Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 113 (2017) 81–88

Procedia Computer Science

www.elsevier.com/locate/procedia

The 8th International Conference on Emerging Ubiquitous Systems and Pervasive Networks (EUSPN 2017)

CoLLIDE: CLoud Latency-based IDEntification

Vanesa Daza[a], Roberto Di Pietro[b], Flavio Lombardi[c,*], Matteo Signorini[b]

[a]*Pompeu Fabra University, Tanger 122-140, 08018 Barcelona, Spain*
[b]*Nokia Bell Labs, Route de Villejust, 91620 Nozay, France*
[c]*IAC-CNR, via dei Taurini 19, 00185 Rome, Italy*

Abstract

As services steadily migrate to the Cloud, the availability of an overarching identity framework has become a stringent need. Moreover, such an identity framework is now critical in the Internet of Things. To address this problem, identification solutions have been proposed in the past leveraging software or hardware properties of devices. While those solutions proved feasible, their root of trust was based either within the device or in a remote server.

In this paper, we overcome the above paradigm and star investigating novel perspectives offered by an overarching identity framework that is not based on client/server properties, but on the network latency of their communications. The core idea behind our approach is to leverage cloud client/server interactions' latency patterns over the network to derive unique and unpredictable identity factors. Such factors can be used to design and implement effective identification schemes especially suitable for cloud-based services. To the best of our knowledge, our approach is the first one ensuring unclonability and unpredictability properties, relying on neither trusted computing bases (TCBs) nor on classical pseudo-random number generators (PRNGs). The experimental tests presented in this paper, conducted on worst case conditions, show that the network latency (generated between two interacting devices) can produce random values with properties close to the ones generated by most of the well-known PRNGs, that are an ideal fit for providing unique identifiers.

Peer-review under responsibility of the Conference Program Chairs.

Keywords: identification; latency; cloud; unpredictability;

1. Introduction

Cloud resources and services are usually accessed by heterogeneous unsecured devices over mostly insecure networks. In addition, with the upcoming Internet of Things (IoT), and the machine-to-machine (M2M) paradigm, most devices will often operate without user control, even though they will directly interact with each other. In this scenario, identification protocols based on knowledge factors such as passwords, PIN codes or even biometric information cannot be applied. This is also true in many other scenarios where client-side applications have to maintain identities to be recognized by the server side, such as for messaging protocols. Hence, the need for novel solutions that could tackle

*Corresponding author. Tel.: +39-06-83393264; fax: +39-06-4404306
E-mail address: flavio.lombardi@cnr.it

1877-0509
Peer-review under responsibility of the Conference Program Chairs.
10.1016/j.procs.2017.08.295

the problem of authenticating the supplicant device. As an example, secure (mobile) payment approaches impose strong requirements which are based on reliable and secure user device identification.

Over the last few years, two main approaches have been proposed to address the above issues: Physical Unclonable Functions (for short, PUFs)[1] and device fingerprinting[2]. Both aimed at deriving digital identities from device properties, that are used for identification purposes. The PUF approach exploits manufacturing process variations to identify different physical properties that lead to measurable differences in terms of electronic properties. Since these process variations are not controllable during manufacturing, the physical properties of a device cannot be cloned; as such, they are unique to that device and can be used for identification purposes. Implementing a PUF requires an electronic circuit that is able to produce hardware responses to given input challenges based on the unique physical properties of the device. Hence, PUFs are functions that are easy to challenge and whose response is easy to measure, but very hard to reproduce.

Furthermore, there exist device fingerprinting approaches, aimed at deriving unique identities from the software layer rather than from the physical one. Indeed, software properties such as browser configurations, screen resolution, stored cookies or any other system data structures can provide (when combined) a high level of unpredictability and randomization that can be exploited to derive device fingerprints, and strong identity factors.

