Privacy is Not an Option: Attacking the IPv6 Privacy Extension

Johanna Ullrich ✉, Edgar Weippl

SBA Research, Vienna, Austria
Email: (firstletterfirstname)(lastname)@sba-research.org

Abstract. The IPv6 privacy extension introduces temporary addresses to protect against address-based correlation, i. e., the attribution of different transactions to the same origin using addresses, and is considered as state-of-the-art mechanism for privacy protection in IPv6. In this paper, we scrutinize the extension's capability for protection by analyzing its algorithm for temporary address generation in detail. We develop an attack that is based on two insights and shows that the notion of protection is false: First, randomization is scarce and future identifiers can be predicted once the algorithm's internal state is known. Second, a victim's temporary addresses form a side channel and allow an adversary to synchronize to this internal state. Finally, we highlight mitigation strategies, and recommend a revision of the extension's specification.

1 Introduction

Snowden's revelations on the National Security Agency's surveillance program startled the global public due to its sheer extent and sophistication. Practically everbody's Internet communication is collected. The gained data is filtered, analyzed, measured and finally stored for the purpose of compounding a precise picture of Internet users [1, 2]. But other actors are also after massive amounts of user data: Western democracies, e. g., in the European Union or Australia, often introduce telecommunication data retention. Commercial enterprises spy on their customers on a massive scale to increase monetary revenue [3, 4], and criminals may do so as well.

The power of such an approach lies in its capability of making sense from large amounts of data that seem unrelated to each other by combing countless pieces of information [5]. This means that a person's different activities on the Internet can be correlated to each other, and this condensed information typically exceeds what people believe can be found out about their lives. Addresses play a sensitive role in this: On the one hand, an address has to accurately identify the receiver so that traffic reaches its intended destination. On the other hand, address-based correlation enables the attribution of different transactions to the same origin and allows to gain insights into others' Internet behavior. General protection strategies against correlation like an attribute's removal or its encryption seem inadequate for addresses as intermediate nodes require access for appropriate data delivery.

Addressing, in turn, is heavily dependent on the protocol, and IPv6 introduced new aspects in the matter of address-based correlation. Initially, all addresses of an interface were defined to include a globally unique identifier and thus allowed simplest address correlation over an interface's full lifetime [6]. In response, temporary addresses that change by default every 24 hours were introduced. This mechanism is known as the privacy extension [7], and is considered as state-of-the-art privacy protection in IPv6 [8]. It is implemented in major desktop and mobile operating systems.

In this paper, we scrutinize the IPv6 privacy extension's capability of protecting against address-based correlation, and therefore focus on the algorithm for temporary address generation. We find that once the algorithm's state is known by an adversary, she is able to accurately predict a victim's future addresses. Beyond that, we develop a way that allows an adversary to synchronize to the victim's state by exploiting observed temporary addresses as a side channel, and appraise the attacker's effort to perform our attack with currently available technology. Our results yield 3.3 years of hashing but advances in technology are going to decrease this time period. We highlight mitigation strategies; however, our most important contribution may be the impetus for a revision of the extension's specification.

The remainder of the paper is structured as follows: Section 2 provides details on addressing in IPv6 and the privacy extension. Section 3 summarizes related work focusing on privacy implications of competing IPv6 addressing standards as well as known vulnerabilities of the privacy extension. Section 4 describes the assumed attack scenario and is followed by a security analysis of the extension's address generation algorithm that identifies four weaknesses in Section 5. Based on these insights, the development of our attack is described in Section 6. Its feasibility is discussed in Section 7, which is followed by an investigation of current operating systems' vulnerability in Section 8. Strategies for mitigation are presented in Section 9, and Section 10 concludes the paper.

2 Background

This section provides background on IPv6 addressing in general: the address structure, address assignment and their implications for address-based correlation. In a second step, we focus on the IPv6 privacy extension and describe its principal idea as well as its algorithm for temporary interface identifier generation.

IPv6 Addressing: IPv6 addresses have a length of 128 bit and are portioned into two distinct parts of equal size as depicted in Figure 1. The first 64 bits form the network prefix, and are dependent on a host's location in the network. The remaining 64 bits form the interface identifier (IID) that enables a subscriber's identification on the link. Address configuration for clients is done via stateless address autoconfiguration [9] and does not require human intervention: Routers advertise the network prefix on the network, and hosts form their global IPv6 addresses by combining the announced prefix with a self-generated interface identifier.

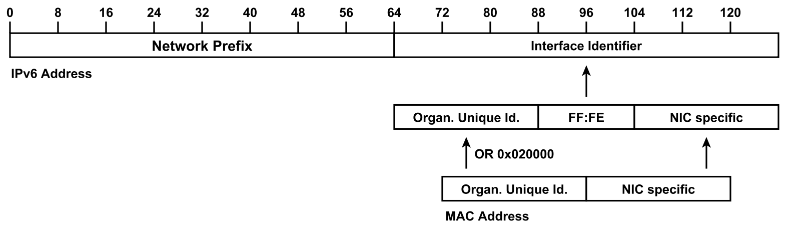


Fig. 1: IPv6 addresses using interface identifiers in modified EUI-64 format

The interface identifier was initially intended to follow the modified EUI-64 format [6] that infers an interface identifier from the 48 bit media access control (MAC) address, see also Figure 1. The MAC address consists of a 24 bit organizationally unique identifier, and a network interface card (NIC)-specific part of equal size. A fixed pattern of two bytes is inserted between these parts and a universal/local bit is set to one in order to form the identifier.

The MAC address is globally unique and typically remains stable over a host's lifetime[1]. Consequently, the interface identifier that is included in every IPv6 address is globally unique and stable as well. All addresses of a certain host have the same second half, while their network prefix changes according to the visited location. An adversary is thus able to attribute various transactions to the same origin based on the identifier and trace a host's Internet behavior even beyond a certain sub-network. The adversary is further able to retrace a host's movement in the network as the included network prefixes allow localization.

The IPv6 Privacy Extension: The privacy extension is presented as a solution that impedes correlation "*when different addresses used in different transactions actually correspond to the same node*" [7]. Its basic principle are interface identifiers that change at a regular interval of typically 24 hours. Hosts form temporary IPv6 addresses from the announced prefix in combination with the current interface identifier, and change the IPv6 address with every newly generated identifier. An expired address is considered deprecated and not used for new connections, but still serves already active transactions.

