Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 8, 2008

C. Jennings
Cisco Systems

July 7, 2007

Computational Puzzles for SPAM Reduction in SIP
draft-jennings-sip-hashcash-06

Status of this Memo

Copyright Notice

Abstract

One of the techniques used in SPAM prevention and various solutions for denial of service attacks is to force the SIP client requesting a service to perform a calculation that limits the rate and increases the cost of the request. This draft defines a way to allow a UAS to ask the UAC to compute a computationally expensive hash based function and present the result to the UAS. Although the computation is expensive for the UAC to compute, it is cheap for the UAS to verify. The solution also allows for proxies to compute and check

the puzzle on behalf of the UAC or UAS.

This draft currently outlines enough information to evaluate and consider this approach or even run experiments. It would need finalization around the forking topics discussed in the open issues before it would be implementable in production system.

Table of Contents

1. Introduction 3
2. Overview 5
3. Definitions 6
4. Puzzles 6
5. Semantics 7
5.1. UAS Creating Puzzle 7
5.2. UAC Receiving Puzzle 7
5.3. Proxy Behavior 8
6. Example 8
7. Syntax 12
8. Open Issues and To Do Items 13
9. Security Considerations 14
10. IANA Considerations 14
10.1. Puzzle Header 14
10.2. 419 Response 14
11. Acknowledgments 15
12. Appendix A: Test Vectors 15
13. References 28
13.1. Normative References 28
13.2. Informational References 28
Author's Address 28
Intellectual Property and Copyright Statements 29

1. Introduction

The SPAM prevention problem is complex and will require many techniques working in combination to balance reducing SPAM to acceptable levels while still fostering efficient communication. The overall problem and various approaches are in [7]. Clearly white lists are a critical part of dealing with SPAM. Any system would first check whether an incoming request for communications was from someone on the white list. The Identity [6] mechanisms are critical for understanding who the caller is and to check whether the caller is on the white list. As well, there still needs to be a way for callers not on the white list to communicate with the user. It is here that this specification becomes relevant.

The problem is how to permit contacts from people with no previous relationship to us without receiving undesirable contacts. This draft uses the idea that it may be possible to make undesirable contacts more expensive than desirable ones.

Different undesirables are willing to spend different amounts of time and money on contacting their markets. Founders of acquired startups are often contacted by random financial companies offering to help manage the new riches. These companies will send people from New York to San Jose and spend hours talking to this very narrow target market. Clothing retailers will mail glossy catalogues worth $1 apiece to houses within the right demographic zip codes. Emails advertising Viagra are sent to random email addresses. As the costs go down, the volume of unsolicited contact goes up.

Often people whose contact is desirable are willing to spend much less than some of the undesirables. The student in Fiji who wants to ask about this draft will send an email but probably will not fly here to talk to me. I would like to receive that email.

Increasing the cost of contact will reduce both desirable and undesirable contact. My assumption is that the cost should be set very low, so that even a person with a pathetic CPU could still make contact in, say, 10 seconds. Key to this draft is that the receiver can set this cost. This low cost will not stop the financial advisers or the telemarketers, but it might stop the Viagra ads. It would also probably stop a single user from ringing every phone of some residential service provider in a five-second window, before any operator or system can react. Deciding what cost to set constitutes a classic type I/type II error problem, and the receiver gets to choose how to balance these two errors.

As is clearly stated in [7], whitelists are the best thing. After that, this is one of the multiple other options that need

consideration.

In general there are two arguments about why the computation puzzles in this specification will not work. The first is that the bad guys have the most powerful CPUs. This issues was addressed above. The other argument is that bad guys have infinite CPU time through using armies of zombie PCs. The problem with this argument is that the goal is not to block particular bad guys but to reduce the overall number of undesirable messages. This second argument is, however, more worrisome than the first.

Assume that some percentage of the world's machines each year get owned and used as zombies. Let's say that a given machine has 1% of having this happen to it in a year, that it sends zombie traffic for 24 hours before getting shut down, and that the mechanism described here limits it to ten messages per second: each machine on the internet would receive an average of about one undesirable message per hour. If you assume there are more users than machines, this looks appealing. If message sending technology detects users that are sending lots of messages and shuts them down in less than 24 hours, it gets better. It gets better still if you hope for improvements in operating systems or for users to choose them more carefully. The next assumption is hard to model statistically but it is true: the people with the best financial incentives to send undesirable messages do not want to be subject to the legal and reputation problems of using zombies to get their message across.

The zombie problem basically comes down to this. If there are a small percentage of machines in the world that are zombies, they do not render this computation puzzle approach useless. If 10% of the machines in the world are zombies, this approach will be useless. This specification does not attempt to deal with how to make the world such that a small percentage of computers are zombies - the is the problem for other work and that work needs to happen for SPAM to be reduced to reasonable level. This specification does assume that the zombie problem is solved to the level where a small percentage of the worlds computers are zombies.

Clearly there is a need to be able to initiate SIP communications from very low power, low cost, devices. They will have relatively slow CPUs and their users will be very impatient and only willing to wait a short time to compute the puzzle. On the other hand there will be attackers with very fast computers and possibly many of them. The relative ratio of these speeds and size of the attacker population will determine how effective this approach is.

So in summary, white listing is the first and best defense. But for dealing with messages from people with whom we have not previous

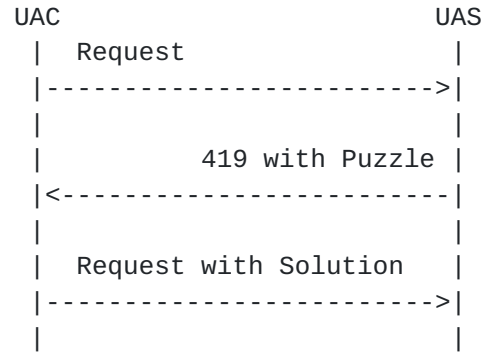
direct or indirect relationship, another approach is necessary. Puzzles cannot stop all bad messages - that is not the goal - but it can raise the cost of messages and thus decrease the number of times it makes economic sense to send undesirable ones. This approach does assume that bad guys will have more CPU power than good guys and that zombies will still send lots of messages. This approach will simply reduce the number of undesirable messages by some amount that cannot be measured.

No one knows if this approach would reduce SPAM noticeably. Right now the only thing that limits the rate at which I can call every SIP phone in the world is proxies getting overloaded. And of course, most SIP phones are not connected to the public internet. The SPAM problem is one reason why many SIP phones are not connected to the public internet. There are some other approaches outlined in [7]. They have different pros and cons, and it is probably necessary to use most of them to ensure SPAM stays at an acceptable level.

