Published as: Ahmad AlSa'deh, Hosnieh Rafiee, Christoph Meinel, “Cryptographically Generated Addresses (CGAs): Possible Attacks and Proposed Mitigation Approaches,” in Proceedings of the 12th IEEE International Conference on Computer and Information Technology (CIT 2012), IEEE Press, Chengdu, Sichuan, China, 27-29 October 2012.

Cryptographically Generated Addresses (CGAs): Possible Attacks and Proposed Mitigation Approaches

Ahmad AlSa’deh
Hasso Plattner Institute
University of Potsdam
Potsdam, Germany
ahmad.alsadeh@hpi.uni-potsdam.de

Hosnieh Rafiee
Hasso Plattner Institute
University of Potsdam
Potsdam, Germany
hosnieh.rafiee@hpi.uni-potsdam.de

Christoph Meinel
Hasso Plattner Institute
University of Potsdam
Potsdam, Germany
christoph.meinel@hpi.uni-potsdam.de

***Abstract*—Cryptographically Generated Addresses (CGAs) were mainly designed to prove address ownership and to prevent the theft of existing IPv6 addresses by binding the owner’s public key to the generated address. The address owner uses a corresponding private key to prove its ownership by using signed messages that are originated from that address. Though the CGA approach is quite useful in providing a means of proving address ownership in IPv6 networks, it does have some limitations and some vulnerabilities. In this paper we will provide a security analysis and descriptions of possible ways of attacking CGA. We found that the CGA verification process is prone mainly to Denial-of-Service (DoS) attacks. We also found that CGAs are still susceptible to privacy related attacks. We will therefore propose some extensions to the CGA standard verification algorithm to mitigate DoS attacks and to make CGA more privacy-conscious.**

Keywords- IPv6 Security; IPv6 Privacy; Authentication of IPv6 addresses; IPv6 addresses ownership

I. INTRODUCTION

In the absence of a reliable authentication mechanism, it is easy to fabricate forged IPv6 messages which lead to various types of attacks. In the IPv6 network, Neighbor Discovery Protocol (NDP) assumes that all nodes on the link trust each other. However, this assumption does not hold for a number of scenarios one of which is over a wireless network where anyone can join a local link either with a minimal or with no link layer authentication. Consequently, an attacker can impersonate legitimate nodes by forging NDP messages in order to generate Denial-of-Service (DoS) attacks, and instigate theft and traffic spoofing.

Cryptographically Generated Addresses (CGAs) [1] were first proposed in order to provide the necessary authentication for IPv6 addresses. CGAs are IPv6 addresses where the interface identifier (ID) portion of the addressing scheme (the 64-rightmost bits of IPv6 address) is created from a cryptographic hash of the address owner’s public key and other auxiliary parameters. The address owner uses a corresponding private key to sign messages sent from that address. In this manner the CGA technique enables the address owner to prove address ownership by binding the public key signature to an IPv6 address.

The self-certified feature of CGA is its main advantage. No third party or additional security infrastructure is needed. The CGA approach can thus be scaled up for large networks. Any node can generate its own CGA address locally and then only the address and the public key are needed to verify the binding between the public key and the address. CGA also works automatically without the need for manual user configuration.

Although CGA is a promising security technique for use with IPv6 addresses, there are some limitations and disadvantages. The main disadvantage of using CGA is the computational time necessary to generate the address. Also, CGA is not a complete security solution; it still exhibits weaknesses and vulnerabilities to threats. For instance, CGA cannot provide the assurance needed with respect to the authority of the node so there is no guarantee that the CGA address was created from the appropriate node. Attackers can thus exploit this weakness to create a new valid address from their own public key. Attackers can also capture Neighbor Discovery (ND) messages and alter the sender’s CGA parameters. When this happens the CGA verification process on the receiver’s side will fail. Thus the communication between a legitimate sender and receiver is prevented. It is also possible for an attacker to conduct a Duplicate Address Detection DoS Attack which will prevent a CGA node from joining a link. An attacker can copy the CGA parameters and the signature and then respond with a Neighbor Advertisement (NA) message that contains the same security parameters. In this way the attacker can prevent the CGA address configuration for all nodes attached to a local link. Another type of attack is one in which the victim’s node is kept busy with the verification process. An attacker will inundate the verifier with valid or invalid CGA signed messages.

