

M801 Research Project and Dissertation

Trusted Email using Hashcash

Email spam is a growing problem; current anti-spam techniques do little to minimise the volumes. Would a new approach such as Hashcash proof-of-work prove a viable and universal standalone solution?

A dissertation submitted in partial fulfilment of the requirements for the Open University's Master of Science Degree in Computing for Commerce and Industry

By

John Stephen Honan

Personal Identifier	T4131552
Date of Submission	14th September 2005
Word Count	15,297

Preface

Like most users of email I am sick of spam. I've tried many of the technical solutions out there, including various blacklisting and filtering solutions. Unfortunately the spammers were keeping ahead of the anti-spammers and their junk was still reaching me. Even worse, legitimate email was getting flagged as spam and blocked.

As the volumes of spam in my in-box increased, I started looking at the new anti-spam approaches and solutions being proposed and developed across the industry. This led to the subject of my research.

With thanks to my supervisor Dr Rob Walker for his invaluable guidance. To Eamon Honan for proof-reading. And to Adam Back and all the members of the Hashcash mailing-list for answering my questions.

Contents

1. Introduction	7
1.1 Overview	7
1.2 The problem domain	7
1.2.1 What is spam?.....	7
1.3 The research question	8
1.4 Objective	9
1.5 Aims	9
1.6 Chapter summary	10
2. How Spammers Operate	12
2.1 Weaknesses in SMTP.....	12
2.1.1 How spammers exploit SMTP	13
2.1.2 Remaining anonymous.....	13
2.1.3 Obtaining email addresses.....	15
2.1.4 Fooling filtering software	15
2.2 Why spammers send spam; profitability	15
2.3 Requirements of an anti-spam solution	16
2.3.1 Block all spam	16
2.3.2 No false positives	16
2.3.3 Transparent to the user.....	16
2.3.4 Can be implemented universally	17
2.3.5 Minimises network spam volume	17
2.4 Summary	17
3. Existing Anti-Spam Methods	18
3.1 Social	21
3.1.1 Social evolution	21
3.1.2 Non-response to spam	22
3.1.3 Non-dissemination of email address	22
3.2 Political	22

3.2.1 Legal	22
3.2.2 Charge for email	24
3.2.3 Policy framework	24
3.2.4 Accountability	24
3.3 Technical	25
3.3.1 Filtering	25
3.3.2 Stamps	28
3.3.3 Trust and policy framework	29
3.4 Hybrid approach	31
3.5 Comparison of anti-spam solutions	32
3.6 Summary	33
4. Hashcash in Detail	35
4.1 What is Hashcash?	35
4.2 How does Hashcash work?	36
4.3 How Hashcash computes hash-collisions	37
4.4 Partial hash-collisions explained	38
4.5 Summary	42
5. Evaluating the Hashcash Approach	43
5.1 Choosing a method	44
5.2 Potential methods	45
5.2.1 Field testing	46
5.2.2 Analytical approach	46
5.2.3 Laboratory simulation	46
5.3 Chosen method	46
5.4 Research instruments	48
5.5 Summary	49
6. Results and Analysis	50
6.1 Part One: Spammer profitability and breakeven point	50
6.2 Part Two: Hashcash stamping performance measurement	56
6.2.1 Setting up the experiment	56

6.2.2 Results of the experiment.....	57
6.2.3 Extrapolating minting times	57
6.3 Review of experimental method.....	58
6.4 Analysis of results	59
6.5 Summary.....	61
7. Conclusion	62
7.1 Answering the research question	62
7.2 The future of proof-of-work	62
7.3 Further work.....	63
7.3.1 Memory-bound proof-of-work function	63
7.3.2 Ticket servers	64
Appendices	65
Appendix A: The SMTP Protocol	65
Appendix B: Full email header including Hashcash stamp.....	69
Appendix C: Hashcash GUI front-ends.....	70
References	71

Abstract

The purpose of this research project is to investigate the problem of email spam, and identify possible methods to minimise the volumes. The analysis focuses on the Hashcash proof-of-work approach, and investigates the feasibility of a Hashcash based solution.

A potential problem with proof-of-work is that disparity across different powered computers may result in some unfortunate users spending a disproportionately long time calculating a stamp. An experiment is carried out to time how long it takes to calculate stamps across a variety of processor speeds.

It is concluded from the analysis of the results that due to this problem of egalitarianism, Hashcash (or CPU-bound proof-of-work in general) is not a suitable approach as a stand-alone anti-spam solution.

It appears that a hybrid anti-spam system in conjunction with a legal and policy framework is the best approach. But this needs to be agreed upon and implemented by the main industry players and technical bodies in order to be successful.

1. Introduction

1.1 Overview

Email spam is a growing problem. There are many techniques in use to minimise it, with varying degrees of success. This research project will assess the effectiveness of existing methods, and then look in detail at proposed new anti-spam approaches, evaluating the feasibility of 'proof-of-work' as a potential solution.

1.2 The problem domain

1.2.1 What is spam?

Spam can be defined as unsolicited e-mail, often of a commercial nature, sent indiscriminately to multiple mailing lists, individuals, or newsgroups. Spam can be broadly categorized as follows (Cranor and LaMacchia, 1998)

- Junk mail - mass mailings from legitimate businesses that is unwanted.
- Non-commercial spam – mass mailings of unsolicited messages without an apparent commercial motive including chain letters, urban legends and joke collections.
- Offensive and Pornographic spam – mass mailings of “adult” advertisements or pornographic pictures.
- Spam scams – mass mailings of fraudulent messages or those designed to con people out of personal information for the purpose of identity theft and other criminal acts.
- Virus spam – mass mailings that contain viruses, Trojans, malicious scripts, etc.

As a communications medium, email has become very useful and practically universal. However, the usefulness of email and its potential for future growth are jeopardized by the rising tide of unwanted email, both spam and viruses. This threatens to wipe out the advantages and benefits of email. Table 1 shows how much of a problem spam has become.

Email Statistics (2003 Averages)	
Daily emails sent:	31 billion
Daily emails sent per email address:	56
Daily emails sent per person:	174
Daily emails sent per corporate user:	34
Daily emails received per person:	10
Spam Statistics (2003 Averages)	
Email considered Spam:	40% of all email
Daily Spam emails sent:	12.4 billion
Daily Spam received per person:	6
Annual Spam received per person:	2,200
Spam costs to all non-corp Internet Users:	\$255 million
Spam costs to US corporations in 2002:	\$8.9 billion
Estimated Spam Increase by 2007:	63%
Annual Spam in 1,000 Employee Company:	2.1 million
<i>Source: http://www.spamfighter.org/bb2/content.php?article.23</i>	

Table 1: Spam Statistics

1.3 The research question

Is Hashcash proof-of-work a single technique which meets all the requirements of an anti-spam solution?

Existing anti-spam measures are not effective against eliminating spam. One problem is that they focus on filtering spam at the recipient end, and do not deter the spammer from sending it in the first place.

Current industry proposals present new approaches to anti-spam, including a technical method called 'proof-of-work', based on the sender's CPU expending a certain amount of processing power to generate a virtual stamp which is attached to outgoing email. This approach was first proposed by Dwork and Naor (1992) in their seminal paper 'Pricing via Processing or Combating Junk Mail'.

Proof-of-work is fundamentally different to existing anti-spam methods as it focuses on impacting the spammer's profitability by slowing down their email transmission rate, and therefore deterring them from sending spam in the first place.

Adam Back developed Dwork's proposal into a working implementation of proof-of-work based on the SHA-1 algorithm entitled 'Hashcash' (Back, 1997). Hashcash is currently in the beta test stage of development (Back, 2002), although the code is freely available it has not yet been deployed or accepted by a wider audience.

An issue with Hashcash is that of disparity of performance across users computers; while calculating stamps may impact a spammers profitability, it may also impact certain users ability to effectively send email. Therefore Hashcash may not be a viable universal approach to anti-spam.

1.4 Objective

The objective is to answer the research question; "Is Hashcash proof-of-work a single technique which meets all the requirements of an anti-spam solution?". The question can be broken down thus:

- What are the requirements of an effective anti-spam solution?
- What are the existing anti-spam techniques and why are they not effective?
- Is Hashcash proof-of-work viable as a standalone anti-spam solution?

1.5 Aims

The following tasks will answer these questions, and help in forming a conclusion to the research question:

- Identify the weaknesses of SMTP and how spammers exploit it
- Define the requirements of an anti-spam solution

- Describe and compare existing anti-spam methods. Do they meet the requirements?
Why aren't they effective?
- Explain how Hashcash works, and why it is different to other anti-spam methods
- Primary data gathering. There are two pieces of primary data which need to be obtained; spammer breakeven points, and Hashcash stamp performance across various PCs;
 - Analyse spammer profitability and breakeven. Explain why spammer profitability and breakeven is core to the Hashcash approach.
 - Determine spammer breakeven points (to be analysed in conjunction with experimental data)
 - Define an experimental approach to testing Hashcash performance across devices
 - Perform the experiment and collect primary data
 - Analyse the results and draw conclusions
- Based on the results of the comparisons and the analysis of the primary data, answer the question: Is Hashcash proof-of-work a single technique which meets all the requirements of an anti-spam solution?

1.6 Chapter summary

Chapter 2 begins by describing weaknesses in SMTP (Simple Mail Transport Protocol), the protocol used to transmit email, and how it is exploited by spammers. The main reason for spammers sending spam is profitability, this concept is introduced (to be built on in later chapters). The chapter concludes with a summary of the main requirements of a successful anti-spam solution.

In Chapter 3 existing and proposed anti-spam solutions are categorised and reviewed in detail. Proof-of-work is identified as a novel and promising approach to anti-spam. Each of the technical anti-spam methods is then compared against the requirements from Chapter 2,

Chapter 4, focuses on a proof-of-work email stamping system called 'Hashcash'. Describes how Hashcash works, and how it is used to slow down the spammer thus impacting their profitability.

The primary experiment methods and research technique is determined and described in Chapter 5.

Chapter 6 presents and analyses the primary data. We calculate how much spam a spammer needs to send to stay in business, and how his profitability model works. If you can slow the spammer down enough to make it unprofitable, then you might stop the flow of spam. The performance of Hashcash across different processors is measured. The results of the experiments are discussed and the feasibility of Hashcash as part of an anti-spam solution is determined.

Chapter 7 contains conclusions and recommendations for further work.

2. How Spammers Operate

This chapter examines how spammers operate. It describes how spammers are able to exploit flaws in SMTP (Simple Mail Transport Protocol) to send spam. Spammer profitability, the main reason behind spamming, is discussed. Finally the requirements for an effective anti-spam solution are outlined.

2.1 Weaknesses in SMTP

The Simple Mail Transport Protocol is the main protocol that email servers use to send and receive email. It was designed and implemented early in the development of the Internet, at a time when spam was unheard of. SMTP has not changed for decades, and the simplicity and openness of the protocol exposes certain weaknesses which are now being exploited by spammers. SMTP is described in more detail in Appendix A.

A major flaw in current email standards (most notably SMTP) is the lack of any technical requirement that ensures the reliable identification of the sender of messages. A message's domain of origin can easily be faked, or 'spoofed'.

Spoofing (Templeton and Levitt, 2003) is a technique often used by spammers to make them harder to trace. Trojan viruses embedded in email messages also employ spoofing techniques to ensure the source of the message is more difficult to locate (Ishibashi et al., 2001).

Spam filters and virus scanners can only eliminate a certain amount of spam and also risk catching legitimate emails. As the SoBig virus (Levy, 2003) has demonstrated, virus scanners themselves actually add to the email traffic through notification and bounceback messages.

SMTP is flawed in that it allows these email headers to be faked, and does not allow for the sender to be authenticated as the 'real' sender of the message. If this problem can be solved, it will result in a reduction in spam email messages, more security for existing emails, and

allow email viruses to be tracked down and stopped more effectively (Schwartz, 1998). This approach is known as 'Trusted Email'.

2.1.1 How spammers exploit SMTP

Spammers exploit SMTP through a number of flaws in the protocol;

- No verification of identity, the SMTP server accepts who you say you are without question
- No consequences for dishonest addressing ('From' line can be anything you want)
- Content filtering requires delivery (the email has to be received by the server before it can be filtered)
- Nothing on which to base delivery routing options (no generic flags in the header or elsewhere to allow an email to be flagged as an advertisement, adult content, newsletter, or otherwise)
- No consequences for dishonest content (where the actual message contents do not match the subject line)

SMTP allows 'untrusted' communications to take place (Tserefos et al, 1997). There is no requirement to prove you are who you say you are when an SMTP communication is instigated. SMTP is also very effective at sending email as quickly as possible to its destination, meaning the spammer is only slowed down by the speed of his connection to the internet.

These problems offer the spammer two main advantages which allow them to continue spamming unhindered; anonymity and volume of spam (Simpson, 2002).

2.1.2 Remaining anonymous

Apart from the obvious methods of supplying incorrect information in the email header, many spammers go to great lengths to remain anonymous. Anonymity is important to a spammer, because if they can be tracked down to an ISP they risk having their email servers shut down

or their web-hosting account terminated. In light of the new 'Can-Spam' anti-spam legislation, spammers are even more determined to remain unaccountable.

One additional technique used by spammers is exploitation of open email relays (Cerf, 2005). These are SMTP servers that are incorrectly configured and which allow email to be forwarded to addresses outside of the server's domain. On inspecting the email headers, it appears as if the relay server is the source of the email (Hastings and McLean, 1996).

In order to trace the spammer, the owner of the open relay needs to be made aware of the activity. However, some sites are either reluctant to act or willingly abet spammers. Furthermore, even when spammers are identified and an ISP removes their account, they will often open a new one immediately and carry on their activities.

Many system administrators are aware of techniques that spammers use and have configured email servers correctly and securely. However, spammers have discovered the capability of using common web mail form handling software as open relays. Many websites provide form applications such as mailto and the FormMail perl scripts to allow users to construct forms, the input to which can be forwarded to a specified email address for collecting information.

Spammers write software which exploits security holes in FormMail scripts to enable them to forward email to an address they specify. This results in the spam appearing to originate from the website of the FormMail software, a most undesirable outcome (Simpson, 2002).