2. Overview

This specification extends RFC 3261 [3] and defines a mechanism for a proxy or UAS to request that a UAC compute the solution to a puzzle. The puzzle is based on finding a value called the pre-image that, when hashed with SHA1 [4], results in a specific value referred to as the image. The goal is for the UAC to find a pre-image that will SHA1 hash to the correct image. The UAS provides a partial pre-image with some of the low order bits set to zero, together with the number of bits in the pre-image that have been set to zero.

The UAS provides the puzzle information using a 419 response, and the UAC resubmits the request along with the solution to the puzzle. The high level flow of information is shown below.



This specification defines the 419 response code along with a new header, called Puzzle, to carry the puzzle and solution.

3. Definitions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [2].

4. Puzzles

The normative definition of a puzzle is as follows. A puzzle is four values: an integer number referred to as work, a pre-image string, an image string, and a integer number referred to as value. There MUST exist a value X such that all but the "work" number of low order bits of X match the pre-image string, and the SHA1 hash of the string formed by the concatenation of "z9hG4bK" and X results in a value Y, where the "value" number of low order bits of Y are the same as those bits in the image string. The SHA1 hash is computed as described in RFC 3174 [4]. The value X is the solution to the puzzle. The 'work' number of low order bits of the pre-image MUST be zero.

This can all be described more mathematically. The notation low(v,x) returns the first v number of low order bits of the value x, and zero(v,x) returns x with the lowest v number of bits set to zero. The | operator signifies string concatenation. The solution to the puzzle can be considered finding an X such that both the following are true:

```
   low( value, image ) = low( value, sha1( "z9hG4bK" | X ) )
       zero( work, X ) = zero( work, pre-image )
```

The pre-image forms a constraint on X. The value of X is the same pre-image, other than the low 'work' bits that are set to zero in the pre-image. The 'value' is the number of bits that match in the solution and is typically set to 160, which is the full size of the SHA1 hash result.

The following is a non-normative way for a UAS or proxy to construct a puzzle. The following strings are concatenated:

1. a secret that only this device knows. This would typically be a crypto random string of bits;
2. the current time
3. the URI of the request, the Call-ID, the From tags, and the branch tag for a proxy or the To tag for a UAS.

The string is hashed with SHA1 to form the pre-image. The pre-image is appended to the string "z9hG4bK", and the SHA1 hash of this is

computed to get the value of the image. A value 'work' indicates how many bits of the pre-image are to be removed. The value 'work' could be a configurable parameter, or it could be dynamically discovered by the software based on how long a hash should take and the speed of the computer it was running on. In the latter case, the resulting software would automatically choose larger values of 'work' as computers get faster. The low order 'work' bits of the pre-image are set to zero. The puzzle consists of the chosen value of 'work', the pre-image (with the low order bits set to zero), the image, and the 'value'. The 'value' would typically be set to 160 as this is the size of the SHA1 hash. Since the time was rounded

Note: Some implementors have pointed out that this approach requires the UAS to do a SHA1 to compute the puzzle and that this creates extra load on the UAS. On a machine with a proxy that could process about thousand sip transactions per second, the approximate rate of puzzle creation was over one million puzzles per seconds. The work to create a puzzle is trivial compared to the work to receive a sip messages and send the response. The advantage of a puzzle in this form seemed apparent at some time in the past but I can not remember why. Big TODO item to recall why this form was used. This form does allow progressively better solutions to be found with a higher "value" without changing the image string.

5. Semantics

5.1. UAS Creating Puzzle

When a UAS wishes to challenge a request, it MAY create a puzzle, encode this puzzle in a Puzzle header field value, and return the puzzle in a 419 response.

5.2. UAC Receiving Puzzle

When a UAC receives a 419 response, it needs to look at the 'work' and 'value' requested and decide whether or not to try to solve this puzzle. This decision can be made based on the programmed policy and possibly human input. The UAC should not tackle a puzzle that will take longer than the age of the universe to solve. If the UAC chooses to try to solve the puzzle then it proceeds along the following steps:

1. Check that the 'work' bottom bits of the pre-image are all zero. If they are not, this is an invalid puzzle and the 419 response MUST be considered an error response.

2. Set Y to low(value, image).
3. Create a loop where X ranges from the value of the pre-image to the value of the pre-image plus 2 raised to power of the 'work'.
4. For each interaction through the loop, check if low(value, sha1("z9hG4bK" | X)) equals Y. If it does, a solution X has been found and the loop can terminate.

If the loop terminates without a solution being found, the puzzle was bad and the 419 response MUST be considered as an error response.

Once the solution to the puzzle, X, is found, a new request is formed by copying the old request and adding an additional puzzle header field value. The new puzzle header field value MUST have the 'work' set to 0, the pre-image set to the value X, the image set to the value of the image in the original puzzle, and the value parameter set to the same as the value parameter in the original puzzle. Note that if a request was challenged by one proxy and a new request was generated with a solution, and then this request was challenged by a second proxy, a third request would be generated that had two Puzzle header field values. If a UAC, through some out of band mechanism, knows that it will be challenged and what the puzzle will be, it MAY include the appropriate puzzle header field value in the initial request.

5.3. Proxy Behavior

SIP allows proxies to act as UASs when generating 4xx responses. This same mechanism can be used to allow a proxy to generate the challenge on behalf of a UAS in its domain.

Proxies may also act on behalf of the UAC and compute the solution to a puzzle on behalf of the UAC in either a request or a response that passes through the proxy. Typically a proxy would only do this for a UAC that had authenticated to the proxy and for which the proxy had a service relationship.

6. Example

In this example, we present a communication establishment between two users, Alice (sip:alice@example.com) and Bob (sip:bob@example.net). First, Alice sends an INVITE to Bob. Bob, who wants to make sure that Alice is not a spammer, replies to Alice with a message "419 - Puzzle Required'', indicating that he wants Alice to pass a challenge before establishing a communication. Bob's reply contains a Puzzle that Bob wants Alice to solve. To get the "preImage'' value, Bob generates a random string that he hashes with SHA1. If we define "SHA1'' as the method that hashes a string:

preImage = SHA1(random string)

To get the "image'', Bob appends the string "z9hG4bK" to his "preImage'', and he hashes this new string with SHA1 again. We can say:

image = SHA1 ("z9hG4bK"|preImage)

where the "|" operator signifies string concatenation.

Bob chooses to fix the value of "value'' to 160, as it is the size (in bits) of the "preImage'' and the "image'' that he's going to send to Alice. As seen before, the value of "work'' determines the difficulty of the problem, and has to be set up depending on the power of Alice's UA. To simplify the problem, let's suppose that Bob chooses to fix "work'' to 15. Before putting these values into the "Puzzle header'' field, Bob saves the value of his "preImage", and applies the method "zero(value, preImage)" to set the "value" bottom bits of "preImage'' to 0.

