                  Support for Multiple Hash Algorithms in
               Cryptographically Generated Addresses (CGAs)

Status of This Memo

   This document specifies an Internet standards track protocol for the
   Internet community, and requests discussion and suggestions for
   improvements. Please refer to the current edition of the "Internet
   Official Protocol Standards" (STD 1) for the standardization state
   and status of this protocol. Distribution of this memo is unlimited.

Copyright Notice

Abstract

   This document analyzes the implications of recent attacks on commonly
   used hash functions on Cryptographically Generated Addresses (CGAs)
   and updates the CGA specification to support multiple hash
   algorithms.

Table of Contents

1. Introduction

   Recent attacks to currently used hash functions have motivated a
   considerable amount of concern in the Internet community. The
   recommended approach [6] [10] to deal with this issue is first to
   analyze the impact of these attacks on the different Internet
   protocols that use hash functions and second to make sure that the
   different Internet protocols that use hash functions are capable of
   migrating to an alternative (more secure) hash function without a
   major disruption in the Internet operation.

   This document performs such analysis for the Cryptographically
   Generated Addresses (CGAs) defined in [2]. The first conclusion of
   the analysis is that the security of the protocols using CGAs is not
   affected by the recently available attacks against hash functions.
   The second conclusion of the analysis is that the hash function used
   is hard coded in the CGA specification. This document updates the
   CGA specification [2] to enable the support of alternative hash
   functions. In order to do so, this document creates a new registry
   managed by IANA to register the different hash algorithms used in
   CGAs.

2. Terminology

   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 [1].

3. Impact of Collision Attacks in CGAs

   Recent advances in cryptography have resulted in simplified attacks
   against the collision-free property of certain commonly used hash
   functions [6] [10], including SHA-1 that is the hash function used by
   CGAs [2]. The result is that it is possible to obtain two messages,
   M1 and M2, that have the same hash value with much less than $2^{(L/2)}$
   attempts. We will next analyze the impact of such attacks in the
   currently proposed usages of CGAs.

   As we understand it, the attacks against the collision-free property
   of a hash function mostly challenge the application of such hash
   functions, for the provision of non-repudiation capabilities. This
   is because an attacker would be capable to create two different
   messages that result in the same hash value and it can then present
   any of the messages interchangeably (for example after one of them
   has been signed by the other party involved in the transaction).
   However, it must be noted that both messages must be generated by the
   same party.

As far as we understand, current usages of CGAs does not include the provision of non-repudiation capabilities, so attacks against the collision-free property of the hash function do not enable any useful attack against CGA-based protocols.

Current usages of the CGAs are basically oriented to prove the ownership of a CGA and then bind it to alternative addresses that can be used to reach the original CGA. This type of application of the CGA include:

o The application of CGAs to protect the shim6 protocol [7]. In this case, CGAs are used as identifiers for the established communications. CGA features are used to prove that the owner of the identifier is the one that is providing the alternative addresses that can be used to reach the initial identifier. This is achieved by signing the list of alternative addresses available in the multihomed host with the private key of the CGA.

o The application of CGAs to secure the IPv6 mobility support protocol [8] as proposed in [9]. In this case, the CGAs are used as Home Addresses and they are used to prove that the owner of the Home Address is the one creating the binding with the new Care-off Address. Similarly to the previous case, this is achieved by signing the Binding Update message carrying the Care-off Address with the private key of the CGA.

o The application of CGA to Secure Neighbour Discovery [4]. In this case, the CGA features are used to prove the address ownership, so that it is possible to verify that the owner of the IP address is the one that is providing the layer 2 address information. This is achieved by signing the layer 2 address information with the private key of the CGA.

Essentially, all the current applications of CGAs rely on CGAs to protect a communication between two peers from third party attacks and not to provide protection from the peer itself. Attacks against the collision-free property of the hash functions suppose that one of the parties is generating two messages with the same hash value in order to launch an attack against its communicating peer. Since CGAs are not currently used to providing this type of protection, it is then natural that no additional attacks are enabled by a weaker collision resistance of the hash function.

4. Options for Multiple Hash Algorithm Support in CGAs

CGAs, as currently defined in [2], are intrinsically bound to the SHA-1 hash algorithm and no other hash function is supported.