2.1.3 Obtaining email addresses

Spammers email in bulk using automatic email-sending programs. They must first obtain email addresses. Sources for email addresses include (Pfleeger and Bloom, 2005)

- Scavenging for them on Web sites and bulletin boards (using software called robots, spiders, or spam-bots),
- Guessing (using a dictionary attack - pairing randomly generated usernames with known domain names),
- Purchasing lists of names from brokers (which can contain millions of addresses)

2.1.4 Fooling filtering software

Spammers ensure their spam is not blocked by spam-filtering software by making the spam look like legitimate email, by avoiding excessive use of HTML or exclamation marks, or by misspelling commonly used spam phrases and words (Goodman and Rounthwaite, 2004).

2.2 Why spammers send spam; profitability

Currently a spammer is only limited by the speed of their uplink and their available hardware as to how many spam messages they can send per day. Spammers are usually specialists in their field. They are employed on a cost per email or sometimes a response commission basis (Boutin, 2004).

In order to make it unprofitable for the spammers to stay in business, it is necessary to reduce the amount of spam they send, or make it costly to send each message. It should be possible based on existing knowledge about spammer's business models to calculate how much the rate of spamming needs to be slowed down (this will be calculated in section 6.1).

If the rate of spam transmission can be slowed below a certain number of emails transmitted per day, some calculations against expected response rate and profitability should allow the

spammers 'break-even' point to be determined. i.e. how much they need to be slowed down in order for their activity to become unprofitable.

2.3 Requirements of an anti-spam solution

2.3.1 Block all spam

Ideally, the system should block all spam. In practice this may not be achievable, as there may always be a small quantity of spam that bypasses whatever system is in place. The number of messages getting through should be perhaps 1 in 1,000.

If a person is sent 200 spam emails a day, then if 1 spam gets through to their in-box per 5 day working week, this equates to 0.1%, which should be acceptable to most email users.

2.3.2 No false positives

The system should not block legitimate email. Not receiving email you are expecting, or important email getting blocked by a spam filter is considered worse by many people than receiving large quantities of spam. (Deepak et al., 2005)

It is important therefore that any anti-spam system ensures that 100% of legitimate email gets through (Damiani et al., 2004).

2.3.3 Transparent to the user

The system should block spam with little or no user interaction. The spam blocking ideally should happen before the email even reaches the client computer. Any filtering that happens on the client computer means the spam has to be downloaded, which results in additional network traffic for the user, and possibly a noticeable delay while the spam is filtered. The system should ideally exist on the email server itself, not on the client computer.

The user should not be aware of any delays in either sending or receiving and filtering email. Users may be tolerant of certain performance issues if they know it is preventing spam

reaching their inbox, but they will not be so accepting if this starts affecting their normal day-to-day operation of their PC to send and receive emails (Bass and Watt, 1997).

2.3.4 Can be implemented universally

Any anti-spam solution should have the ability to be availed of by all email users, regardless of device or method of accessing the internet.

2.3.5 Minimises network spam volume

If spam is filtered at the recipient end, this does not decrease the volume of spam on the internet. If the spam can be stopped at the sender end, or by deterring/preventing the spammer from sending it in the first place, it has the added advantage of minimising spam volumes.

2.4 Summary

This chapter described the SMTP protocol, its weaknesses, and how these weaknesses are exploited by spammers. Techniques that spammers use were also discussed. The main requirements of an anti-spam solution were identified.

The next chapter will look at existing and proposed anti-spam solutions. Each method will be examined, and compared against the requirements criteria from section 2.3

3. Existing Anti-Spam Methods

Most anti-spam methods currently in use are based on some type of filtering at the recipient end. New proposals attempt to take a different approach to stopping spam, by deterring the spammer from sending the email in the first place; either through impacting their profitability (section 2.2) or some other policy/legal/framework approach. This chapter will look at the existing and proposed anti-spam methods in more detail.

Filman (2003) identifies three major categories for anti-spam solutions. Social Evolution, Political Evolution, and Technical Evolution.

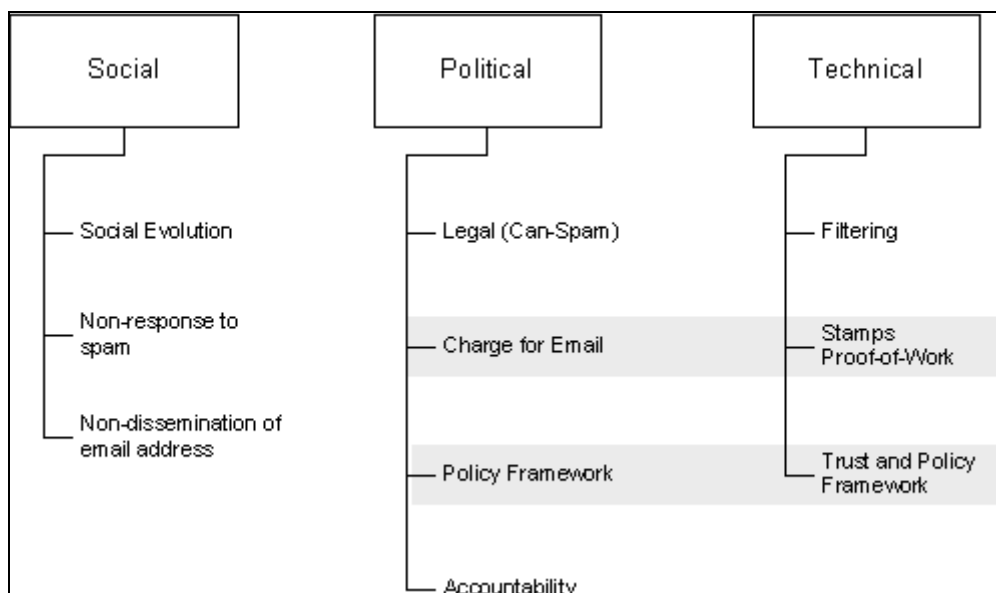


Figure 1: Expanded view of Filman (2003) categories

Figure 1 shows these three categories, with expanded sub-categories branching out.

Social Evolution: we will simply learn not to respond to the offers provided by spam. If there are no takers, spam will die out. However only a small percentage of respondents are all that is needed to make spam profitable for the spammer.

Political Evolution: this depends on regulation, including establishing policy frameworks to make spammers identifiable and accountable for their spam.

Technical Evolution: this encompasses approaches such as email filtering, electronic stamps and blacklists.

The 'technical' category is the one which has attracted the most research and development work in the past few years. This category warrants a more detailed break-down as shown in Figure 2.

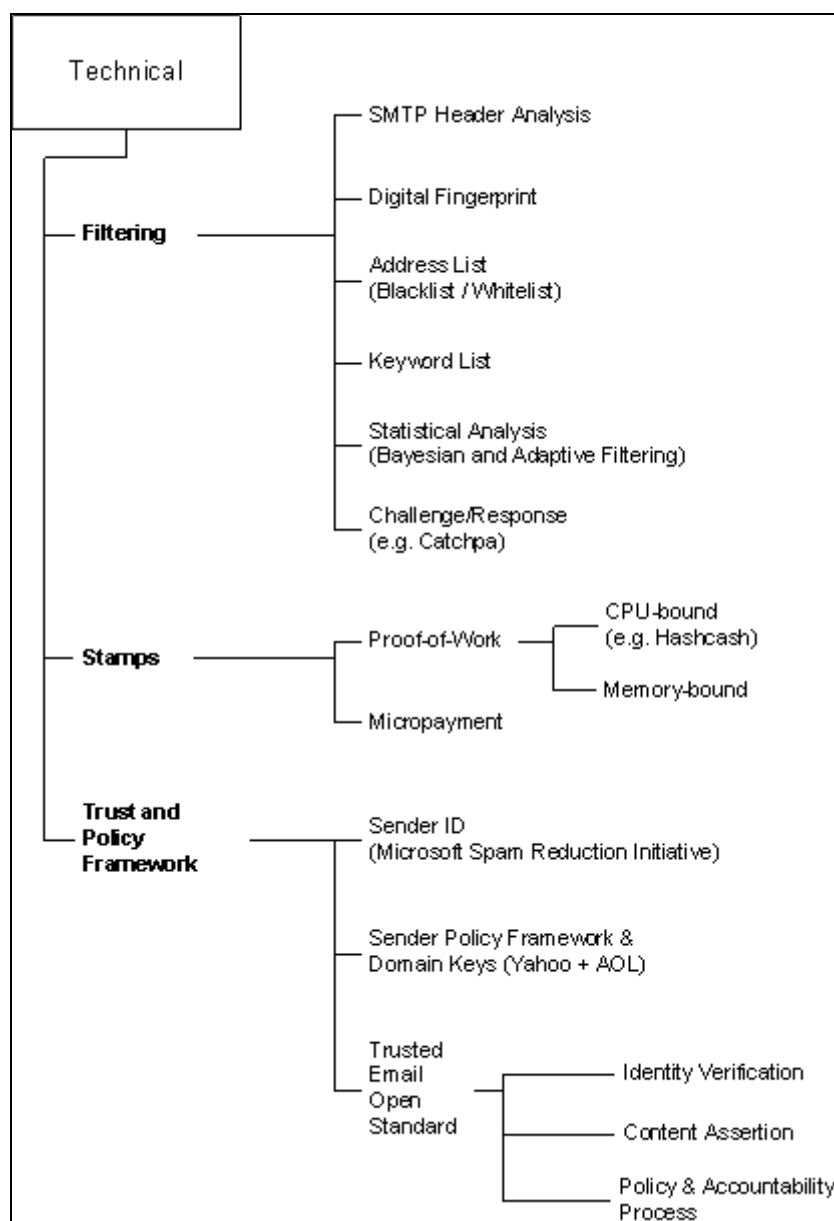


Figure 2: Technical approaches to anti-spam

An important difference between the filtering methods and stamp (proof-of-work) methods is in the reduction of spam volumes on the network. In most filtering methods, the spammer sends massive volumes of email, which then must be filtered at the recipient end. The recipient's hardware (or their ISP's servers) is where the filtering processes happen.

The analysis of existing filter-based approaches to anti-spam show that although they are having an impact on how much spam is reaching in-boxes, they are not impacting the volume of spam being sent. In fact, this continues to increase. With a proof-of-work approach, the email is stamped (i.e. the work is done) at the senders end, before the email is sent. This should result in reduced spam volumes, as the following diagrams illustrate;

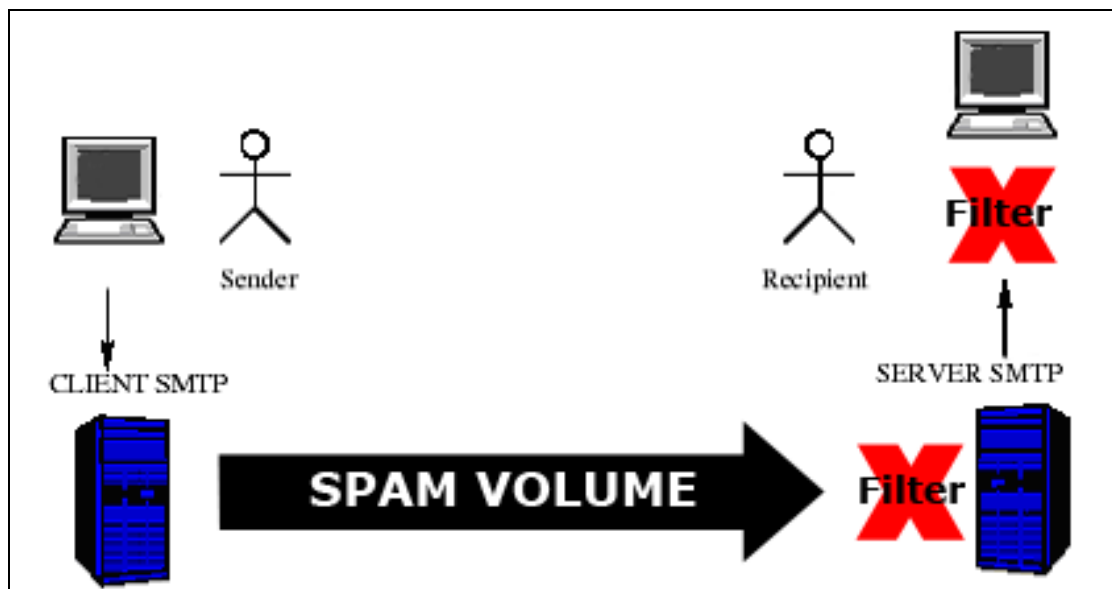


Figure 3: Recipient-based filtering

Figure 3 shows that with existing filtering solutions based at the recipient end, the spammer is not restricted in any way, and will continue to transmit large volumes of spam.

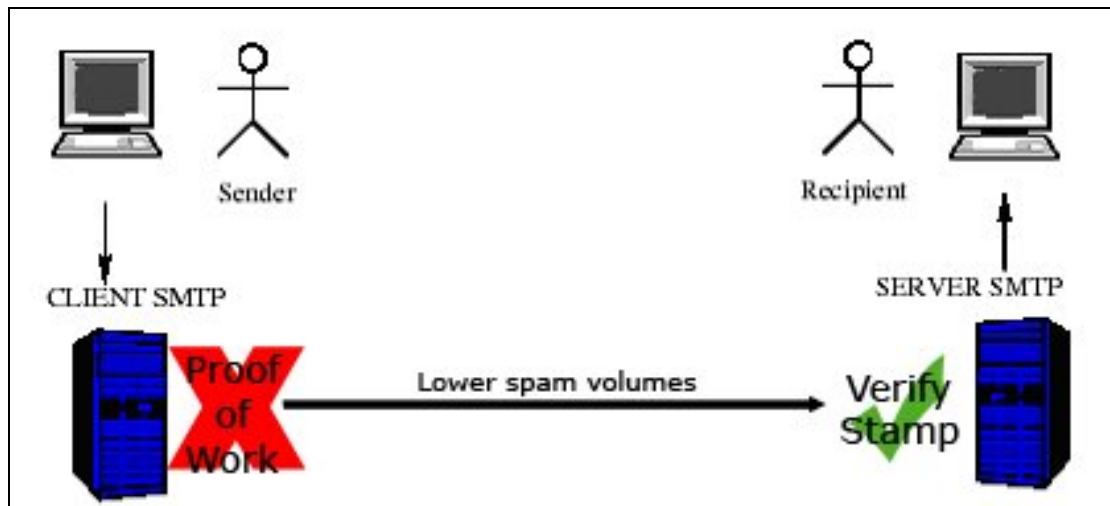


Figure 4: Sender based solution - Proof of work

Proof of work shifts the burden to the sender. Figure 4 shows that by slowing down the rate of sending spam at the senders end, this lowers overall spam volumes, and leaves the recipient with a very simple stamp validation to perform. All the processing happens with the sender.