If we assume that Bob has picked the random string "itjjyfdubtpneggrdsaavouy", he has the following values:

random string
= "itjjyfdubtpneggrdsaavouy"

original preImage
= SHA1(random string)
= "VgVGYixbRg0mdSwTY3YIfCBuYmg=" (base-64 encoded)
= "01010110 00000101 01000110 01100010 00101100 01011011 01000110
   00001101 00100110 01110101 00101100 00010011 01100011 01110110
   00001000 01111100 00100000 01101110 01100010 01101000" (binary)

sent preImage
= zero(work, preImage) where work = 15
= "VgVGYixbRg0mdSwTY3YIfCBuAAA=" (base-64 encoded)
= "01010110 00000101 01000110 01100010 00101100 01011011 01000110
   00001101 00100110 01110101 00101100 00010011 01100011 01110110
   00001000 01111100 00100000 01101110 00000000 00000000" (binary)

image
= SHA1 ("z9hG4bK"|original preImage)
= "NhhMQ2l7SE0VBmZFKksUC19ia04=" (base-64 encoded)

Then Bob constructs the Puzzle header field, that has this form:

Puzzle: work=15; pre="VgVGYixbRg0mdSwTY3YIfCBuAAA=";
        image="NhhMQ2l7SE0VBmZFKksUC19ia04="; value=160

At this point Bob answers to Alice's INVITE, with a message "419 Puzzle required" containing the "Puzzle header" field that he has just built. When Alice receives this message, she looks at the value of "work" and "value", and in our case she decides to try to solve this puzzle. She decodes the "preImage'' and the "image'' with a base64-decoder. Then she checks that the "work" bottom bits of preImage are set to 0. If it is not the case, she would have to consider the 419 message as an error message. Alice creates a variable Y, set to low(value, image). She creates another variable, X, that will contain the solution, and that is initialized to the preImage that she has read in the "Puzzle header" field.

The binary value of X, before starting to search for a solution, is:

01010110 00000101 01000110 01100010 00101100 01011011 01000110
00001101 00100110 01110101 00101100 00010011 01100011 01110110
00001000 01111100 00100000 01101110 00000000 00000000

In fact it's the same value as the "preImage'' that Bob has put in the "Puzzle header" field, because Alice has initialized X to this value. This "X" is the start-point of solving the puzzle.

Then she starts looping...

While low(value, SHA1("z9hG4bK" | X)) doesn't equal Y, she must "increment" X, ie binary add 1 to the bit-representation of X. In other words, if during a given iteration the binary value of X is:

01010110 00000101 01000110 01100010 00101100 01011011 01000110
00001101 00100110 01110101 00101100 00010011 01100011 01110110
00001000 01111100 00100000 01101110 00010010 10010111

then Alice has to add 1 to this value, and the new binary value of X must be:

01010110 00000101 01000110 01100010 00101100 01011011 01000110
00001101 00100110 01110101 00101100 00010011 01100011 01110110
00001000 01111100 00100000 01101110 00010010 10011000

The maximum number of iterations is "2^work", because Alice has received a "preImage'' with the "work" bottom bits set to 0, and that the biggest solution would can be the "preImage'' with the "work" bottom bits set to 1. This configuration is accessible via "2^work" iterations.

If Alice has finished looping without finding a solution, she must consider that the puzzle was invalid, and then consider the "419 Puzzle Required" as an error message.

If Alice finds an X such as low(value, SHA1("z9hG4bK" | X)) equals Y, she has the solution! She can break out of the loop and build a response for Bob.

As a response, Alice will send a copy of her initial request, but she will insert the same "puzzle header field" as the one she has received in the "419 Puzzle Required", except the "work" field that she sets to 0, and the "preImage" field where she puts the solution of the problem. She won't forget to base64-encode her solution, X, before putting it in the "Puzzle header" field.

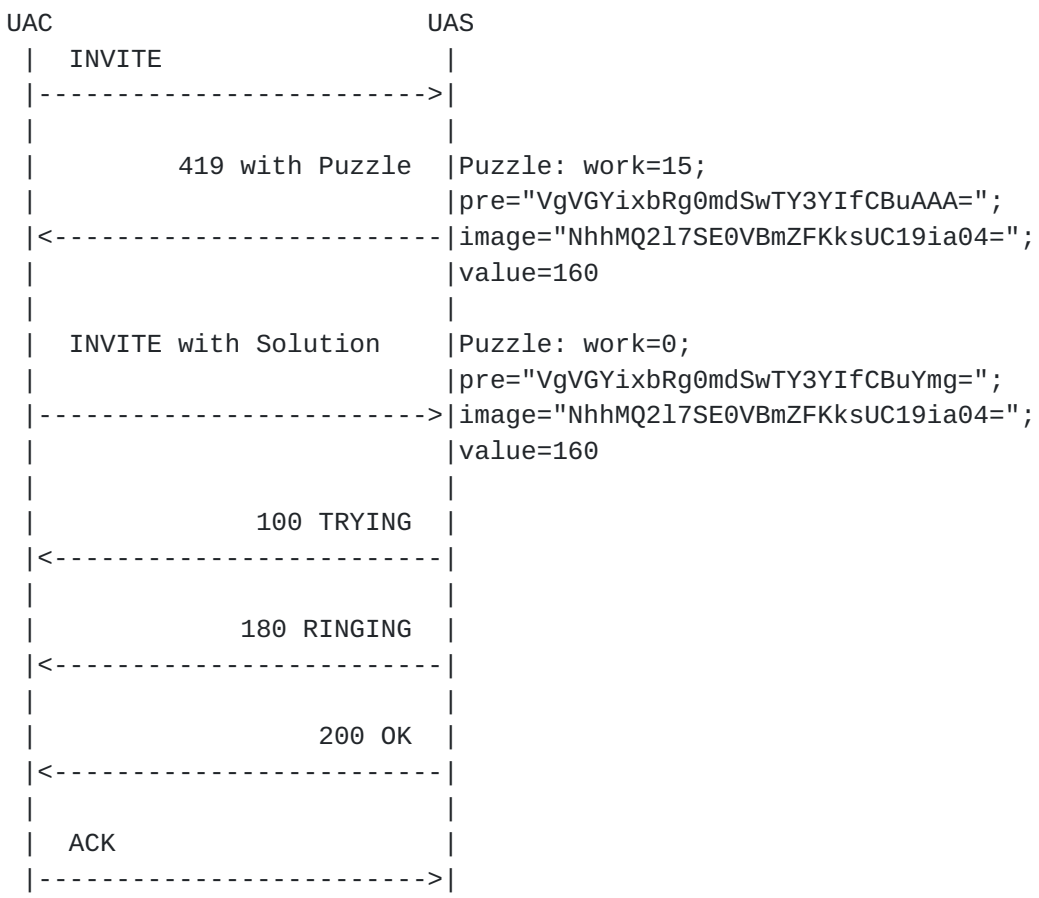
The "Puzzle header" field of Alice's answer has the form:

```
Puzzle: work=0; pre="VgVGYixbRg0mdSwTY3YIfCBuYmg=";
        image="NhhMQ2l7SE0VBmZFKksUC19ia04="; value=160
```

When Bob receives this message, he can compare the "preImage'' value that he has used to build the Puzzle with the "preImage'' value that he can read in the "Puzzle header" field of Alice's answer.

