy for their email. Chiefly, they can identify which servers are allowed to send mail for their domain. SPF records show verifiers that an email is likely fake if it originated from an unauthorized server address. Also, in contrast to DKIM, SPF does not make any assertions about the integrity of message content.

This simple record asserts that all mail purporting to originate from the bright-armor.net domain must come from one of the IP addresses published in DNS as the "MX" or mail exchanger record for that domain.

```
bright-armor.net. IN  TXT  "v=spf1 mx -all"
```

11.2. Sender ID

Sender ID (Microsoft Corporation, 2004) was created by Microsoft. It is based on SPF and uses similar DNS syntax. It also specifies which server addresses are authorized senders for an email domain. However, while SPF authenticates the envelope sender address, Sender ID authenticates the Purported Responsible Address (PRA), which it derives from several mail header fields.

11.3. DMARC

DMARC (DMARC.org) is a relatively new framework that provides add-on functionality to both SPF and DKIM. It allows senders to specify a handling policy in DNS that tells verifiers what to do if email from this domain fails the SPF or DKIM verification. Finally, senders can

request feedback on rejections so they can troubleshoot and update their implementation accordingly.

The Messaging, Malware and Mobile Anti-Abuse Working Group (MAAWG) best practices (Messaging, Malware and Mobile Anti-Abuse Working Group, 2012) recommend leveraging DMARC to monitor how other organizations use your DKIM authentications.

12. Security Concerns

As mentioned, organizations are not obligated to use DKIM. However, once an organization makes the deliberate decision to implement DKIM, they must take steps to protect their implementation and in particular, their private keys. Not using DKIM is far preferable to using DKIM that has been compromised by attackers who are using it to send fraudulent but still authenticated email using your domain name.

Consider delegation carefully when developing security controls for DKIM. The specification (Crocker, Hansen, & Kucherawy, 2011) allows email signing by intermediate organizations so an organization can contract some external entity to sign and forward emails for them. In practice, this key delegation is most often used by organizations that outsource email marketing to third party firms. This represents a risk that security personnel must consider. In addition to implementing strict controls around your DKIM keys, you must also ensure that a third party entity is protecting your keys with the same diligence (Messaging, Malware and Mobile Anti-Abuse Working Group, 2012).

The practical threat of a compromised DKIM is interruption of reliable mail delivery. If SPAM is authenticated as leaving your organization, your domain may be blacklisted (Bonar, 2010). But there is a more intangible threat that may have more insidious consequences--the loss of trust. If you are cryptographically certifying ownership of unsolicited or malicious email originating from your domain, in essence you are advertising that do not have adequate controls protecting your infrastructure.

RFC 6376 describes several esoteric attacks against DKIM that would likely have minimal impact on organizations with well-designed, resilient mail infrastructures.

An attacker could exploit the relaxed canonicalization algorithm by intercepting a signed message from the target and manipulating the whitespace in the message body to create a

visually recognizable overlaid image or message. When the recipient system verifies the message, it will ignore extra whitespace according to the canonicalization rules and the signature will verify successfully. However, the target would see the unintended overlaid message in addition to the original authenticated message.

Two other attacks describe possible denial of service scenarios. In one, the attacker could send a signed message to the target after manipulating the exponent on his key in a way that causes the verifier to use significant processing resources. In another, an attacker could flood other recipients with spoofed but signed messages from the target hoping that the distributed recipients will all try to verify at the same time and overwhelm the target's DNS while querying the public key.

These attacks should not concern incident handlers and other information security personnel too much. Theft or misappropriation of the public and private keys are more likely and would have greater security impact. One far more practical attack—an offline crypto attack against the public key--should concern security personnel since a real case has already happened.

In 2011, a mathematician and IT consultant named Zachary Harris received an email message from a Google recruiter (Zetter, 2012). Suspecting the message was fraudulent, he examined the headers and noticed the DKIM public key was only 512-bits long. He knew that factoring a 512-bit key was achievable using computing resources that are readily available, so he did. After he factored the public key using a cluster of Amazon EC2 servers, he derived the private key and used it to send a signed (though still spoofed) message from one of the Google founders to the other. Shortly afterwards, Google revoked its 512-bit DKIM key and replaced it with a much stronger 2048-bit key.

This incident led to a CERT advisory (US-CERT, 2012) and the release of a DKIM best practice recommendations by the MAAWG (Messaging, Malware and Mobile Anti-Abuse Working Group, 2012). Several other high profile Internet services also replaced their 512-bit keys (Harris Z.).

Though a *Wired Magazine* interview of Harris did not provide specific details on how he cracked Google's key, it did mention that he used Amazon's EC2 cloud computing network. On a cryptography mailing list, he revealed he used the CADO-NFS software to factor the public

key and the StarCluster software to manage multiple Amazon instances at once to parallelize the factoring job (Harris Z. , Thanks, 2012).

An attacker can easily do the same without knowing high-level mathematics or the Number Field Sieve algorithm. One site lists a very helpful recipe (Ruiz Duarte, 2011) to follow to factor a public key and with enough time and computing power, derive the private key.