A host's successive interface identifiers have to be chosen in a way that appears random to outsiders and hinders them in attributing different identifiers to the same origin. Thus the IPv6 privacy extension defines an algorithm for

[1] Technically speaking the MAC remains stable over the NIC's lifetime, but we suppose that personal computers, laptops, tablets and mobiles keep their NIC over their whole lifetime.

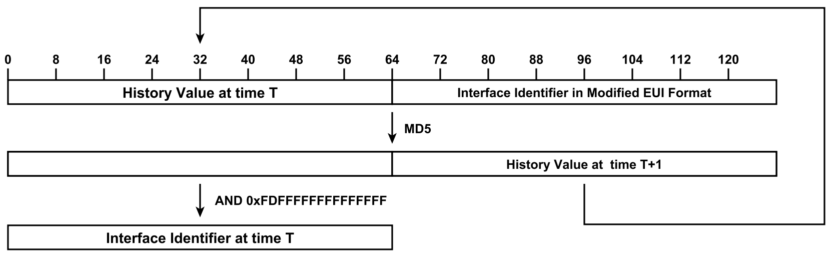

0
8
16
24
32
40
48
56
64
72
80
88
96
104
112
120
History Value at time T
Interface Identifier in Modified EUI Format
MD5
History Value at time T+1
AND 0xFDFFFFFFFFFFFFFF
Interface Identifier at time T

Fig. 2: Interface identifier generation according to the privacy extension

a pseudo-random generation of these temporary identifiers as described in the following and depicted in Figure 2:

1. A 64 bit history value is concatenated with the interface identifier in the modified EUI-64 format.
2. An MD5 digest is calculated over the concatenation of the previous step to gain a digest of 128 bit length.
3. The digest's leftmost 64 bits are extracted and bit 6 is set to zero in order to form the temporary interface identifier.
4. The digest's rightmost 64 bits form the next iteration's history value and are stored.
5. In case the generated interface identifier is found to be used by other local devices or reserved, the process is restarted to gain another identifier.

The very first history value is initialized with a random value the first time a system boots. This algorithm is defined for systems with present stable storage, which is necessary to keep the history value across system restarts. Devices like stationary PCs, laptops, tablets and smart phones are typically considered to have such storage. However, in its absence, it is allowed to randomly re-initialize the history value after every system restart.

Temporary IPv6 addresses are assigned in addition to stable addresses in modified EUI-64 format, and do not replace them in order to prevent negative impacts on addressing. Temporary addresses are used in outgoing connections to stay private, while stable addresses make it possible to stay reachable for incoming requests.

3 Related Work

Our research has a two-pronged foundation: First, we discuss various IPv6 address structures with respect to privacy, and highlight the IPv6 privacy extension's outstanding positions due to its capability to protect against geographical as well as temporal address-based correlation. This further emphasizes why the extension's secure standardization and implementation is an important aspect of IPv6 privacy. Second, we summarize previously discovered vulnerabilities of the privacy extension, and illustrate their minor importance in comparison to the new attack that we present in this paper.

3.1 IPv6 Address Formats and Address Correlation

There are ways to form IPv6 interface identifiers for stateless address autoconfiguration beyond the modified EUI-64 format and the privacy extension: (1) manually configured stable identifiers, (2) semantically opaque identifiers [10] and (3) cryptographically generated addresses (CGAs) [11]. CGAs, however, require authenticated messages as defined by Secure Neighbor Discovery (SeND) [12] instead of plain Neighbor Discovery [13].

We discussed these alternatives with respect to an adversary's capability for address correlation, and consider two distinct aspects of address correlation:

- Temporal correlation refers to address-based correlation over multiple sessions of a stationary host.
- Geographical correlation refers to address-based correlation over multiple sessions of a mobile node.

The difference is the network prefix: A stationary host stays in the same sub-network and includes the same network prefix in all its addresses. A mobile node wanders and changes the network prefix when moving.

Addresses using the modified EUI-64 format include the globally unique MAC address, and all of a host's addresses are equivalent in their second part. This fact allows the correlation of multiple sessions of a stationary or mobile node, i. e., this type of address is vulnerable to both forms of address correlation and, beyond that, also for active host tracking [14, 15]. Apart from global uniqueness, the same is valid for (manually configured) interface identifiers that remain static.

Semantically opaque interface identifiers are generated by hashing the network prefix and a secret key among other parameters. As the hash calculation includes the address prefix, the interface identifier changes from subnet to subnet and prevents geographical correlation. The identifier, however, remains stable in a certain network, even when returning from another network, and allows temporal correlation over long periods of time. Due to their recent standardization their availability in current operating systems is limited.

Cryptographically generated addresses are generated by hashing the public key and other parameters and are bound to certain hosts. Ownership is verified by signing messages that originate from this address with the corresponding private key. The network prefix is included as a parameter into hashing, and a node's CGA changes from network to network, preventing geographical correlation of traffic. However, their generation comes at high computational costs, and prevents address changes as a means of protection against temporal correlation in practise [16]. An approach to overcome the limitation with respect to frequent address change has been proposed [17]. However, CGAs and SeND lack acceptance and are neither widely implemented nor deployed.

	Modified EUI-64	Stable (Manual)	Sem. Opaque Id.	CGA	Privacy Extension
Temporal Correlation	-	-	-	-	✓
Geographical Correlation	-	-	✓	✓	✓

Table 1: IPv6 address formats with respect to their capability of protecting against different forms of address correlation

Mac OS X Yosemite	✓	✓	-	-	✓
Ubuntu 14.10	✓	✓	-	-	✓
Windows 8.1	✓	✓	-	-	✓

Table 2: IPv6 address formats with respect to their native availability in current client operating systems

The discussion is summarized in Table 1, and is accompanied by the capabilities' native availability in the current client operating systems Mac OS X Yosemite, Ubuntu 14.10 (Utopic Unicorn) and Windows 8.1, see Table 2. The results emphasizes the unique position of the privacy extension: First, it is the only mechanism using stateless address autoconfiguration that is currently deployed at a larger scale that is intended to protect against traffic correlation. Second, it is the only mechanism that considers protection against temporal as well as geographical address correlation.

In this paper, we develop an attack that overcomes the belief that the privacy extension provides adequate protection against address correlation. The attacks leaves a gap that cannot be filled by another address mechanism, and highlights the importance of revisiting the extension's current definition.

3.2 Known Vulnerabilities of the Privacy Extension

Drawbacks of the IPv6 privacy extension were discussed before, and follow two principal directions. First, its design does not impede active tracking, e. g., by using ping. Temporary addresses are assigned in addition to stable ones, and an adversary can still actively probe multiple subnets for a certain interface identifier in order to trace a host's movement. The respective specification, however, explicitly states its intention to protect solely against passive eavesdroppers, and not against active adversaries [7].