Even though the attacks against the collision-free property of the
hash functions do not result in new vulnerabilities in the current
applications of CGAs, it seems wise to enable multiple hash function
support in CGAs. This is mainly for two reasons: first, potential
future applications of the CGA technology may be susceptible to
attacks against the collision-free property of SHA-1. Supporting
alternative hash functions would allow applications that have
stricter requirements on the collision-free property to use CGAs.
Second, one lesson learned from the recent attacks against hash
functions is that it is possible that one day we need to start using
alternative hash functions because of successful attacks against
other properties of the commonly used hash functions. Therefore, it
seems wise to modify protocols in general and the CGAs in particular
to support this transition to alternative hash functions as easy as
possible.

4.1. Where to Encode the Hash Function?

The next question we need to answer is where to encode the hash
function that is being used. There are several options that can be
considered:

One option would be to include the hash function used as an input to
the hash function. This basically means to create an extension to
the CGA Parameter Data Structure, as defined in [3], that codifies
the hash function used. The problem is that this approach is
vulnerable to bidding down attacks or downgrading attacks as defined
in [10]. This means that even if a strong hash function is used, an
attacker could find a CGA Parameter Data Structure that uses a weaker
function but results in an equal hash value. This happens when the
original hash function H1 and CGA Parameters Data Structure
indicating H1 result in value X, and another hash function H2 and CGA
Parameters Data Structure indicating H2 also result in the same value
X.

In other words, the downgrading attack would work as follows: suppose
that Alice generates a CGA CGA_A using the strong hash function
HashStrong and using a CGA Parameter Data Structure CGA_PDS_A. The
selected hash function HashStrong is encoded as an extension field in
the CGA_PDS_A. Suppose that by using a brute force attack, an
attacker X finds an alternative CGA Parameter Data Structure
CGA_PDS_X whose hash value, by using a weaker hash function, is
CGA_A. At this point, the attacker can pretend to be the owner of
CGA_A and the stronger hash function has not provided additional
protection.

The conclusion from the previous analysis is that the hash function
used in the CGA generation must be encoded in the address itself.

   Since we want to support several hash functions, we will likely need
   at least 2 or 3 bits for this.

   One option would be to use more bits from the hash bits of the
   interface identifier. However, the problem with this approach is
   that the resulting CGA is weaker because less hash information is
   encoded in the address. In addition, since those bits are currently
   used as hash bits, it is impossible to make this approach backward
   compatible with existent implementations.

   Another option would be to use the "u" and the "g" bits to encode
   this information, but this is probably not such a good idea since
   those bits have been honoured so far in all interface identifier
   generation mechanisms, which allow them to be used for the original
   purpose (for instance we can still create a global registry for
   unique interface identifiers). Finally, another option is to encode
   the hash value used in the Sec bits. The Sec bits are used to
   artificially introduce additional difficulty in the CGA generation
   process in order to provide additional protection against brute force
   attacks. The Sec bits have been designed in a way that the lifetime
   of CGAs are extended, when it is feasible to attack 59-bits long hash
   values. However, this is not the case today, so in general CGA will
   have a Sec value of 000. The proposal is to encode in the Sec bits,
   not only information about brute force attack protection but also to
   encode the hash function used to generate the hash. So for instance,
   the Sec value 000 would mean that the hash function used is SHA-1 and
   the 0 bits of hash2 (as defined in RFC 3972) must be 0. Sec value of
   001 could be that the hash function used is SHA-1 and the 16 bits of
   hash2 (as defined in RFC 3972) must be zero. However, the other
   values of Sec could mean that an alternative hash function needs to
   be used and that a certain amount of bits of hash2 must be zero. The
   proposal is not to define any concrete hash function to be used for
   other Sec values, since it is not yet clear that we need to do so nor
   is it clear which hash function should be selected.

   Note that since there are only 8 Sec values, it may be necessary to
   reuse Sec values when we run out of unused Sec values. The scenario
   where such an approach makes sense is where there are some Sec values
   that are no longer being used because the resulting security has
   become weak. In this case, where the usage of the Sec value has long
   been abandoned, it would be possible to reassign the Sec values.
   However, this must be a last resource option, since it may affect
   interoperability. This is because two implementations using
   different meanings of a given Sec value would not be able to
   interoperate properly (i.e., if an old implementation receives a CGA
   generated with the new meaning of the Sec value, it will fail and the
   same for a new implementation receiving a CGA generated with the old
   meaning of the Sec value). In case the approach of reassigning a Sec

value is followed, a long time is required between the deprecation of
the old value and the reassignment in order to prevent
misinterpretation of the value by old implementations.