In this paper we propose CoLLIDE, a new way of identifying devices based on the round-trip time latency [1]. To the best of our knowledge, our solution is the first one which does not rely on devices' properties but rather on communication channels' properties. Indeed, in this paper, we show how the intrinsic network latency, embedded in the channel, can be leveraged to mimic a random number generator which is highly unpredictable and can be used for the identification of devices in the Cloud or more generally in the Internet. The experimental results of our tests (see Section 5), obtained in the worst case scenario, show the effectiveness of this new approach.

The remainder of this paper is organized as follows: the pros and cons of the above-introduced approaches are analyzed in Section 2; in Section 3 the adopted threat model is discussed; Section 4 introduces our novel software-based approach; Section 5 describes the experimental settings and evaluates our approach; finally, conclusions and hints for future work are given in Section 6.

2. Related Work

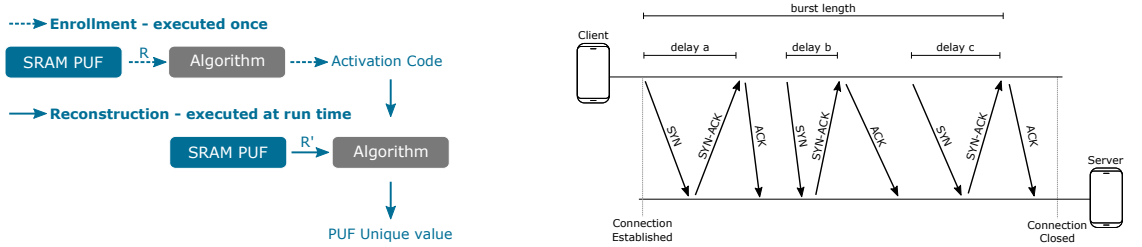
In this section, an analysis of PRNG issues and of both physical unclonable functions and device fingerprinting solutions is provided together with a discussion of both pros and cons for the listed solutions, in order to better understand how their weaknesses have been addressed in this paper.

Most Security features of a computing system are based on the unpredictability of some generated values. Given that computing systems are deterministic, generating as-random-as-possible values usually requires leveraging some kind of input from the more chaotic real-world. In fact, the quality of PseudoRandom Number Generator (PRNG) output is crucial for the security of almost any modern application. Unfortunately, the actual security/randomness of implemented software-based PRNG is far from optimal[3]. Possible attacks include Cryptanalytic, input-based, and state compromission attacks. Some specific issues are discussed in the following.

Dorrendorf et al.[4] analyzed the binary code of the Windows 2000/XP PRNG, and reconstructed the algorithm used by the pseudorandom number generator (namely, the function CryptGenRandom). They found a forward security attack: given the internal state of the generator, the previous state can be computed in 223 steps. Furthermore, the generator run in user mode giving easy to access its state. The implication of these findings is that a buffer overflow attack or a similar attack can be used to learn a single state of the generator, which can then be used to predict all random values, such as SSL keys, used by a process in all its past and future operations.

Kumari et al.[5] consider performance of Linux Random Number Generator (RNG) in virtualized environments and as if emulated hardware can provide sufficient entropy sources for the RNG. They show that, as expected, hosts have higher entropy sources available and generate entropy at a higher rate. Further, an adversary can find the output of the host RNG by controlling the guest by generating high amounts of disk activity on the guest itself.

[1] we are considering here the actual entire/global RTT+SystemDelay+AppDelay Latency Pattern



(a) An overview of PUF values extraction.

(b) Latency-based 3-way handshake timing computation.

Fig. 1: PUF vs Latency-based approaches.

For many years, silicon Physical Unclonable Functions (PUFs) have been seen as a promising and innovative security technology. Today, Static Random-Access Memory (SRAM)-based PUFs are no more a promise but a reality which offers mature roots of trust for commercial products. PUFs can be found in devices ranging from tiny sensors and microcontrollers to high-performance Field-Programmable Gate Arrays (FPGAs) and secure elements. All of them, share the same property of presenting micro structural differentiation, ranging from optical properties[6] to SRAM configurations[7], that can be used to derive digital identities. In fact, as an example, although SRAM cells are symmetrical, small and random manufacturing variations exist that can cause an intrinsic mismatch that, at boot process, is responsible for cell preferences to either bias towards a logic 0 or 1. As for SRAM PUFs, other hardware-based solutions proved to be useful in deriving unique, per-device characteristics such as the ones able to distinguish devices through their TCP and ICMP timestamps[8], radio frequency[9] or accelerometer idiosyncrasies[10].

