Network Working Group
Request for Comments: 3972
Category: Standards Track

T. Aura
Microsoft Research

March 2005

Cryptographically Generated Addresses (CGA)

Status of This Memo

Copyright Notice

Abstract

This document describes a method for binding a public signature key to an IPv6 address in the Secure Neighbor Discovery (SEND) protocol. Cryptographically Generated Addresses (CGA) are IPv6 addresses for which the interface identifier is generated by computing a cryptographic one-way hash function from a public key and auxiliary parameters. The binding between the public key and the address can be verified by re-computing the hash value and by comparing the hash with the interface identifier. Messages sent from an IPv6 address can be protected by attaching the public key and auxiliary parameters and by signing the message with the corresponding private key. The protection works without a certification authority or any security infrastructure.

Table of Contents

1. Introduction

This document specifies a method for securely associating a cryptographic public key with an IPv6 address in the Secure Neighbor Discovery (SEND) protocol [RFC3971]. The basic idea is to generate the interface identifier (i.e., the rightmost 64 bits) of the IPv6 address by computing a cryptographic hash of the public key. The resulting IPv6 address is called a cryptographically generated address (CGA). The corresponding private key can then be used to sign messages sent from the address. An introduction to CGAs and their application to SEND can be found in [Aura03] and [AAKMNR02].

This document specifies:

- how to generate a CGA from the cryptographic hash of a public key and auxiliary parameters,
- how to verify the association between the public key and the CGA, and
- how to sign a message sent from the CGA, and how to verify the signature.

To verify the association between the address and the public key, the verifier needs to know the address itself, the public key, and the values of the auxiliary parameters. The verifier can then go on to verify messages signed by the owner of the public key (i.e., the address owner). No additional security infrastructure, such as a public key infrastructure (PKI), certification authorities, or other trusted servers, is needed.

Note that because CGAs themselves are not certified, an attacker can create a new CGA from any subnet prefix and its own (or anyone else's) public key. However, the attacker cannot take a CGA created by someone else and send signed messages that appear to come from the owner of that address.

The address format and the CGA parameter format are defined in Sections 2 and 3. Detailed algorithms for generating addresses and for verifying them are given in Sections 4 and 5, respectively. Section 6 defines the procedures for generating and verifying CGA signatures. The security considerations in Section 7 include limitations of CGA-based security, the reasoning behind the hash extension technique that enables effective hash lengths above the 64-bit limit of the interface identifier, the implications of CGAs on privacy, and protection against related-protocol attacks.

In this document, the key words MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT, RECOMMENDED, MAY, and OPTIONAL are to be interpreted as described in [RFC2119].

2. CGA Format

When talking about addresses, this document refers to IPv6 addresses in which the leftmost 64 bits of a 128-bit address form the subnet prefix and the rightmost 64 bits of the address form the interface identifier [RFC3513]. We number the bits of the interface identifier starting from bit zero on the left.

A cryptographically generated address (CGA) has a security parameter (Sec) that determines its strength against brute-force attacks. The security parameter is a three-bit unsigned integer, and it is encoded in the three leftmost bits (i.e., bits 0 - 2) of the interface identifier. This can be written as follows:

```
Sec = (interface identifier & 0xe000000000000000) >> 61
```

The CGA is associated with a set of parameters that consist of a public key and auxiliary parameters. Two hash values Hash1 (64 bits) and Hash2 (112 bits) are computed from the parameters. The formats of the public key and auxiliary parameters, and the way to compute the hash values, are defined in Section 3.

A cryptographically generated address is defined as an IPv6 address that satisfies the following two conditions:

- The first hash value, Hash1, equals the interface identifier of the address. Bits 0, 1, 2, 6, and 7 (i.e., the bits that encode the security parameter Sec and the "u" and "g" bits from the standard IPv6 address architecture format of interface identifiers [RFC3513]) are ignored in the comparison.

- The 16*Sec leftmost bits of the second hash value, Hash2, are zero.