The technical methods listed in Figure 2 are now explained in more detail (Graham, 2003).

We also examine the weaknesses of each method.

3.1 Social

3.1.1 Social evolution

Spinello (1999) identifies the ethical implications of sending spam. Spam is different from normal junk mail in that it shifts the burden of cost to the recipient. Social evolution depends on the internet community as a whole to assume the sense of shared responsibility (Spinello, 1999) for this communications medium and recognise that sending spam is morally and ethically wrong. Holmes (2005) also highlights the importance of understanding the underlying social problems associated with spam; technology alone will not solve the problem.

Unfortunately, for as long as spammers remain anonymous, and continue to make money out of sending spam, then social evolution can not be depended on to slow down spam.

3.1.2 Non-response to spam

It only requires a tiny number of respondents to spam to make it worthwhile for spammers (Cerf, 2005). If users of email do not respond to spam, and reduce the spammers response rate to 0%, then it will no longer be profitable for the spammer. However, to educate every email user in the world about not responding to spam is a formidable task.

If the spammers know they might still reach a few gullible people and generate some responses, they will just increase their sending volumes and thus clog the network even more.

3.1.3 Non-dissemination of email address

This approach involves educating email users about the importance of protecting their email addresses. Do not sign up to newsletters or allow their email address to appear on websites, which will make it more difficult for the spammer to obtain addresses.

However, spammers may also use 'brute force' attacks (described in section 2.1.3) to circumvent this. No matter how well protected an email address is, all it takes is one lucky 'hit' by a spammers dictionary attack, and the email address is forever on their spam list.

3.2 Political

3.2.1 Legal

Description: In 2003 the US Senate passed into law the 'Can Spam' act as an attempt to cut down spam (Baker, 2003). The main points are;

- Allows businesses to continue to send commercial emails to their customers and prospects, but gives consumers the right to require that a sender of unsolicited commercial email cease sending commercial emails to them. This is known as the "one free shot" approach;

- Requires senders of commercial email to include a functional "opt-out" mechanism in commercial emails. This is not required in "transactional or relationship messages;"
- Requires senders of commercial email messages to provide "clear and conspicuous" identification that the message is an advertisement or solicitation, unless the recipient has given prior "affirmative consent" to receipt of the message;
- Generally allows the sale or rental of email lists, subject to certain restrictions, including that an email address of a recipient who has "opted out" of further commercial emails cannot be transferred to a third party;
- Creates criminal and civil sanctions (up to \$2 million, which can be trebled) for a number of common practices of spammers, including the use of deceptive or misleading origins information, headers, sender identity, transmission information, subject lines, and falsely registered IP addresses.

This is potentially a very effective deterrent against spammers. If a spammer thinks they are going to get taken to court and possibly fined, they might think twice about their activities. However, a loophole in spam laws is usually in the exact definition of 'spam'.

Most spam laws allow the sending of unsolicited email to recipients who have a prior relationship with the sender. This is reasonable, but it must be defined carefully what a prior relationship consists of. There is a type of spammer ("permission-based email marketers") who obtain email addresses by buying them from websites with unethical privacy policies. By calling the site they bought your email address from a "partner" or "affiliate", the spammers can claim that they too have a prior relationship with you, and are therefore exempt from spam laws.

In November 2004, (Levine, 2004) the USA held its first criminal trial concerning spam in Leesburg, Virginia with a conviction of Jeremy Jaynes. The case was brought under Virginia's state anti-spam law, not the weaker Federal Can-Spam act. Virginia's law makes it a crime to send unsolicited bulk mail using forgery, so the Commonwealth had to show first that Jaynes sent lots of unsolicited mail and second that it was sent using forgery. While most of society

welcomes rulings like in Virginia, it must be noted that spammers are often based in different countries, which have different internet laws. Applying a law on an international basis, or prosecuting spammers in other countries is full of difficulties (Goodman and Heckerman, 2004; Holmes, 2005).

3.2.2 Charge for email

This is a system which charges users a fee for each email they send, or for using their email address for specific time periods (per month billing). This would be administered by the users ISP.

Since the cost of sending email is negligible at the moment, most legitimate users would not accept this type of payment system. Legitimate users could feel they were paying the penalty as a result of spammer activity.

3.2.3 Policy framework

This is a set of regulations and recommendations which ensure spammers activities can be monitored and controlled. For example, spammers are allowed to send commercial emails as long as they contain certain information in the subject line which identifies them as commercial. There are also guidelines for removing email addresses from spam lists at the request of the user.

Policy framework forms part of a number of industry proposals discussed in section 3.3.3. It is also used in conjunction with technical measures to ensure email senders can be identified and held accountable.

3.2.4 Accountability

This approach is used in conjunction with a Policy Framework (3.2.3 and 3.3.3) to ensure that senders are held accountable for their actions. It requires a technical method to positively identify the source of email, ensure the sender can be held accountable for any email they send, and a policy regarding how to deal with spammers who are in violation of the standards.

3.3 Technical

3.3.1 Filtering

SMTP Header Analysis

The email header provides some clues which indicate if the received message is a spam. For example, an IP which does not match the domain name indicates that the header may be forged. A forged header is a good indication that an email is spam. Other clues in the header might include; use of distribution list, and the presence of error lines (Pelletier et al. 2004).

Digital Fingerprint

This method compares incoming emails against a signature database of known spam emails. The system calculates a checksum signature of an incoming spam message, and adds it to the database. Any incoming emails are then compared to this database to see if they are spam.

This is an accurate way of matching spam. It can achieve very low 'false positives' since only definite spam is matched based on the hash signature of its contents. But in order to be detected as spam, the message will have to exist in the database of pre-sent spam messages first. If the spam is new, it may not exist in the database yet, and therefore won't get blocked (Graham, 2003).

Address List

An IP blacklist is a list of the IP addresses of suspected spammers' mail servers, or relay servers (unsecured servers which allow spammers to forward email). These lists are maintained by volunteer groups and anti-spam organisations. ISPs can then subscribe to these lists and refuse to accept email from any listed IP addresses (Jung and Sit, 2004).

This is a very precise method of blocking potential spam. Unfortunately, these blacklists can never hope to list every single IP address that spammers use. Also, they often end up listing legitimate IP addresses, or blacklisting an entire domain (1,000 ordinary users could get

blacklisted for the actions of one spammer). The source IP is often spoofed by the spammer, which means they can bypass the blacklist.

Keyword List

Until Bayesian filtering was introduced, this was probably the most flexible method to help identify spam. Rule-based filters look for patterns that indicate spam: specific words and phrases, lots of uppercase and exclamation points, malformed headers, dates in the future or the past, etc. This is how nearly all spam filters worked until 2002 (Graham, 2003).

Again, this method is very easy for the spammer to bypass. The clever spammer even runs their spam through keyword-based filters before sending it to ensure it doesn't trigger the spam alert. Many spammers have now learnt how to make their email not look like spam to the filters, and use techniques to ensure it is at least opened by the recipient when it reaches their in-box. (e.g. making the 'From' look like a real person's name, and making the 'Subject' something like 'Hi' or 'Long time no see'). This method is often used in conjunction with Bayesian filtering.

Statistical Analysis

This method uses a statistical analysis technique (typically a Bayesian approach) to analyse the words contained in each received email. It uses predefined lookup tables to determine the probability that an email is spam. For example, the word 'Viagra' would have a high weighting, since it commonly appears in spam.

Bayesian filtering is a relatively new approach to spam prevention. This approach seems to trigger less false positives than other types of filtering, as it is self-training based on spam it receives. In some cases the user can update the filter if it misclassifies an email, thus improving the detection accuracy (Graham, 2003).

Spammers often bypass these types of filters by making the spam look less spammy; using less spam-related words (often making the email read like a personal email from a friend), or

putting random non-spam words at the end of the message to cause the Bayesian calculation to misclassify them as non-spam. Another technique spammers use is to misspell spam-related words. For example, Figure 5 demonstrates the spammer's technique of misspelling spam-related words, and putting blocks of non-spam words in the email to trick the filters:

```
@247!-Online Doctorz!  
up to 70% of the best pain killers out!  
_Soma_vioxx_viagraaa_Floriceet-Phentremine  
and other popular meds.!valium_XXanax_Cialis%  
  
--  
slab usc heel strophe thalia koinonia checkpoint slump florida  
ileum buttress borroughs beatnik rent aaas allocate laxative  
ask indigo switzerland maddox armload cougar bella embed buggy  
cereus equilibria
```

Figure 5: Example of spammers trying to fool Bayesian filter

Challenge / Response

When you get an email from someone you haven't had email from before, a challenge-response filter sends an email back to them, telling them they must go to a web page and fill out a form before the email is delivered. Once the sender has verified themselves, they get added to a senders whitelist to ensure any future emails get through without requiring verification (Gburzynski and Maitan, 2004).

This method ensures you only receive email from people who really want to correspond with you. The chances are that a spammer is not going to spend the time filling out the web form. However, this can be quite inconvenient for the sender, as they have to remember to fill in the form before the recipient gets the email. In some cases the sender might not bother, and this method will always result in email being delayed. This approach has not been widely adopted.

3.3.2 Stamps

Proof of work - slowing down spammers

Spam has low response rates (on the order of 15 per million) but spammers make up for it with high volumes, sending millions of emails per day (Schwartz, 1998). If you could slow down the rate at which they send email, you could put them out of business. One way to do this would be to make any computer used to send email perform an easily verifiable time-consuming computation before you would accept that email.

Whatever these computations are, they should be within acceptable, controllable levels of complexity, because legitimate corporate email servers have to be able to send high volumes of email. And corporate email servers would be running on standard hardware. Many computations can be made hundreds or thousands of times faster by custom hardware.

This is the first approach that directly attacks the spammers' profitability model. Instead of trying to block or filter spam that has already been sent it makes it more costly for the spammer to send each message. It also helps reduce false positives caused by other types of spam filtering. It is likely that if an email has a 'proof of work' stamp, then it has been sent by a genuine sender, and can bypass the standard Bayesian filters.

For this idea to work, you'd need to figure out a kind of computation that couldn't easily be speeded up by custom hardware.

Even with a suitable computation, this idea would require new email protocols. Any new protocol has a problem: no one is inclined to adopt it till everyone else does. As a result, it is practically impossible to get a new protocol adopted for anything. How are you going to get system administrators who don't even bother to install patches for years-old security holes to switch to a new email protocol?

Micropayment

A technical implementation of the 'charge for email' method (section 3.2.2). Micropayment schemes, small transfers of cash to accompany transactions, have been promoted by companies like Cybercash. But the user community hasn't been receptive to paying for what has been previously freely available (Pfleeger and Bloom, 2005).

A sender payment scheme would require international treaties, which can be difficult to enforce. Also free email services like Hotmail might not be willing to absorb micropayment costs. The new costs might be more than the current costs of dealing with spam.

3.3.3 Trust and policy framework

The approaches outlined above take a 'technology only' approach to the spam problem. What is required is an approach which encompasses technology with a policy-based solution. Any approach also needs support from the major ISPs (such as Yahoo, Hotmail, and AOL), and should be aligned with existing anti-spam laws. There are a number of industry proposals, most of which encompass hybrid multi-layer spam blocking/filtering technologies with trust-based systems and often cover the areas of policy as well.

Sender ID

Microsoft's proposal covers a number of areas, but the main focus is on a system called Caller ID (Microsoft, 2004). Microsoft and the Internet Engineering Task Force have proposed changes to the way SMTP verifies the sender of an email by looking up the source via DNS. This involves modifications of DNS standards (a central part of the internet itself).

Caller ID allows Internet domain owners to publish the IP (Internet Protocol) address of their outgoing email servers in an XML (Extensible Markup Language) format email "policy" in the DNS (Domain Name System) record for their domain. Email servers can query the DNS record and match the source IP address of incoming email messages to the address of the

approved sending servers. This results in email being verifiable as coming from who it says it's from.

Sender Policy Framework & Domain Keys

These proposals (supported by Yahoo and AOL) are essentially the same as the Caller ID proposals. They use a DNS challenge/response mechanism to allow look-up and verification of the sender of the email to ensure they are who they claim to be. Weaknesses are already becoming apparent in this system.

It was implemented by Yahoo in the first quarter of 2004, but it has resulted in no reduction in spam volumes to Yahoo email addresses. The spammers are validating themselves as legitimate email senders to ensure their emails get through, and legitimate emailers are sending from servers who have not actually implemented the Domain Keys technology.

Although the spammers no longer have anonymity on their side, there is no solid legal framework in place, or even a way to prevent them from continuing to send spam if they are using Domain Keys. Systems like Domain Keys and Caller ID only offer a part of the anti-spam solution. They ensure the email sender can be identified. But they do not offer a way to stop a spammer sending spam (Geer, 2004).

Trusted Email Open Standard (TEOS)

In May 2003, the ePrivacy Group announced the Trusted Email Open Standard (TEOS) to fight spam, spoofing, and email fraud (Schiavone et al, 2003). TEOS is a staged approach towards a trusted email system built upon and extending the SMTP protocol. TEOS takes a two-track approach comprising an identify verification system, content assertions (flags in the subject line which identify the type of content) in conjunction with a policy-based trust and accountability process.

TEOS creates a framework of trusted identity for email senders based on secure, fast, lightweight signatures in email headers, optimized with DNS-based systems for flexibility and

ease of implementation. TEOS also provides a common-language framework for making trusted assertions about the content of each individual message. ISPs and email recipients can rely on these assertions to manage their email.

3.4 Hybrid approach

This approach consists of using an intelligent combination of technologies to provide an effective approach by using the strengths of one method to counter the weaknesses of another. There are a quartet of complementary technologies to consider which are; proof of work, signatures with whitelist, domain certificates and content filters. This cocktail of anti-spam measures is known as the 'hybrid' system.