In this paper we analyze the possible methods an attacker could use to attack the standard CGA [2]. The conclusions that we have reached are first, that the CGA verification process is still vulnerable to DoS attacks. Therefore we proposed an extension to the CGA verification algorithm in order to eliminate this attack. Second, an attacker can capture and replay the sender’s CGA parameters so that the verification process fails on the verifier side. To combat this attack the CGA should include a *Timestamp Option* in order to mitigate this type of attack. Third, that CGA may be susceptible to

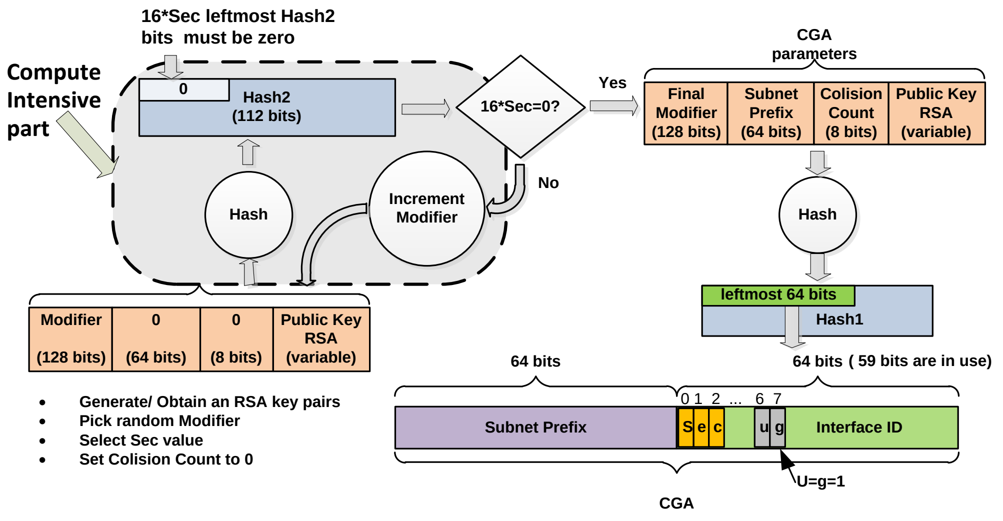

Figure 1. CGA Generation Algorithm.

privacy related attacks. Thus we extend it to make it more privacy-conscious by changing the addresses over time. Changing the addresses over time makes it more difficult for eavesdroppers to correlate when different addresses are used for different activities corresponding to the same node.

The remainder of this paper is organized as follows: Section II briefly summarizes the CGA algorithm, Section III analyzes the CGA attack types and costs and suggests modifications that can be made to the standard CGA. Section IV presents the implementation of the proposed extensions, besides implementing some attacks against CGA in practice. Finally, Section V summarizes our conclusions.

II. CRYPTOGRAPHICALLY GENERATED ADDRESSES (CGAs)

In this section we will provide a review of the works related to the basic idea of CGA and then briefly introduce the standard CGA generation and verification algorithms being used. At the end of this section we will introduce a security analysis of the standard CGA.

A. *CGA Related Work*

The idea of using Cryptographically Generated Addresses first appeared in the Child-proof Authentication for MIPv6 (CAM) which was proposed by O'Shea and Roe [3]. In the CAM approach, the hash of the owner's public key is added to the interface ID portion of IPv6 address. Later Nikander [4] suggested an improvement and an extension to the CAM approach to make it more resistant to *birthday collision* by adding some "random" data to the hash input. Montenegro and Castelluccia [5] worked on a similar proposal for Mobile IPv6. The final model of CGA was proposed by Aura [1] and was standardized in RFC 3972 [2].

Introduction of the *Hash Extension* is the main difference between Aura's proposal [1] and the earlier proposals. The use of a 64-bit value does not adequately protect the address from a security standpoint. The *Hash Extension* technique increases the hash length beyond the 64-bit limit without actually increasing its length. This technique increases both the cost of generating a new CGA address and the cost of initiating a brute-force attack against the address. This is realized by a scaling factor called the Security Parameter (*Sec*) which determines the level of security for each generated address. Henceforth the term CGA will be used to refer to the standardized CGA which appears in RFC 3972.

B. *CGA Algorithm*

Instead of a single hash value, the standard CGA [2] computes two independent one-way hash values (Hash1 and Hash2). The purpose of the second hash (Hash2), *Hash Extension*, is to increase the cost of the brute-force attack without increasing the length of the hash output value which is written to the interface ID portion of IPv6 address.

The computational complexity of Hash2 depends on the *Sec* value. The *Sec* is an unsigned 3-bit integer having a value between 0 and 7 (0 being the least secure while 7 the most) which indicates the security level of the generated address against brute-force attacks.

The procedure for the CGA generation process is depicted in Figure 1. The algorithm uses as input values; *Public Key*, *Modifier*, *Subnet Prefix*, and *Sec* value. The output from the CGA algorithm is CGA address and CGA parameters which are comprised of the following fields:

- *Modifier* (128-bit): initialized to a random value.
- *Subnet Prefix* (64-bit): set to the routing prefix value advertised by the router at the local subnet.
- *Collision Count* (8-bits): a collision counter used for Duplicate Address Detection (DAD) to ensure the uniqueness of the generated address.
- *Public Key* (variable length): set to the Distinguished Encoding Rules (DER) encoded public key of the address owner.

- *Extension Field*: variable length field for future use.