The above definition can be stated in terms of the following two bit masks:

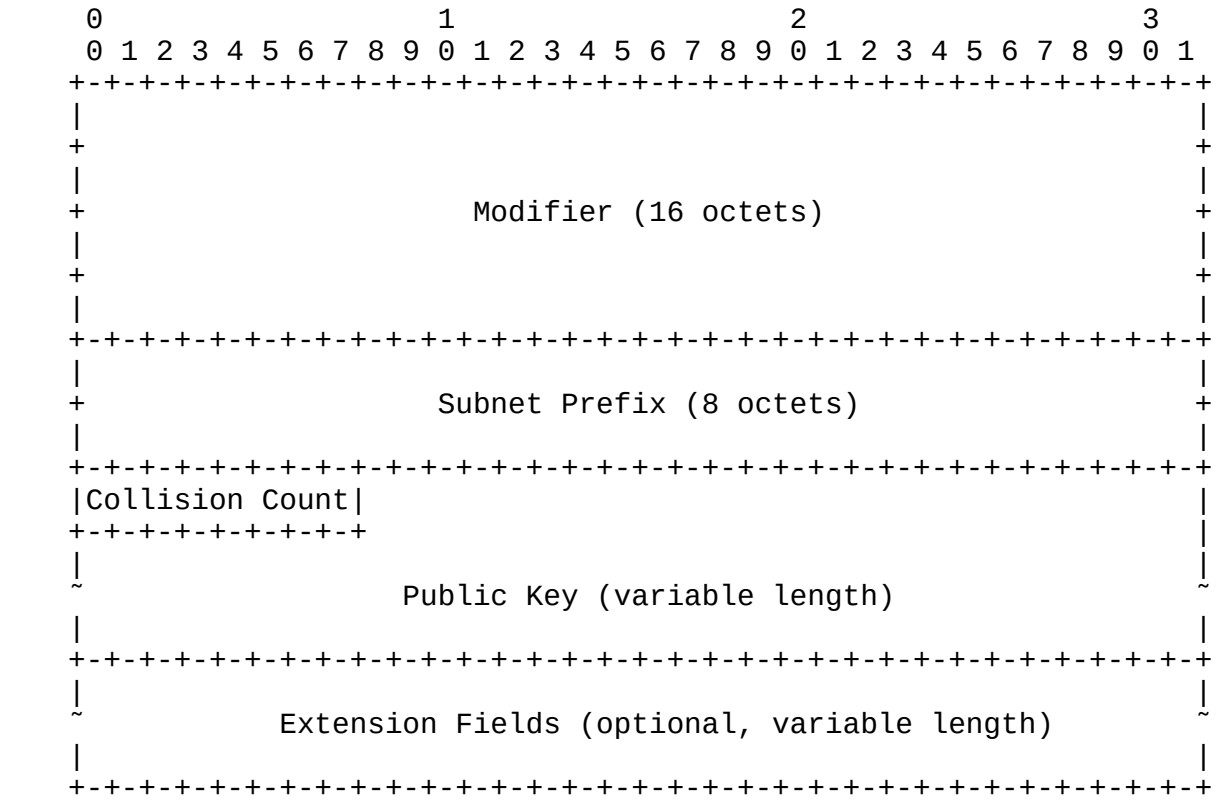
```
Mask1 (64 bits)  = 0x1cffffffffffffff

Mask2 (112 bits) = 0x00000000000000000000000000000000  if Sec=0,
                   0xffff0000000000000000000000000000  if Sec=1,
                   0xffffffff00000000000000000000000  if Sec=2,
                   0xffffffffffff000000000000000000  if Sec=3,
                   0xffffffffffffffff0000000000000  if Sec=4,
                   0xffffffffffffffffffff00000000  if Sec=5,
                   0xffffffffffffffffffffffff0000  if Sec=6, and
                   0xffffffffffffffffffffffffffff  if Sec=7
```

A cryptographically generated address is an IPv6 address for which the following two equations hold:

```
Hash1 & Mask1  ==  interface identifier & Mask1
Hash2 & Mask2  ==  0x00000000000000000000000000000000
```

3. CGA Parameters and Hash Values

Each CGA is associated with a CGA Parameters data structure, which has the following format:



Modifier

This field contains a 128-bit unsigned integer, which can be any value. The modifier is used during CGA generation to implement the hash extension and to enhance privacy by adding randomness to the address.

Subnet Prefix

This field contains the 64-bit subnet prefix of the CGA.

Collision Count

This is an eight-bit unsigned integer that MUST be 0, 1, or 2. The collision count is incremented during CGA generation to recover from an address collision detected by duplicate address detection.

Public Key

This is a variable-length field containing the public key of the address owner. The public key MUST be formatted as a DER-encoded [ITU.X690.2002] ASN.1 structure of the type SubjectPublicKeyInfo, defined in the Internet X.509 certificate profile [RFC3280]. SEND SHOULD use an RSA public/private key pair. When RSA is used, the algorithm identifier MUST be rsaEncryption, which is 1.2.840.113549.1.1.1, and the RSA public key MUST be formatted by using the RSAPublicKey type as specified in Section 2.3.1 of RFC 3279 [RFC3279]. The RSA key length SHOULD be at least 384 bits. Other public key types are undesirable in SEND, as they may result in incompatibilities between implementations. The length of this field is determined by the ASN.1 encoding.