Second, shortcomings in the extension's protection against address correlation are known. A node does not have to change its interface identifier when moving to a new network prefix. Thus, tracking a host's movement remains feasible within an identifier's lifetime of typically 24 hours [14, 18]. For mitigation, the inclusion of the network prefix into the interface identifier calculation was proposed [18]. The respective specification also allows the change of an identifier in such a situation [7]. Our attack supports the second direction, and highlights that adversaries are able to perform address correlation even when the privacy extension is used. In comparison to known attacks, our attack cannot be fully mitigated within the specification's limitations.

4 Attack Scenario

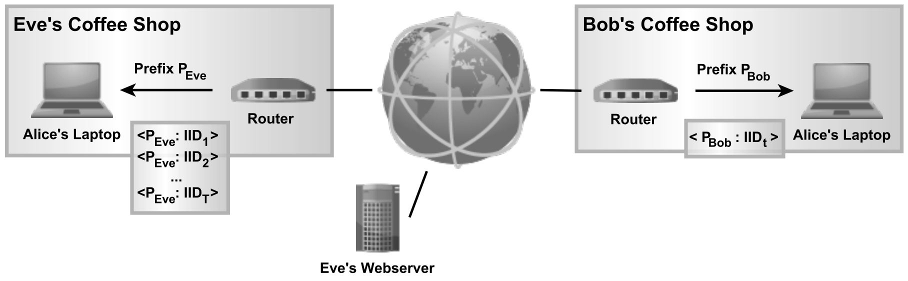


Fig. 3: Attack Scenario

Our attack scenario is depicted in Figure 3 and assumes full IPv6 deployment. We assume three stakeholders named Alice, Bob and Eve. Alice loves coffee, and regularly visits coffee shops. Then, she brings her laptop with her, and uses the offered Internet access to read mails or to chat. Bob and Eve each run a coffee shop, and provide Internet access to their guests. They deployed stateless address autoconfiguration, and their routers advertise the respective IPv6 network prefix so that customers are able to configure their global IPv6 addresses by connecting the prefix with their self-generated interface identifiers. Bob's router advertises the prefix P_{Bob}, Eve's router advertises P_{Eve}. Eve further runs a webserver to advertise current offers. She records her coffee shop's local traffic, and logs visits to her webserver.

Alice visits Eve's coffee shop for T successive days[2], and connects her laptop to the coffee shop's local network. Eve's router advertises P_{Eve}, and Alice's laptop configures a stable IPv6 address from this prefix and the stable interface identifier. Alice has enabled the IPv6 privacy extension, and thus temporary

[2] Although the T days do not necessarily have to be successive, we claim so here for better readability. In case days are missing, e. g., due to weekends, one simply has to consider these gaps when calculating the current state.

addresses are created in addition to the stable address by combining the prefix with the interface identifier of the day. Alice's temporary addresses are $< P_{Eve} : IID_1 >, < P_{Eve} : IID_2 >, ..., < P_{Eve} : IID_T >$ for day $1, 2, ..., T$.

After T days, Alice stops going to Eve's coffee shop. On an arbitrary day t $(t > T)$, Alice visits Eve's competitor Bob. She connects her laptop to Bob's local network. Bob's router announces the prefix P_{Bob}, and Alice's laptop forms a stable identifier from this prefix. In addition, the privacy extension generates a temporary address $< P_{Bob} : IID_t >$. On this day, Alice visits Eve's website to check current offers and causes a respective log entry.

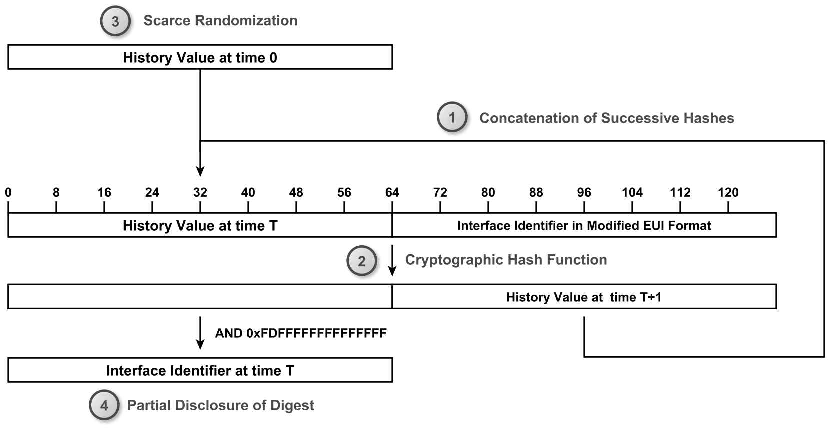
Eve is interested tracing her customers' activities, and wants to find out whether (1) Alice is still interested in her offers and visits the webserver, and whether (2) Alice is drinking coffee at a competitor.

We refer to this scenario in the remainder of the paper for illustration of our attack. This scenario was developed due to its representativeness for day-to-day life, but we are sure that there are plenty of alternative scenarios. The preconditions for an adversary are moderate: She has to gain a victim's MAC address and T successive interface identifiers that have been generated by the privacy extension. The MAC address is gained from local traffic as in the presented scenario, or inferred from the stable IPv6 address in case the latter is in modified EUI-64 format. Interface identifiers are included in the temporary addresses, and are inferred from there.

5 Security Analysis

In this section, we perform a manual security analysis of the privacy extension's algorithm for temporary interface identifier generation as defined in [7] and presented in Section 2. Our analysis revealed four striking characteristics that facilitate the prediction of future interface identifiers. While some of them might seem minor in isolation, their combination forms a reasonable attack vector as described in Section 6. In this section, we consider each characteristic separately: First, we describe the characteristic and highlight the specification's argumentation in its favor. Next, we infer implications on security. Figure 4 contrasts the algorithm for temporary address generation with the discussed characteristics; the depicted numbers are consistent with the following paragraphs.