A combination of anti-spam solutions which are inadequate when used individually, but when used together can compensate for the inadequacies inherent in each part of the system. A method of combining three of these technologies might be as in Figure 6

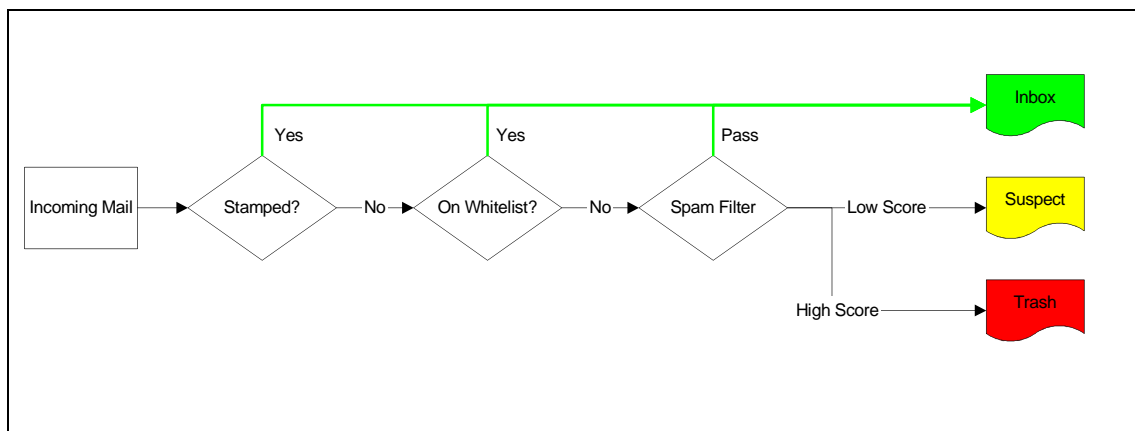


Figure 6: Hybrid approach to anti-spam

First, the incoming mail is checked for a valid Hashcash stamp, if it passes this step it goes straight to the inbox. If not, it goes to the next stage to see if the sender is listed on the recipients list of 'allowed senders' (i.e. a conversation and level of trust has previously been established between the two parties) – If the email is from a whitelisted sender, it can be directed to the users inbox.

Finally, a standard content filter is applied. If an email does not have a stamp and is not from a whitelisted sender, then there is a possibility it might be spam. So, a Bayesian filter check is applied to the contents to check it for 'spaminess', if it scores highly on this check (i.e. it is undoubtedly spam) it is routed to the trash folder. Otherwise it is routed to the 'suspect' folder for review by the recipient. Hopefully, most spam will be filtered out in step one and two, to minimise the chance of false positives in the final filtering step (the filtering step is the part most likely to introduce false positives in this setup).

Camram (Johansson and Dawson, 2003) is an example of a hybrid system, although this is still in the beta/development phase. Camram blends anti-spam solutions to compensate for the weaknesses in each. Hashcash forms the foundation of the Camram system.

An all-encompassing anti-spam system needs to address a number of additional areas however, and ideally consists of a multi-layered approach:

- Policies - Mass-mailers agree to conform to certain standards
- Accountability – If spammers don't conform, then some sort of internet 'sanctions' should be applied
- Traceability – Email should be verifiable as coming from who it says it is from, to ensure senders can be traced
- Legal – For repeat or spammers who refuse to comply with any of the policies, then legal discourse can be used as the fall-back position
- Technical – Proof-of-work stamps to minimise spam volumes. Traditional Bayesian filters to trap anything that gets through (this is the point where false positives could occur – so whether Bayesian filters are still required in this model is debatable).

3.5 Comparison of anti-spam solutions

Table 2 summarises and compares the technical anti-spam solutions discussed in this chapter. The five main requirements (from section 2.3) for an anti-spam system are shown,

with each method evaluated against each of the requirements. It can be seen that no one method fits all the requirements perfectly.

Method	Type	Spam Blocking Efficiency	Causes False Positives	User Transparency Rating	Universal Solution	Minimise Network Volumes
SMTP Header Analysis	Filtering	Poor	Yes	Good	Yes	No
Digital Fingerprint	Filtering	Poor	Yes	Good	Yes	No
Address List	Filtering	Good	Yes	Good	Yes	No
Keyword List	Filtering	Poor	Yes	Good	Average	No
Statistical Analysis	Filtering	Good	Yes	Good	Average	No
Challenge / Response	Filtering	Average	No	Poor	Average	No
Proof of Work	Stamp	Good	No	Unknown	Unknown	Yes
Micropayment	Stamp	Good	No	Good	Yes	Yes

Table 2: Comparison of technical anti-spam solutions

In the case of ‘Proof-of-work’, specifically Hashcash, it is not known if it can be implemented universally in a manner which is transparent to the user. It is suspected that due to the disparity of performance across email devices, Hashcash may not be a universal stand-alone solution.

Hashcash will now be examined and tested in more detail to determine if it is a feasible stand-alone solution to spam (i.e. to find out if it meets the ‘transparent’ and ‘universal’ requirements).

3.6 Summary

This chapter presented an analysis of the existing body of work in anti-spam research. Deployed methods were shown, together with their weaknesses and the latest proposed solutions.

Hashcash offers a potential solution to spam. But there are some unknowns about its suitability against the requirements (Can it be used universally? Does it impact legitimate email senders unfairly?) These questions will be answered as part of the primary experiment.

The mode of operation of Hashcash is to impact spammer profitability by slowing down their rate of email transmission. The next chapter will show how this is achieved with a more detailed description of how Hashcash operates.

4. Hashcash in Detail

A novel proposal to anti-spam is the proof-of-work method called 'Hashcash'. This presents an interesting line of investigation as it is still in development and has not been deployed.

There also exist some doubts over its feasibility due mainly to the problem of CPU-bound functions not performing equally across platforms.

4.1 What is Hashcash?

Hashcash (Dwork and Naor, 1992), (Back, 1997, 2002) is a method of adding a textual stamp to the header of an email to prove you have expended a certain amount of CPU time calculating the stamp prior to sending the email. In other words, as you have taken a certain amount of time to generate the stamp and send the email, it is unlikely that you are a spammer. The stamp can be validated quickly at the recipient end. If the stamp is valid, then it is possible to flag the message as 'not spam' and direct it straight to the recipient's inbox.

Hashcash inserts a X-Hashcash: header into the email headers section of the email the user sends. Figure 7 shows an example of an email with a Hashcash stamp in the header.

```
From: Someone <test@test.invalid>
To: John Honan <jhonan@silveronion.com>
Subject: test Hashcash
Date: Thu, 24 Jun 2004 11:59:59 +0000
X-Hashcash: 0:040624: jhonan@silveronion.com:1fea2abb184ebf0a

blah blah

-Someone
```

Figure 7: Hashcash stamp in email header

To Hashcash-stamp an outgoing email, the recipient email address is supplied to the Hashcash minter program together with how many 'bits' the stamp should be (larger bit stamps take longer to calculate). The minter program, after a certain amount of calculation time, will return a valid Hashcash stamp in the format shown in Figure 7. The stamp is then

attached to the header of the email, and the email is sent as usual. This process should happen as a background job so the sender is not even aware a stamp is being generated and attached to the outgoing email.

On the recipient end, the server reads the Hashcash stamp and performs the following verification;

- 1) Is the stamp is the required number of bits?
- 2) Is the address in the stamp the person the email is intended for?
- 3) Is the date in the stamp today's date?

If the above three tests are passed, then the stamp is valid, and the email is directed to the recipients in-box. The important point is that it takes a lot longer to calculate a stamp than to verify it. It could take a number of minutes for the sender's PC to calculate a valid stamp, but it only takes a few milliseconds for the recipient's email server to check the stamp is valid. Therefore the 'CPU cost' of calculating a stamp is borne at the sender's end.

4.2 How does Hashcash work?

Hashcash is based on the SHA-1 hashing algorithm. SHA-1 generates a unique 160 bit number from any input text. It is easy to validate the input text generates the output hash. This method is often used to check that executable files or documents which are downloaded from the internet are intact and not tampered with when they reach their destination, by validating their SHA-1 hash signature is correct at the receiving end. Even if one character of the input text changes, this results in a drastically different SHA-1 hash.

Hashcash makes use of the fact that it is highly unlikely that any two input strings will generate the same output hash signature (160 bit hashes give 2^{160} possible hash combinations)

4.3 How Hashcash computes hash-collisions

Basically on the recipient's email address and the date. What Hashcash actually does is look for bit collisions on strings such as:

```
0:040624:jhonan@silveronion.com:1fea2abb184ebf0a
```

against an all '0' string. The stamp contains a date (040624= Jun 24th 2004), and an email address (jhonan@silveronion.com). The first field (the 0:) is the stamp version number. The last field, a string of random letters, is some random text used to in the collision detection part of the stamp generation process. Lots of different random strings must be tried to find the required bit-collision, approximately 2^{16} attempts for a 16 bit collision.

One of the powerful features of Hashcash is that it is defined using SHA1, a commonly used and effective hashing algorithm. The above stamp hashed gives:

```
echo -n 0:040624:jhonan@silveronion.com:1fea2abb184ebf0a | sha1
00000000c70db7389f241b8f441fcf068aead3f0
```

It can be seen that the first 8 hex digits are 0. Each hex digit is 4 bits. This means the above stamp has a collision on $(8 \times 4 \text{ bits}) = 32 \text{ bits}$. So the above stamp is a 32-bit collision. This is a big collision which would take a high-end 2Ghz PC many hours to compute. But for normal email it is expected to use stamps in the range of 20 - 25 bits (which will take under a minute to calculate on most hardware).

The self-contained nature of a Hashcash stamp means that once the stamp is generated, then any tampering to the stamp will result in the collision bit count changing, and thus rendering the stamp useless. This is very important, as it means a spammer cannot just generate one stamp and slot multiple email addresses into it. Also, they cannot generate hundreds of stamps in advance for a single email address, as the date forms part of the stamp.

For example, assuming a spammer legitimately generated the following 20-bit stamp;

```
0:040624:jhonan@silveronion.com:r7rk+d1/34
```

This stamp can only be attached to an email being sent to 'jhonan@silveronion.com' on Jun 24th 2004. Any other recipient will reject this stamp (as it doesn't contain their email address), and if it is sent on any other date to jhonan@silveronion.com, then it will be rejected as the date is incorrect.

Now, the spammer decides they want to use the same stamp, but doesn't want to spend the CPU time generating it. So why not just change the stamp slightly?

```
0:040624:someone@yahoo.com:r7rk+d1/34
```

The email address has been modified to read 'someone@yahoo.com'. However, if this stamp is validated with SHA1, it is no longer a 20-bit stamp;

```
SHA1 (0:040624:someone@yahoo.com:r7rk+d1/34)
= 155DD9CAD53322C7FA119A3AFB6851501D06A7E8
= 3 bit collision (unacceptable)
```

This is because changing the stamp itself actually results in a different SHA1 hash output. Once the stamp has been generated, it cannot be tampered with.

4.4 Partial hash-collisions explained

Computing a full hash-collision (finding two text inputs that generate the same 160-bit hash output) is computationally infeasible - there isn't enough computing power on the planet to create one in the next 100 years. Hashcash instead works on the basis of partial-collisions. A full-collision would be that all bits of SHA-1(x) must match SHA-1(y); a k-bit partial collision would be where only the k most-significant bits match. The only way to attempt to calculate

these partial collisions is by using a brute force method; this is where the CPU-time expenditure (proof-of-work) takes place.

Taking the hash we need to find a match on, loop through a process of appending randomly generated text until checking the output hash gives the required number of collision bits as shown in Figure 8.

```
Take input text to generate stamp for "0:040624:jhonan@silveronion.com:"
Start loop
    Add random string to text "0:040624:jhonan@silveronion.com:r7rfjs0fsa"
    Check for collision (Does new string cause X bits collision?)
No collision found, continue loop
Yes ,collision found! output final string
    "0:040624:jhonan@silveronion.com:r7rk+dl/34"
```

Figure 8: Pseudo code for partial hash collision function

The loop can take millions of attempts before it finds a matching x-bit collision. And it is this brute force iterative approach that takes so much computational effort to find a collision and therefore generate a valid stamp. There is no other mathematical way of quickly finding hash collisions; this brute-force looping method is the only way. If one could see the output of this loop, it would look something like Figure 9.

Attempt 1

Try "0:040624: jhonan@silveronion.com: r7rfs0fsa" (iterate through characters)

SHA1(0:040624: jhonan@silveronion.com: r7rfs0fsa) =

155DD9CAD53322C7FA119A3AFB6851501D06A7E8 : 3 bit collision (unacceptable)

Attempt 2

Try "0:040624: jhonan@silveronion.com: r7rfs0fsb"

SHA1(0:040624: jhonan@silveronion.com: r7rfs0fsb) =

036048B1A1AA5FF78EE35965EB37C45A7E9A626C : 6 bit collision (unacceptable)

Attempt 3

Try "0:040624: jhonan@silveronion.com: r7rfs0fsc"

SHA1(0:040624: jhonan@silveronion.com: r7rfs0fsc) =

07E18141DACC9A9F2F39419BA36823CB1F420271 : 5 bit collision (unacceptable)

***Continues looping and testing string hash collisions until a 20-bit collision is found,
1.6 seconds later...***

Attempt 1,048,000

Try "0:040624: jhonan@silveronion.com: r7rk+dl/32"

SHA1(0:040624: jhonan@silveronion.com: r7rk+dl/32) =

00016EACE3E12A186394D5AA2FBC8FA0177AAFC6 : 12 bit collision (unacceptable)

Attempt 1,048,001

Try "0:040624: jhonan@silveronion.com: r7rk+dl/33"

SHA1(0:040624: jhonan@silveronion.com: r7rk+dl/33) =

0A581544F62B8C7EACCA37A0AAE2D495F78B4587 : 4 bit collision (unacceptable)

Attempt 1,048,002

Try "0:040624: jhonan@silveronion.com: r7rk+dl/34"

SHA1(0:040624: jhonan@silveronion.com: r7rk+dl/34) =

00000B6D7BDF8F3DF424A24F1B2BC4BBFCBF3EDE : 20 bit collision (success!)

20-bit Collision Found on string "0:040624: jhonan@silveronion.com: r7rk+dl/34"

Figure 9: Brute-force looping method

To take a 20 most significant bits collision as an example (which is a practical target). A 2GHz P4 can compute one in about 1.6 seconds, but this equates to about 1,048,576 (2^{20}) iterations, on average, through the loop. Because Hashcash uses a brute-force algorithm to find hash collisions, there is no guarantee as to exactly how many iterations it will take before finding a collision. It might be 'lucky' and start the loop close to a successful collision, and

thus calculate the collision in a matter of milliseconds. Or, the loop might require a large number of iterations, and actually take longer than expected to achieve the required collision.