Extension Fields

This is an optional variable-length field that is not used in the current specification. Future versions of this specification may use this field for additional data items that need to be included in the CGA Parameters data structure. IETF standards action is required to specify the use of the extension fields. Implementations MUST ignore the value of any unrecognized extension fields.

The two hash values MUST be computed as follows. The SHA-1 hash algorithm [FIPS.180-1.1995] is applied to the CGA Parameters. When Hash1 is computed, the input to the SHA-1 algorithm is the CGA Parameters data structure. The 64-bit Hash1 is obtained by taking the leftmost 64 bits of the 160-bit SHA-1 hash value. When Hash2 is computed, the input is the same CGA Parameters data structure except that the subnet prefix and collision count are set to zero. The 112-bit Hash2 is obtained by taking the leftmost 112 bits of the 160-bit SHA-1 hash value. Note that the hash values are computed over the entire CGA Parameters data structure, including any unrecognized extension fields.

4. CGA Generation

The process of generating a new CGA takes three input values: a 64-bit subnet prefix, the public key of the address owner as a DER-encoded ASN.1 structure of the type SubjectPublicKeyInfo, and the security parameter Sec, which is an unsigned three-bit integer. The cost of generating a new CGA depends exponentially on the security parameter Sec, which can have values from 0 to 7.

A CGA and associated parameters SHOULD be generated as follows:

1. Set the modifier to a random or pseudo-random 128-bit value.

2. Concatenate from left to right the modifier, 9 zero octets, the encoded public key, and any optional extension fields. Execute the SHA-1 algorithm on the concatenation. Take the 112 leftmost bits of the SHA-1 hash value. The result is Hash2.

3. Compare the 16*Sec leftmost bits of Hash2 with zero. If they are all zero (or if Sec=0), continue with step 4. Otherwise, increment the modifier by one and go back to step 2.

4. Set the 8-bit collision count to zero.

5. Concatenate from left to right the final modifier value, the subnet prefix, the collision count, the encoded public key, and any optional extension fields. Execute the SHA-1 algorithm on the concatenation. Take the 64 leftmost bits of the SHA-1 hash value. The result is Hash1.

6. Form an interface identifier from Hash1 by writing the value of Sec into the three leftmost bits and by setting bits 6 and 7 (i.e., the "u" and "g" bits) to zero.

7. Concatenate the 64-bit subnet prefix and the 64-bit interface identifier to form a 128-bit IPv6 address with the subnet prefix to the left and interface identifier to the right, as in a standard IPv6 address [RFC3513].

8. Perform duplicate address detection if required, as per [RFC3971]. If an address collision is detected, increment the collision count by one and go back to step 5. However, after three collisions, stop and report the error.

9. Form the CGA Parameters data structure by concatenating from left to right the final modifier value, the subnet prefix, the final collision count value, the encoded public key, and any optional extension fields.

The output of the address generation algorithm is a new CGA and a CGA Parameters data structure.

The initial value of the modifier in step 1 SHOULD be chosen randomly to make addresses generated from the same public key unlinkable, which enhances privacy (see Section 7.3). The quality of the random number generator does not affect the strength of the binding between

the address and the public key. Implementations that have no strong random numbers available MAY use a non-cryptographic pseudo-random number generator initialized with the current time of day.

For Sec=0, the above algorithm is deterministic and relatively fast. Nodes that implement CGA generation MAY always use the security parameter value Sec=0. If Sec=0, steps 2 - 3 of the generation algorithm can be skipped.

For Sec values greater than zero, the above algorithm is not guaranteed to terminate after a certain number of iterations. The brute-force search in steps 2 - 3 takes O(2^(16*Sec)) iterations to complete. The algorithm has been intentionally designed so that the generation of CGAs with high Sec values is infeasible with current technology.

Implementations MAY use optimized or otherwise modified versions of the above algorithm for CGA generation. However, the output of any modified versions MUST fulfill the following two requirements. First, the resulting CGA and CGA Parameters data structure MUST be formatted as specified in Sections 2 - 3. Second, the CGA verification procedure defined in Section 5 MUST succeed when invoked on the output of the CGA generation algorithm. Note that some optimizations involve trade-offs between privacy and the cost of address generation.

One optimization is particularly important. If the subnet prefix of the address changes but the address owner’s public key does not, the old modifier value MAY be reused. If it is reused, the algorithm SHOULD be started from step 4. This optimization avoids repeating the expensive search for an acceptable modifier value but may, in some situations, make it easier for an observer to link two addresses to each other.