If these two values are the same, he can consider that Alice has spent time to solve the puzzle, and that she passed the challenge. So he can accept her initial INVITE request!

In conclusion, we saw different messages going between Alice and Bob. Here is a summary of these messages, and the content of the "Puzzle header" field" for messages that use this header:



7. Syntax

The Puzzle header field carries the puzzle and solution information. It has a parameter called 'work' that has the number of bits of the pre-image that have been set to zero for this puzzle. It has a parameter called 'pre' that carries the pre-image string base64 encoded, and a parameter called 'image' that carries the image string base64 encoded. In addition there is a parameter called 'value' that indicates how many bits of the resulting hash will match the 'image' string. The base64 encoding is done as described in RFC 3548 [1].

When the header field value is carrying a solution to a puzzle, the work parameter will be set to zero.

Example:

```
Puzzle: work=10; pre="XPokF1n0+NG6iwRcYzeXuETrtDo=";
        image="XPokF1n0+NG6iwRcYzeXuETrtDo="; value=160
```

The ABNF for the header is:

```
Puzzle       = "Puzzle" HCOLON puzzle-parm *(COMMA puzzle-param)

puzzle-param =  puzzle-bits SEMI puzzle-pre SEMI puzzle-image
                SEMI puzzle-value *( SEMI generic-param )

puzzle-work  = "work=" 1*DIGIT
puzzle-value = "value=" 1*DIGIT
puzzle-pre   = "pre=" quoted-string
puzzle-image = "image=" quoted-string
```

This document updates the dreaded Table 2 of RFC 3261 to be:

Header field	where	proxy	ACK	BYE	CAN	INV	OPT	REG
Puzzle		amr	o	o	-	o	o	o

SUB	NOT	REF	INF	UPD	PRA
o	o	o	o	o	o

8. Open Issues and To Do Items

The current mechanism has poor interaction with the HERD forking problem. If several endpoints sent a 419, the proxy would need to aggregate the results and add something like the realm to the challenges to keep them sorted out. Need to add this in next revision. In many cases the solution would work out better if the proxy that was doing the forking applied the policy and did the 419 before forking. This approach has the usual HERD problem that if some UAs do a 419, and some UAs don't, the request will only reach the UAs that don't do the 419.

What is the transition model here. Not everything is going implement this right away: how to differentiate non-implementers from purposeful non-implementors? Is it realistic to just say no to non-implementors? Especially when you consider that as a PANT replacement there is a general expectation of call success rather than call failure (unlike, say, IN).

Need to add a parameter to the puzzle that specifies which hash algorithm to use.

Update what happens when a UAS receives a puzzle with an incorrect solution.

9. Security Considerations

Still TBD.

The concatenation with "z9hG4bK" is done so that this mechanism cannot be used as a distributed computation to reverse arbitrary hash values, as that would present a security risk for other hash based security schemes.

TODO - Advice on selecting the size of 'work'.

There may be ways of using this to effectively perform DOS attacks on system by asking them to solve many puzzles. Need to consider these attacks and make sure that puzzles are only needed to be solved by the device the initiated the requested action.

TODO - discuss rational for design of the puzzle

Some applications like "reverse 911" (community emergency alert systems that notify all the UAs in a given group or geographic region) would be severely hampered by being challenged with puzzles. These systems will require some other authorization system and SHOULD NOT use this approach.

10. IANA Considerations

This specification registers a new header and a new response code. IANA is requested to make the following updates in the registry at: http:///www.iana.org/assignments/sip-parameters

10.1. Puzzle Header

Add the following entry to the header sub-registry.

Header Name	compact	Reference
Puzzle		[RFC-XXXX]

10.2. 419 Response

Add the following entry to the response code sub-registry under the "Request Failure 4xx" heading.

419	Puzzle Required	[RFC-XXXX]

11. Acknowledgments

The test vectors in Appendix A and the example text were provided by Geoffrey Dawirs. This approach was motivated by [5]. Michael Thomas has pointed out some significant problems with this idea and perhaps the whole approach. I have tried to paraphrase some of his concerns into the discussion in this document. Henning Schulzrinne pointed out the important reverse 911 consideration.