CGA generation begins by determining the address owner's public key and selecting the proper *Sec* value to use. The process continues with the Hash2 computation loop which finds the *Final Modifier* which satisfies the condition where the *16×Sec* leftmost bits of the Hash2 are equal to zero. The Hash2 value is a Secure Hash Algorithm (SHA)-1 hash value over all CGA parameters (the *Public Key* and *Collision Counts* are zeros). The address generator tries different values for the *Modifier* until the 16×Sec-leftmost-bit of Hash2 computes to zero. Once a match is found, the loop for the Hash2 computation terminates. At this point the *Final Modifier* value is saved and used as an input for the Hash1 computation. The Hash1 value is a hash created by the combination of all of the CGA parameters. Then the interface ID is derived from the Hash1 value. The hash value is truncated to the appropriate length (64-bit). The *Sec* value is encoded into the three leftmost bits of the interface ID. The 7th and 8th (*u* and *g*) bits, from the left of interface ID, are reserved for a special purpose; they are equal to 1 to identify a field as a CGA address. Thus, the hash output of the CGA parameters will be distributed across the remaining 59 bits of the interface ID. The concatenation of the subnet prefix (64-bit leftmost bits) with the interface ID portion forms the completed IPv6 address. The subnet prefix can be a routable global prefix which is obtained by listening for the local Router Advertisement (RA) or local link prefix. Finally a DAD algorithm is executed against this tentative address to ensure that there is no address collision within the same subnet. If an address conflict does occur, then the *Collision Count* will be incremented and the Hash1 process will be repeated until a link-unique address is obtained or the *Collision Count* reaches 2 (after three collisions).

The fact is that fulfilling the condition of Hash2 is the computationally expensive part of CGA generation. Selecting a high *Sec* value may cause an unacceptable delay in address generation. For a *Sec* value greater than zero there is a probabilistic guarantee that the process will stop after a certain number of iterations but not exactly when. Thus, the required time to find the final *Modifier* that satisfies the condition where *16×Sec = 0* is very diverse for the same *Sec* value. We measured the generation time of 1000 CGAs with *Sec =1* and with 1024-bit RSA key size. We found that the required time to satisfy the Hash2 condition and to find a valid CGA varied between 10 to 2060 milliseconds. Our CGA calculation was performed using a computer with a 2.67 GHz CPU speed.

To assert the ownership of an address and to protect the message, the address owner uses a corresponding private key to sign messages sent from that address. Signing a message using CGA requires the combined use of the CGA address, the associated CGA parameters, the message, and the private key that corresponds to the *Public Key* in the CGA parameters. Finally, the node will send the message, the CGA parameters, and the signature to the receiver node.

CGA verification takes as an input the IPv6 address and the CGA parameters. If the verification succeeds, the verifier knows that the public key belongs to that address. The verifier can then use the *Public Key* to authenticate the signed messages received from that address. According to RFC3972, the verification process is achieved by executing the following steps:

- Check that the *Collision Count* value is 0, 1 or 2, and that the *Subnet Prefix* value is equal to the subnet prefix of the address. The CGA verification fails if either check fails.
- Concatenate the CGA parameters and execute the hash algorithm (SHA-1) on the concatenation. The 64 leftmost bits of the result make up Hash1.
- Compare Hash1 with the interface ID of the address. Differences in the *u* and *g* bits and in the three leftmost bits are ignored. If the 64-bit values differ (other than in the five ignored bits), the CGA verification fails.
- Read the security parameter (*Sec*) from the three leftmost bits of the interface ID of the address.
- Concatenate the *Modifier*, 64+8 zero bits and the *Public Key*. Execute the hash algorithm on the concatenation. The leftmost 112 bits of the result make up Hash2.
- Compare the *16×Sec* leftmost bits of Hash2 with zero. If any one of these is non-zero, CGA verification fails. Otherwise, the verification succeeds. If Sec=0, verification never fails in this step.

In addition to the CGA verification process outlined above, the verifier uses the *Public Key* to verify the signature on the message. If the signature is valid, the verifier knows that the message was sent by the specific IPv6 address.

The CGA algorithm increases the computational cost for both the attacker and the address generator (owner). For *Sec* values greater than zero, the address generator needs, on average, $O(2^{16\times Sec})$ iterations to complete a brute-force search in order to satisfy the Hash2 condition and to find the *Final Modifier*. This requires that the attacker performs a brute-force search for (*16×Sec+59*) hash bits which costs, on average, $O(2^{16\times Sec + 59})$. Hence, increasing the *Sec* value by 1 adds 16 bits to the length of hash that the attacker must break.

III. Possible Attacks to Comromise a CGA Node

There are several possible ways for an attacker to compromise a CGA node. Some of these methods are known and are mentioned in the literature while others are not. These methods include; discovering an alternative key pair hashing for the victim's node address, performing a Global Time-Memory Trade-Off attack, and carrying out DoS attacks on the verification process. In this section we will discuss, in more detail, these attacks and possible approaches that may be used to mitigate them. The proposed mitigation approaches come solely within the CGA domain without needing to rely on other deployments.