Note that this document does not specify whether duplicate address detection should be performed and how the detection is done. Step 8 only defines what to do if some form of duplicate address detection is performed and an address collision is detected.

Future versions of this specification may specify additional inputs to the CGA generation algorithm that are concatenated as extension fields to the end of the CGA Parameters data structure. No such extension fields are defined in this document.

5. CGA Verification

CGA verification takes an IPv6 address and a CGA Parameters data structure as input. The CGA Parameters consist of the concatenated modifier, subnet prefix, collision count, public key, and optional extension fields. The verification either succeeds or fails.

The CGA MUST be verified with the following steps:

1. Check that the collision count in the CGA Parameters data structure is 0, 1, or 2. The CGA verification fails if the collision count is out of the valid range.

2. Check that the subnet prefix in the CGA Parameters data structure is equal to the subnet prefix (i.e., the leftmost 64 bits) of the address. The CGA verification fails if the prefix values differ.

3. Execute the SHA-1 algorithm on the CGA Parameters data structure. Take the 64 leftmost bits of the SHA-1 hash value. The result is Hash1.

4. Compare Hash1 with the interface identifier (i.e., the rightmost 64 bits) of the address. Differences in the three leftmost bits and in bits 6 and 7 (i.e., the "u" and "g" bits) are ignored. If the 64-bit values differ (other than in the five ignored bits), the CGA verification fails.

5. Read the security parameter Sec from the three leftmost bits of the 64-bit interface identifier of the address. (Sec is an unsigned 3-bit integer.)

6. Concatenate from left to right the modifier, 9 zero octets, the public key, and any extension fields that follow the public key in the CGA Parameters data structure. Execute the SHA-1 algorithm on the concatenation. Take the 112 leftmost bits of the SHA-1 hash value. The result is Hash2.

7. Compare the 16*Sec leftmost bits of Hash2 with zero. If any one of them is not zero, the CGA verification fails. Otherwise, the verification succeeds. (If Sec=0, the CGA verification never fails at this step.)

If the verification fails at any step, the execution of the algorithm MUST be stopped immediately. On the other hand, if the verification succeeds, the verifier knows that the public key in the CGA Parameters is the authentic public key of the address owner. The

verifier can extract the public key by removing 25 octets from the beginning of the CGA Parameters and by decoding the following SubjectPublicKeyInfo data structure.

Note that the values of bits 6 and 7 (the "u" and "g" bits) of the interface identifier are ignored during CGA verification. In the SEND protocol, after the verification succeeds, the verifier SHOULD process all CGAs in the same way regardless of the Sec, modifier, and collision count values. In particular, the verifier in the SEND protocol SHOULD NOT have any security policy that differentiates between addresses based on the value of Sec. That way, the address generator is free to choose any value of Sec.

All nodes that implement CGA verification MUST be able to process all security parameter values Sec = 0, 1, 2, 3, 4, 5, 6, 7. The verification procedure is relatively fast and always requires at most two computations of the SHA-1 hash function. If Sec=0, the verification never fails in steps 6 - 7 and these steps can be skipped.

Nodes that implement CGA verification for SEND SHOULD be able to process RSA public keys that have the algorithm identifier rsaEncryption and, key length between 384 and 2,048 bits. Implementations MAY support longer keys. Future versions of this specification may recommend support for longer keys.

Implementations of CGA verification MUST ignore the value of any unrecognized extension fields that follow the public key in the CGA Parameters data structure. However, implementations MUST include any such unrecognized data in the hash input when computing Hash1 in step 3 and Hash2 in step 6 of the CGA verification algorithm. This is important to ensure upward compatibility with future extensions.

6. CGA Signatures

This section defines the procedures for generating and verifying CGA signatures. To sign a message, a node needs the CGA, the associated CGA Parameters data structure, the message, and the private cryptographic key that corresponds to the public key in the CGA Parameters. The node also must have a 128-bit type tag for the message from the CGA Message Type name space.

To sign a message, a node SHOULD do the following:

- Concatenate the 128-bit type tag (in network byte order) and the message with the type tag to the left and the message to the right. The concatenation is the message to be signed in the next step.

- Generate the RSA signature by using the RSASSA-PKCS1-v1_5 [RFC3447] signature algorithm with the SHA-1 hash algorithm. The private key and the concatenation created above are the inputs to the generation operation.

The SEND protocol specification [RFC3971] defines several messages that contain a signature in the Signature Option. The SEND protocol specification also defines a type tag from the CGA Message Type name space. The same type tag is used for all the SEND messages that have the Signature Option. This type tag is an IANA-allocated 128 bit integer that has been chosen at random to prevent an accidental type collision with messages of other protocols that use the same public key but that may or may not use IANA-allocated type tags.