(1) Concatenation of Successive Hashes: Interface identifiers are based on MD5 digests that are chained with each other because an iteration's result is partly included into the next hash calculation. The RFC states that *"In theory, generating successive randomized interface identifiers using a history scheme [...] has no advantages over generating them at random,"* [7] but claims an advantage in case two hosts have the same flawed random number generators. Performing duplicate address detection would make both hosts recognize their identical identifiers and trigger the generation of new identifiers. However, the flawed random number generators would again provide identical numbers, leading to identical identifiers. The presented algorithm is said to avoid this as the inclusion of the



3 Scarce Randomization
History Value at time 0
1 Concatenation of Successive Hashes
0 8 16 24 32 40 48 56 64 72 80 88 96 104 112 120
History Value at time T
Interface Identifier in Modified EUI Format
2 Cryptographic Hash Function
History Value at time T+1
AND 0xFDFFFFFFFFFFFFFF
Interface Identifier at time T
4 Partial Disclosure of Digest

Fig. 4: The privacy extension's characteristics impacting its quality of protection

(globally unique) interface identifier in modified EUI-64 format leads to different temporary interface identifiers in the next iteration.

It remains unclear why the inclusion of a globally unique identifier, e. g., in modified EUI-64 format, requires working with a history scheme, i. e., the concatenation of successive hashes. We believe that inclusion of a globally unique interface identifier and a random value into MD5 digest calculation is sufficient. It seems unlikely that sequences of equivalent random numbers result in successive collision in case a globally unique identifier is included into calculation.

The concatenation does not only appear dispensable with respect to the discussed aspect, but also negatively impacts the algorithm's quality of protection. Successive interface identifiers are dependent on each other, and today's state influences future identifiers. An adversary might exploit this to predict a victim's future identifiers.

(2) Cryptographic Hash Function: The privacy extension aims to create random-appearing interface identifiers, but states that pseudo-randomness suffices *"so long as the specific sequence cannot be determined by an outsider examining information that is readily available or easily determinable"* [7]. For the algorithm, MD5 with its adequate properties with respect to randomization has been *"chosen for convenience"* [7].

MD5 is considered broken, but a general dissolution would be an overshooting reaction: MD5 turned out to be prone to collisions that can be found within seconds on commodity hardware [19]. Pre-image attacks are still of high complexity and remain practically infeasible. The privacy extension uses MD5 for randomization, and neither relies on collision resistance nor pre-image resistance. Taking these considerations into account, the extension's choice of MD5 is justifiable.

MD5 is, however, a comparably fast hash function and the more hashes per second, the more feasible brute-force search becomes. This especially holds in combination with a limited input range. In 2012, a cluster of four servers hosting 25 off-the-shelf graphics processing units (GPU) achieved 180 Gigahashes per second [20], and time is usually in favor of the adversary as technology moves forward.

(3) Scarce Randomization: The RFC claims that *"To make it difficult to make educated guesses as to whether two different interface identifiers belong to the same node, the algorithm for generating alternate identifiers must include input that has an unpredictable component from the perspective of the outside entities that are collecting information"* [7].

Our analysis, however, identifies only scarce unpredictability in the algorithm for temporary address generation. Every iteration includes 128 bits into MD5 digest calculation:

- 64 bit of the former iteration's result, i. e., the remainder of the MD5 hash that was not used for the temporary interface identifier, and
- the 64 bit interface identifier in modified EUI-64 format. This identifier is not kept secret. An adversary might infer it from the stable IPv6 address that is assigned in addition to temporary addresses or from the MAC address. 17 bit of this identifier are fixed and thus the same for all nodes anyway.

In conclusion, there is no entropy added per iteration and this fact makes prediction of future identifiers easier as there are less possibilities. The only unpredictable component of the presented algorithm is the very first history value of 64 bit that should *"be generated using techniques that help ensure the initial value is hard to guess"* [7].

(4) Partial Disclosure of Digest: A temporary interface identifier is generated by taking *"the leftmost 64-bits of the MD5 digest and set bit 6 [...] to zero"* [7]. The gained interface identifier forms a temporary IPv6 address when combined with the current network prefix. The address is present in packets' address fields and accessible by others.

As a consequence, an eavesdropper gains 63 bit (one bit is overwritten with zero as mentioned above) of the calculated MD5 digest. This eavesdropped part does not present the algorithm's internal state, i. e., the history value, but both are part of the same MD5 digest. In conclusion, 63 bit of every iteration's MD5 digest is readily available to outsiders without any further processing effort and form a side channel of the algorithm's internal state. The algorithm leaks information but does not add entropy in an iteration.

6 Attack Design

We will now explain the steps of our attack in detail. We will include the characteristics that have been found in the security analysis of Section 5. In a first step, we will analyse the predictability of future addresses if the current state (history value) is known. As this turns out to be promising, we investigate methods to gain the current state. Finally, we summarize our full attack.

Predictability of Future Identifiers: For rather unambiguous prediction of future temporary identifiers, two requirements have to be met. First, future identifiers have to be dependent on the current state of the algorithm. Second, the calculation of the next identifier should include little randomness. The less random input, the better predictability.

We know from the previous section that both conditions apply to the IPv6 privacy extension: Interface identifiers are based on concatenated hashes. A part of the digest is used for the identifier, the other is called the history value and used as an input for the next calculation. An iteration's input is twofold – the mentioned history value and the interface identifier in modified EUI-64 format that is inferred from the MAC address. This means that there are no unpredictable components that are included. In conclusion, an adversary that is aware of the victim's history value and its MAC address is able to calculate the next temporary interface identifier according to the following recipe:

1. Infer the interface identifier in modified EUI-64 format from the victim's MAC address. This requires the insertion of a fixed pattern of two byte, and setting bit 6 as described in Section 2.
2. Concatenate the victim's current history value with the interface identifier in modified EUI-64 format generated in step 1.
3. Calculate the MD5 digest of the concatenation of step 2.
4. Extract the first 64 bits from the calculated digest and unset bit 6 to form the next temporary interface identifier.
5. Extract the remaining 64 bits from the digest and form the next history value.

This way an adversary is not only able to compute the next interface identifier, but all future identifiers by repeating the described steps. As a consequence, it seems worth developing methods to gain the algorithm's internal state.

Synchronization to the Current State: The internal state could be leaked, e. g., by means of malware, but this approach would imply an active adversary that does not simply eavesdrop. In the following paragraphs, we show that eavesdropping over a number of consecutive days is sufficient to gain the internal state: As described in Section 5, a temporary interface identifier that is included into an IPv6 address inherently discloses 63 bit of an iteration's MD5 digest. While the disclosed part is not the internal state, it is nevertheless related to the latter as both are clips of the same MD5 digest. The disclosed interface identifier can be considered a side channel of the internal state.