An erroneous interpretation of a reused Sec value, both on the CGA
owner's side and the CGA verifier's side, would have the following
result, CGA verification would fail in the worst case and both nodes
would have to revert to unprotected IPv6 addresses. This can happen
only with obsolete CGA parameter sets, which would be considered
insecure anyway. In any case, an implementation must not
simultaneously support two different meanings of a Sec value.

5. CGA Generation Procedure

   The SEC registry defined in the IANA considerations section of this
   document contains entries for the different Sec values. Each of
   these entries points to an RFC that defines the CGA generation
   procedure that MUST be used when generating CGAs with the associated
   Sec value.

   It should be noted that the CGA generation procedure may be changed
   by the new procedure not only in terms of the hash function used but
   also in other aspects, e.g., longer Modifier values may be required
   if the number of 0s required in hash2 exceed the currently defined
   bound of 112 bits. The new procedure (which potentially involves a
   longer Modifier value) would be described in the RFC pointed to by
   the corresponding Sec registry entry.

   In addition, the RFC that defines the CGA generation procedure for a
   Sec value MUST explicitly define the minimum key length acceptable
   for CGAs with that Sec value. This is to provide a coherent
   protection both in the hash and the public key techniques.

6. IANA Considerations

   This document defines a new registry entitled "CGA SEC" for the Sec
   field defined in RFC 3972 [2] that has been created and is maintained
   by IANA. The values in this name space are 3-bit unsigned integers.

   Initial values for the CGA Extension Type field are given below;
   future assignments are to be made through Standards Action [5].
   Assignments consist of a name, the value, and the RFC number where
   the CGA generation procedure is defined.

The following initial values are assigned in this document:

```
     Name              | Value |  RFCs
------------------+-------+------------
SHA-1_0hash2bits  |  000  | 3972, 4982
SHA-1_16hash2bits |  001  | 3972, 4982
SHA-1_32hash2bits |  010  | 3972, 4982
```

7. Security Considerations

   This document is about security issues and, in particular, about
   protection against potential attacks against hash functions.

8. Acknowledgements

   Russ Housley, James Kempf, Christian Vogt, Pekka Nikander, and Henrik
   Levkowetz reviewed and provided comments about this document.

   Marcelo Bagnulo worked on this document while visiting Ericsson
   Research Laboratory Nomadiclab.

9. References

9.1. Normative References

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

   [2]   Aura, T., "Cryptographically Generated Addresses (CGA)",
          RFC 3972, March 2005.

   [3]   Bagnulo, M. and J. Arkko, "Cryptographically Generated
          Addresses (CGA) Extension Field Format", RFC 4581,
          October 2006.

   [4]   Arkko, J., Kempf, J., Zill, B., and P. Nikander, "SEcure
          Neighbor Discovery (SEND)", RFC 3971, March 2005.

9.2. Informative References

   [5]   Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA
          Considerations Section in RFCs", BCP 26, RFC 2434,
          October 1998.

   [6]   Hoffman, P. and B. Schneier, "Attacks on Cryptographic Hashes
          in Internet Protocols", RFC 4270, November 2005.

   [7]   Nordmark, E. and M. Bagnulo, "Multihoming L3 Shim Approach",
          Work in Progress, July 2005.

   [8]   Johnson, D., Perkins, C., and J. Arkko, "Mobility Support in
          IPv6", RFC 3775, June 2004.

   [9]   Arkko, J., "Applying Cryptographically Generated Addresses and
          Credit-Based Authorization to Mobile IPv6", Work in Progress,
          June 2006.

   [10]   Bellovin, S. and E. Rescorla, "Deploying a New Hash Algorithm",
          NDSS '06, February 2006.

Authors' Addresses

   Marcelo Bagnulo
   Universidad Carlos III de Madrid
   Av. Universidad 30
   Leganes, Madrid 28911
   SPAIN

   Phone: 34 91 6249500
   EMail: marcelo@it.uc3m.es
   URI: http://www.it.uc3m.es

   Jari Arkko
   Ericsson
   Jorvas 02420
   Finland

   EMail: jari.arkko@ericsson.com

Full Copyright Statement

Intellectual Property

Acknowledgement