The CGA, the CGA Parameters data structure, the message, and the signature are sent to the verifier. The SEND protocol specification defines how these data items are sent in SEND protocol messages. Note that the 128-bit type tag is not included in the SEND protocol messages because the verifier knows its value implicitly from the ICMP message type field in the SEND message. See the SEND specification [RFC3971] for precise information about how SEND handles the type tag.

To verify a signature, the verifier needs the CGA, the associated CGA Parameters data structure, the message, and the signature. The verifier also needs to have the 128-bit type tag for the message.

To verify the signature, a node SHOULD do the following:

- Verify the CGA as defined in Section 5. The inputs to the CGA verification are the CGA and the CGA Parameters data structure.

- Concatenate the 128-bit type tag and the message with the type tag to the left and the message to the right. The concatenation is the message whose signature is to be verified in the next step.

- Verify the RSA signature by using the RSASSA-PKCS1-v1_5 [RFC3447] algorithm with the SHA-1 hash algorithm. The inputs to the verification operation are the public key (i.e., the RSAPublicKey structure from the SubjectPublicKeyInfo structure that is a part of the CGA Parameters data structure), the concatenation created above, and the signature.

The verifier MUST accept the signature as authentic only if both the CGA verification and the signature verification succeed.

7. Security Considerations

7.1. Security Goals and Limitations

The purpose of CGAs is to prevent stealing and spoofing of existing IPv6 addresses. The public key of the address owner is bound cryptographically to the address. The address owner can use the corresponding private key to assert its ownership and to sign SEND messages sent from the address.

It is important to understand that an attacker can create a new address from an arbitrary subnet prefix and its own (or someone else's) public key because CGAs are not certified. However, the attacker cannot impersonate somebody else's address. This is because the attacker would have to find a collision of the cryptographic hash value Hash1. (The property of the hash function needed here is called second pre-image resistance [MOV97].)

For each valid CGA Parameters data structure, there are 4*(Sec+1) different CGAs that match the value. This is because decrementing the Sec value in the three leftmost bits of the interface identifier does not invalidate the address, and the verifier ignores the values of the "u" and "g" bits. In SEND, this does not have any security or implementation implications.

Another limitation of CGAs is that there is no mechanism for proving that an address is not a CGA. Thus, an attacker could take someone else's CGA and present it as a non-cryptographically generated address (e.g., as an RFC 3041 address [RFC3041]). An attacker does not benefit from this because although SEND nodes accept both signed and unsigned messages from every address, they give priority to the information in the signed messages.

The minimum RSA key length required for SEND is only 384 bits. So short keys are vulnerable to integer-factoring attacks and cannot be used for strong authentication or secrecy. On the other hand, the cost of factoring 384-bit keys is currently high enough to prevent most denial-of-service attacks. Implementations that initially use short RSA keys SHOULD be prepared to switch to longer keys when denial-of-service attacks arising from integer factoring become a problem.

The impact of a key compromise on CGAs depends on the application for which they are used. In SEND, it is not a major concern. If the private signature key is compromised because the SEND node has itself been compromised, the attacker does not need to spoof SEND messages from the node. When it is discovered that a node has been compromised, a new signature key and a new CGA SHOULD be generated.

On the other hand, if the RSA key is compromised because integer-factoring attacks for the chosen key length have become practical, the key has to be replaced with a longer one, as explained above. In either case, the address change effectively revokes the old public key. It is not necessary to have any additional key revocation mechanism or to limit the lifetimes of the signature keys.

7.2. Hash Extension

As computers become faster, the 64 bits of the interface identifier will not be sufficient to prevent attackers from searching for hash collisions. It helps somewhat that we include the subnet prefix of the address in the hash input. This prevents the attacker from using a single pre-computed database to attack addresses with different subnet prefixes. The attacker needs to create a separate database for each subnet prefix. Link-local addresses are, however, left vulnerable because the same prefix is used by all IPv6 nodes.

To prevent the CGA technology from becoming outdated as computers become faster, the hash technique used to generate CGAs must be extended somehow. The chosen extension technique is to increase the cost of both address generation and brute-force attacks by the same parameterized factor while keeping the cost of address use and verification constant. This also provides protection for link-local addresses. Introduction of the hash extension is the main difference between this document and earlier CGA proposals [OR01][Nik01][MC02].