Figure 5 depicts a situation like our attack scenario from Section 4. The victim's very first history value is randomly initialized at day 0 and determines the history value and the temporary interface identifier of day 1; the history value

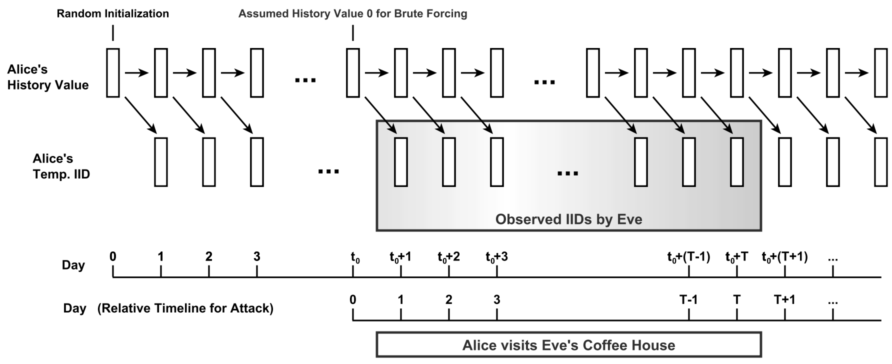


Fig. 5: Synchronization to current state

of day 1 in turn determines history value and temporary interface identifier of day 2 and so on. The randomly assigned history value at day 0 determines one of 2^{64} deterministic sequences that the victim's interface identifiers follow.

An adversary might probe all possible values for this history value at day 0, and compare the first half of the MD5 digest with the interface identifier of day 1. If they are equal this value might represent an appropriate candidate. As it is only possible to compare 63 bit of the MD5 digest, it is likely that numerous candidates remain. The adversary thus extracts the second half of the digest as a candidate for the history value at day 1, includes it in another iteration containing an MD5 calculation, compares the result with the interface identifier at day 2 and further shrinks the candidate set until a single candidate remains. Then, the adversary has identified the internal state.

It is, however, unlikely that an adversary observes the very first temporary addresses that a victim generates after its installation; an adversary rather observes an arbitrary sequence of T successive addresses starting at day $t_0 + 1$ as indicated in Figure 5. Due to the algorithm's structure, the adversary then assumes the history value at day t_0 to be randomly initialized without loss of generality. The adversary does not have to know the number of temporary addresses that the victim has generated before being recorded. For this reason, we added an relative time line for the attack in the figure for readability.

Composite Attack: Based on the attack scenario of Section 4, the gained insights of the previous paragraphs and Figure 5, we summarize Eve's steps towards predicting Alice's future identifiers.
On Alice's first visit at Eve's coffee shop on day 1, Eve has to perform the following steps:

– *Data Extraction from Traffic Records:*
Eve records the local traffic in her coffee shop, and is thus able to read Alice's

MAC address from Ethernet headers as well as her temporary IPv6 address. From this temporary IPv6 address, Alice extracts the last 64 bits that are the interface identifier for the first day IID_1.
- *Generation of Modified EUI-64 Interface Identifier:*
  Eve infers Alice's interface identifier in modified EUI-64 Format from the MAC address by inserting a fixed pattern of two bytes and setting bit 6 as described in Section 2. Alternatively, she might read the identifier in modified EUI-64 format directly from Alice's stable IPv6 address.
- *Reduction of Candidate Set:*
  Eve probes all possible values for the assumed initial history value at day 0, concatenates the value with the stable identifier in modified EUI-64 format, and calculates the MD5 digest. If the first part of the MD5 digest equals Alice's current temporary address[3], the remainder of the digest forms a candidate for the next iteration's history value and is added to the candidate set of the first day C_1. In this step, Eve reduces the initial candidate set C_0 of 2^{64} alternative sequences to a smaller set C_1 that is stored for the next day.

On every further visit of Alice at Eve's on subsequent days t with $1 < t \leq T$, Eve performs:

- *Data Extraction from Traffic Records:*
  Eve extracts today's temporary interface identifier IID_t from Alice by reading the traffic records.
- *Further Reduction of Candidate Set:*
  Eve probes all values for the history value that are present in yesterday's candidate set C_{t-1}, concatenates the values with the stable identifier in modified EUI-64 format, and calculates the MD5 digest. If the first part of the MD5 digest equals Alice's current temporary address IID_t, the remainder of the digest forms a candidate for the next iteration's history value and is added to the candidate set C_t. In this step, Eve further reduces the number of alternative sequences to a smaller set that is again stored for the next day.

This is performed whenever a new temporary address is available until a single candidate remains. This single candidate represents the algorithm's internal state, the history value, and allows to predict future addresses from now on. On every further day t with $t > T$, Eve is able to anticipate Alice's temporary interface identifier for this day:

- *Anticipation of Current Temporary Address:*
  Eve concatenates the history value of day T with the stable identifier in modified EUI-64 format and calculates the MD5 digest. She extracts the history value, and repeats the calculation with the new history value. In total, $(t - T)$ MD5 digest calculation are performed.

[3] The comparison is done on 63 different bits (0-5 and 7-63); bit 6 is always set to zero in temporary addresses, see Section 2.

– *Assemblage of the Interface Identifier:*
  Eve forms Alice's interface identifier IID_t from the first part of the last MD5 digest by setting bit 6 to zero.

With this knowledge, Eve is able to search her web server's logs for the calculated temporary identifier and attributes certain visits to Alice. At the same time, the prefix that the temporary identifier is concatenated with to form an IPv6 address provides information on the sub-network that Alice resided at the time of the page visit. If this is equivalent to Bob's assigned prefix, Eve is able to infer that Alice drank coffee at Bob's coffee shop.

7 Feasibility

In the previous sections, we identified weaknesses of the IPv6 privacy extension and developed an attack exploiting these characteristics. The question on the attack's practicability with respect to today's technology remains, and is discussed in this section. Three aspects have to be considered: (1) the minimum number of observed interface identifiers, i. e., the number of days that Alice has to visit Eve's coffee shop, (2) the expenditure of time for brute-forcing, and (3) the storage capacity to save the candidate set for the next day. Finally, a modified version of our attack for limited storage capabilities is presented.

Number of Address Observations: Alice has to visit Eve's coffee shop so often that Eve gains enough temporary identifiers for synchronization to the internal state. We assume that Alice generates one temporary address per day as recommended by the RFC [7], and an iteration of the attack corresponds to a day.