The following steps represent one way an attacker could derive an organization's private key and abuse it to send a spoofed, but legitimately signed, message on their behalf.

- Acquire a recent email message from the organization you want to attack
- Examine the mail headers and identify the selector name
- Use the selector to query the public key from DNS
- Use the openssl library to print the exponent and modulus of the public key
- Reformat the public key, converting the modulus from hexadecimal to decimal using the Linux bc utility
- Run factoring software, such as CADO-NFS, on an Amazon EC2 virtual computing cluster to factor the 155 decimal number that represents the public key
- When factoring is complete, compute the private key
- Reformat the decimal value back into hexadecimal, and then reformat the hex into PEM format
- Copy the now formatted private key to a DKIM setup on a mail server you control and reference it in the configuration
- Send a spoofed but signed message from mail server under your control

First, the attacker would need an authenticated, DKIM-signed message from the organization. If he didn't already have one, he could send a request to address designed to accept public inquires such as marketing@ or recruiting@ or privacy@. From the DKIM message headers, he would locate the selector name and use that to query DNS for the public key.

```
DKIM-Signature: v=1; a=rsa-sha256; c=relaxed/relaxed; d=bright-armor.net;
       s=weak; t=1375154633;
       bh=OZ4aeXAQ1gpugfnnEY41b7u8BL0PT56vmkJolBnwRDM=;
       h=To:Subject:Message-Id:Date:From;
       b=dJoOh5u6wdpbKNBgtj84EiLGYExOzTfT+JrnNiWHx4q45bmMkjE0bMv5gxb3G+nos
        dDqqni3SYtdg03zCxnYPA==
```

```
admin@ip-10-154-135-41:~$ dig -t TXT weak._domainkey.bright-armor.net +short
"v=DKIM1\; k=rsa\;
p=MFwwDQYJKoZIhvcNAQEBBQADSwAwSAJBAMdR5n8In3WHqtJ1XGngXbYsItYu6ydNr81xNlw1wa
KBzgGv9095yGJQmD7b01qStxlFn3vBJqq6TQ44TyeaxUsCAwEAAQ"
```

The attacker could use the openssl command to determine the exponent and the modulus of the public key.

```
admin@ip-10-154-135-41:~$ openssl rsa -in ref-weak -pubin -text -modulus
Public-Key: (512 bit)
Modulus:
    00:c7:51:e6:7f:08:9f:75:87:aa:d2:75:5c:69:e0:
    5d:b6:2c:22:d6:2e:eb:27:4d:af:cd:71:36:5c:35:
    c1:a2:81:ce:01:af:f7:4f:79:c8:62:50:98:3e:db:
    d3:5a:92:b7:19:45:9f:7b:c1:26:aa:ba:4d:0e:38:
    4f:27:9a:c5:4b
Exponent: 65537 (0x10001)
Modulus=C751E67F089F7587AAD2755C69E05DB62C22D62EEB274DAFCD71365C35C1A28
1CE01AFF74F79C86250983EDBD35A92B719459F7BC126AABA4D0E384F279AC54B
writing RSA key
-----BEGIN PUBLIC KEY-----
MFwwDQYJKoZIhvcNAQEBBQADSwAwSAJBAMdR5n8In3WHqtJ1XGngXbYsItYu6ydN
r81xNlw1waKBzgGv9095yGJQmD7b01qStxlFn3vBJqq6TQ44TyeaxUsCAwEAAQ==
-----END PUBLIC KEY-----
```

He would reformat the public key by converting the modulus from hexadecimal to decimal using the Linux bc utility.

```
admin@ip-10-154-135-41:~$ echo "ibase=16;
C751E67F089F7587AAD2755C69E05DB62C22D62EEB274DAFCD71365C35C1A281CE01
AFF74F79C86250983EDBD35A92B719459F7BC126AABA4D0E384F279AC54B" | bc
1043923144061564876149064247393924005754990444495789238949414157546\
1652917871449170431046940619511182819537772429174926682462813974201\
0635962343103703301963
```

The result is the decimal integer that the attacker supplies to CADO-NFS to factor:

```
104392314406156487614906424739392400575499044449578923894941415754616529178714491704310469406195111828195377724291749266824628139742010635962343103703301963
```

This is the full command (see Appendix II for more notes about CADO-NFS usage):

```
perl ./cadofactor.pl params=params/params.c155 wdir=cadowork
n=104392314406156487614906424739392400575499044449578923894941415754616529178714491704310469406195111828195377724291749266824628139742010635962343103703301963
```

Chris Murphy, chrismrph0@gmail.com

```
[…previous output snipped…]

    Info:Factorization was successful!

Info:9467592566042455828159569807202728161118317379236558752541712307
5076172576717 [prime]

Info:1102627871636363336724570329030822239932141235919360404000343819
76964178930039 [prime]
    Info:Doing gcds with previously known composite factors
    Info:non-trivial factor:
9467592566042455828159569807202728161118317379236558752541712307507617
72576717 [prime]
    Info:non-trivial factor:
1102627871636363336724570329030822239932141235919360404000343819769641
78930039 [prime]
    Info:Now: 2 prime factors, 0 composite factors
    Info:Factorization complete
Info:----------------------------------------------------------------------
----------
Info:All done!
Info:----------------------------------------------------------------------
----------
9467592566042455828159569807202728161118317379236558752541712307507617
72576717
1102627871636363336724570329030822239932141235919360404000343819769641
78930039
```

After many stages, CADO-NFS finishes and provides the attacker with the two prime numbers. When both are multiplied together, they equal the original 155 digit number that was factorized.