A. Discover an Alternative Key Pair Hashing of the Victim's Node Address (Second Pre-image Attack)

In this case an attacker would have to find an alternate key pair hashing of the victim's address. The success of this attack will rely on the security properties of the underlying hash function, i.e., an attacker will need to break the second pre-image resistance of that hash function. The standard CGA, RFC 3972, proposes the use of SHA-1 which may be vulnerable to collision attacks [6]. The RFC 4982 [7] analyzes the implications of attacks against hash functions and updates the CGA's specifications in support of multiple hash algorithms. We thus recommend the use of an alternative hash function instead of SHA-1, such as SHA-256 which has proven to be safe against a collision attack.

The attacker will perform a second pre-image attack on a specific address in order to match other CGA parameters with Hash1 and Hash2. When the underlying hash function has no weaknesses, the following equation, authored by Bos, Özen, and Hfubaux [8], can be used to define the cost (T_A: hash function evaluation) of the attack:

$$T_A = \begin{cases} 2^{59} & \text{if Sec} = 0, \\ (2^{59}+1)2^{16\times Sec} & \text{if } 1 \leq \text{Sec} \leq 3, \\ (2^{16\times Sec}+1)2^{59} & \text{if } 4 \leq \text{Sec} \leq 7. \end{cases} \quad (1)$$

It is clear that the strength of the CGA depends on the *Sec* value. If the user uses a sufficient security level, it will be not feasible for an attacker to carry out this attack due to the cost involved. On the other hand, a large *Sec* value may lead to significant and undesirable address generation delay. We found that for a *Sec* value 2, the CGA address computation takes several hours, on average, on a computer with 2.67 GHz CPU speed, but it can take several days. Therefore users will probably opt for a *Sec* value of 0 or 1. They will accept that "good enough security" is better than very strong security with a cost of waiting for days or years to achieve this level of security. One proposed modification to the standard CGA in order to limit the time that CGA generation may takes is proposed by AlSa'deh et al. [9]. The authors modified the CGA generation algorithm to take the time that the user is willing to wait for CGA generation and determine the *Sec* value as an output of Hash2 computation.

B. Find the Victim Node's Private Key

Another possible method an attacker can use is to find the private key for a given public key. In this case the attacker will be able to impersonate the CGA address and forge signatures. Here we will not talk about breaking the RSA by factoring the public key modulus because we assume that the RSA is strong enough as long as an appropriate key length is used. Our focus here is on the situation where the attacker exploits an insecure, specific implementation of the RSA. For instance, if the private key is stored in an insecure place, an attacker may discover it. In this case choosing a long key size will not guarantee that the RSA scheme is safe.

We propose the generation of the key pairs automatically inside the CGA code so that the keys are not stored in a particular path before starting the CGA generation. After generating the CGA, the key pairs are stored in the device's memory (RAM) for quick accessibility. In this manner the node is forced to regenerate a new key pair after rebooting the system. The recommended default key length for the RSA is 1024 bits [10]. The user will, however, be able to change the default value of the CGA generation input parameters.

C. Global Time-Memory Trade-Off Attack

As shown by Bos et al. [8], CGAs are vulnerable to global Time-Memory Trade-Off (TMTO) attacks. The attacker needs to do an exhaustive search for hash collision or create a large pre-computed database of interface IDs from the attacker's own public key(s) which are used to find matches for many addresses. Therefore, they proposed a more secure version, called CGA++, to resist this type of attack. In CGA++, the *Subnet Prefix* is included in the calculation of Hash2, and then the *Modifier*, *Collision Count,* and *Subnet Prefix* values are signed by the private key corresponding to that public key. In this way TMTO cannot be applied globally. The attacker would have to do a brute-force search for each address prefix separately.

However, we believe that the global Time-Memory Trade-Off attack against CGA is not an easy attack due to the practical problems in carrying it out. It is not easy to impersonate a random node in a network because a large amount of storage is required in order to carry this attack out. For a network of the size 2^{16}, the attacker would need to have 128 terabytes of storage [8]. The CGA++ enhancement also comes with a new, additional signature. This new signature adds new additional cost due to the number of signature generations/verifications necessary and the size of the attached signature to each message. CGA++, therefore, requires much more computational time than that for the generation of a CGA using the same *Sec* value. CGA++ does enhance the security of the CGA global address against a TMTO attack but it does not solve the problem with local link addresses because the local link prefix is the same for all subnets. To obtain a more compact CGA++, the work in [11] adopts the Elliptic Curve Cryptograph (ECC) keys in CGA++ instead of standardized RSA keys in order to minimize the size of CGA parameters and reduce CGA generation time.

The probability of success for the attacks outlined above in subsection III.A to C is quite small, but not zero, and the cost of the attacks may exceed the benefits gained from them. For instance, the attacker may need to do a brute-force search for a long period of time using a powerful processor in order to break the second pre-image resistance. Also, for a TMTO attack the attacker would need a very large storage capacity. However there are still other ways to compromise CGA nodes as will be shown in the following subsections.