On average, it will take 2^n attempts to find a collision (RSA Laboratories, 2000), where n is the number of collision bits required. If we know how long each iteration takes (by simply timing the collision detection loop), then we can work out on average how long it will take to calculate the required stamp on that particular hardware.

The goal of introducing a proof-of-work system like Hashcash is to reduce the amount of spam the average person receives to below some fraction of their legitimate email.

Independent testing of the best commercial spam filtering solutions shows that they currently achieve an spam-to-legitimate ratio of 0.06 (Snyder, 2003)

There are some¹ who say that Hashcash will encourage spammers to utilise large collections of zombie computers, which run malware to send spam unbeknownst to the user. These large networks have a large total computational power, which can be used to generate legitimate hashes for the spam emails they send. It can be argued that the extra CPU usage will often be noticed by the owners of the machines, who will be more likely to fix them. (Laurie and Clayton, 2004) argue that while Hashcash for email is attractive, nonetheless spammer profit margins per sale mean that they will be able to afford the PCs to do the proof of work required. This claim will be tested by the spammer profitability analysis (chapter 6)

We believe however that their economic analysis of spammer charge per email and hardware costs are weighed in favour of the spammer (i.e. lower than we would state them) and they themselves mention elsewhere in the paper that spam response rates to date are badly documented.

¹ <http://en.wikipedia.org/wiki/Hashcash>

4.5 Summary

In this chapter we described Hashcash proof-of-work in detail. In chapter 5, the methods for the main experiment, measuring the performance and feasibility of Hashcash across multiple platforms, will be explained.

5. Evaluating the Hashcash Approach

The aim of the experiment is to determine if the Hashcash proof of work approach meets all the requirements of an anti-spam solution. In Table 2, we were able to match some of the requirements, but there is still doubt over the suitability of Hashcash as a universal solution.

To restate the research question “Is Hashcash proof-of-work a single technique which meets all the requirements of an anti-spam solution?”. We can deduce by simple analysis of the way Hashcash operates that it **does** meet the following requirements;

(2.3.1) Block all spam – Either the email has a valid Hashcash stamp, or it doesn't. We assume it will not be profitable for a spammer to stamp every outgoing email (analysis in chapter 6); so either the spam will not be sent in the first place, or Hashcash will reject it at the recipient end, therefore blocking all spam.

(2.3.2) No false positives –The email either has a stamp or it doesn't. If it is stamped it will reach the users in-box, otherwise it is rejected. Assuming Hashcash is implemented universally (2.3.4)

(2.3.5) Minimises network spam volume – Hashcash works to deter spammers from sending spam in the first place by impacting their profitability, so there will be less spam transmitted over the internet, therefore reducing volumes. Refer to Figure 3 and Figure 4 for graphical representation of this process.

However, due to the potential problem of performance disparity, the following requirements may not be met;

(2.3.3) Transparent to the user – It might take too long for the sender to calculate a stamp, especially if they have a slow machine. This depends on what the minimum stamp value needs to be set to, which is dependant on spammer breakeven analysis (chapter 6)

(2.3.4) Can be implemented universally – If the problem of performance disparity is proven, then Hashcash can not be implemented universally. If there are only some users using it, then the spammers will stay in business, and it will not be a viable standalone solution.

The primary data gathering will address requirements 2.3.3 and 2.3.4. The experiment will be approached in 2 parts;

- 1) Calculate spammer profitability and breakeven point to find out how long (in seconds) the Hashcash stamp needs to be to deter spammers
- 2) Take time measurements of stamp calculation times across for different bit sizes across different devices (processor speeds)

We can then combine and analyse the results of both parts of the experiment and draw conclusions for the ‘transparent’ and ‘universal’ requirements. This chapter narrows down and defines the method and research instruments required for primary data collection.

5.1 Choosing a method

The most common method used in measuring the performance of a filter-based anti-spam system is using a corpus of spam messages, passing them through the system, and measuring success rates and false positive reading (Lai and Tsai, 2004; Kim et al., 2004; Stuart et al., 2003).

This method is particularly useful for evaluating Bayesian and adaptive filtering approaches (discussed in section 3.3.1) and spam testing methodologies are emerging, such as measurements of Spam Precision, Legitimate Precision and Spam Recall (Stuart et al., 2003). However, Hashcash takes a different approach to anti-spam, so these testing methods cannot be used.

The central concept of Hashcash is that it takes a certain amount of time to generate a stamp, which can be attached to any email and then quickly validated at the recipient end. There is no ‘filtering’ as such at the recipient end, just a simple check that the stamp is valid. Every

email with a legitimate stamp is accepted, and every email without a stamp, or with an invalid stamp, is rejected.

The objective of this experiment is to compare the time taken to generate stamps across platforms, in order to prove that Hashcash is not feasible due to the wide disparity of stamp generation times from one system to the next.

There are a number of possible approaches to performance testing the Hashcash system. For this analysis, we need to measure the following;

- Calculate (based on spammer profitability) the minimum stamp size required in bits to make Hashcash an effective solution
- Measure how long it takes to calculate various size Hashcash stamps on different powered CPUs
 - Test from the 'slowest' CPU to the 'fastest' available for testing, and some in-between
 - Calculate average time (seconds) to generate an n-bit stamp

For the analysis of the data, we can then compare the minimum stamp size required against the slower machines and reach some conclusions about the feasibility of a universal Hashcash-based solution.

5.2 Potential methods

Part 1 of the data gathering concentrates on calculating spammer breakeven points. This will be performed by obtaining secondary data regarding spammer response rates and revenue, and then some Excel-based goal-seek sheets to determine breakeven points (i.e. when profit = 0).

Part 2 is more complex. Here we have to measure Hashcash minting performance across multiple platforms. The following are potential approaches;

5.2.1 Field testing

This would involve installing an end-to-end Hashcash based system, from sending client to receiving client via a SMTP sending server and receiving server (over the internet). This would most closely represent the actual use of Hashcash. It would also allow load testing of the client machines. Disadvantages of this approach are the variability of field test conditions, for example different processes running on different Hashcash client machines.

The only data we are really interested in measuring in is the generation of the stamp at the sender end. End-to-end field testing, including the recipient machine, will add no primary data of value to the experiment. Too many variables (such as network load) would be introduced into the measurements, thus affecting the reliability of the data.

5.2.2 Analytical approach

This method would involve a more detailed analysis of the Hashcash algorithms and underlying code in conjunction with required processor-cycles per step. This would result in very exact measurements, and the ability to extrapolate measurements across different processor architectures. However, this method would require an in-depth technical knowledge of processor architecture and machine-code execution cycles. The same method may not apply across all architectures (e.g. parallel processing architectures)

5.2.3 Laboratory simulation

This involves running Hashcash in standalone mode on individual PCs, and timing the minting performance. There is no need to recreate a live environment. Measuring the performance of Hashcash under laboratory conditions would allow for the most accurate set of results.

However, this method may not accurately represent how a server would respond under load

5.3 Chosen method

Spam corpus –Although this is the standard way of measuring the detection rates of anti-spam filtering systems, for our purposes we are really interested in measuring the stamp

minting performance of Hashcash. Running a spam corpus through a Hashcash system would actually generate 100% spam detection with 0 false positives. **Discarded.**

Field testing – It is unfeasible to setup an end-to-end SMTP client-server system running over the internet, as I do not have access to the appropriate hardware and technical expertise. This would require 2 correctly configured SMTP servers (sender and receiver), with Hashcash decoding running on the receiver. **Discarded**

Analytical approach – The technical steps and detail required to obtain precise understanding of machine code cycles across multiple processor architectures would extend the scope and timescales of this project considerably. **Discarded**

Laboratory simulation – This is the most straightforward way to obtain minting performance across multiple processors. It involves the least amount of technical configuration and will generate accurate results. **Chosen method.**

Various stamp calculation times are to be tested across different types of CPU and operating system in a standalone (i.e. not internet-connected or part of an SMTP system). It is assumed that a spammer will be willing to invest in a stack of 'fast CPUs' running optimised minting code in order to generate stamps as quickly as possible, and comparing this to the 'poor' Pentium 2 user who just wants to send email in the most efficient way.

Since a collision detection of the required number of bits has an element of 'luck', there is no exact measurement of how long it takes to calculate a stamp with a certain bit size (described in 4.4). What we do know is that the time taken averages out over a number of samples.

The most accurate way of determining the average time per x-bit stamp is to measure the number of collision detections (hashes) per second, and then using the following formula;

Average time taken to calculate X-bit stamp (seconds) = Collisions per second / 2^X

(RSA Laboratories, 2000)

5.4 Research instruments

In order to get a like-for-like comparison, a version of the Hashcash minter which can run from a DOS command line was used. Hashcash version 0.28² was used to run the experiments and obtain the primary data. The '-s' command line flag tells Hashcash to calculate and output the number of collision tests per second, as seen in Figure 10;

```
C:\>hashcash -s
speed: 1277100 collision tests per second

C:\>hashcash -s
speed: 1262800 collision tests per second

C:\>hashcash -s
speed: 1256400 collision tests per second

C:\>hashcash -s
speed: 1258600 collision tests per second

C:\>hashcash -s
speed: 1266700 collision tests per second

C:\>hashcash -s
speed: 1272800 collision tests per second
```

Figure 10: Obtaining minting performance data

The full list of machines used for performance testing is shown in Table 3. The slowest specification machine that the test was run on was an Intel 486 25Mhz, the highest specification machine was the Apple Xserve G5. The data obtained from these machines was the 'number of collision tests per second' they can compute. A stamp size parameter range of between 20-bits and 31-bits was selected.

² <http://www.Hashcash.org/binaries/win32/>

In the case of the slower processors, actually running a full stamp generation for a 30-bit stamp was unfeasible as it could take over 2.5 days to calculate. Therefore we calculated how long one hash takes to calculate, and then used the 2^n formula (section 5.3) to obtain an average minting time.

Processor Model	Speed
Intel 486	25MHz
Intel Pentium	200MHz
Intel Pentium 2	400MHz
Intel Pentium 3	750MHz
Intel Pentium 4	2GHz
Intel Pentium 4	3GHz
Apple G-5 X-Serve Dual	Dual 2.3GHz

Table 3: Processors used for testing

5.5 Summary

In this chapter we determined the form of the primary research method, and identified the research instruments. In chapter 6 we will progress to gathering the primary data and performing the analysis.

6. Results and Analysis

6.1 Part One: Spammer profitability and breakeven point

Proof-of-work is based on the concept that if it takes x seconds for a spammer to stamp and send an email, then if x is large enough it should be possible to impact the spammers profitability enough to deter them from sending spam in the first place.

For the first part of the analysis, we need to work out what this breakeven point is. In other words, how long it should take to calculate a Hashcash stamp to make sending spam unprofitable. There are only a certain number of seconds in a day, which means the spammer is limited in the maximum amount of stamped email they can send per CPU per day. It should be possible to combine both these ideas together.

We need to make some assumptions about average costs to the spammer, and establish some base variables. These base variables are required throughout the calculations which follow.

- a) There are $(3600*24)*365.25 = 31,557,600$ seconds in a year
- b) We will assume the average cost of acquiring, powering, housing, maintaining, and administering one CPU for one year is approximately €200
- c) Therefore, we can divide b) by a) to determine the cost of running a CPU for one second

Base variables		
a)	Seconds in a year	31,557,600
b)	CPU cost per year	€200.00
c)	CPU cost per second	€0.000006338

Normally, the time taken to transmit one email (without stamping it) is in the fractions of a second. Fast spam programs can send hundreds of emails a second.

- d) To continue with the calculation, we will assume the spammer has to stamp every mail, and that each stamp takes 15 seconds to calculate (this number will change later on, as we will use goal-seek to find what the minimum value needs to be, but lets say 15 seconds for the moment to demonstrate the formula as a first pass)
- e) The CPU cost to send one message is then (cost per second / seconds per message)

Cost of sending one message				
d)	Seconds per message	15	This is the value we need to find using goal-seek	
e)	CPU cost per message	€0.00009506		

Spammer revenue and response rate figures are difficult to obtain. We know that spammers operate different business models with different success rates and profits. Mangalindan (2002) gave real-world examples of spam response rates of 0.013% and 0.0023% (1 response per 7,692 emails, and 1 response per 43,478 emails). These were obtained through interviews with only one spammer so will have to be viewed as 'typical' response rates.

Yerazunis (2003) quotes a response rate of 0.01% (1 response per 10,000 emails). Which appears to uphold Mangalindan's claimed response rates. Overall this secondary data is not very accurate or reliable as it was derived from limited interviews or rough estimates around response rates. Since it is very difficult to obtain exact spammer response rates we will have to use these approximations.

Spammers deal with their clients using a variety of models; fixed price per mailing to fixed price per email sent, to pay-per-response, where the spammer gets paid once a recipient clicks on a link to a webpage or actually makes a purchase (Boutin, 2004). We will use the commission-based and click-through based models as examples in the profitability calculations which follow.

Click-through model: The spammer receives a fixed payment for every recipient who clicks on a link in the spam and is taken to a website (regardless of whether they purchase anything or the value of the final item).

Since it is easier to get someone to click a link on an email, the response rates could be between 1-click-for-every-2000-messages-sent and 1-click-for-every-10000-messages-sent. However, the revenue-per-click could be quite low, between €1 and €2 per click. Again, these values will vary considerably depending on how successful the spammer is, and what type of campaign they run. However, we will be averaging out these minimum and maximum values, so exact figures are not as critical for this part of the calculation.

Commission-based model: The spammer receives a fixed payment for every completed sale.

Fewer recipients will actually see the purchase through to completion once they click through to the website. So we can assume the response rates for this model are lower, say between 1-purchase-for-every-50,000-messages-sent and 1-purchase-for-every-100,000-messages-sent. But, the payment the spammer gets could be much higher, we will use the range €20-€40 per response for this model.