To achieve the effective extension of the hash length, the input to the second hash function, Hash2, is modified (by changing the modifier value) until the leftmost 16*Sec bits of the hash value are zero. This increases the cost of address generation approximately by a factor of 2^(16*Sec). It also increases the cost of brute-force attacks by the same factor. That is, the cost of creating a CGA Parameters data structure that binds the attacker's public key with somebody else's address is increased from O(2^59) to O(2^(59+16*Sec)). The address generator may choose the security parameter Sec depending on its own computational capacity, the perceived risk of attacks, and the expected lifetime of the address. Currently, Sec values between 0 and 2 are sufficient for most IPv6 nodes. As computers become faster, higher Sec values will slowly become useful.

Theoretically, if no hash extension is used (i.e., Sec=0) and a typical attacker is able to tap into N local networks at the same time, an attack against link-local addresses is N times as efficient as an attack against addresses of a specific network. The effect could be countered by using a slightly higher Sec value for link-

local addresses. When higher Sec values (such that 2^(16*Sec) > N) are used for all addresses, the relative advantage of attacking link-local addresses becomes insignificant.

The effectiveness of the hash extension depends on the assumption that the computational capacities of the attacker and the address generator will grow at the same (potentially exponential) rate. This is not necessarily true if the addresses are generated on low-end mobile devices, for which the main design goals are to lower cost and decrease size, rather than increase computing power. But there is no reason for doing so. The expensive part of the address generation (steps 1 - 3 of the generation algorithm) may be delegated to a more powerful computer. Moreover, this work can be done in advance or offline, rather than in real time, when a new address is needed.

To make it possible for mobile nodes whose subnet prefixes change frequently to use Sec values greater than zero, we have decided not to include the subnet prefix in the input of Hash2. The result is weaker than it would be if the subnet prefix were included in the input of both hashes. On the other hand, our scheme is at least as strong as using the hash extension technique without including the subnet prefix in either hash. It is also at least as strong as not using the hash extension but including the subnet prefix. This trade-off was made because mobile nodes frequently move to insecure networks, where they are at the risk of denial-of-service (DoS) attacks (for example, during the duplicate address detection procedure).

In most networks, the goal of Secure Neighbor Discovery and CGA signatures is to prevent denial-of-service attacks. Therefore, it is usually sensible to start by using a low Sec value and to replace addresses with stronger ones only when denial-of-service attacks based on brute-force search become a significant problem. If CGAs were used as a part of a strong authentication or secrecy mechanism, it might be necessary to start with higher Sec values.

The collision count value is used to modify the input to Hash1 if there is an address collision. It is important not to allow collision count values higher than 2. First, it is extremely unlikely that three collisions would occur and the reason is certain to be either a configuration or implementation error or a denial-of-service attack. (When the SEND protocol is used, deliberate collisions caused by a DoS attacker are detected and ignored.) Second, an attacker doing a brute-force search to match a given CGA can try all different values of a collision count without repeating the brute-force search for the modifier value. Thus, if higher values are allowed for the collision count, the hash extension technique becomes less effective in preventing brute force attacks.

7.3. Privacy Considerations

CGAs can give the same level of pseudonymity as the IPv6 address privacy extensions defined in RFC 3041 [RFC3041]. An IP host can generate multiple pseudo-random CGAs by executing the CGA generation algorithm of Section 4 multiple times and by using a different random or pseudo-random initial value for the modifier every time. The host should change its address periodically as in [RFC3041]. When privacy protection is needed, the (pseudo)random number generator used in address generation SHOULD be strong enough to produce unpredictable and unlinkable values. Advice on random number generation can be found in [RFC1750].

There are two apparent limitations to this privacy protection. However, as will be explained below, neither is very serious.

First, the high cost of address generation may prevent hosts that use a high Sec value from changing their address frequently. This problem is mitigated because the expensive part of the address generation may be done in advance or offline, as explained in the previous section. It should also be noted that the nodes that benefit most from high Sec values (e.g., DNS servers, routers, and data servers) usually do not require pseudonymity, and the nodes that have high privacy requirements (e.g., client PCs and mobile hosts) are unlikely targets for expensive brute-force DoS attacks and can make do with lower Sec values.

Second, the public key of the address owner is revealed in the signed SEND messages. This means that if the address owner wants to be pseudonymous toward the nodes in the local links that it accesses, it should generate not only a new address but also a new public key. With typical local-link technologies, however, a node's link-layer address is a unique identifier for the node. As long as the node keeps using the same link-layer address, it makes little sense to change the public key for privacy reasons.

7.4. Related Protocols

Although this document defines CGAs only for the purposes of Secure Neighbor Discovery, other protocols could be defined elsewhere that use the same addresses and public keys. This raises the possibility of related-protocol attacks in which a signed message from one protocol is replayed in another protocol. This means that other protocols (perhaps even those designed without an intimate knowledge of SEND) could endanger the security of SEND. What makes this threat even more significant is that the attacker could create a CGA from someone else's public key and then replay signed messages from a protocol that has nothing to do with CGAs or IP addresses.