The mathematics involved in converting the two prime numbers (representing *p* and *q*) back into an RSA private key are complex. But the attacker could use a Python script created by Tom Ritter (Ritter, prime2pem) that uses Python RSA crypto libraries to do the math.

He would edit the prime2pem.py script and replace the "p" and "q" values with the factors. Remember that the exponent value was displayed earlier with the openssl command.

```
p@stark:~/scripts$ cat prime2pem.py
#!/usr/bin/python

import sys
from prime2pemutils import RSAKey


p = 9467592566042455828159569807202728161118317379236558752541712307507617257671 7
q = 1102627871636363336724570329030822239932141235919360404000343819769641789300 39
e = 0x10001

if __name__ == "__main__":
        if p == 0 or q == 0:
                print "Error: You need to set p and q in the code before running this"
                sys.exit(1)

        r = RSAKey(p, q)
        print r.getPEM()
```

With the details inserted, running the prime2pem script produces an RSA private key in PEM format.

```
p@stark:~/scripts$ python prime2pem.py
-----BEGIN RSA PRIVATE KEY-----
MIIBOQIBAAJBAMdR5n8In3WHqtJ1XGngXbYsItYu6ydNr81xNlw1waKBzgGv9095
yGJQmD7b01qStxlFn3vBJqq6TQ44TyeaxUsCAwEAAQJADkzj+q1Fs4r+SIc/ECGW
16EnBrLrEDDUsiqzOb5pnB0PhrNUvTEB1BKw2gd6P97rvl4yKULZEEIaJMmYo6lK
8QIhAPPGhqNkmkzPEPbOl8ndY7VZvq80BVwqQnR5ElerjGl3AiEA0VCsydEUdN/F
FEcyRdl1XuM/CuD+2XfpSY/TrV78d80CIE2ucbEXmePoCCvd4Zi+J8vecVk7Zonc
HZkoC6RMUk45AiA4BsYZRxOYrQTNhrkYJTrbaSxItyy//O6+t/bK4y/kyQIgPyK2
D27thW4tU8nKffSA38x8lfS27dQxXlp/mxBIKoE=
-----END RSA PRIVATE KEY-----
```

The attacker would place this key in a file on the mail server under his control. In the case of OpenDKIM, you edit the /etc/opendkim/SigningTable and /etc/opendkim/KeyTable and reference this new key (Simon, 2013). Then, restart the OpenDKIM service.

He could then use any mail client but a quick way to test is to use the mailx (Bergantino, 2008) command.

```
admin@ip-10-154-135-41:~$ echo "This is an authentic message from
chris@bright-armor.net" | mail -s "extremely important discussion
notes" joe@bright-armor.net -- -f chris@bright-armor.net
```

Finally, success! The attacker sent joe@bright-armor.net a DKIM signed message purportedly from chris@ that originated from an Amazon EC2 instance under the attacker's control. Joe's mail server successfully authenticated the message as having originated from the bright-armor.net domain.

```
Return-Path: <chris@bright-armor.net>
Delivered-To: joe@bright-armor.net
Received: from ip-10-154-135-41.ec2.internal (ec2-54-225-21-1.compute-
1.amazonaws.com [54.225.21.1])
       by mail.bright-armor.net (Postfix) with ESMTP id 54CCD13795
       for <joe@bright-armor.net>; Sun, 11 Aug 2013 18:09:55 -0400 (EDT)
Authentication-Results: mail.bright-armor.net; dkim=pass
       (512-bit key; insecure key) header.i=@bright-armor.net;
       dkim-adsp=pass
Received: by ip-10-154-135-41.ec2.internal (Postfix, from userid 1000)
       id A238F20364; Sun, 11 Aug 2013 22:10:14 +0000 (UTC)
DKIM-Signature: v=1; a=rsa-sha256; c=relaxed/relaxed; d=bright-armor.net;
       s=weak; t=1376259014;
       bh=/z1eq3+fq9muVVRGucBnjqmBAl0Vtx5Y/ZSYwDR239Y=;
       h=To:Subject:Message-Id:Date:From;
       b=TWPlVM+Xo2+8p+tqG7ytKgL3bl5+hqpvCvsow2VFwXve3kzNYIUcT0wBz3KsgLWm7
        4UVJUdKr4E3PDZrmT0UzA==
To: joe@bright-armor.net
Subject: extremely important discussion notes
Message-Id: <20130811221014.A238F20364@ip-10-154-135-41.ec2.internal>
Date: Sun, 11 Aug 2013 22:10:14 +0000 (UTC)
From: chris@bright-armor.net
```

Note that the "insecure key" string in the authentication header is misleading. Older versions of OpenDKIM warn of an "insecure key" when the signer is not also using DNSSEC to sign their DNS records (Kitterman, 2013).