D. Denial-of-Service (DoS) Attack Against the CGA Verification Process

An attacker can conduct DoS attacks on some particular steps within the CGA verification process. He can perform a DoS attack against the DAD check and the CGA parameter verification.

1) DoS Attack Against the DAD-CGA

It is well known that the DAD algorithm defined in IPv6 is susceptible to a DoS attack as was stated in section 4.1.3 of RFC 3756 [12]. Each time the victim's node performs a DAD on a tentative address, an attacker can reply by saying that the address is already in use. Thus the victim's node will be unable to configure the IP address so that it can join the network.

It was proposed that to counter this type of attack the CGA process use signed DAD and Neighbor Advertisement (NA) messages [2]. If the NA message sent in response to a DAD does not fulfill the validity check, then the verification fails and the node performing the DAD discards that response. The verifier checks the validity of both the CGA parameters and the signature.

However, it is still possible to conduct the DoS attack on DAD in order to prevent a CGA node from joining a link. If the attacker replays a NA message in order to gain a valid signature, then this security protection will be void. In this way the attacker does not need to do any brute-force searches against the CGA in order to carry out this attack. Below is the detailed description of this type of attack.

A CGA node (victim) that wants to configure a CGA will generate a DAD and send it over the local link. When an attacker receives this packet he can immediately copy the IP address, CGA parameters, and signature. Then he replies with an NA message that has the same valid signature and CGA parameters. The victim's node receives the reply via a secure NA message and checks its validity as defined by the CGA verification algorithm. If it is valid, because the signature is valid and the reply was received in the allotted time frame, then the victim will increment its *Collision Count* value, and try another address. If the attacker replies again and the *Collision Count* reaches 2, the victim stops and reports an error. Even though this attack is limited to DoS attacks because the attacker does not have the private key needed to sign messages, the attacker can prevent the CGA address configuration for all new nodes that want to attach to the local link.

There are some constraints on the attacker's ability to carry out the above attack. First, the attacker needs to have access to the link resources and must be able to listen to multicast packets sent by the victim's node. Second, the attacker needs to conduct the attack within a short period of time. The attacker's response should be fast enough, i.e., to replay the packet and send it before the end of the DAD process. RFC 4862 [13] specifies this delay by using a *RetransTimer* variable. The default value for the *RetransTimer* is 1,000 milliseconds.

One possible solution would be for the host to discard the same DAD, i.e., the DAD with the same tentative address, CGA parameters, and signature that was sent before. The probability that two nodes would generate the same RSA key pair is very small [14]. Having the same public key and the same CGA parameters is a clear indication of an attack. Therefore, if a node receives a CGA protected message with the same CGA parameters and signature as its own, the node assumes this message was sent by an attacker and discards it.

In the case of receiving a DAD message from a non-CGA (unsecured) node or from a tentative address with different CGA parameters and signature, one heuristic solution is to trust the first three DAD failures that occurred with a specific node in a given link layer. After that, ignore any DAD failures and consider the other node as a malicious node and use tentative address in DAD process as a valid address.

The probability that two legitimate CGA nodes will generate the same interface ID is very low. The following formula, which was defined by Bagnulo, et al. [15], defines the probability of having at least two nodes generate the same interface ID:

$$Pb(n,k) \le 1 - \left(\frac{n-k+1}{n}\right)^{k-1} \qquad (2)$$

Where CGA addresses, $n=2^{59}$ and k is the number of interfaces in the same link. For a large subnet with one hundred thousand (100,000) interface IDs, the $Pb(2^{59},100000) <=1.7e\text{-}08$. Accordingly, this very small probability makes the heuristic approach reasonable. Receiving three NA messages as a response to DAD is a strong indication of malicious activity.

The CGA DAD verification process can be extended as follows. When a DAD is detected, the node should check to see whether or not the NA message contains the exact CGA parameters as those used in the NS message that it sent. If the message received as a response to the NS DAD contains the same CGA parameters and signature as its own, then discard this message and consider the tentative address as a valid address and start using it. When the received response to the NS DAD comes from a non-CGA, then the node generates another tentative CGA. If after three consecutive attempts a non-unique address is generated, the CGA node will consider it as an attack and will discard the NA message from that non-CGA node and will start using this tentative address. Figure 2 depicts a flowchart of DAD extension used to eliminate the DoS in the CGA-DAD process.