On the first day, Eve probes 2^{64} potential values for the history value and compares their MD5 digest to the observed interface identifier of Alice. The unequal ones are excluded, and the appropriate ones form the candidate set C_1 of potential values for the next day. The size of the candidate set is dependent on the ratio of candidates that Eve is able to reject per day. With p being this ratio, the size of the candidate set C_t for day t is calculated as follows

$$|C_t| = 2^{64} \cdot (1-p)^t \tag{1}$$

Eve has to repeat the explained step until a single candidate remains, i. e., $|C_t| = 1$, and the minimum number of days T_{min} is calculated as follows

$$T_{min} = ceil\ \frac{log(2^{64})}{log(p-1)} \tag{2}$$

The more candidates can be excluded per iteration, the less successive interface identifiers have to be known by Eve. If Eve is able to reduce the candidate set by only 50 % every day, the minimum number of days is 64. A reduction by 99 %, 99.99 %, 99.9999 % shortens this to 10, 5, 4 days.

Time Expenditure for Brute-forcing: Every iteration requires brute-forcing the current candidate set C_t, and means an MD5 digest calculation for every candidate. Assuming a hash rate r indicating the number of calculated hashes per second, the total time T_{Brute} for brute-forcing is calculated as follows

$$T_{Brute} = \frac{1}{r} \sum_{i=0}^{T_{min}} |C_i| = \frac{2^{64}}{r} \sum_{i=0}^{T_{min}} (1-p)^i \tag{3}$$

Assuming $1 - p < 1$[4], the equation is bounded as follows and allows an estimation of the total time expenditure for MD5 brute-forcing

$$T_{Brute} < \frac{2^{64}}{r} \sum_{i=0}^{\infty} (1-p)^i = \frac{2^{64}}{r} \cdot \frac{1}{p} \tag{4}$$

A hash rate of 180 G/s with MD5 is feasible [20]. The more candidates can be excluded, the less time is required. If Eve is able to reduce the candidate set on average by only 50 % every day, the time for brute-forcing remains 6.5 years, a reduction by 99 % shortens this to 3.3 years. Time expenditure appears high at the first sight, but time plays for the adversary, and advances in technology are likely to decrease this effort. It is likely that faster hashing is already feasible today as the given hash rate was measured at a cluster of 25 consumer GPUs back in the year 2012 and GPUs have recently experienced extensive advancement.

Storage of Candidate Set: Appropriate candidates for the history value have to be stored for the next iteration. The history value size is 8 byte, and the storage demand S_t is dependent on the size of the candidate set.

$$S_t = |C_t| \cdot 8\,byte = 2^{64} \cdot (1-p)^t \cdot 8\,byte \tag{5}$$

The following calculation considers the first iteration due to its worst case character[5]: If Eve is able to reduce the candidate set on average by only 50 % every day, the storage demand for the first iteration is 74 Exabyte, a reduction of 99 %, 99.99 %, 99.9999 % reduces the storage demand to 1.5 Exabyte, 15 Petabyte, 148 Terabyte.

This storage demand, however, can be circumvented by a modification of the attack. In our initial design of Section 6, Eve synchronized to Alice's state simultaneously to her coffee shop visits, but Eve might alternatively perform the attack retroactively. Therefore, she stores Alice's successive interface identifiers for T_{min} days before starting the attack. Instead of storing an appropriate candidate after the first iteration, she performs the second, third, etc. iteration with this candidate as long as it appears appropriate. Otherwise, it is rejected. This way the storage demand is reduced to a few bytes for execution of the algorithm for temporary interface identifier generation.

[4] p is the portion of candidates that can be excluded per iteration.

[5] The candidate set C_0 does not have to be stored as it contains all 2^{64} possible values.

8 Implementation in Operating Systems

In this section, we assess current operating systems that support the IPv6 privacy extension with respect to their individual vulnerability. We tested Mac OS X Yosemite, Ubuntu 14.10 (Utopic Unicorn) and Windows 8.1 Enterprise as representatives of the three major ecosystems on clients. In doing so, we faced the challenge that we cannot access the respective sources of all operating systems, and had to rely on the externally observable pattern of successively generated interface identifiers. A machine running an operating systems that implemented the privacy extension as described in the respective RFC has to generate the same sequence of successive interface identifiers whenever originating from a defined initial state. The sequence appears unchanged when faced with some external factors, while changing in dependence of other factors. The specific influencing factors are discussed later in this section.

For checking the stated premise, we created a setup of two virtual machines running in VMWare Workstation 11 and Fusion Pro 7. The machines were virtually connected for networking. One ran the tested operating system; we refer to this machine as the testee. To save time, we decreased the preferred lifetime on all operating systems and forced the generation of a new temporary address at an interval of twelve minutes. We finally created a snapshot of the testee that made it possible to return it to the initial state after every test. The testee generated temporary addresses after a router's announcement of a network prefix. The second virtual machine thus ran Ubuntu 14.10 simulating this router; to send ICMPv6 Router Advertisements the tool fake_router6 from the thc-ipv6 toolkit [21] was used. We recorded the temporary addresses of the testee by means of local scripts.

Using the above premise, we tested the operating systems for five criteria. First, repeating the test without any changes multiple times has to result in the same sequence of successive interface identifiers due to the algorithm's determinism. If this holds, the sequence is checked for their dependence on various influencing factors. The algorithm has to be invariant to time, the announced prefix as well as system restarts and provide the same sequence of identifiers, while it has to be variant to a change of the MAC address. These conditions are inferred from the algorithm's definition in the respective RFC: Neither the point in time of address generation is included into the calculation nor the identifier's lifetime. Thus, a later repetition of the experiment or a change in the interval may not have an impact on the identifiers. The same holds for the announced network prefix. The algorithm has to be invariant to system restarts as the current state has to be stored in stable storage; all the tested operating systems require the availability of such a storage. In contrast, the MAC address is included into the calculation, and its change should result in different identifiers. These are necessary criteria, and are not sufficient criteria. The results of our tests are shown in Table 3.

Ubuntu 14.10 does not generate deterministic sequences, and its temporary interface identifiers appear to be assigned by a random number generator without

	Deterministic Sequence	*Time-Invariance*	*Prefix-Invariance*	*Restart-Invariance*	*MAC-Variance*
Windows 8	✓	✓	✓	✗	✓
Ubuntu 14.10	✗				
Mac OS 10.10	✗				

Table 3: Temporary Address Characteristics wrt to different Operating Systems

following the defined algorithm. A review of the source code[6] supports this. Mac OS X Yosemite showed the same behavior.

Windows 8.1 provides the same sequence whenever originating from the same state, and further fulfills the conditions of time and prefix invariance as well as MAC variance. Restarting the machine or the interface, however, influences the sequence. Thus, we assume that Windows 8.1 implements the privacy extension's version for systems without presence of stable storage. In such a case, the first history value after a restart is randomly assigned. This assumption coincides with the fact that we could not find any appropriate history value in the Windows Registry analysing Registry diffs. Further signs supporting our assumption are the collaboration of Microsoft in the definition of the RFC, as well as the algorithm's description in older TechNet Library articles [22].