Why would an organization implement DKIM but use a weak encryption key in spite of recommended best practices? There are two likely reasons. First, some organizations may have originally adopted the pre-cursor framework to DKIM, Domain Keys. Older Domain Keys implementations were more likely to use keys less than 1024-bits (Yahoo! Inc.). DKIM offers backwards compatibility with encryption keys used by Domain Keys. Therefore, it is possible that some organizations re-used the same key when they upgraded from Domain Keys to DKIM. Secondly, large organizations who send significant amounts of e-mail (millions per day) may be leery of the computational cost of the cryptographic operations supporting DKIM (hashing and private key signing) (Crocker, Hansen, & Kucherawy, 2011). A 1024-bit key or higher, though preferred for security reasons, may negatively tax CPU resources on systems processing large volumes of mail.

The weak key issue has been mostly solved now. Large Internet service providers have already changed to 1024-bit or higher keys. However, RFC 6376 still requires DKIM implementations to be able to *verify* 512-bit keys. But since old installation software packages

are still in circulation and neglected implementations have been "switched on" but receive little ongoing maintenance or attention, there are likely still installations of DKIM using 512-bit keys.

13. Stealing a Private Key

In addition to the offline attack described above, attackers can compromise your DKIM implementation by stealing or replacing your private key as well.

An attacker could steal a private key from your mail server or that of your delegate. Most implementations of DKIM do not protect the private key with a passphrase since it would have to be entered each time the key was accessed and used, making its use impractical on a mail server. At the same time, this means that an attacker only needs access to the key file in order to appropriate it.

As previously discussed, in DKIM, key management is largely informal. By design, there is no public key infrastructure with accompanying revocation procedures to manage keys. So, organizations must protect the keys using alternate processes. At the most basic level, keys stored on mail servers should be configured with appropriate file permissions. Using the OpenDKIM implementation as an example, the private key should be owned by the 'opendkim' user and group and have mode 600 permissions.

```
-rw------- 1 opendkim opendkim  887 Mar 31 15:24 one.private
```

Like most other service or daemon accounts, no one should be able to logon interactively as the opendkim user. The opendkim install package will set the opendkim's shell to /bin/false.

As mentioned before, delegates have to be carefully monitored. An attacker may target their network in search of the private key instead of your own.

14. Attacking DNS

If an attacker cannot gain access to your mail server, he could attack your DNS instead. An attacker could generate a key pair and selector and associate it with your domain name by replacing your public key in DNS with his own. That would allow him to send emails from systems under his control that would successfully authenticate while yours would not. That attack would be somewhat noisy; eventually you would receive feedback that your DKIM-signed messages were failing to verify. So, a smart attacker might create a new subdomain with which

to associate his selector and key. That way, the target's mail would function as normal and the attacker would be able to send signed email.

Finally, SPF and Sender-ID were reviewed earlier. Since they are extremely simple to configure, they offer a valuable complement to DKIM. In the offline attack example described above, if SPF or Sender-ID were enabled, they would have reported a failure status in the Authentication-Results header field since the message came from an unknown mail host. Though the DKIM signature was successfully verified, the other failures might have triggered some action by the SPAM filtering software or else drawn the attention of an incident handler if this were evidence in an investigation.

15. Closing

DKIM offers an easy method for organizations to protect their email reputation. But the purpose of DKIM encompasses more than just increasing the chance that your mail will be reliably delivered. DKIM provides more intangible benefits for those who take responsibility for email originating from their domain. In addition, organizations who verify DKIM signatures can fortify their SPAM filtering operations and increase protections against Phishing attacks. Information security personnel should appreciate that the integrity of DKIM depends solely on encryption keys that must be protected with the same care used to protect encryption keys for other services.

Chris Murphy, chrismrph0@gmail.com

16. Appendix I: Using StarCluster to Create and Manage a Cluster on Amazon's EC2 Network

Zach Harris mentioned using StarCluster on the CADO-NFS mailing list (Harris Z. , Multiple MPIs, 2012). Another crypto researcher (Ritter, factoring-howto.txt) uses the BOINC grid-computing software to control a large cluster of Amazon EC2 instances.

Though you can use the Amazon AWS web management console, managing your cluster with one of the above tools will save considerable time and let you concentrate on your main task instead of laborious cluster system administration.

StarCluster (Software Tools for Academics and Researchers) is indispensable if you are going to be creating your own cluster using Amazon EC2 instances. It is a Python framework created at MIT that automates most of the tedious EC2 management tasks. You can create clusters of EC2 instances, create Amazon Machine Instances (AMI), create Elastic Bock Storage (EBS) volumes, mount volumes, etc. with simple one-line commands.

The StarCluster User Manual is excellent: http://star.mit.edu/cluster/docs/latest/manual/

16.1. Configuration file

The first time you run StarCluster, it will prompt you to create a well-commented configuration file in your home directory that you can edit. Thereafter, StarCluster will reference this configuration file for options unless you specify a different one to use.