To prevent the related-protocol attacks, a type tag is prepended to every message before it is signed. The type tags are 128-bit randomly chosen values, which prevents accidental type collisions with even poorly designed protocols that do not use any type tags. Moreover, the SEND protocol includes the sender's CGA address in all signed messages. This makes it even more difficult for an attacker to take signed messages from some other context and to replay them as SEND messages.

Finally, a strong cautionary note has to be made about using CGA signatures for purposes other than SEND. First, the other protocols MUST include a type tag and the sender address in all signed messages in the same way that SEND does. Each protocol MUST define its own type tag values as explained in Section 8. Moreover, because of the possibility of related-protocol attacks, the public key MUST be used only for signing, and it MUST NOT be used for encryption. Second, the minimum RSA key length of 384 bits may be too short for many applications and the impact of key compromise on the particular protocol must be evaluated. Third, CGA-based authorization is particularly suitable for securing neighbor discovery [RFC2461] and duplicate address detection [RFC2462] because these are network-layer signaling protocols for which IPv6 addresses are natural endpoint identifiers. In any protocol that uses other identifiers, such as DNS names, CGA signatures alone are not a sufficient security mechanism. There must also be a secure way of mapping the other identifiers to IPv6 addresses. If the goal is not to verify claims about IPv6 addresses, CGA signatures are probably not the right solution.

8. IANA Considerations

This document defines a new CGA Message Type name space for use as type tags in messages that may be signed by using CGA signatures. The values in this name space are 128-bit unsigned integers. Values in this name space are allocated on a First Come First Served basis [RFC2434]. IANA assigns new 128-bit values directly without a review.

The requester SHOULD generate the new values with a strong random-number generator. Continuous ranges of at most 256 values can be requested provided that the 120 most significant bits of the values have been generated with a strong random-number generator.

IANA does not generate random values for the requester. IANA allocates requested values without verifying the way in which they have been generated. The name space is essentially unlimited, and any number of individual values and ranges of at most 256 values can be allocated.

CGA Message Type values for private use MAY be generated with a strong random-number generator without IANA allocation.

This document does not define any new values in any name space.

9. References

9.1. Normative References

[RFC3971] Arkko, J., Ed., Kempf, J., Sommerfeld, B., Zill, B., and P. Nikander, "SEcure Neighbor Discovery (SEND)", RFC 3971, March 2005.

[RFC3279] Bassham, L., Polk, W., and R. Housley, "Algorithms and Identifiers for the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 3279, April 2002.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC3513] Hinden, R. and S. Deering, "Internet Protocol Version 6 (IPv6) Addressing Architecture", RFC 3513, April 2003.

[RFC3280] Housley, R., Polk, W., Ford, W., and D. Solo, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 3280, April 2002.

[ITU.X690.2002] International Telecommunications Union, "Information Technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)", ITU-T Recommendation X.690, July 2002.

[RFC3447] Jonsson, J. and B. Kaliski, "Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1", RFC 3447, February 2003.

[RFC2434] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 2434, October 1998.

[FIPS.180-1.1995] National Institute of Standards and Technology, "Secure Hash Standard", Federal Information Processing Standards Publication FIPS PUB 180-1, April 1995, <http://www.itl.nist.gov/fipspubs/fip180-1.htm>.

9.2. Informative References

[AAKMNR02] Arkko, J., Aura, T., Kempf, J., Mantyla, V., Nikander, P., and M. Roe, "Securing IPv6 neighbor discovery and router discovery", ACM Workshop on Wireless Security (WiSe 2002), Atlanta, GA USA , September 2002.

[Aura03] Aura, T., "Cryptographically Generated Addresses (CGA)", 6th Information Security Conference (ISC'03), Bristol, UK, October 2003.

[RFC1750] Eastlake, D., Crocker, S., and J. Schiller, "Randomness Recommendations for Security", RFC 1750, December 1994.

[MOV97] Menezes, A., van Oorschot, P., and S. Vanstone, "Handbook of Applied Cryptography", CRC Press , 1997.

[MC02] Montenegro, G. and C. Castelluccia, "Statistically unique and cryptographically verifiable identifiers and addresses", ISOC Symposium on Network and Distributed System Security (NDSS 2002), San Diego, CA USA , February 2002.

[RFC3041] Narten, T. and R. Draves, "Privacy Extensions for Stateless Address Autoconfiguration in IPv6", RFC 3041, January 2001.