So we now have a broad range of minimum and maximum parameters, summarised in Table 4;

	Low	High
Click-through rate	1-in-2,000	1-in-10,000
Revenue per click	€1	€2
Conversion rate	1-in-50,000	1-in-100,000
Revenue per conversion	€20	€40

Table 4: Approximate parameter range values

From e), we know how much it will cost the spammer to send one 15-second stamped message. It can now be derived how much it will cost the spammer to achieve one revenue-generating response.

f) **Table 4** Messages for 1 response is taken from

g) **Table 4** Revenue per response is taken from

h) Cost per response is (Messages for 1 response / CPU Cost per message) (f / e)

Total cost needed to achieve one response					
		Click-based commission		Conversion-based commission	
		Min	Max	Min	Max
f)	Messages for 1 response	2,000	10,000	50,000	100,000
g)	Revenue per response	€1.00	€2.00	€20.00	€40.00
h)	Cost per response	€0.19	€0.95	€4.75	€9.51

i) Revenue minus Cost gives the gross profit per response (i = g – h)

		Click-based commission		Conversion-based commission	
		Min	Max	Min	Max
i)	Gross profit per response	€0.81	€1.05	€15.25	€30.49

However, because of the stamp slow-down there is now a 'cap' on the maximum number of messages the spammer can send in one day, this needs to be introduced into the calculation;

j) Max messages sendable in one day = Number of seconds in a day / number of seconds per message (3600*24)/15 (from d)

j)	Max messages per day	5,760
----	----------------------	-------

We can now apply j) to the gross profit per response calculation, to calculate maximum achievable profit per day for each of the models;

- k) Maximum messages per day divided by number of messages for 1 response (j/f)
- l) Profit per response (i) multiplied by max number of responses per day (k)
- m) Average of 4 values in row l)

		Click-based commission		Conversion-based commission	
		Min	Max	Min	Max
k)	Max responses per day	2.9	0.6	0.1	0.1
l)	Max profit per day	€2.33	€0.60	€1.76	€1.76
m)	Average max profit per day	€1.61			

By using the Excel goal-seek function, we now need to make m) equal to zero by modifying d) (seconds per message). We know that to make the spammer unprofitable, then we must reduce their profit per day figure (m) to zero, or less than zero.

The MS Excel goal-seek function (Making m = 0 by changing d) provides the results show in Table 5.

Base variables					
a)	Seconds in a year	31,557,600			
b)	CPU cost per year	€200.00			
c)	CPU cost per second	€0.000006338			
Cost of sending one message					
d)	Seconds per message	59			
e)	CPU cost per message	€0.00037447			
Total cost needed to achieve one response					
		Click-based commission		Conversion-based commission	
		Min	Max	Min	Max
f)	Messages for 1 response	2,000	10,000	50,000	100,000
g)	Revenue per response	€1.00	€2.00	€20.00	€40.00
h)	Cost per response	€0.75	€3.74	€18.72	€37.45
		Click-based commission		Conversion-based commission	
		Min	Max	Min	Max
i)	Gross profit per response	€0.25	-€1.74	€1.28	€2.55
j)	Max messages per day	1,462			
		Click-based commission		Conversion-based commission	
		Min	Max	Min	Max
k)	Max responses per day	0.7	0.1	0.0	0.0
l)	Max profit per day	€0.18	-€0.26	€0.04	€0.04
m)	Average max profit per day	€0.00			

Table 5: Spammer stamp breakeven point calculation

It can be seen that with a value of 59 for 'd) seconds per message', the average profit is reduced to zero. Given the uncertainty around the minimum and maximum range values in table, it is safe to round this up to 60 seconds to be sure of making the spammer unprofitable regardless of business model.

Therefore, in order for Hashcash to deter spammers, the minimum time required to calculate one stamp should be 60 seconds.

6.2 Part Two: Hashcash stamping performance measurement

Now that the minimum stamp time has been determined, the second part of the experiment sets out to collect data across processor types to find out how long various bit-lengths take to calculate.

6.2.1 Setting up the experiment

As described in 5.2.3, this part of the experiment involves running Hashcash performance measurement tests on a variety of processors and collecting the data for further analysis.

I had direct access to all the processors on the list except the Apple G-5 X-Serve (as this is quite an expensive machine). To obtain the data for this processor I contacted a developer from the Hashcash mailing list (Jonathan Morton) who was able to run the experiment and supply me with the required data.

The setup consisted of creating a DOS boot disk to boot the PC to a C:\> command line prompt. This was primarily to ensure that all the processing power was available to the Hashcash stamp generation, and not interrupted by operating system multi-tasking (which may have introduced inaccuracies).

The boot disk also contained a copy of the Hashcash executable (hashcash.exe) version 0.28, which was the latest stable build at the time of the experiment.

The 'hashcash -s' instruction was then typed at the C:\> prompt:. The '-s' flag instructs Hashcash to perform a loop function to supply a simple estimate for hash collision tests per second on that particular hardware, from which we can extrapolate how long an x-length stamp will take to calculate.

The 'hashcash -s' command was executed 10 times in succession and an average of the 10 results was recorded as the result for each processor type. Typical command-line output can be seen in Figure 10.

6.2.2 Results of the experiment

The results of the experiment are shown in Table 6. 'Collision tests per second' was the primary data measurement collected for each processor type.

Processor	Collision tests / sec	Seconds required to mint stamp of bit value:												
		20	21	22	23	24	25	26	27	28	29	30	31	
Intel 486	25MHz	5,000	209.7	419.4	838.9	1,677.7	3,355.4	6,710.9	13,421.8	26,843.5	53,687.1	107,374.2	214,748.4	429,496.7
Intel Pentium	200MHz	77,500	13.5	27.1	54.1	108.2	216.5	433.0	865.9	1,731.8	3,463.7	6,927.4	13,854.7	27,709.5
Intel Pentium 2	400MHz	210,521	5.0	10.0	19.9	39.8	79.7	159.4	318.8	637.6	1,275.1	2,550.2	5,100.4	10,200.8
Intel Pentium 3	750MHz	399,521	2.6	5.2	10.5	21.0	42.0	84.0	168.0	335.9	671.9	1,343.8	2,687.6	5,375.1
Intel Pentium 4	2GHz	658,915	1.6	3.2	6.4	12.7	25.5	50.9	101.8	203.7	407.4	814.8	1,629.6	3,259.1
Intel Pentium 4	3GHz	1,264,580	0.8	1.7	3.3	6.6	13.3	26.5	53.1	106.1	212.3	424.5	849.1	1,698.2
Intel Pentium 4 (opt)	3GHz	2,396,776	0.4	0.9	1.7	3.5	7.0	14.0	28.0	56.0	112.0	224.0	448.0	896.0
Apple G-5 X-Serve Dual	Dual 2.3Ghz	18,000,000	0.1	0.1	0.2	0.5	0.9	1.9	3.7	7.5	14.9	29.8	59.7	119.3

Table 6: Seconds taken to mint stamps on different processors

For example, the P2 400MHz can perform 210,521 collision tests per second compared to the 1,264,580 collision tests per second of the P4 3GHz machine.

6.2.3 Extrapolating minting times

An n-bit stamp will take on average 2^n collision tests until a partial hash collision is obtained. We now know how many collision tests per second can be performed by each processor type, so it is possible to calculate how many seconds each processor type will take on average to mint a stamp of n-bits (described in section 5.3). Example;

- To mint a 20-bit stamp, it will take on average $2^{20} = 1,048,576$ collision tests before a partial collision is found.

- The 486 25MHz can perform 5,000 collision tests per second (data obtained from direct measurement)
- Therefore the 486 25MHz will take, on average, $(1,048,576 / 5,000) = \mathbf{209.7 \text{ seconds}}$ to find a partial collision for a 20-bit stamp (as seen in the results in Table 6)

6.3 Review of experimental method

Overall the experiment and the results were accurate, and stood up to analysis providing the necessary conclusions for the research question. However, there were two potential problem areas; Different version releases of Hashcash, and extrapolation of data.

The developer community is continuously working on and optimising the performance of the Hashcash code so stamp generation times will be faster from one version release to the next. Although this will speed up the time taken for stamp generation, it will do so equally across all platforms, so the conclusions of this research should not be impacted. However, this variance across software versions should be considered if these tests are to be duplicated. I picked version 0.28 for the experiment as it was the most stable release available at the time.

As an example, I executed a later version of Hashcash (v0.30) on the Intel P4 3GHz. The result is shown in Table 6. It can be seen that the performance of Hashcash for this processor has almost doubled in just two version releases. (1,264,580 per second for v0.28 versus 2,396,776 for v0.30)

The data in Table 6 was obtained by extrapolating the collision test per second figures for each processor across the range of bit-lengths (as described in 6.2.3). This was necessary as it would take too long to calculate average minting times for the slower processors within the timescales of the research. To allow these processors to calculate full 31-bit stamps multiple times (for averages) would have meant running the Hashcash algorithm for weeks on each machine, which was unfeasible. Hence the extrapolation method was used.

One area which the extrapolation method did not allow investigation of was the 'variance problem'. When a full Hashcash stamp is generated we can estimate on average how long it will take (based on the 2^n formula 5.3), in reality it could be either faster or slower (described in 4.4). The 'full-stamp' generation results would have demonstrated this variance problem, but would have required lengthy iteration on the slower machines which was not feasible in the project timescales. The average (extrapolated) method was a better fit for the purposes of this analysis.

6.4 Analysis of results

Selected results are shown graphically in Figure 11 which focuses on the generation of a 24-bit stamp, showing how long it takes to mint on various processors. Even before incorporating spammer breakeven data, we can clearly see the disparity between the fastest and the slowest machine.

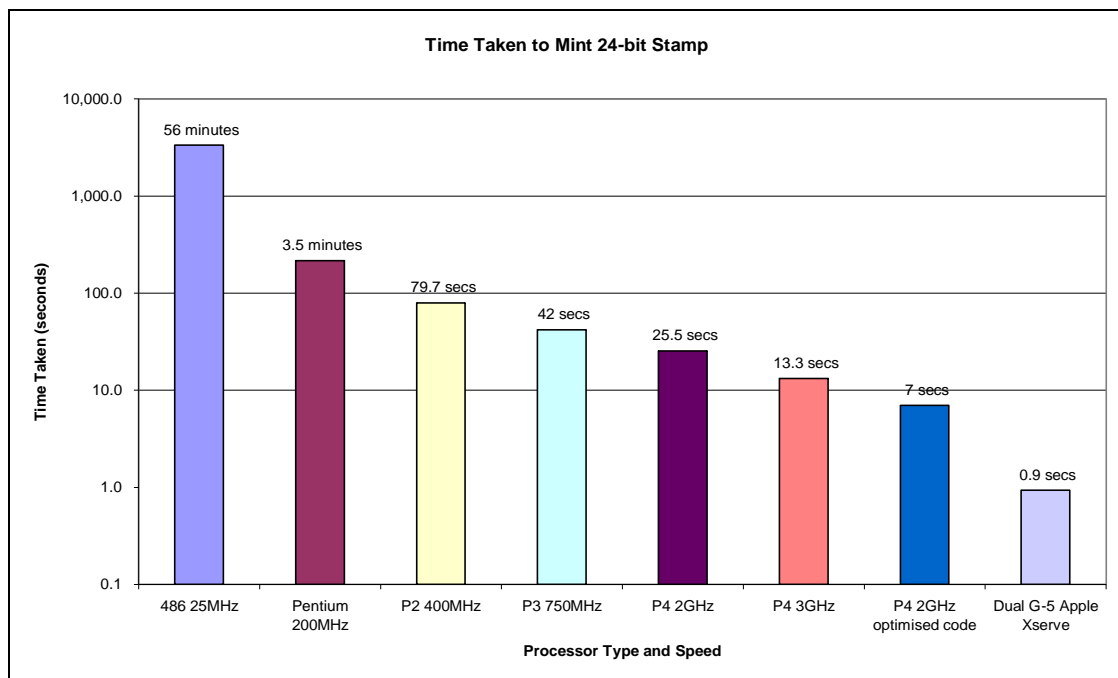


Figure 11: Time taken to calculate 24 bit stamp on various processors

Taking the full range of results, and incorporating the spammer breakeven point of one minute calculated in 6.1 gives us the chart shown in Figure 12;

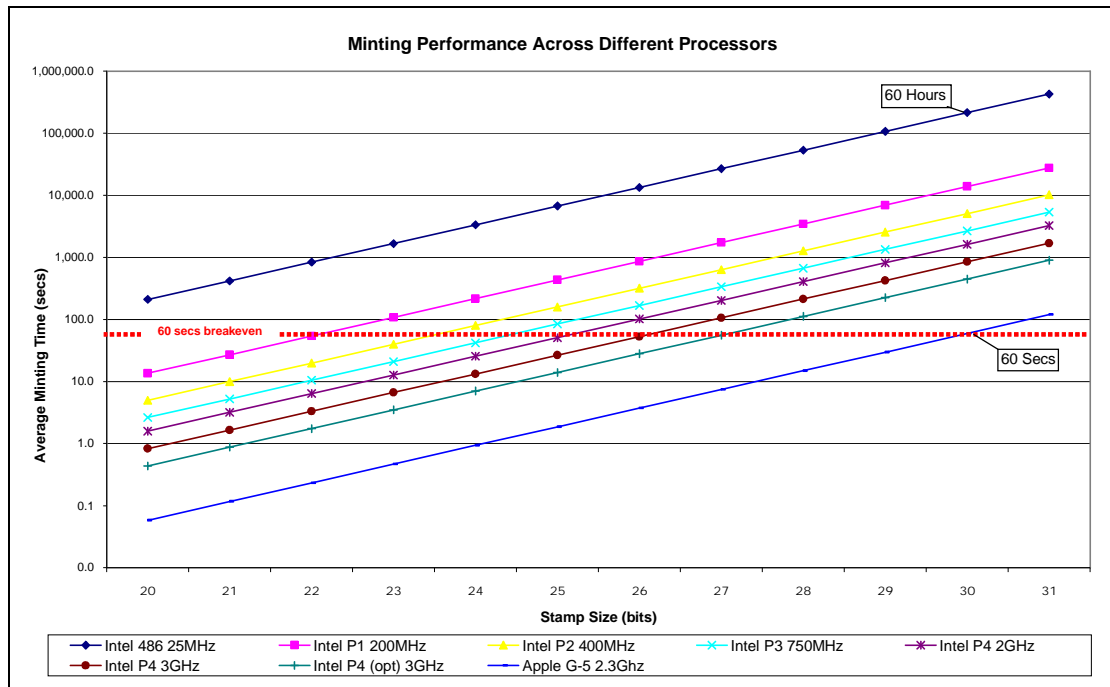


Figure 12: Hashcash minting performance across processors

The point at which even the fastest processor takes **60 seconds** to generate a stamp is at **30 bits**. So, for Hashcash to work as a deterrent for all spammers (assuming spammers are likely to afford fast processors like the Apple G-5), **a global stamp value of 30 bits needs to be used.**

Any less than 30 bits and the spammer is profitable (since it now takes him less than 60 seconds to generate a stamp which is below his breakeven point) and by just increasing the number of stamp-generating processors he will soon be sending large volumes of spam again. Which would render Hashcash useless as a solution.