Here is one of the config files I used, with my AWS authentication credentials redacted.

```
[global]
DEFAULT_TEMPLATE=smallcluster
[aws info]
AWS_ACCESS_KEY_ID = <removed>
AWS_SECRET_ACCESS_KEY = <removed>
AWS_USER_ID= <removed>
[key <removed>]
KEY_LOCATION=/home/p/aws/<removed>.pem
[cluster smallcluster]
KEYNAME = <removed>
CLUSTER_SIZE = 4
CLUSTER_USER = sgeadmin
CLUSTER_SHELL = bash
NODE_IMAGE_ID = ami-ada3d3c4
NODE_INSTANCE_TYPE = cc2.8xlarge
VOLUMES = fdata
[volume fdata]
VOLUME_ID = vol-814ccedb
MOUNT_PATH = /fdata
```

Note that the config file comments list the different EC2 instance types you can use. The Amazon EC2 web site has more details on the specs and importantly, the cost, of each.

```
# instance type for all cluster nodes
# (options: cg1.4xlarge, c1.xlarge, m1.small, c1.medium, m2.xlarge,
t1.micro, cc1.4xlarge, m1.medium, cc2.8xlarge, m1.large, m1.xlarge,
m2.4xlarge, m2.2xlarge)
```

You will want to carefully decide if you want a large cluster of many inexpensive instances or a small cluster of expensive instances. Note also that while the cc2.8xlarge is one of the most expensive instances per hour, it is also the most cost effective since the cost per processing unit is among the lowest of all the offerings.

16.2. Modify an Amazon Machine Image (AMI)

StarCluster provides a collection of cluster-friendly Ubuntu and CentOS AMIs to use.

You will have to start with an existing AMI and modify it to your needs. This may include installing any specialized software your need or just running *apt-get update*; *apt-get dist-upgrade* to ensure you have the latest security updates. To factor a number, you will have to install and configure the CADO-NFS software.

Once you are finished modifying your AMI, you can save your modified version as a new AMI to use for the nodes in your cluster.

```
        starcluster start -o -s 1 -i m1.small -n ami-765b3e1f cmfactor

        id: i-18a30b73
        dns_name: ec2-50-19-188-93.compute-1.amazonaws.com
        private_dns_name: ip-10-194-19-184.ec2.internal
        state: running
        public_ip: 50.19.188.93
        private_ip: 10.194.19.184
        zone: us-east-1c
        ami: ami-765b3e1f
        type: m1.small
        groups: @sc-cmfactor
        keypair: <removed>
        uptime: 0 days, 00:27:35


        starcluster ebsimage i-18a30b73 cmfactor-img

        >>> Removing private data...
        >>> Creating EBS image...
        >>> Waiting for AMI ami-5fadc436 to become available...
        >>> create_image took 2.888 mins
>>> Your new AMI id is: ami-5fadc436
```

Note that you have to select a hardware virtual machine (HVM)-based AMI if you are using the any of the computer cluster instances (e.g., cc2.8xlarge). You can run *starcluster listpublic* to see all the AMIs provided by the project.

16.3. Create an EBS volume

You will need to create an EBS volume to store the output files created by CADO-NFS while factoring a 155-digit number. Based on your config file, StarCluster will share the EBS volume amongst all your nodes using NFS.

```
starcluster createvolume --name=my-fdata 70 us-east-1c
>>> New volume id: vol-a1a1fbf9
```

16.4. Create a cluster

When your custom AMI and EBS volume have been created, and you selected both the EC2 instance type and the number of nodes you want in your cluster, you can edit your config file to reflect your choices and start the cluster.

```
p@stark:~$ starcluster start cmfactor0603
StarCluster - (http://web.mit.edu/starcluster) (v. 0.93.3)
Software Tools for Academics and Researchers (STAR)
Please submit bug reports to starcluster@mit.edu

>>> Using default cluster template: smallcluster
>>> Validating cluster template settings...
>>> Cluster template settings are valid
>>> Starting cluster...
>>> Launching a 5-node cluster...
>>> Creating security group @sc-cmfactor0603...
Reservation:r-e58c518e
>>> Waiting for cluster to come up... (updating every 30s)
>>> Waiting for all nodes to be in a 'running' state...
[…output snipped…]
20/20 |||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||| 100%
>>> Configuring cluster took 1.629 mins
>>> Starting cluster took 3.414 mins
```

Once a cluster is running, you can list it. This command is more valuable if you are managing multiple clusters at one time.

```
p@stark:~$ starcluster listclusters
StarCluster - (http://web.mit.edu/starcluster) (v. 0.93.3)
Software Tools for Academics and Researchers (STAR)
Please submit bug reports to starcluster@mit.edu

-----------------------------------------------
cmfactor0603 (security group: @sc-cmfactor0603)
-----------------------------------------------
Launch time: 2013-06-03 21:40:45
Uptime: 0 days, 00:04:07
Zone: us-east-1c
Keypair: <removed>
EBS volumes:
    vol-a1a1fbf9 on master:/dev/sdz (status: attached)
Cluster nodes:
     master running i-de8a38bf ec2-107-20-56-28.compute-1.amazonaws.com
    node001 running i-d88a38b9 ec2-50-19-34-35.compute-1.amazonaws.com
    node002 running i-da8a38bb ec2-54-234-70-64.compute-1.amazonaws.com
    node003 running i-d48a38b5 ec2-23-20-56-40.compute-1.amazonaws.com
    node004 running i-d68a38b7 ec2-75-101-187-145.compute-1.amazonaws.com
Total nodes: 5
```