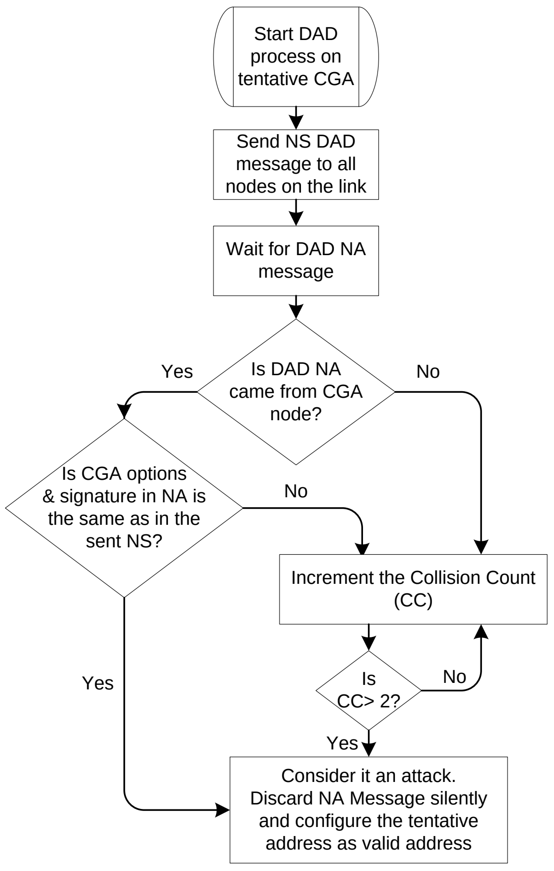
Start DAD process on tentative CGA
Send NS DAD message to all nodes on the link
Wait for DAD NA message
Is DAD NA came from CGA node?
Yes
No
Is CGA options & signature in NA is the same as in the sent NS?
No
Yes
Increment the Collision Count (CC)
Is CC> 2?
No
Yes
Consider it an attack. Discard NA Message silently and configure the tentative address as valid address

Figure 2. The flowchart of a DAD extension used to eliminate a DoS attack in the CGA-DAD

2) DoS Attack by Replaying the Sender CGA Parameters

CGA is vulnerable to a replay attack where an attacker can sniff and store signed messages from the victim's node and replay them later. It is easy for an attacker to conduct a DoS attack against a CGA-enabled host by replaying the sender's CGA parameter. The CGA-enabled receiver is required to calculate Hash1in order to verify the interface ID of the sender. This verification requires that the sender sends its CGA parameters to the receiver. If the attacker modifies any of the CGA parameters, Hash1 will fail. It would be easy for the attacker to modify the *Collision Count* so that it exceeds 2 thereby making the verification process fail. In this way the attacker can prevent the communication between a CGA-enabled sender and receiver.

The use of the *Timestamp Option* of SEcure Neighbor Discovery (SEND) [10] eliminates the possibility of a replay attack. We therefore recommend the use of this option with CGA even when CGA is deployed alone. The verifier will thus not receive two messages with the same signature since two successive messages from the same node will have a different timestamp and thus different signatures. Therefore, we suggest not using CGA as a lone option. The *Timestamp option* must be used with CGA. Details describing these options can be found in RFC 3972.

3) DoS to Kill a CGA Node

Sending a valid or invalid CGA signed message with high frequency across the network can keep the destination node(s) busy with the verification process. This type of DoS attack is not specific to CGA but it can apply to any request-response protocol. One possible solution to mitigate this attack is to add a controller at the verifier side to determine the maximum number of messages that the receiver can accept within a certain period of time from a specific node. If this threshold rate is exceeded, the receiver drops the new incoming messages from that node.

E. CGA Privacy Implication

Due to the high computational complexity necessary for the creation of a CGA, it is likely that once a node generates an acceptable CGA it will continue to use it at that subnet. The result is that nodes using CGAs are still susceptible to privacy related attacks. Using the same address for a long time makes it possible for an attacker to violate the users' privacy by tracking an individual's device(s) online. In practice, a lot of devices (e.g., laptops, cell phones, etc.) are associated with individual users. Therefore, changing the CGA should be done often enough to prevent the attacker from collecting enough information about the node to mount an attack.

We think that the CGA privacy implication can be resolved by setting a lifetime (the length of time the address can be used) for a CGA address. When this time has elapsed, a new CGA, with a new CGA parameter, should be generated. We also propose that the key generation be included in the CGA code. This will force the node to generate a new address when it is rebooted or moved to a new location. When using changeable CGA addresses, it makes no sense to select a high security parameter (*Sec*). There should be a balance between the CGA lifetime and the security level. We do not recommend the use of a *Sec* value greater than 1.

To protect the users' privacy we propose changing the CGA addresses periodically. These temporary CGA addresses would be used for a certain period of time (hours to days as recommended in RFC 4941 [16]) and would then be deprecated. Deprecated addresses can continue to be used for connections that are already established, but they are not to be used to initiate new connections. Once a CGA address is deprecated, a new CGA should be used.

The lifetime of a temporary CGA address depends on several parameters and actions. For instance, the lifetime will be dependent on the time needed for a host to generate a new CGA address (T_G), the time needed for an attacker to break the CGA address (T_A) and a user desired setting for security and privacy. The following lists the conditions under which a new temporary CGA address should be generated:

- When a host joins a new subnet. In this case, the new CGA parameters will be used to generate the new address. A new public key will be used for calculating both the Hash1 and Hash2 values.
- Before the lifetime for the in-use CGA address has expired. To ensure that the CGA address is always available and valid, new CGAs should be generated in advance before the predecessor is deprecated. In practice, a valid time should not be zero. We recommend a minimum value for a lifetime to be one hour.
- If the prefix has expired, a new CGA address will need to be generated and it must include the newly received prefix in the Hash1 calculation.
- When the user needs to override the default value in order to generate a new CGA address. The CGA implementation should offer the user the ability to override the default values and force the CGA algorithm to generate a new address.