Now, the problem can be seen. With a global stamp value set at 30-bits, the person running the 25MHz 486 will take 60 hours, or 2.5 days, to generate a single stamp. Even for the most patient user, this is unacceptable. Other machines, like the P2 400MHz (a common configuration still in use) would take 5,100 seconds, or over 1.5 hours to generate a stamp. Even if the user was willing to wait this long for the stamp to be generated, it would still limit his email sending to $(24/1.5) = 16$ emails per day. Which might not be enough for one user.

Moore's law states (roughly) that chip density doubles every eighteen months. This means that memory sizes, processor power, etc. all follow the same curve. Therefore in order to keep the stamp calculation times at spammer breakeven levels, it will be necessary to increase the global stamp postage value on a regular basis.

To be fully successful as a standalone anti-spam solution the Hashcash method will need to be universally adopted. If Hashcash is used in isolation as an anti-spam measure, then it will need to be installed across all servers and email clients in order to be successful, otherwise legitimate email senders without Hashcash stamps will not be able to send email to Hashcash enabled recipients.

6.5 Summary

In this chapter we applied the experimental methods described in chapter 5 to obtain two sets of primary data, from which we obtained two key figures; the spammer breakeven point (60 seconds) and the related Hashcash minimum bit value (30 bits)

Further analysis of the '30-bit' limitation and the performance data we obtained in part 2 allowed us to complete the missing requirements (2.3.3 transparency and 2.3.4 universally implementable). From this complete view of Hashcash against the requirements a conclusion to the research question was reached. This will be summarised in the next chapter.

7. Conclusion

7.1 Answering the research question

Is Hashcash proof-of-work a single technique which meets all the requirements of an anti-spam solution?

From the results and analysis in chapter 6, it is proven that Hashcash does not meet all the requirements of a standalone anti-spam system (as defined in 2.3). Specifically it fails in the areas of 'user transparency' (2.3.3) as it takes too long to generate the required stamp size on slower machines, and 'universally implementable' (2.3.4), performance disparity means it would be unfeasible for low-powered devices.

It is becoming increasingly clear that a single technology isn't going to solve the problem of spam (Roberts, 2004), (Ishibashi et al, 2003). I can concur with this statement, as unfortunately Hashcash doesn't offer a single-technology solution.

7.2 The future of proof-of-work

In some cases (for example low specification PCs, or other devices used to send email like handheld PDAs or mobile phones) the processing power in the hardware itself would not be enough to generate a reasonable size stamp. It may be possible to move the minting process to a separate minting server. The minting server would have to be powerful enough to handle multiple mint generation requests from multiple clients. And as Hashcash is designed to be processor intensive, then a rack of minting servers may be required, depending on the demand. It is important to protect these minting servers from attacks by spammers intent on using the power of these servers to generate stamps.

Proof of work could be used as a foundation for signatures. Obviously, it is cheaper for a sender to add a signature than a proof-of-work token, but if a signature relationship hasn't yet

been set up, the proof-of-work token is the way to go. Hashcash is a good candidate for the proof-of-work part of a hybrid solution. Content filters are the last line of defence. They should only come into play if a message passes the domain certificate stage but gets an inconclusive result from the signature and proof-of-work filters (perhaps because neither are present).

Any hybrid anti-spam solutions such as these discussed in this paper go far beyond the technically 'simple' anti-spam solutions currently in use, in that to be successful they need to be adopted on a global scale. This means the main industry players have to reach agreement on the type of solution to implement, and how it should be implemented.

Currently the main stumbling block in any of these proposed industry solutions is achieving agreement between the industry partners on the best way forward. Spammers have learned to adapt to overcome many of the anti-spam measures used against them thus the greater urgency for industry partners to work together sooner to implement a global hybrid solution.

7.3 Further work

7.3.1 Memory-bound proof-of-work function

An alternative suggestion to CPU-bound proof-of-work functions like Hashcash is the memory-bound function. This uses memory reads/writes to determine the speed of the stamp calculation instead of CPU-intensive calculations.

This has the possible advantage of eliminating the performance disparity problem of Hashcash. Memory performance has not increased as drastically as CPU performance over the years, and so performance disparity might not be as wide.

At the time of writing, I was unable to find an implementation of a memory-bound function (either in beta test or development) – If a memory-bound function could be developed, it would be interesting to perform the stamping performance experiments against it.

7.3.2 Ticket servers

This approach was proposed by Abadi et al. (2003) in a Microsoft whitepaper. Again, there is no known implementation of the 'ticket-server' approach, but it does offer a potential solution to the disparity problem, and would be a promising route of further research.

Using this method the sender of an email can choose their method of stamping their email (proof-of-work, micropayments, captcha graphic recognition etc). So if somebody has a slower device, they can choose an alternative method of stamping their outgoing email. Whichever method the sender chooses, the 'ticket-server' acts as a central controlling mechanism which issues the stamp challenge, confirms the response, and issues and verifies the stamps.

Appendices

Appendix A: The SMTP Protocol

The SMTP protocol described

Simple Mail Transport Protocol is the basic protocol used by servers to send email messages to each other (Schwartz, 1998). It defines how the conversation should take place, and the format of the data that is exchanged during the conversation. Figure 13 shows an SMTP conversation taking place between a sender and a recipient.

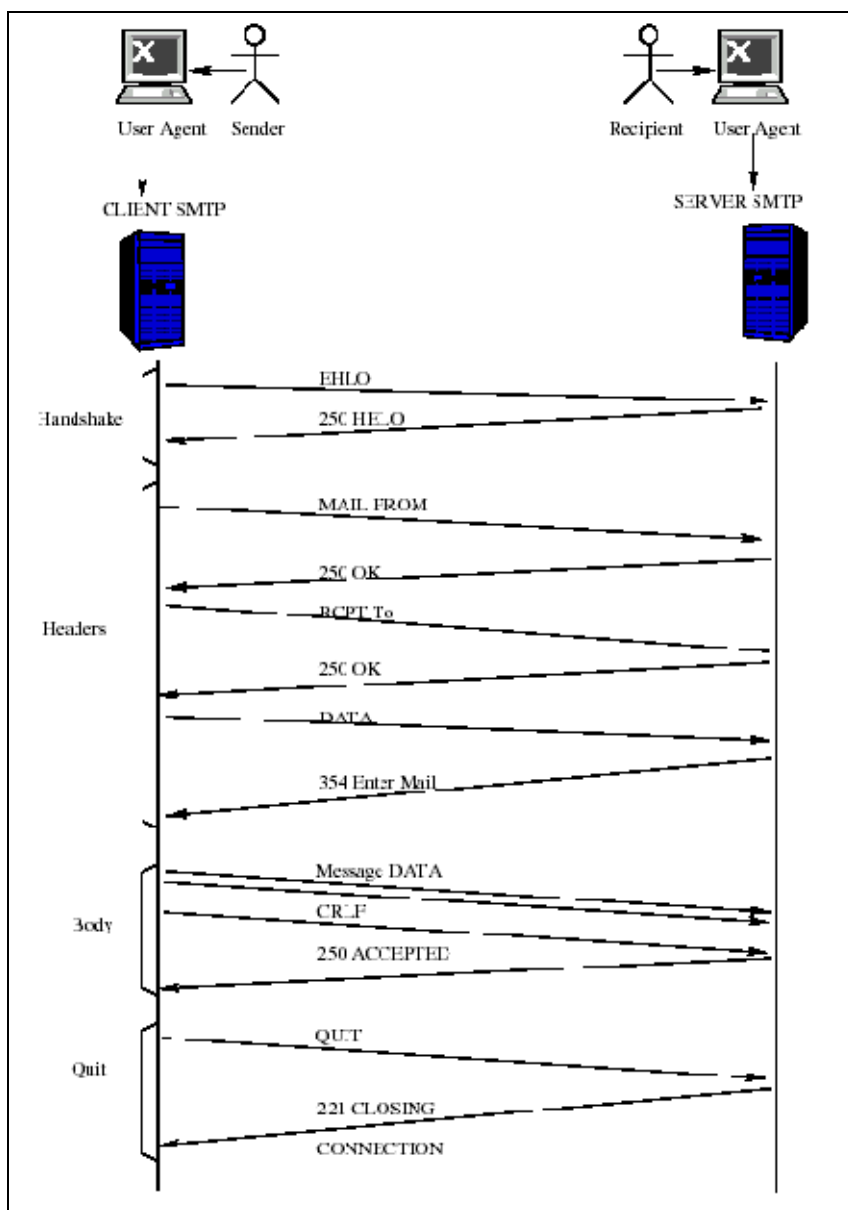


Figure 13: SMTP conversation. (Kaushik et al., 2004)

Mail User Agent

The email is composed in the senders Mail User Agent (MUA), this is usually a piece of software on the senders PC such as Outlook or Eudora, but can also consist of a web-based email system such as Yahoo or Hotmail. The message might look like this;

```
Date: Thursday, 1 Apr 2004 12:40:30 -0000
From: you@yourhost.com
To: Paul <paul@pluto.com>
Subject: Party on Sat night

There's a party on Saturday night, would you like to go?
```

Figure 14: Simple email header and body

There are two parts to the message in Figure 14; the header and the body. The header contains information about the message, such as who it is to be sent to. The body contains the actual text of the message itself.

When the sender clicks the 'Send' button in their MUA, some additional headers are automatically added to the message by the MUA. Figure 15 shows the new headers added by the MUA in bold.

```
Date: Thursday, 1 Apr 2004 12:40:30 -0000
From: you@yourhost.com
To: Paul <paul@pluto.com>
Subject: Party on Sat night
Message-Id: <002d01c444ca$bdaa3e70$5b92cbc1@yourhost.com>
X-Mailer: Microsoft Outlook Express 6.00.2800.1409

There's a party on Saturday night, would you like to go?
```

Figure 15: Message ID added by MUA

The message Id is a unique identifier added by the MUA, the X-Mailer is the name and version of the MUA software used to compose the email.

Mail Transport Agent

To deliver the email, the MUA needs to contact a Mail Transport Agent (MTA). The MTA is responsible for routing and delivering email.

When the MTA receives the email, it adds a header of its own, the **Received** header. This is like a postmark. Every MTA that handles a message adds this received header (Tserrefos et al, 1997; Clyman, 2004). Figure 16 shows how the message might look after the MTA has received it.

```
Received: from yourhost.com (HELO yourhost.com)
(193.203.146.91) by pluto.com with SMTP; 1 Apr 2004 12:40:40
Date: Thursday, 1 Apr 2004 12:40:30 -0000
From: you@yourhost.com
To: Paul <paul@pluto.com>
Subject: Party on Sat night
Message-Id: <002d01c444ca$bdaa3e70$5b92cbc1@yourhost.com>
X-Mailer: Microsoft Outlook Express 6.00.2800.1409

There's a party on Saturday night, would you like to go?
```

Figure 16: Header added by MTA

The Received header shows from which server the message was received, it often includes the IP address, in case the server supplied an incorrect or a faked hostname (yourhost.com). Other information stored in the Received header includes the name of the receiving MTA server, and may include other information such as the version of the MTA software the server is running.

If a message is forwarded between multiple servers on the way to its destination, they all add a Received header. So a message may have many received headers. They are read in reverse order. The header at the top indicates the last server which added its received header.

Finally, if the email message has reached its eventual destination, an SMTP From header is added as shown in Figure 17.

```
From: you@yourhost.com Thurs Apr 1 2004 12:40:30
Received: from yourhost.com (HELO yourhost.com)
(193.203.146.91) by pluto.com with SMTP; 1 Apr 2004 12:40:40
Date: Thursday, 1 Apr 2004 12:40:30 -0000
From: you@yourhost.com
To: Paul <paul@pluto.com>
Subject: Party on Sat night
Message-Id: <002d01c444ca$bdaa3e70$5b92cbc1@yourhost.com>
X-Mailer: Microsoft Outlook Express 6.00.2800.1409

There's a party on Saturday night, would you like to go?
```

Figure 17: Header added by final recipient

The From header lists the address as supplied by the sender in the MAIL part of the SMTP conversation. Some MTA agents store this in a 'Return-Path:' header instead. This final message is how it looks when it reaches the inbox of paul@pluto.com. The recipients email client software parses the header and presents the mail to the user in a more readable format (the final recipient often doesn't see all the headers in their email software, being mainly interested in the 'From', 'Subject' lines, and the body.)

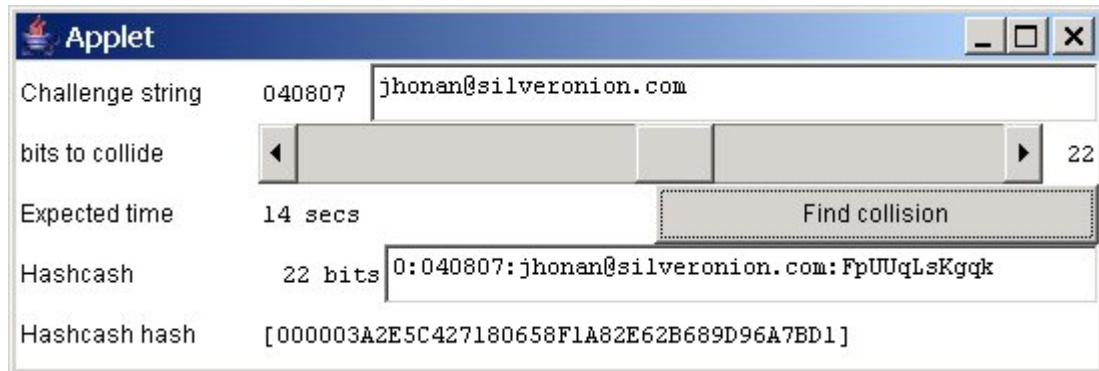
The headers as described above are standard SMTP headers. Any header beginning with 'X-' (such as X-Mailer) is a freeform header, and can be used for any purpose (Clyman, 2004).