[RFC2461] Narten, T., Nordmark, E., and W. Simpson, "Neighbor Discovery for IP Version 6 (IPv6)", RFC 2461, December 1998.

[Nik01] Nikander, P., "A scaleable architecture for IPv6 address ownership", draft-nikander-addr-ownership-00 (work in progress), March 2001.

[OR01] O'Shea, G. and M. Roe, "Child-proof authentication for MIPv6 (CAM)", ACM Computer Communications Review 31(2), April 2001.

[RFC2462] Thomson, S. and T. Narten, "IPv6 Stateless Address Autoconfiguration", RFC 2462, December 1998.

Appendix A. Example of CGA Generation

We generate a CGA with Sec=1 from the subnet prefix fe80:: and the following public key:

```
305c 300d 0609 2a86 4886 f70d 0101 0105 0003 4b00 3048 0241
00c2 c2f1 3730 5454 f10b d9ce a368 44b5 30e9 211a 4b26 2b16
467c b7df ba1f 595c 0194 f275 be5a 4d38 6f2c 3c23 8250 8773
c786 7f9b 3b9e 63a0 9c7b c48f 7a54 ebef af02 0301 0001
```

The modifier is initialized to a random value 89a8 a8b2 e858 d8b8 f263 3f44 d2d4 ce9a. The input to Hash2 is:

```
89a8 a8b2 e858 d8b8 f263 3f44 d2d4 ce9a 0000 0000 0000 0000 00
305c 300d 0609 2a86 4886 f70d 0101 0105 0003 4b00 3048 0241
00c2 c2f1 3730 5454 f10b d9ce a368 44b5 30e9 211a 4b26 2b16
467c b7df ba1f 595c 0194 f275 be5a 4d38 6f2c 3c23 8250 8773
c786 7f9b 3b9e 63a0 9c7b c48f 7a54 ebef af02 0301 0001
```

The 112 first bits of the SHA-1 hash value computed from the above input are Hash2=436b 9a70 dbfd dbf1 926e 6e66 29c0. This does not begin with 16*Sec=16 zero bits. Thus, we must increment the modifier by one and recompute the hash. The new input to Hash2 is:

```
89a8 a8b2 e858 d8b8 f263 3f44 d2d4 ce9b 0000 0000 0000 0000 00
305c 300d 0609 2a86 4886 f70d 0101 0105 0003 4b00 3048 0241
00c2 c2f1 3730 5454 f10b d9ce a368 44b5 30e9 211a 4b26 2b16
467c b7df ba1f 595c 0194 f275 be5a 4d38 6f2c 3c23 8250 8773
c786 7f9b 3b9e 63a0 9c7b c48f 7a54 ebef af02 0301 0001
```

The new hash value is Hash2=0000 01ca 680b 8388 8d09 12df fcce. The 16 leftmost bits of Hash2 are all zero. Thus, we found a suitable modifier. (We were very lucky to find it so soon.)

The input to Hash1 is:

```
89a8 a8b2 e858 d8b8 f263 3f44 d2d4 ce9b fe80 0000 0000 0000 00
305c 300d 0609 2a86 4886 f70d 0101 0105 0003 4b00 3048 0241
00c2 c2f1 3730 5454 f10b d9ce a368 44b5 30e9 211a 4b26 2b16
467c b7df ba1f 595c 0194 f275 be5a 4d38 6f2c 3c23 8250 8773
c786 7f9b 3b9e 63a0 9c7b c48f 7a54 ebef af02 0301 0001
```

The 64 first bits of the SHA-1 hash value of the above input are Hash1=fd4a 5bf6 ffb4 ca6c. We form an interface identifier from this by writing Sec=1 into the three leftmost bits and by setting bits 6 and 7 (the "u" and "g" bits) to zero. The new interface identifier is 3c4a:5bf6:ffb4:ca6c.

Finally, we form the IPv6 address fe80::3c4a:5bf6:ffb4:ca6c. This is the new CGA. No address collisions were detected this time. (Collisions are very rare.) The CGA Parameters data structure associated with the address is the same as the input to Hash1 above.

Appendix B. Acknowledgements

The author gratefully acknowledges the contributions of Jari Arkko, Francis Dupont, Pasi Eronen, Christian Huitema, James Kempf, Pekka Nikander, Michael Roe, Dave Thaler, and other participants of the SEND working group.

Author's Address

Tuomas Aura
Microsoft Research
Roger Needham Building
7 JJ Thomson Avenue
Cambridge CB3 0FB
United Kingdom

Phone: +44 1223 479708
EMail: tuomaura@microsoft.com

Full Copyright Statement

Intellectual Property

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.