12. Appendix A: Test Vectors

The test vectors were run with 17 different levels of work ranging from 1 to 17. For each level of work three puzzles are created and solved and are labeled TEST 1,2, or 3.

```
Level 1 / 17 --- TEST 1 / 3
Random string: gdxvmcdcovpejyrabkxgyqciaxu
preImage (base64-encoded): dA0CRElXfnIcdntrKyxmK29HKAA=

image (base64-encoded): VjRfVFoFLzFRICRyMS0pOV9cNDc=

work=1
value=160
preImage after zero (base64-encoded): dA0CRElXfnIcdntrKyxmK29HKAA=

solution (base64-encoded): dA0CRElXfnIcdntrKyxmK29HKAA=

level 1 / 17 --- TEST 2 / 3
Random string: vdbfkvjtsutahbzgejiqnii
preImage (base64-encoded): Ek8iQC9oLEslA1ljQGsfNGx4PRc=

image (base64-encoded): SHddMGEpcjUecGUNbwtFE2Jpfls=

work=1
value=160
preImage after zero (base64-encoded): Ek8iQC9oLEslA1ljQGsfNGx4PRY=

solution (base64-encoded): Ek8iQC9oLEslA1ljQGsfNGx4PRc=

level 1 / 17 --- TEST 3 / 3
Random string: gknevljqdowdehqixzrbnvjjavcco
preImage (base64-encoded): AE0hWXEnJUNtbAUgSkc4VVtEQW8=

image (base64-encoded): HFpTEh4WMXpBViRmVzEmXDoPeGg=

work=1
```

```
   value=160
   preImage after zero (base64-encoded): AE0hWXEnJUNtbAUgSkc4VVtEQW4=

   solution (base64-encoded): AE0hWXEnJUNtbAUgSkc4VVtEQW8=

   level 2 / 17 --- TEST 1 / 3
   Random string: gcpcerwaprqwfdrpzaqbpgkxbfglw
   preImage (base64-encoded): UnE7YXFOLGckXDJ6HTBaYH8tMXs=

   image (base64-encoded): cwRPKT4yBlUnXEw5dRBiHjBiECU=

   work=2
   value=160
   preImage after zero (base64-encoded): UnE7YXFOLGckXDJ6HTBaYH8tMXg=

   solution (base64-encoded): UnE7YXFOLGckXDJ6HTBaYH8tMXs=

   level 2 / 17 --- TEST 2 / 3
   Random string: zeyszfhbfasaedjztugfojwgijdc
   preImage (base64-encoded): PDonUmxvPAQcLRwVClppQgZRQRo=

   image (base64-encoded): OQ4+XnQSAUMDc20PeQMaYjxKME0=

   work=2
   value=160
   preImage after zero (base64-encoded): PDonUmxvPAQcLRwVClppQgZRQRg=

   solution (base64-encoded): PDonUmxvPAQcLRwVClppQgZRQRo=

   level 2 / 17 --- TEST 3 / 3
   Random string: wmvpjcjnwkchtyquvsawz
   preImage (base64-encoded): FCBUD0wmVRFVATxxI3NAXhtZAVk=

   image (base64-encoded): ZHQ8OxY8DHdgYQ4WNmMkLUdyTA4=

   work=2
   value=160
   preImage after zero (base64-encoded): FCBUD0wmVRFVATxxI3NAXhtZAVg=

   solution (base64-encoded): FCBUD0wmVRFVATxxI3NAXhtZAVk=

   level 3 / 17 --- TEST 1 / 3
   Random string: mwqoaoazhakqsxmujkhjjezkhwvgfz
   preImage (base64-encoded): Qll6REckHzB9VRQZCHFuZW4UPVM=

   image (base64-encoded): DUs/bRUcfDViZgtzP1xDKQZ0a18=

   work=3
```

```
   value=160
   preImage after zero (base64-encoded): Qll6REckHzB9VRQZCHFuZW4UPVA=

   solution (base64-encoded): Qll6REckHzB9VRQZCHFuZW4UPVM=

   level 3 / 17 --- TEST 2 / 3
   Random string: oepdjqztwxixkpxyagdhykvwwc
   preImage (base64-encoded): LBZZekRMSU95AE15S15kJU4UJVM=

   image (base64-encoded): M10xHBI0Ok1gOX80DyUQFSAaNw0=

   work=3
   value=160
   preImage after zero (base64-encoded): LBZZekRMSU95AE15S15kJU4UJVA=

   solution (base64-encoded): LBZZekRMSU95AE15S15kJU4UJVM=

   level 3 / 17 --- TEST 3 / 3
   Random string: slczwdgfazorhwoasymaoepvf
   preImage (base64-encoded): JRoPcHV5CgMvJ1R7bjYqfyVoFig=

   image (base64-encoded): PHYgIEsrDRc8EEQBUjAbcBYEHAU=

   work=3
   value=160
   preImage after zero (base64-encoded): JRoPcHV5CgMvJ1R7bjYqfyVoFig=

   solution (base64-encoded): JRoPcHV5CgMvJ1R7bjYqfyVoFig=

   level 4 / 17 --- TEST 1 / 3
   Random string: qgksjhphjligddrspyrc
   preImage (base64-encoded): U0FaV00YIHx9CyBbO0FSTiw/bXQ=

   image (base64-encoded): DyV2EwktcWgQPEA+XmwHOT0UYE0=

   work=4
   value=160
   preImage after zero (base64-encoded): U0FaV00YIHx9CyBbO0FSTiw/bXA=

   solution (base64-encoded): U0FaV00YIHx9CyBbO0FSTiw/bXQ=

   level 4 / 17 --- TEST 2 / 3
   Random string: buwdwqyxzhejgbbkbjeqaipaqsbrp
   preImage (base64-encoded): GwY1Mm02El8kKE9ve15MbllEKkw=

   image (base64-encoded): WE8IJksYW1oFeBx6WTVaLQR/dhU=

   work=4
```

```
   value=160
   preImage after zero (base64-encoded): GwY1Mm02El8kKE9ve15MbllEKkA=

   solution (base64-encoded): GwY1Mm02El8kKE9ve15MbllEKkw=

   level 4 / 17 --- TEST 3 / 3
   Random string: hdxbcrbgodseongcjeownmovmqzny
   preImage (base64-encoded): UwgDYCgRETIGKhtsZBcTNWoGKyg=

   image (base64-encoded): fiU0fHsBOFxDFlxnC1FgD2I0HCA=

   work=4
   value=160
   preImage after zero (base64-encoded): UwgDYCgRETIGKhtsZBcTNWoGKyA=

   solution (base64-encoded): UwgDYCgRETIGKhtsZBcTNWoGKyg=

   level 5 / 17 --- TEST 1 / 3
   Random string: anbkdsxaubmulkktrgupv
   preImage (base64-encoded): AngUeyYuTGQkL3lNHVNIelslJT8=

   image (base64-encoded): YgQpFS0Wb25SHRtGPR91Un9VUXM=

   work=5
   value=160
   preImage after zero (base64-encoded): AngUeyYuTGQkL3lNHVNIelslJSA=

   solution (base64-encoded): AngUeyYuTGQkL3lNHVNIelslJT8=

   level 5 / 17 --- TEST 2 / 3