Appendix B: Full email header including Hashcash stamp

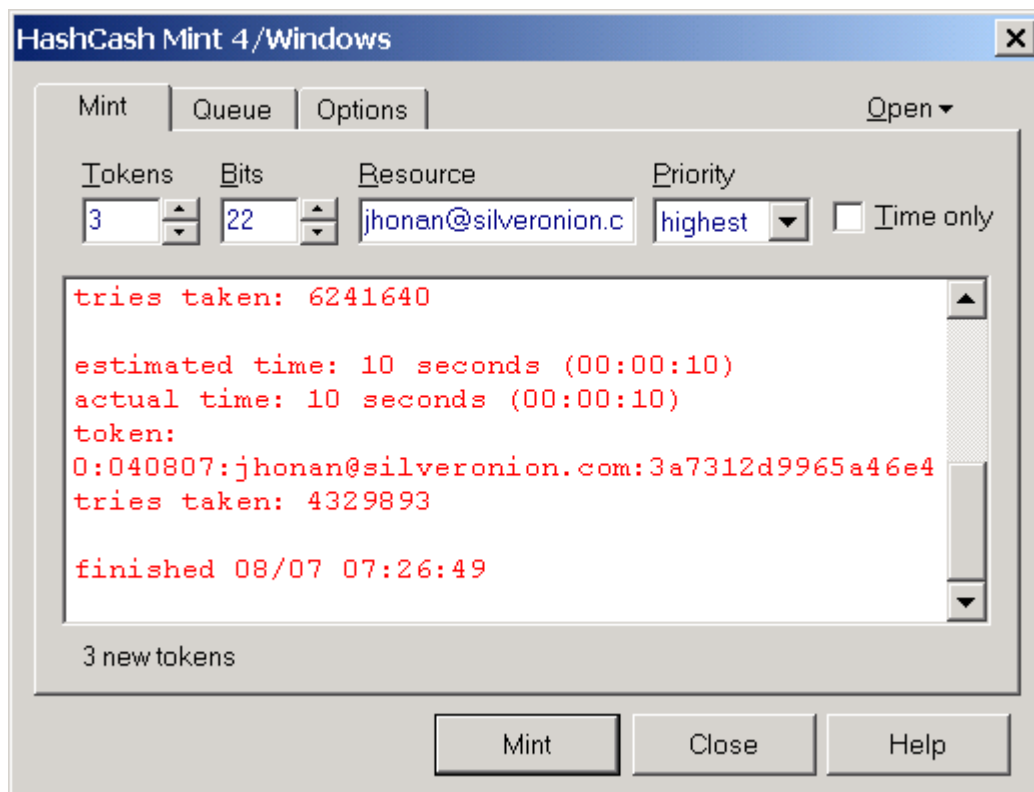
Subject: Re: [Hashcash] Re: Low end benchmarks
From: Adam Back <adam@cypherspace.org>
Date: Sat, 28 Aug 2004 19:34:46 -0400
To: John Honan <jhonan@silveronion.com>
Return-Path: adam@mail.off.net
Delivered-To: 38-jhonan@silveronion.com
Received: (qmail 25147 invoked from network); 28 Aug 2004 23:34:54 -0000
Received: from ms1.mail-shield.com (69.57.175.42) by lamda.control-access6.com with SMTP; 28 Aug 2004 23:34:54 -0000
Received: (qmail 10981 invoked by uid 105); 28 Aug 2004 23:33:32 -0000
Received: from adam@mail.off.net by ms1.mail-shield.com by uid 102 with qmail-scanner-1.22st (f-prot: 4.4.2/3.14.11); 28 Aug 2004 23:33:32 -0000
X-Spam-Status: No, hits=0.0 required=5.0
Message-ID: 20040828233446.GA5537@bitchcake.off.net
References: References: <32901.81.111.84.187.5558.squirrel@webmail.silveronion.com>
MIME-Version: 1.0
Content-Type: text/plain; charset=us-ascii
Content-Disposition: Inline
In-Reply-To: 32901.81.111.84.187.1093735558.squirrel@webmail.silveronion.com
User-Agent: Mutt/1.4.1i
X-Hashcash: 1:20:040828;jhonan@silveronion.com::9d8cee4c7ef5eb8d:98c0
X-Spam-Checker-Version: SpamAssassin 2.63 (2004-01-11) on ms1.mail-shield.com

Appendix C: Hashcash GUI front-ends

Simple Java applet to calculate a) expected time to get hash collision, based on number of collision bits required, b) actually calculate the hash collision and stamp.



Windows application to generate Hashcash stamps of a required bit value



References

Abadi, M., Birrell A., Burrows M., Dabek F., Wobber T. (2003) 'Bankable Postage for Network Services', Whitepaper published at

<http://research.microsoft.com/~birrell/papers/TicketServer.pdf> (1st Mar 2004)

Back, A. (1997) 'Hashcash', Published at <http://www.cypherspace.org/Hashcash/> (1st Mar 2004)

Back, A. (2002) 'Hashcash - A Denial of Service Counter-Measure'. Technical Report at <http://www.cypherspace.org/adam/Hashcash/Hashcash.pdf> (1st Mar 2004)

Baker, W. and Kamp, J. (2003) 'Summary of Can Spam Act', December. Report at <http://www.wrf.com/publications/publication.asp?id=1623481222003> (15th Aug 2004)

Bass, T. and Watt, G. (1997) 'Simple Framework for Filtering Queued SMTP Mail (Cyberwar Countermeasures)', Proceedings of IEEE MILCOM '97, Nov. 1997.

Boutin, P. (2004) 'Interview with a Spammer', *InfoWorld*. Article at http://www.infoworld.com/article/04/04/16/16FEfuturerichter_1.html?s=feature (16th Apr 2004)

Cerf, V. (2005) 'Spam, Spim, and Spit', *Communications of the ACM*, Vol. 48, No. 4 (April), pp. 39-43.

Clyman, J. (2004) 'The Problem with Protocols', *PC Magazine*, Ziff Davies Publishers, February 2004

Cranor L. and LaMacchia B. (1998) 'Spam!', *Communications of the ACM*, Vol. 41, No. 8 (August), pp. 74-83.

Damiani, E., De Capitani di Vimercati, S., Paraboschi, S., Samarati, P. (2004) 'P2P-Based Collaborative Spam Detection and Filtering', *Proceedings of the Fourth International Conference on Peer-to-Peer Computing (P2P'04)*, 25-27 Aug. 2004, pp. 176-183.

Deepak, P. Parameswaran, S. (2005) 'Spam Filtering using Spam Mail Communities', *Applications and the Internet, 2005. Proceedings. The 2005 Symposium on*, 31 Jan.-4 Feb. 2005, pp 377-383.

Dwork, C. and Naor, M. (1992) 'Pricing via Processing or Combating Junk Mail', *Crypto '92*, pp.139-147.

Filman, R. (2003) 'When email was good', *IEEE Internet Computing*, May 2003, pp. 4-6.

Gburzynski P., Maitan J. (2004) 'Fighting the Spam Wars: A Remailer Approach with Restrictive Aliasing', *ACM Transactions on Internet Technology* 4(1): 1-30, 2004.

Geer, D. (2004) 'Will New Standards Help Curb Spam?', *IEEE Computer Magazine*, Volume 37, Issue 2, Feb 2004 Page(s):14 - 16

Goodman, J. and Rounthwaite, R. (2004) 'Stopping Outgoing Spam'. *ACM Conference on Electronic Commerce, EC'04*, New York, pp: 20-39, ISBN: 1-58113-7110, 2004.

Goodman, J. and Heckerman, D. (2004) 'Fighting spam with statistics', *Significance*, June 2004, Vol. 1, No. 2, pp. 69-72.

Graham, P. (2003) 'Different methods of stopping spam'. October 2003. Article at http://www.secinf.net/anti_spam/Stopping_Spam.html (15th Jan 2004)

Hastings, N. and McLean, P. (1996) 'TCP/IP Spoofing Fundamentals', *IEEE IPCCC'96, IEEE International Phoenix Conference on Computers and Communications*, March 27-29, 1996, Phoenix, Arizona, USA IEEE 1996 0-7803-3255-5/96 pp.218-224

Holmes, N. (2005) 'In Defense of Spam', *Computer, IEEE*, Vol. 38, Issue 4, April 2005, pp. 86 - 88

Ishibashi, H., Yamai, N., Abe, K., Matsuura, T. (2001) 'Protection Method against Unauthorised Access and Address Spoofing for Open Network Access Systems', *IEEE 0-7803-7080-5/01*, pp.10-13.

Johansson, E. and Dawson, K. (2003) 'Camram'. Technical paper at <http://harvee.org:6080/~esj/output/camram.pdf> (Mar 14th 2004)

Juels, A. and Brainard, J. (1999) 'Client puzzles: A Cryptographic Countermeasure Against Connection Depletion Attacks', *Proceedings of NDSS '99 (Networks and Distributed Systems Security)*, pp. 151-165.

Jung J., Sit E. (2004) 'An Empirical Study of Spam Traffic and the Use of DNS Black Lists', *Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement, Taormina, Sicily, Italy*, Oct 25-27 2004, pp. 370-375.

Kaushik, S., Ammann, P., Wijesekera, D., Winsborough, W., Ritchey, R (2004) 'A Policy Driven Approach to Email Services', *Policies for Distributed Systems and Networks, 2004. POLICY 2004. Proceedings. Fifth IEEE International Workshop on 7-9 June 2004*, pp. 169-178.

Kim, H.J., Kim, H.N., Jung, J.J., Jo, G.S. (2004) 'On Enhancing the Performance of Spam Mail Filtering System Using Semantic Enrichment', *Lecture Notes in Computer Science*, Volume 3339, Jan 2004, pp. 1095-1100.

Lai, Chih-Chin and Tsai, Ming-Chi (2004) 'An Empirical Performance Comparison of Machine Learning Methods for Spam E-mail Categorization', *Proceedings of the Fourth International Conference on Hybrid Intelligent Systems*, 05-08 Dec 2004, pp. 44-48.

Laurie, B. and Clayton, R. (2004) 'Proof of Work proves not to work', *The Third Annual Workshop on Economics and Information Security (WEIS04)*, May 13–14, 2004, University of Minnesota, Digital Technology Center

Levine, J. (2004) 'Putting a spammer in jail', *CircleID*, November 16th, 2004. Article at http://www.circleid.com/article/804_0_1_0_C/ (Nov 16th 2004)

Levy, E. (2003) 'The Making of a Spam Zombie Army', *Security & Privacy Magazine, IEEE*, Vol. 1, Issue 4, July-Aug. 2003, pp. 58-59.

Mangalindan, M. (2002) 'For Bulk E-Mailer, Pestering Millions Offers Path to Profit', *Wall Street Journal*, 13 Nov 2002.

Microsoft (2004) 'The Coordinated Spam Reduction Initiative – A technology and policy proposal', *Microsoft Corporation*. Whitepaper available at <http://www.microsoft.com/downloads> (Feb 13th 2004)

Pelletier, L., Almhana, J., Choulakian, V. (2004) 'Adaptive Filtering of SPAM', *Communication Networks and Services Research, 2004. Proceedings.*, Second Annual Conference on 19-21 May 2004, pp 218-224

Pfleeger, S.L., Bloom, G. (2005) 'Canning Spam: Proposed Solutions to Unwanted Email', *Security & Privacy Magazine, IEEE* Vol. 3, Issue 2, March-April 2005, pp 40-47

Roberts, P. (2004) 'Experts Question Microsoft's Caller ID Patents', *InfoWorld*, March 2004. Article available at http://www.infoworld.com/article/04/03/05/HNcalleridpatents_1.html (20th Apr 2004)

RSA Laboratories (2000) 'RSA Laboratories Frequently Asked Questions About Today's Cryptography', *Version 4.1*, *RSA Security Inc.* FAQ available at <http://www.rsasecurity.com/rsalabs/node.asp?id=2152> (July 10th 2005)

Schiavone, V., Brussin, D., Koenig, J., Cobb, S., Everett, R. (2003) 'Trusted Email Open Standard – A Comprehensive Policy and Technology Proposal for Email Reform', *ePrivacy Group*, Whitepaper available at <http://www.eprivacygroup.net/teos/TEOSwhitepaper1.pdf> (15th Jan 2004)

Schryen, G. (2004). 'Approaches Addressing Spam', *Proceedings of IPSI*, Hawaii 2004

Schwartz, A., Garfinkel, S. (1998). 'Stopping Spam'. *O'Reilly*, ISBN:156592388X

Simpson, P. (2002) 'Putting Spam Back in the Can', *ITsecurity.com*. Article at <http://www.itsecurity.com/archive/papers/mime6.htm> (13th May 2004)

Snyder, J (2003) 'Test: Spam in the wild'. *Network World Fusion*, 9 Sep 2003. Article at <http://www.nwfusion.com/reviews/2003/0915spam.html>

Spinello, R.A. (1999) 'Ethical reflections on the problem of spam', *Ethics and Information Technology*, Vol. 1, Issue 3, Sep 1999, pp. 185-191.

Stuart, I., Cha, S.H., Tappert, C. (2004) 'A Neural Network Classifier for Junk E-Mail', *Lecture Notes in Computer Science*, Vol. 3163, Jan 2004, pp. 442-450.

Templeton, S., Levitt, K. (2003) 'Detecting spoofed packets', *IEEE DARPA Information Survivability Conference and Exposition - Volume I*, April 22 - 24, 2003, Washington, DC, pp. 164-177, 2003

Tserefos, P., Smythe, C., Stergiou, I., Cvetkovic, S. (1997) 'A Comparative Study of Simple Mail Transfer Protocol (SMTP), POP and X.400 Email Protocols', *22nd Annual IEEE Conference on Local Area Networks*, Minneapolis, USA, 1997, pp.545-55

William S. Yerazunis, W (2003) 'Sparse Binary Polynomial Hashing and the CRM114 Discriminator', *Spam Conference Presentation*, January 2003. Slide available at http://crm114.sourceforge.net/crm_slides/img39.html (15th Jan 2004)