16.5. Connect to a cluster

This command logs you onto the master node as root.

```
starcluster sshmaster cmfactor0603
```

From that ssh session, you can easily logon to other nodes with "*ssh node004*" or run remote commands like "*ssh node008 ps –ef*". The /etc/hosts file is populated by StarCluster so you don't have to reference Amazon host names to manage the cluster.

When you are done with your problem or task, and you want to stop the EC2 charges, you have to terminate the cluster. Using *stop* instead of *terminate* will shut down all the nodes but the virtual resources are still reserved so you will still be charged.

```
starcluster terminate cmfactor0603
```

The above listed command will destroy the cluster and stop the EC2 charges. Your EBS volume will remain available to attach to another host until you destroy it. Monthly storage charges for EBS volumes are minimal.

17. Appendix II: Using CADO-NFS on Amazon's EC2 Network

If you are going to run CADO-NFS on Amazon EC2 instances, I recommend installing and running it locally to become familiar with it first. It does not make sense to learn the software *while* you are paying for expensive cloud resources. It installs easily (make; make install) on Ubuntu Linux after installing the following packages: build-essential, cmake, libgmp3-dev and optionally mpich2 (CADO-NFS Project). Note that if you use one of the StarCluster Ubuntu AMIs, you do not need to install the mpich2 package. The StarCluster AMIs already have OpenMPI installed so if you run CADO-NFS in parallel on several computers, it will crash when it starts the linear algebra stage since it won't know which Message Passing Interface (MPI) binary to use (Harris Z. , Multiple MPIs, 2012).

For testing CADO-NFS locally, I recommend factoring a 100-digit number. On a Core i7 laptop, you can factor a 100-digit number in roughly one hour and forty-five minutes. That gives you the opportunity to watch all stages invoked by the cadofactor.pl script and examine the output files it creates. Thus, when you do try to factor a larger number in the cloud, you will have a sense of the overall progress based on the current running stage. You can also test the recovery process by simulating a crash with Ctrl + C. You can use the RSA-100 challenge number and compare your results to those published to make sure it worked (Aoki, Kazumaro, Kida, Yuji, Shimoyama, & Ueda, 2004).

The screen program on Linux will prove useful whether you are running locally or in the cloud. Though CADO-NFS creates numerous log and output files, the script runs interactively and the current status is reported to stdout. Thus, if you disconnect your ssh session, the

cadofactor.pl script will be killed. Using screen, you can leave it running when you log off your ssh session.

When running CADO-NFS inside a screen session, you can press Ctrl + A and then D to disconnect so you can do other things in your terminal. Then, when you want to reconnect to back to the running cadofactor.pl script, you can type "screen –r <session name>".

If CADO-NFS crashes or is interrupted for any reason, it is designed to recover the existing work and pick up where it left off. Running the exact same command will prompt you with a recovery menu. If the software crashed due to a bug, you may have to ask for help on the CADO-NFS mailing list.

Though I was not able to achieve the relative low-cost or quick completion times reported by other researchers (Ritter, factoring-howto.txt) (Kleinjung, Lenstra, Page, & Smart, N.P., 2011), I finished a factorization on Amazon EC2 in roughly 113 hours, or just under five days.

You will need to create an Elastic Block Storage (EBS) volume on which to store the CADO-NFS output files. I created a 70 GB drive and used 49 GB of it to complete the job.

```
root@master:~# du -hc /fdata
17G    /fdata/cadowork/c155.bwc
4.1G   /fdata/cadowork/c155.nodup/1
4.1G   /fdata/cadowork/c155.nodup/0
8.2G   /fdata/cadowork/c155.nodup
49G    /fdata/cadowork
16K    /fdata/lost+found
49G    /fdata
49G    total
```

The StarCluster documentation includes instructions to create a single-node volume host cluster to create and configure this volume. Once it is created, you terminate the host cluster and thereafter, you can reference the volume in your StarCluster config file so it will be mounted on the master node and shared via NFS to all the other nodes in your cluster. Most of the AMIs have only 8 GB local drives so there will not be enough space to complete factorization without a dedicated, shared EBS volume.