   Random string: anrwzqdsobsqibrwfmquabctqna
   preImage (base64-encoded): aX9wTT1rIVJwG25WMh5MZzsedUQ=

   image (base64-encoded): QlRNbwQrGVVmPDEPZUASTmUVa2s=

   work=5
   value=160
   preImage after zero (base64-encoded): aX9wTT1rIVJwG25WMh5MZzsedUA=

   solution (base64-encoded): aX9wTT1rIVJwG25WMh5MZzsedUQ=

   level 5 / 17 --- TEST 3 / 3
   Random string: gjnwilitpznhmaidkyloeg
   preImage (base64-encoded): Mi5eChcsKCUoTVhyfBgQSydhfXo=


   image (base64-encoded): QV1GCh8mGW1WOWlMdgAERmEzH2M=
```

```
   work=5
   value=160
   preImage after zero (base64-encoded): Mi5eChcsKCUoTVhyfBgQSydhfWA=

   solution (base64-encoded): Mi5eChcsKCUoTVhyfBgQSydhfXo=

   level 6 / 17 --- TEST 1 / 3
   Random string: hrvgonrrmphifwrfqcho
   preImage (base64-encoded): AQ9TTCNgUnU6eDh1CDk8LlYcIE8=

   image (base64-encoded): Zi5Fcn9MJiFGaXwqbw10cXEXSQA=

   work=6
   value=160
   preImage after zero (base64-encoded): AQ9TTCNgUnU6eDh1CDk8LlYcIEA=

   solution (base64-encoded): AQ9TTCNgUnU6eDh1CDk8LlYcIE8=

   level 6 / 17 --- TEST 2 / 3
   Random string: gcxmhamuasvxfzfljclkuslv
   preImage (base64-encoded): L3h/YCUDFTVCUndTbB1bDzVWGik=

   image (base64-encoded): Y1QmWndFXG5fPHNPHE10aDliDBY=

   work=6
   value=160
   preImage after zero (base64-encoded): L3h/YCUDFTVCUndTbB1bDzVWGgA=

   solution (base64-encoded): L3h/YCUDFTVCUndTbB1bDzVWGik=

   level 6 / 17 --- TEST 3 / 3
   Random string: azkroxnsxoasmlalcrjgsfy
   preImage (base64-encoded): bT09cypeJlMkPGpgRwQMfVYsAS4=

   image (base64-encoded): NRxzQjh7ClFsWWp5IhoyFxN6aGw=

   work=6
   value=160
   preImage after zero (base64-encoded): bT09cypeJlMkPGpgRwQMfVYsAQA=

   solution (base64-encoded): bT09cypeJlMkPGpgRwQMfVYsAS4=

   level 7 / 17 --- TEST 1 / 3
   Random string: afxoapxwehorxjxczgyokgfllwtmwv
   preImage (base64-encoded): PEA1J0l9FyVpFR03ITVOciY5TR4=

   image (base64-encoded): CBspElUeEV4zMEsqHU86PGN0N10=
```

```
   work=7
   value=160
   preImage after zero (base64-encoded): PEA1J0l9FyVpFR03ITVOciY5TQA=

   solution (base64-encoded): PEA1J0l9FyVpFR03ITVOciY5TR4=

   level 7 / 17 --- TEST 2 / 3
   Random string: kbycttlwmbuwkgijafwehmxwqoc
   preImage (base64-encoded): Elx9ZyoGVSUnNAJRSkwdS2o/KiU=

   image (base64-encoded): KXELayx+CBRvCi8+DWdiFn8rYXs=

   work=7
   value=160
   preImage after zero (base64-encoded): Elx9ZyoGVSUnNAJRSkwdS2o/KgA=

   solution (base64-encoded): Elx9ZyoGVSUnNAJRSkwdS2o/KiU=

   level 7 / 17 --- TEST 3 / 3
   Random string: xdabcoapunmukhnpszrqtb
   preImage (base64-encoded): IwYALWI4MTRPOwYyDFxLVU8kYjk=

   image (base64-encoded): CSYOZxU3cQZKRSNrawEIUCMAPAc=

   work=7
   value=160
   preImage after zero (base64-encoded): IwYALWI4MTRPOwYyDFxLVU8kYgA=

   solution (base64-encoded): IwYALWI4MTRPOwYyDFxLVU8kYjk=

   level 8 / 17 --- TEST 1 / 3
   Random string: euztuexgapbmvwzhdnyj
   preImage (base64-encoded): Xm8xcmdkeW02TWNvBH0XXmU/E0g=

   image (base64-encoded): AzgIeENNC2UKbx8cdQs7ZBwPLxA=

   work=8
   value=160
   preImage after zero (base64-encoded): Xm8xcmdkeW02TWNvBH0XXmU/EwA=

   solution (base64-encoded): Xm8xcmdkeW02TWNvBH0XXmU/E0g=

   level 8 / 17 --- TEST 2 / 3
   Random string: pgiksjjmwdhfvzqtzmocef
   preImage (base64-encoded): OhU7BF4gCnQzay5HCl8GGRF7ci8=

   image (base64-encoded): bjovP2MCQDN9OgZzZ28WTmgZDVY=
```

```
   work=8
   value=160
   preImage after zero (base64-encoded): OhU7BF4gCnQzay5HCl8GGRF7cgA=

   solution (base64-encoded): OhU7BF4gCnQzay5HCl8GGRF7ci8=

   level 8 / 17 --- TEST 3 / 3
   Random string: wnbfdqnpgfaccosflhufgcud
   preImage (base64-encoded): eWw4WDk6T0EkTERJVxJqbngQU04=

   image (base64-encoded): dUg5JxssUnFZEiVyREMeGxd9Px8=

   work=8
   value=160
   preImage after zero (base64-encoded): eWw4WDk6T0EkTERJVxJqbngQUwA=

   solution (base64-encoded): eWw4WDk6T0EkTERJVxJqbngQU04=

   level 9 / 17 --- TEST 1 / 3
   Random string: mzihxmckhemkrrdxkvhrjo
   preImage (base64-encoded): GD0cEWh3PiNjGF4BVn8JREJsJGU=

   image (base64-encoded): DT0MBy01WBxBBRITBksEbXkoOH4=

   work=9
   value=160
   preImage after zero (base64-encoded): GD0cEWh3PiNjGF4BVn8JREJsJAA=

   solution (base64-encoded): GD0cEWh3PiNjGF4BVn8JREJsJGU=

   level 9 / 17 --- TEST 2 / 3
   Random string: cvzkodyvqjhotainrlqvwyuyi
   preImage (base64-encoded): GDoYYhIwC0AUETRMGG5TdiA3IU8=

   image (base64-encoded): P1sTfWYbLnA8PBREUkEVXFJLVT4=

   work=9
   value=160
   preImage after zero (base64-encoded): GDoYYhIwC0AUETRMGG5TdiA3IAA=

   solution (base64-encoded): GDoYYhIwC0AUETRMGG5TdiA3IU8=

   level 9 / 17 --- TEST 3 / 3
   Random string: zgjxoxqgohakmrxgqtrhhlyjso
   preImage (base64-encoded): DWAFYDIRFRMCYE8zS0ppRx40cjM=

   image (base64-encoded): ADZtDHsif3QMQT4ie1IPeU1LeEA=
```

```
   work=9
   value=160
   preImage after zero (base64-encoded): DWAFYDIRFRMCYE8zS0ppRx40cgA=

