Network Working Group Adam Back
draft-hashcash-back-00.txt Category: INTERNET-DRAFT Created: Jun 2003
Expires: Dec 2003

The Hashcash Proof-of-Work Function
draft-hashcash-back-00.txt

Status of this Memo

This document is an Internet-Draft. Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

Abstract

Hashcash is a denial-of-service counter measure tool. It's main current use is to help hashcash users avoid losing email due to content based and blacklist based anti-spam systems.

Email senders attach hashcash tokens with the X-Hashcash: header. Vendors and authors of anti-spam tools are encouraged to exempt mail sent with hashcash from their blacklists and content based filtering rules.

This memo describes how to use hashcash to increase email reliability.

Distribution of this memo is unlimited.

1. Introduction

The Hashcash Proof-of-Work function is an algorithm that can be used as a building block in the construction of network protocols with defenses against Denial-of-Service attacks.

The Hashcash function can be used with any iterative cryptographic hash function, e.g., SHA1. The cryptographic strength of Hashcash

depends on the properties of the underlying hash function. This draft specifies SHA1 for the underlying hash function.

The Hashcash proof-of-work function is non-interactive and so is suitable for use in store-and-forward systems such as email and USENET news. Senders attach Hashcash tokens to their sent email and USENET posts. Mail Filtering Agents verify the hashcash tokens attached to email and alter their filtering behavior in some way. For example a spam detecting system such as SpamAssassin may choose to exempt all mail sent with hashcash from other filtering rules. A mail-client may filter mail without hashcash into a potential spam folder.

2. Definition of Hashcash

The definition of Hashcash requires a cryptographic hash function, and the version 0 format Hashcash token specifies the SHA1 [SHA1] hash function. SHA1 hashes variable length octet-strings to produce a 160 bit (or 20 octet) output. We denote the bit-length of SHA1 hash output L=160 bits.

Hashcash is parameterised by:

- a work factor w, is an integer 0 <= w <= L
- a version number: ver (this document specifies version '0')
- a time-stamp parameter: time
- a resource identifier: resource (variable length ascii string)
- a trial value string: trial (variable length ascii string)

The hashcash token has a number of fields separated by the ':' character.

The token has form:

```
token = ver:time:resource:trial
```

The ver and time paramters are defined as the first two ':' delimited fields. The trial parameter is defined as the last ':' delimited field. The resource string is the string between the ':' following the time field and the ':' preceding the trial field. This definition allows the resource string to contain ':' characters.

See also the section below on double-spending for a use of the time parameter which MAY be used to reduce the size of the double-spending database.

The trial parameter is a variable length ascii strings consisting of

any printable character except whitespace, linefeed, newline and the subset of valid trial characters of a specific length (for example hex or base64 encoding characters only, say being an encoding of a 64 bit number), however all implementations MUST accept any valid characters, and MUST accept trial strings of any length up to a maximum of 128 characters.

We define a target string 'target' against which partial hash collisions are found, and arbitrarily set this to the L-bit all 0 string which we denote: 0^L (the binary string composed of L 0 bits):

```
chal = 0^L
```

We define a w-bit partial collision between the hash of the trial token and the zero string chal where the w most significant bits of trial value hash are equal. We use msb(w, s) to denote the w most-significant bits of bit string s. The client tries different trial values until a w-bit partial collision is found such that:

```
msb( w, SHA1( token ) ) == 0^w;
```

An example algorithm to find such a token is given by the following pseudo-code:

```
trial = random_start();      REPEAT          trial := trial + 1
UNTIL msb( w, SHA1( vers:time:resource:trial ) ) == 0
```

In practice trial may be for example a 64 bit number and the addition would be mod 2^64. In this case the trial used in the SHA1 hash would be some printable encoding of a 64 bit number (for example hex or base64 encoding).

The trial value MUST be chosen randomly. If the trial were not chosen randomly collisions may accidentally occur, or be maliciously be induced between different senders to the same recipient.

The length of the trial value SHOULD be chosen to be large enough that the probability of collision between to senders to the same recipient is negligible. It is recommended that a range of at minimum 2^64 possible values be used.

3. Double-spending

Tokens SHOULD NOT be accepted more than once. In the non-interactive setting to implement this the server MAY implement a double-spend database. As tokens are spent they are added to the double-spend database. Tokens SHOULD only considered valid if they have not been

already spent (if they are not in the double-spend database).

3.1 Reducing the size of the double-spend database

To reduce the storage requirements for the double-spend database non-interactive tokens MAY use the optional time-stamp parameter. The server will then define an expiry period for a token based on it's time-stamp, and can purge from it's database any tokens which are expired.

Servers MAY elect to communicate the expiry period they impose on tokens they receive to the clients.

Servers SHOULD NOT expire tokens earlier than the typical worst case communication delay for the associated message.

The time-stamp format shall be based on UTCTime: 2-12 digits with the trailing 'Z' character omitted. Reading from left to right in pairs of digits the date/time stamp is: year (modulo 100), month, day, hour (24 hour), minute, seconds. The timezone is GMT (UTC). (The 2 digit year SHOULD be converted into a 4 digit year by comparing to the current year. The interpretation that is closest to the current date SHOULD be used.)

Author's Address

Adam Back
Email: adam@cypherspace.org