The lifetime of a CGA address (T_l) should be safe enough so that the attacker is not able to impersonate the other nodes' addresses. We recommend that T_A be at least nT_l (where n is an integer) in order to have a safe margin. Clearly, the speed of hash function computation depends on the CPU speed of the computing device. Reading the CPU speed by using the CGA code makes it possible to determine whether or not the selected lifetime is suitable. On the other hand, the T_l time should be greater than the time required for the node to generate a CGA address. It is not feasible to invest the time and resources of the computing device to create an address and then, after a very short period of time, deprecate this address. We recommend that T_l be greater than mT_G (where m is an integer). Therefore, T_l can be described by the following equation:

$$mT_G \le T_l \le \frac{T_A}{n}, \qquad (3)$$

Where m and n are integers

IV. Evaluation and implementation

According to RFC 4861 [17], the maximum time a node holds an IP address of neighboring nodes in its neighbor cache is 30 seconds. This time is defined by a constant called REACHABLE_TIME. Based on REACHABLE_TIME and the CGA message verification time ($T_{Verification}$), the attacker can determine the number of ND messages (N_{ND}) necessary for the generation of a DoS attack against the CGA verification process. Formula 4 calculates the required number of packets for the generation of this DoS attack.

$$N_{ND} \geq \frac{\text{REACHABLE\_TIME}}{\text{T}_{\text{Verification}}} \tag{4}$$

To carry out this attack against the CGA verification process, we implemented a small program that calls NA generation and verification functions and then writes the average time to execute these functions to a file. Table I shows the average (Avg.), minimum (Min.), and maximum (Max.) times, in milliseconds, required for NA packet generation and for the verification process of a NA message. We obtained this data by running the NA packet generation 100 times.

TABLE I. NA GENERATIONAND VERIFICATION FOR 100 SAMPLES

NA generation time (milliseconds)			*NA verification time (milliseconds)*
Avg.	Min.	Max.	Avg.
51.35	5	221	12.5

By referring to formula 4 and Table I, one can see that the attacker needs to generate a minimum of 2400 NA messages with different CGA values before he starts sending them over the network in sequential order within an infinite loop. The attacker can do this before starting his DoS attack against the verification process on the victim node. This prevents the victim node from detecting old NA messages and thus for each message a new verification process would be started. The victim node first checks whether or not that address exists in its neighbor cache. If the address does not exist, the victim node starts the verification process to check whether or not the node is authorized for that address. Then, if the verification is successful, it adds that IP address to its neighbor cache. Based on our experimental results, the victim node CPU is kept busy processing the attacker's packet and, then, finally, the victim's buffer will overflow making it impossible to process more packets.

We evaluated the aforementioned attack in a test scenario on a computer with three Virtual Machines (VMs) running windows 7 as a guest Operating System (OS). All VMs are in the same local network and have 2 GB RAM with a 2.6 GHz CPU. All the VMs use the default values of the CGA parameters, i.e., RSA key size 1024 and *Sec* value 1. VM1 plays the role of the attacker. It runs our small application that generates different CGA values and sends NA messages in order to execute DoS attacks against the CGA verification process on the victim node. VM2 runs a simulated router that can send RA messages. VM3 is the victim node.

In our tests we extended the standard CGA to mitigate DoS and privacy-related attacks. We modified the CGA function of our WinSEND implementation [18]. WinSEND is a user space SEND implementation for the Windows family of software. It has all the functionality of SEND and can be easily installed on systems. We modified the CGA part and disabled the other SEND options in WinSEND in order to focus on CGA attacks.

The modified version of WinSEND uses the parameters necessary to generate a temporary CGA address as the default. The default value for the minimum valid lifetime is up to one day, the public key size is 1024-bit, and the default value for *Sec* is 1. The user can override these parameters because such default values may not always fits with the users requirements.

V. Conclusion

Cryptographically Generated Addresses (CGAs), as defined in RFC 3972, offer a means of authenticating the identity of communication nodes in a network. This is accomplished by finding the relationship between the addresses and public keys. In CGA, the interface ID is a cryptographic one-way hash of the node's public key and other parameters.