The gained insights lead to the following conclusion: While Ubuntu 14.10 and Mac OS X Yosemite seem to be immune to our attack, Windows 8.1 appears to be vulnerable – admittedly to a decreased extent as reinitialization of the history value is performed with every restart. However, systems that are continuously running for longer periods or using sleep mode remain vulnerable; and sleep mode is widely used for laptops. For interest, the operating systems' protection to our attack is gained by disobeying the privacy extension's standard. Ubuntu and Mac OS seem to totally ignore the proposed generation algorithm, while Windows 8.1 appears to implement the alternative for systems without stable storage albeit it assumes such storage according to its system requirements.

9 Mitigation

In this section, we recommend changes to the address generation mechanism for mitigation of our attack. We propose two kinds of strategy: The first aims at impeding synchronization to the algorithm's current state, while the other removes the predictability of future identifiers in general.

Restraint of Synchronization: Our attack is based on the fact that an adversary is able to learn a victim's state by observating them over multiple days,

[6] Kernel 3.16.0, /net/ipv6/addrconf.c, line 1898

and one might hamper an adversary's synchronization to the algorithm's internal state for mitigation. These strategies do not offer protection in case the state is leaked. The following explanations are supported by Figure 6; the numbers in the figure match those provided in the following paragraphs.

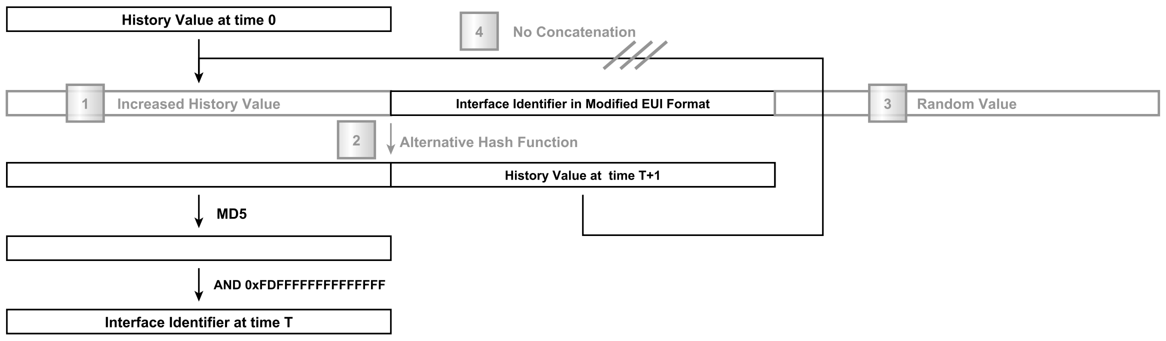

History Value at time 0
4
No Concatenation
1
Increased History Value
Interface Identifier in Modified EUI Format
3
Random Value
2
Alternative Hash Function
History Value at time T+1
MD5
AND 0xFDFFFFFFFFFFFFFF
Interface Identifier at time T

Fig. 6: Mitigation strategies for generation of temporary IIDs

(1) An increased history value would imply improved randomization and increase the size of the initial candidate set C_0, see Equation 1. As a consequence, the adversary has to observe more successive identifiers according to Equation 2, and time expenditure for brute-forcing increases, see Equations 3 and 4. The algorithm's current design, however, does not allow an isolated increase of the history value. The MD5 digest's first half forms the temporary interface identifier and its second the current history value. Beyond, there are no bits available that could serve as additional bits for an increased history value. Thus, this strategy would require the replacement of MD5 by another hash function.

(2) MD5 is considered insecure, and its replacement by a state-of-the-art hash function seems tempting. MD5 is vulnerable to collision attacks, and insecure for applications that rely on collision resistance, e. g., as necessary for certificates [23]. The IPv6 privacy extension, however, exploits a hash function's randomization, and replacing MD5 with the currently used SHA-265 would only modestly increase brute-force effort [24].

Removal of Identifiers' Predictability: Another precondition of our attack is the dependency of future identifiers on the current state and predictable inputs only. The following mitigation approaches tackle this issue by removing the predictability of future identifiers in different ways.

(3) Including a random value in every iteration makes the digest dependent on more inputs, and adds unpredictability with every new interface identifier. This is the major difference to an increased history value as mentioned above that solely increases randomization at the algorithm's initialization. Even if the

current state is leaked, it is impossible to accurately predict future interface identifiers. Moreover, this measure does not require a dissolution of MD5.

(4) A removal of the concatenation would result in successive addresses that are not related to each other; instead, the history value could be randomly initialized for every new address. A similar but more limited approach is defined by the privacy extension's standard, but only for devices without stable storage [7]. As such systems are not able to store the history value across system restarts, they are allowed to randomly initialize the first history value after a reboot. Their vulnerability is thus dependent on their typical restart intervals in comparison to the temporary addresses' lifetime. Nevertheless, it seems curious that an alternative algorithm for specific devices is more secure than the standard algorithm.

Alternatively, temporary interface identifiers could be randomly assigned without such a complex algorithm. A host's vulnerability to address correlation is then dependent on the quality of its random number generator. We see advantages in this approach because high-quality random number generators are necessary in modern operating systems on personal computers, laptops and mobiles anyway. The privacy extension would benefit from this quality and further be updated automatically with every improvement of the number generator. For systems without an appropriate random number generator, an alternative would have to be available. This practice is opposed to today's standard that defines a rather complex algorithm *"to avoid the particular scenario where two nodes generate the same randomized interface identifier, both detect the situation via DAD, but then proceed to generate identical randomized interface identifiers via the same (flawed) random number generation algorithm"* [7] and lowers security for all systems that implement the privacy extension.

Finally, we considered the question which mitigation strategies are in accordance with the current specification, and have drawn the following conclusions: (1) It is allowed to use another hash function instead of MD5. The brute-force effort would, however, increase only modestly, and a replacement brings only limited protection. (2) The history value is allowed to be randomly re-initialized after every system restart, but this behavior is restricted to systems without stable storage. However, a variety of systems that implement the privacy extension like personal computers, laptops, tablets or mobiles do not lack stable storage, and have to follow the standard variety of the algorithm. (3) The privacy extension is considered more secure the shorter the temporary addresses' lifetime. This inherent belief has to be revised with respect to the presented attack because more addresses are provided to the adversary within the same time interval, making synchronization to the current state easier.