   solution (base64-encoded): DWAFYDIRFRMCYE8zS0ppRx40cjM=

   level 10 / 17 --- TEST 1 / 3
   Random string: zkirsotmjnjxevjgefwhnojuop
   preImage (base64-encoded): UgI6BW1VY3FBWwJMDzwrbndeHHQ=

   image (base64-encoded): KW90XUBWcFY5Fw8FJSoJY1AvUH4=

   work=10
   value=160
   preImage after zero (base64-encoded): UgI6BW1VY3FBWwJMDzwrbndeHAA=

   solution (base64-encoded): UgI6BW1VY3FBWwJMDzwrbndeHHQ=

   level 10 / 17 --- TEST 2 / 3
   Random string: ujgqqxhqvbuvexjqsvaui
   preImage (base64-encoded): P3xQZVBgcydHW3clE1BndiBaeSk=

   image (base64-encoded): DGRobH0Qe2xoKFdFYBsIUSNsfnM=

   work=10
   value=160
   preImage after zero (base64-encoded): P3xQZVBgcydHW3clE1BndiBaeAA=

   solution (base64-encoded): P3xQZVBgcydHW3clE1BndiBaeSk=

   level 10 / 17 --- TEST 3 / 3
   Random string: hvrumrohnozihssygzsgppfp
   preImage (base64-encoded): eC5eAUIvSlVYDwQJFy04PwcmdS8=

   image (base64-encoded): G3Z4ZQ9lHSJ9Vx4rRVpiLjlMTFA=

   work=10
   value=160
   preImage after zero (base64-encoded): eC5eAUIvSlVYDwQJFy04PwcmdAA=

   solution (base64-encoded): eC5eAUIvSlVYDwQJFy04PwcmdS8=

   level 11 / 17 --- TEST 1 / 3
   Random string: tovdorusdqfovtifyunvpy
   preImage (base64-encoded): H2F9HVUgAjINaGANCmdHWVEEZ04=

   image (base64-encoded): LE5qbTIcYxRHQCN9NBRPZ1gDGxE=
```

```
   work=11
   value=160
   preImage after zero (base64-encoded): H2F9HVUgAjINaGANCmdHWVEEYAA=

   solution (base64-encoded): H2F9HVUgAjINaGANCmdHWVEEZ04=

   level 11 / 17 --- TEST 2 / 3
   Random string: umfktxivqlkczwoceidez
   preImage (base64-encoded): elwLLA4PFjUZGGN6PEZwRwUJOVM=

   image (base64-encoded): K1d2WR5MXjFRCC04YgIgBl5WAEU=

   work=11
   value=160
   preImage after zero (base64-encoded): elwLLA4PFjUZGGN6PEZwRwUJOAA=

   solution (base64-encoded): elwLLA4PFjUZGGN6PEZwRwUJOVM=

   level 11 / 17 --- TEST 3 / 3
   Random string: nxlbmswquztwldpwokmnqnbcrxkpoo
   preImage (base64-encoded): WGgzajJFQx5nVFFsGhURFgxXCk8=

   image (base64-encoded): BXQXNH05B3NLLQhlFS0DV3cXUgU=

   work=11
   value=160
   preImage after zero (base64-encoded): WGgzajJFQx5nVFFsGhURFgxXCAA=

   solution (base64-encoded): WGgzajJFQx5nVFFsGhURFgxXCk8=

   level 12 / 17 --- TEST 1 / 3
   Random string: tnqqshsvqxwdattkguseouu
   preImage (base64-encoded): amUaNAZCFDd6LkRkNAkMZ118FXc=

   image (base64-encoded): KA4fBQ1Ya0Q1UVZSJWcjG1cWIUY=

   work=12
   value=160
   preImage after zero (base64-encoded): amUaNAZCFDd6LkRkNAkMZ118EAA=

   solution (base64-encoded): amUaNAZCFDd6LkRkNAkMZ118FXc=

   level 12 / 17 --- TEST 2 / 3
   Random string: pjzuzvhidheobinckecwlvfl
   preImage (base64-encoded): NAliEBMPFiBYOlwPdGldQgEMZkU=

   image (base64-encoded): Kgo9IG40RX1MMVELTSMGT1hUVy4=
```

```
   work=12
   value=160
   preImage after zero (base64-encoded): NAliEBMPFiBYOlwPdGldQgEMYAA=

   solution (base64-encoded): NAliEBMPFiBYOlwPdGldQgEMZkU=

   level 12 / 17 --- TEST 3 / 3
   Random string: ywctzidwhouvwpzjjvqrkhvlf
   preImage (base64-encoded): bHJeC39oe00RRS1OKyICFnMdO2I=

   image (base64-encoded): BQJHBiBqVVJ2e29gYy0LaRsLHmM=

   work=12
   value=160
   preImage after zero (base64-encoded): bHJeC39oe00RRS1OKyICFnMdMAA=

   solution (base64-encoded): bHJeC39oe00RRS1OKyICFnMdO2I=

   level 13 / 17 --- TEST 1 / 3
   Random string: arnfaqqzzxqujmpirwfsiktor
   preImage (base64-encoded): c2gsSEBzH11yGmlCMBoqKBQeImI=

   image (base64-encoded): OUU/Jx8aRANPamhZRT8tOW8qAS8=

   work=13
   value=160
   preImage after zero (base64-encoded): c2gsSEBzH11yGmlCMBoqKBQeIAA=

   solution (base64-encoded): c2gsSEBzH11yGmlCMBoqKBQeImI=

   level 13 / 17 --- TEST 2 / 3
   Random string: lsdnchahbpppnphzmkkcsuoj
   preImage (base64-encoded): c1EsT3lOGjgFIWZac0sTKXEKESc=

   image (base64-encoded): ORV5eSVTIV5qIV5PS3lTM0sdIWU=

   work=13
   value=160
   preImage after zero (base64-encoded): c1EsT3lOGjgFIWZac0sTKXEKAAA=

   solution (base64-encoded): c1EsT3lOGjgFIWZac0sTKXEKESc=

   level 13 / 17 --- TEST 3 / 3
   Random string: mqlqeaweiupmzjtldlkskbqtmlj
   preImage (base64-encoded): HAoxPiE1K3xwbGEIdH5TcwxrOwA=

   image (base64-encoded): VVdEekMjRgR/CTZtGFBIIDJaZg4=
```

```
   work=13
   value=160
   preImage after zero (base64-encoded): HAoxPiE1K3xwbGEIdH5TcwxrIAA=

   solution (base64-encoded): HAoxPiE1K3xwbGEIdH5TcwxrOwA=

   level 14 / 17 --- TEST 1 / 3
   Random string: bjnbntkkltdlhgvsxptpjq
   preImage (base64-encoded): FhcYHQwdJ0giX1EOOBE4S0xHZx4=

   image (base64-encoded): MXtOLmtoRWJiIVlKHx4vCDwwMEk=

   work=14
   value=160
   preImage after zero (base64-encoded): FhcYHQwdJ0giX1EOOBE4S0xHQAA=

   solution (base64-encoded): FhcYHQwdJ0giX1EOOBE4S0xHZx4=

   level 14 / 17 --- TEST 2 / 3
   Random string: ukzsazeyuxczkfxibrerffk
   preImage (base64-encoded): NFt0WVkjHHhUeDIjfEp0ajJbPXg=