Cadofactor.pl is the parent script that will call all the other binaries for each stage in the factoring process. The stages include polynomial selection, sieving, merge, linear algebra,

square root, etc. and each performs a single important stage in the factoring process. The status is still reported to stdout, but an empty file will be created after each stage completes.

```
root@master:/fdata/cadowork# ls -la *_done
-rw-r--r-- 1 root root 0 2013-06-30 19:31 c155.chars_done
-rw-r--r-- 1 root root 0 2013-06-27 02:27 c155.dup_done
-rw-r--r-- 1 root root 0 2013-06-23 07:09 c155.factbase_done
-rw-r--r-- 1 root root 0 2013-06-23 07:09 c155.freerels_done
-rw-r--r-- 1 root root 0 2013-06-30 19:27 c155.linalg_done
-rw-r--r-- 1 root root 0 2013-06-27 02:56 c155.merge_done
-rw-r--r-- 1 root root 0 2013-06-23 07:09 c155.polysel_done
-rw-r--r-- 1 root root 0 2013-06-27 02:27 c155.purge_done
-rw-r--r-- 1 root root 0 2013-06-27 03:05 c155.replay_done
-rw-r--r-- 1 root root 0 2013-06-27 02:27 c155.sieve_done
-rw-r--r-- 1 root root 0 2013-06-30 20:17 c155.sqrt_done
```

If you are going to distribute the work of CADO-NFS across multiple hosts, you need to use MPI, a library for sharing jobs among multiple computers. By default, CADO-NFS does not enable MPI support so you must enable it. Before you compile the software, copy the local.sh.example file to local.sh. Go to the end of the file and add "MPI=1". Then compile and install CADO-NFS with *make* and *sudo make install*.

The command I used to start the factoring was simple. I specified the path to the params.c155 file since I was factoring a 155-digit integer. Normally, you should not have to edit this parameters file since it has been optimized. I also specified my working directory where the logs and output files will be stored. Finally, I specified the number to factor.

```
perl ./cadofactor.pl params=params/params.c155
wdir=cadowork
n=104392314406156487614906424739392400575499044449578923894941415754616529178714491704310469406195111828195377724291749266824628139742010635962343107033019 63
```

cadofactor.pl will also assume the wdir contains an important file called mach_desc. Otherwise, you can point to this explicitly on the command line. This configuration file details the names of the cluster machines it will pass jobs to. Even if you are running it on a single machine, you still need to create a mach_desc to record important details.

Here is an example of mach_desc for a cluster:

```
tmpdir=/tmp
bindir=/opt/cado-nfs/build/ip-10-155-196-80
localhost cores=32 mpi=1

node001 cores=32 mpi=1
node002 cores=32 mpi=1
node003 cores=32 mpi=1
[…]
```

This example works on a single machine:

```
[localhost]
tmpdir=/tmp
bindir=/opt/cado-nfs/build/$HOSTNAME
localhost cores=1
```

18. References

amavisd-new. (n.d.). Retrieved August 14, 2013, from www.amavis.org: http://www.amavis.org/

Aoki, Kazumaro, Kida, Yuji, Shimoyama, T., & Ueda, H. (2004). GNFS Factoring Statistics of RSA-100, 110, ..., 150. *Cryptology ePrint Archive, Report 2004/095*, 1.

Bergantino, P. (2008, September 23). *Specify the from user when sending email using the mail command*. Retrieved August 5, 2013, from StackOverflow: http://stackoverflow.com/questions/119390/specify-the-from-user-when-sending-email-using-the-mail-command

Bonar, A. (2010, May 14). *IP & Sender Reputation: What is it, Why do you care and What is your rating?* Retrieved July 25, 2014, from emailexpert.org: http://emailexpert.org/ip-sender-reputation-what-is-it-why-do-you-care-and-what-is-your-rating

CADO-NFS Project. (n.d.). *CADO-NFS.* Retrieved March 25, 2013, from cado-nfs.gforge.inria.fr: http://cado-nfs.gforge.inria.fr

Chapter 56 - Support for DKIM. (n.d.). Retrieved August 14, 2013, from www.exim.org: http://www.exim.org/exim-html-current/doc/html/spec_html/ch-support_for_dkim_domainkeys_identified_mail.html

Comprehensive Perl Archive Network. (n.d.). *MAIL-DKIM*. Retrieved August 14, 2013, from CPAN: http://search.cpan.org/dist/Mail-DKIM/]

Crocker, D., Hansen, T., & Kucherawy, M. (2011, September). RFC 6376: DomainKeys Identified Mail (DKIM) Signatures.

DKIM Software and Services Deployment Reports. (n.d.). Retrieved August 14, 2013, from DKIM.org: http://dkim.org/deploy/index.html

DMARC.org. (n.d.). *DMARC Overview*. Retrieved May 9, 2013, from DMARC.org: http://www.dmarc.org/overview.html

Eland Systems. (2009). *DKIM*. Retrieved March 29, 2013, from www.elandsys.com: http://www.elandsys.com/resources/mail/dkim/opendkim.html

Harris, Z. (2012, July 24). *Multiple MPIs*. Retrieved March 25, 2013, from Cado-nfs-discuss: http://lists.gforge.inria.fr/pipermail/cado-nfs-discuss/2012-July/000077.html

Harris, Z. (2012, October 24). *Thanks*. Retrieved March 25, 2012, from Cado-nfs-discuss: http://lists.gforge.inria.fr/pipermail/cado-nfs-discuss/2012-October/000098.html

Harris, Z. (n.d.). *Category:Cracked DKIM keys*. Retrieved April 15, 2013, from mt633.redirectme.net/wiki: http://mt633.redirectme.net/wiki/Category:Cracked_DKIM_keys

Kitterman, S. (2013, January 8). *opendkim: reports "insecure key" in all AR headers*. Retrieved August 14, 2013, from bugs.debian.org: http://bugs.debian.org/cgi-bin/bugreport.cgi?bug=697583

Kleinjung, T., Lenstra, A., Page, D., & Smart, N.P. (2011). Using the Cloud to Determine Key Strenghts. *http://www.cs.bris.ac.uk/~nigel/Cloud-Keys/*.