10 Conclusions

The IPv6 privacy extension aims to protect privacy by regularly changing the address, and defines an algorithm for the generation of interface identifiers that are combined with the advertised network prefix to form temporary IPv6 addresses.

In this paper, we presented an attack that questions the extension's capability of protection: An adversary is able to predict future temporary interface identifiers once the internal state is known, and is further able to synchronize to this internal state by observing the victim's previous interface identifiers. In consequence, an adversary knows interface identifiers belonging to the same host; in turn, she is able to perform address-based correlation of different transactions and infer (private) details about people's Internet behavior. Moreover, an adversary might even retrace a host's movement in the network based on the network prefixes that are included in the respective addresses.

The presented attack is worthwhile as it does not solely identify a privacy vulnerability but questions a whole measure for privacy protection. The privacy extension was developed with the intention to impede address-based correlation, and our attack shows that it does not meet its goal. Nevertheless, we believe that the general idea of temporary addresses is valuable, and recommend a revision of the algorithm for interface identifier generation. We want to highlight the fact that merely replacing MD5 does not solve the problem, as the vulnerability arises from the concatenation of successive interface identifiers, scarce randomization and information leakage via a side channel. MD5 just makes the attack easier due to its fast nature. Proper mitigation within the current definition appears impractical, and we want to stress the importance of strategies beyond today's specification.

Operating systems appeared less vulnerable than originally assumed. This does not, however, oppose a revision, as their robustness is gained by silently disobeying the standard and should not be held as a virtue. The standard in its current form can tempt developers to implement a vulnerable version of the privacy extension, and should be adapted soon. This utmost concern is further emphasized by the fact that the privacy extension is the only widely deployed IPv6 mechanism using stateless address autoconfiguration that is intended to protect against temporal as well as geographical address correlation.

Acknowledgments

The authors thank Peter Wurzinger, Dimitris E. Simos, Georg Merzdovnik and Adrian Dabrowski for many fruitful discussions. This research was funded by P 842485 and COMET K1, both FFG - Austrian Research Promotion Agency.

References

1. S. Landau, "Making Sense from Snowden: What's Significant in the NSA Surveillance Relevations," *IEEE Security & Privacy Magazine*, vol. 4, pp. 54–63, 2013.
2. ——, "Making Sense from Snowden, Part II: What's Significant in the NSA Surveillance Relevations," *IEEE Security & Privacy Magazine*, vol. 1, pp. 62–64, 2014.
3. J. Leber, "Amazon Woos Advertisers with What It Knows about Consumers," January 2013. [Online]. Available: http://www.technologyreview.com/news/509471/amazon-woos-advertisers-with-what-it-knows-about-consumers/

4. V. Blue, “Facebook turns user tracking ’bug’ into data mining ’feature’ for advertisers,” June 2014. [Online]. Available: http://www.technologyreview.com/news/509471/amazon-woos-advertisers-with-what-it-knows-about-consumers/
5. A. Cooper, H. Tschofenig, B. Aboba, J. Peterson, J. Morris, M. Hansen, and R. Smith, “Privacy Considerations for Internet Protocols,” RFC 6973, July 2013.
6. R. Hinden and S. Deering, “IP Version 6 Addressing Architecture,” RFC 4291, February 2006.
7. T. Narten, R. Draves, and S. Krishnan, “Privacy Extensions for Stateless Address Autoconfiguration in IPv6,” RFC 4941, September 2007.
8. J. Ullrich, K. Krombholz, H. Hobel, A. Dabrowski, and E. Weippl, “IPv6 Security: Attacks and Countermeasures in a Nutshell,” in *USENIX Workshop on Offensive Technologies (WOOT)*, 2014.
9. S. Thomson, T. Narten, and T. Jinmei, “IPv6 Stateless Address Autoconfiguration,” RFC 4862, September 2007.
10. F. Gont, “A Method for Generating Semantically Opaque Interface Identifiers with IPv6 Stateless Address Autoconfiguration (SLAAC),” RFC 7217, April 2014.
11. T. Aura, “Cryptographically Generated Addresses (CGA),” RFC 3972, March 2005.
12. J. Arkko, J. Kempf, B. Zill, and P. Nikander, “SEcure Neighbor Discovery (SEND),” RFC 3971, March 2005.
13. T. Narten, E. Nordmark, W. Simpson, and H. Soliman, “Neighbor Discovery for IP version 6 (IPv6),” RFC 4861, September 2007.
14. M. Dunlop, S. Groat, R. Marchany, and J. Tront, “IPv6: Now You See Me, Now You Don’t,” in *International Conference on Networks (ICN)*, 2011, pp. 18–23.
15. S. Groat, M. Dunlop, R. Marchany, and J. Tront, “IPv6: Nowhere to Run, Nowhere to Hide,” in *Hawaii International Conference on System Sciences (HICSS)*, 2011.
16. A. Alsadeh, H. Rafiee, and C. Meinel, “Cryptographically Generated Addresses (CGAs): Possible Attacks and Proposed Mitigation Approaches,” in *IEEE International Conference on Computer and Information Technology (CIT)*, 2012.
17. A. AlSadeh, H. Rafiee, and C. Meinel, “IPv6 Stateless Address Autoconfiguration: Balancing between Security, Privacy and Usability,” in *Foundations and Practice of Security*, 2013, pp. 149–161.
18. D. Barrera, G. Wurster, and P. C. Van Oorschot, “Back to the Future: Revisiting IPv6 Privacy Extensions,” *LOGIN: The USENIX Magazine*, vol. 36, no. 1, pp. 16–26, 2011.
19. S. Turner and L. Chen, “Updated Security Consideration for the MD5 Message-Digest and the HMAC-MD5 Algorithms,” RFC 6151, March 2011.
20. J. M. Gosney, “Password Cracking HPC,” in *Passwords Security Conference*, 2012.
21. M. Heuse, “thc-ipv6 toolkit v2.7,” April 2015. [Online]. Available: https://www.thc.org/thc-ipv6/
22. TechNet, “IPv6 Addressing (Tech Ref),” April 2011. [Online]. Available: https://technet.microsoft.com/en-us/library/dd392266%28v=ws.10%29.aspx
23. M. Stevens, A. Sotirov, J. Appelbaum, A. Lenstra, D. Molnar, D. Osvik, and B. de Weger, “Short Chosen-Prefix Collisions for MD5 and the Creation of a Rogue CA Certificate,” in *Advances in Cryptology (CRYPTO)*, 2009.
24. “eBASH (ECRYPT Benchmarking of All Submitted Hashes,” March 2015. [Online]. Available: http://bench.cr.yp.to/results-hash.html