In this study we have analyzed the security of and threats to the standard CGA. Several types of attacks and their countermeasures are explained. We found that CGAs are still vulnerable to some types of attack, such as DoS and replay attacks. We thus proposed extensions to RFC 3972 in order to eliminate the standard CGA's vulnerability to the kinds of attacks that we uncovered. Even though DoS attacks against CGA were known there were no countermeasures proposed by other researchers. We suggest the use of extensions and enhancements to the CGA verification process in order to defeat the DoS attack against the DAD algorithm. We also propose the use of the *Timestamp Option* within CGA when it is run alone and not as a part of SEND. CGA may also be susceptible to privacy related attacks. Specifying a lifetime for a CGA address can resolve this privacy issue. However, this approach definitely involves tradeoffs between privacy and security, but it is a very viable solution. The new extensions can easily be added to the original CGA without affecting its performance or its security level. We will write these extensions as an update to RFC 3972.

References

[1] T. Aura, "Cryptographically Generated Addresses (CGA)," in *Information Security*, vol. 2851, C. Boyd and W. Mao, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 29–43.

[2] T. Aura, "Cryptographically Generated Addresses (CGA)," RFC 3972, *IETF*, Mar-2005. [Online]. Available: http://tools.ietf.org/html/rfc3972.

[3] G. O'Shea and M. Roe, "Child-proof authentication for MIPv6 (CAM)," *SIGCOMM Comput. Commun. Rev.*, vol. 31, no. 2, pp. 4–8, Apr. 2001.

[4] P. Nikander, "Denial of Service, Address Ownership, and Early Authentication in the IPv6 World," in *Security Protocols*, vol. 2467, B. Christianson, J. Malcolm, B. Crispo, and M. Roe, Eds. Springer Berlin / Heidelberg, 2002, pp. 22–26.

[5] G. Montenegro and C. Castelluccia, "Statistically Unique and Cryptographically Verifiable (SUCV) Identifiers and Addresses," in *In Proceedings of the 9th Annual Network and Distributed System Security Symposium (NDSS)*, 2002.

[6] X. Wang, Y. L. Yin, and H. Yu, “Finding collisions in the full SHA-1,” in *Proceedings of the 25th annual international conference on Advances in Cryptology*, Berlin, Heidelberg, 2005, pp. 17–36.

[7] M. Bagnulo and J. Arkko, “Support for Multiple Hash Algorithms in Cryptographically Generated Addresses (CGAs),” *IETF*, RFC 4982, Jul-2007. [Online]. Available: http://tools.ietf.org/html/rfc4982.

[8] J. W. Bos, O. Özen, and J.-P. Hubaux, “Analysis and Optimization of Cryptographically Generated Addresses,” in *Proceedings of the 12th International Conference on Information Security*, Berlin, Heidelberg, 2009, pp. 17–32.

[9] A. Alsa’deh, H. Rafiee, and C. Meinel, “Stopping time condition for practical IPv6 Cryptographically Generated Addresses,” in *2012 International Conference on Information Networking (ICOIN)*, 2012, pp. 257 –262.

[10] J. Arkko, J. Kempf, B. Zill, and P. Nikander, “SEcure Neighbor Discovery (SEND),” *IETF*, RFC 3971, Mar-2005. [Online]. Available: http://tools.ietf.org/html/rfc3971.

[11] A. AlSa’deh, F. Cheng, and C. Meinel, “CS-CGA: Compact and more Secure CGA,” in *17th IEEE International Conference on Networks (ICON)*, 2011, pp. 299 –304.

[12] P. Nikander, J. Kempf, and E. Nordmark, “IPv6 Neighbor Discovery (ND) Trust Models and Threats,” *IETF*, RFC 3756, May-2004. [Online]. Available: http://tools.ietf.org/html/rfc3756.

[13] S. Thomson, T. Narten, and T. Jinmei, “IPv6 Stateless Address Autoconfiguration.” *IETF*, RFC 4862, Sep-2007 [Online]. Available: http://tools.ietf.org/html/rfc4862.

[14] A. K. Lenstra, J. P. Hughes, M. Augier, J. W. Bos, T. Kleinjung, and C. Wachter, “Ron was wrong, Whit is right,” . IACR Cryptology ePrint Archive 2012: 64, 2012.

[15] M. Bagnulo, I. Soto, A. Azcorra, and A. Garcia-Martinez, “Random generation of interface identifiers,” *IETF*, Jan-2002. [Online]. Available: http://tools.ietf.org/html/draft-soto-mobileip-random-iids-00.

[16] T. Narten, R. Draves, and S. Krishnan, “Privacy Extensions for Stateless Address Autoconfiguration in IPv6,” *IETF*, RFC 4941. [Online]. Available: http://tools.ietf.org/html/rfc4941.

[17] T. Narten, E. Nordmark, W. Simpson, and H. Soliman, “Neighbor Discovery for IP version 6 (IPv6),” *IETF*, RFC 4861, Sep. 2007. [Online]. Available: http://tools.ietf.org/html/rfc4861.

[18] H. Rafiee, A. AlSa’deh, and Ch. Meinel, “WinSEND: Windows SEcure Neighbor Discovery”, 4th International Conference on Security of Information and Networks (SIN 2011), 14-19 November 2011, Sydney, Australia, pp.: 243-246, November 2011.