Kucherawy, M. (2009, April). RFC 6577: Message Header Field for Indicating Message Authentication Status.

Mail-DKIM and DKIMproxy. (n.d.). Retrieved August 14, 2013, from dkimproxy.sourceforge.net: http://dkimproxy.sourceforge.net

Mehnle, J. (n.d.). *Sender Policy Framework Introduction*. Retrieved May 9, 2013, from www.openspf.org: http://www.openspf.org/Introduction

Messaging, Malware and Mobile Anti-Abuse Working Group. (2012, November). M3AAWG Best Practices for Implementing DKIM To Avoid Key Length Vulnerability. San Francisco, CA, USA.

Microsoft Corporation. (2004, September 30). *Sender ID Framework Overview*. Retrieved May 9, 2013, from www.microsoft.com: http://www.microsoft.com/mscorp/safety/technologies/senderid/overview.mspx

OpenDKIM. (n.d.). Retrieved August 14, 2013, from www.opendkim.org: http://www.opendkim.org/

Piasecki, N. (2010, December 23). *DKIM Signing Outbound Messages in Exchange 2007*. Retrieved March 29, 2013, from Simply Does Not Work: Confessions of a Small Business Software Developer: http://nicholas.piasecki.name/blog/2010/12/dkim-signing-outbound-messages-in-exchange-server-2007

Profanter, S. (n.d.). *dkim-exchange*. Retrieved March 29, 2013, from github.com: https://github.com/Pro/dkim-exchange

Resnick, P. (2008, October). RFC 5322: Internet Message Format.

Ritter, T. (n.d.). *factoring-howto.txt.* Retrieved July 23, 2013, from github.com: https://github.com/tomrittervg/cloud-and-control/blob/5165cf6c2112467c14a702b1e63f1e0cff6beb23/gnfs-info/factoring-howto.txt

Ritter, T. (n.d.). *prime2pem*. Retrieved July 23, 2013, from github.com: https://github.com/tomrittervg/prime2pem/blob/master/prime2pem.py

Ruiz Duarte, E. (2011, June 13). *How to break RSA explicitly with OpenSSL keys*. Retrieved April 3, 2013, from beck's site - Cryptography, algebraic geometry, and some algorithms: http://b3ck.blogspot.com/2011/06/how-to-break-rsa-explicitly-with.html

Santos, H. (2009, July 23). The good ol' "t=" tag in key records. *ietf-dkim mailing list*.

Chris Murphy, chrismrph0@gmail.com

Simon. (2013, July 3). *Install and Configure OpenDKIM on Debian Squeeze*. Retrieved March 29, 2013, from RoseHosting Linux Blog: http://www.rosehosting.com/blog/install-and-configure-opendkim-on-debian-squeeze/

Software Tools for Academics and Researchers. (n.d.). *StarCluster User Manual.* Retrieved March 25, 2014, from star.mit.edu: http://star.mit.edu/cluster/docs/latest/manual/index.html

Spiezle, C., & Nikolayev, A. (2006, December). Fighting Spam and Phishing with Sender ID. *TechNet Magazine*, p. 1.

Takahashi, K. (2013, February 1). *Google Doubles Down on Weak DKIM Keys: What You Need to Do Now to be Compliant*. Retrieved from Return Path Blog: http://blog.returnpath.com/blog/ken-takahashi/google-doubles-down-on-weak-dkim-keys-what-you-need-to-do-now-to-be-compliant

The Regents of the University of California. (n.d.). *Wepawet*. Retrieved April 7, 2013, from wepawet.iseclab.org: http://wepawet.iseclab.org/

US-CERT. (2012, October 24). *DomainKeys Identified Mail (DKIM) Verifiers may inappropriately convey message trust.* Retrieved October 24, 2012, from Vulnerability Notes Database: http://www.kb.cert.org/vuls/id/268267

Yahoo! Inc. (2004). *Yahoo! DomainKeys Patent License Agreement v1.1.* Retrieved August 14, 2013, from domainkeys.sourceforge.net: http://domainkeys.sourceforge.net/license/patentlicense1-1.html

Yahoo! Inc. (n.d.). *DomainKeys Public/Private Key-pair Generation*. Retrieved August 14, 2013, from domainkeys.sourceforge.net: http://domainkeys.sourceforge.net/keygen.html

Zetter, K. (2012, October 24). How a Google Headhunter's E-Mail Unraveled a Massive Net Security Hole. *Wired Magazine*.