   image (base64-encoded): LF9Hcy9nIW9jdjVhPX8mIU5SXx4=

   work=14
   value=160
   preImage after zero (base64-encoded): NFt0WVkjHHhUeDIjfEp0ajJbAAA=

   solution (base64-encoded): NFt0WVkjHHhUeDIjfEp0ajJbPXg=

   level 14 / 17 --- TEST 3 / 3
   Random string: yoagiwgujbfongpncloonmlaztnb
   preImage (base64-encoded): HC4maTtxIFcUc39QdTgOHl8tAAQ=

   image (base64-encoded): HGkuLj56JTVgeAJ8D1QnVG0AV3Q=

   work=14
   value=160
   preImage after zero (base64-encoded): HC4maTtxIFcUc39QdTgOHl8tAAA=

   solution (base64-encoded): HC4maTtxIFcUc39QdTgOHl8tAAQ=

   level 15 / 17 --- TEST 1 / 3
   Random string: ksuvudvnlwebrotnrszczjyvyrf

   preImage (base64-encoded): XHReelpBHk89YxYlWjZ7ejssflg=

   image (base64-encoded): bSJOAUdUYnpPWm4uLyUNSzkVaQM=
```

```
   work=15
   value=160
   preImage after zero (base64-encoded): XHReelpBHk89YxYlWjZ7ejssAAA=

   solution (base64-encoded): XHReelpBHk89YxYlWjZ7ejssflg=

   level 15 / 17 --- TEST 2 / 3
   Random string: qmgbajbocksfunnusggf
   preImage (base64-encoded): H302EGtQfVFrIUI3AlVDX05dJ00=


   image (base64-encoded): GnlOakQVBHMKUUYidGUfSkZBWUw=

   work=15
   value=160
   preImage after zero (base64-encoded): H302EGtQfVFrIUI3AlVDX05dAAA=

   solution (base64-encoded): H302EGtQfVFrIUI3AlVDX05dJ00=

   level 15 / 17 --- TEST 3 / 3
   Random string: whmvlwzipmcarouqfqckr
   preImage (base64-encoded): VEhCREFNLwhMPiYlYQYMYBgpD3c=

   image (base64-encoded): FSBYHmlfdTkXIzRmXmAtS21SQ2Q=

   work=15
   value=160
   preImage after zero (base64-encoded): VEhCREFNLwhMPiYlYQYMYBgpAAA=

   solution (base64-encoded): VEhCREFNLwhMPiYlYQYMYBgpD3c=

   level 16 / 17 --- TEST 1 / 3
   Random string: iallnhuydqzoujkjuumu
   preImage (base64-encoded): DlA/dCB3IFlcFmIPKQQTCAB+eGY=

   image (base64-encoded): PRcQIjNKUEFXPwhoW11KXygYJxY=

   work=16
   value=160
   preImage after zero (base64-encoded): DlA/dCB3IFlcFmIPKQQTCAB+AAA=

   solution (base64-encoded): DlA/dCB3IFlcFmIPKQQTCAB+eGY=

   level 16 / 17 --- TEST 2 / 3
   Random string: cucykaaltpxnpdfbkwiakdlvjt
   preImage (base64-encoded): ABx7YQd/RG1NG1JyCAliFAdOfgE=

   image (base64-encoded): Jhc8Y3ZwMTNZM0FHFnApRjszXDc=
```

```
   work=16
   value=160
   preImage after zero (base64-encoded): ABx7YQd/RG1NG1JyCAliFAdOAAA=

   solution (base64-encoded): ABx7YQd/RG1NG1JyCAliFAdOfgE=

   level 16 / 17 --- TEST 3 / 3
   Random string: awlssoylwkjdldygglgrn
   preImage (base64-encoded): ZyE7fD9wBRlQGgNZSnFhLlMSOAs=

   image (base64-encoded): WXcTQ3gRX1xIOkUbAQRQUi5bKy8=

   work=16
   value=160
   preImage after zero (base64-encoded): ZyE7fD9wBRlQGgNZSnFhLlMSAAA=

   solution (base64-encoded): ZyE7fD9wBRlQGgNZSnFhLlMSOAs=

   level 17 / 17 --- TEST 1 / 3
   Random string: ctwlrtmezmjgjpfmeuzeusnzrbk
   preImage (base64-encoded): KEEYBVBfFjZeGE9lZS9qbmJuAyU=

   image (base64-encoded): IXpbLnMLBTFXfkwdaxYvdEckQ1c=

   work=17
   value=160
   preImage after zero (base64-encoded): KEEYBVBfFjZeGE9lZS9qbmJuAAA=

   solution (base64-encoded): KEEYBVBfFjZeGE9lZS9qbmJuAyU=

   level 17 / 17 --- TEST 2 / 3
   Random string: vggwtxejthimazyoxkfsyeawiber
   preImage (base64-encoded): K2RqCBcqcXMBKlNJFncURmhEOiI=

   image (base64-encoded): dTFDAwokR1keFwlMIlVsE3sKay8=

   work=17
   value=160
   preImage after zero (base64-encoded): K2RqCBcqcXMBKlNJFncURmhEAAA=

   solution (base64-encoded): K2RqCBcqcXMBKlNJFncURmhEOiI=

   level 17 / 17 --- TEST 3 / 3
   Random string: aocgnkxbjleairqeossghdkoix
   preImage (base64-encoded): cCkTaHpsS1RUEmd8MVwEAjg5G1I=

   image (base64-encoded): djscbnkvBAFQAjccYkU6F21DMCM=
```

```
work=17
value=160
preImage after zero (base64-encoded): cCkTaHpsS1RUEmd8MVwEAjg4AAA=

solution (base64-encoded): cCkTaHpsS1RUEmd8MVwEAjg5G1I=
```

13. References

13.1. Normative References

[1] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 3548, July 2003.

[2] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[3] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[4] Eastlake, D. and P. Jones, "US Secure Hash Algorithm 1 (SHA1)", RFC 3174, September 2001.

13.2. Informational References

[5] Black, A., "http://www.hashcash.org/", February 2005.

[6] Peterson, J. and C. Jennings, "Enhancements for Authenticated Identity Management in the Session Initiation Protocol (SIP)", RFC 4474, August 2006.

[7] Rosenberg, J., "The Session Initiation Protocol (SIP) and Spam", draft-ietf-sipping-spam-04 (work in progress).

Author's Address

Cullen Jennings
Cisco Systems
170 West Tasman Drive
MS: SJC-21/2
San Jose, CA 95134
USA

Phone: +1 408 421 9990
Email: fluffy@cisco.com

Full Copyright Statement

Intellectual Property

Acknowledgment

Funding for the RFC Editor function is provided by the IETF Administrative Support Activity (IASA).