SRAM PUFs have been largely qualified for automotive, industrial and military use[2]. Millions of measurements were performed to prove that their noisy behaviors (below 15%) were still enough to reconstruct high-entropy, unique and unpredictable values by using error correction techniques like helper data algorithms[11] or fuzzy extractors[12]. Figure 1a shows an overview of PUF values extraction composed by an *enrollment* phase and a *reconstruction* phase. In the enrollment phase (a one-time process), the PUF behavior and its produced output is mapped onto a codeword of an error correcting code and stored in a so called Activation Code (AC) or helper data. Due to PUFs physical properties, each AC is only valid for the PUF that generated it and it can be stored either on-the-chip or off-the-chip.

PUF-based approaches can use values generated in the *reconstruction* phase to turn the device into an authentication token. It supersedes the necessity to protect sensitive data within the device, which would be prone to attacks where an adversary tries to extract them. Instead, PUF devices generate ephemeral values on-the-fly on the basis of their unique physical characteristics, minimizing the attack surface. However, as based on hardware root of trust, they are usually tied to ad-hoc elements which can hamper their adoption as expensive and mostly of the time not compatibile with legacy systems. Furthermore, it has been shown that hardware approaches have issues related to **Unclonability**: the clonability of "unclonable" functions has already been shown[13]; **Time/Space limitations**: PUFs have a limited number of challenge-response pairs; **Cost**: the securer the PUF, the more complex has to be the hardware within the PUF, thus requiring a higher number of logical ports; **Non upgradeability**: even though reconfigurable PUFs are able to change their configuration, it is not possible to perform such change on-the-fly.

Device fingerprinting solutions aim at providing secure identification and authentication, leveraging the software configuration and/or the device behavior, thus being complementary to PUFs. However, even though they can solve some issues related to PUFs, they still suffer from weaknesses. The most important one is their predictability. In fact, even though device configurations and behaviors can be complex, fingerprinting information can be predicted and/or stolen by an adversary, rendering him able to mimic the victim device behavior thus claiming its identity.

The most common solution is then to leverage PRNGs to render fingerprinting unique and unpredictable. However, PRNGs suffer from attacks such as direct cryptographic, input-based and state compromise extension[14]. These issues render cheap and low power devices unable to generate high quality random values[15]. As such, identity stealing is possible on PRNGs-based identification schemes.

[2]https://www.intrinsic-id.com

3. Threat Model

Our solution involves a server, one or multiple clients, and one or multiple adversaries which pretend to be identified on behalf of their victim(s). Involved entities are defined as follows: **Client**: also known as the *claimant*, this is the device that needs to be remotely identified by another device; **Server**: also known as the *verifier*, it accepts requests from client devices and tries to build their digital identities based on their network latencies; **Adversary**: a third device interacting with both the client and the server and aiming at being identified as the client.

We can define the i) entry points, ii) life cycle and iii) attack approaches being used by the adversary.

For the entry points, we have: **Client/Server Devices:** the adversary can either physically or remotely reach the client and/or server devices; **Communication Channel:** the adversary can access the communication channel between the client and the server.

As Regards the life cycle of the adversary, we can have: **Temporary Attack:** the adversary can only temporarily get access to the desired entry points. As such, once left the entry point, the adversary can only rely on previously-collected information to compute and/or steal other information for further attacks; **Persistent Attack:** the adversary can stay hidden within the client/server device or within the communication channel as long as necessary. In this case it is then possible for the adversary to attack the victim at run-time, i.e. while the victim is interacting with the server.

Last but not least, as concerns the attack type we have the following two cases: **Passive Attack:** the adversary can eavesdrop information being sent over the channel or to steal information which are stored within the client/server devices (depending on the attack entry points available to the adversary); **Active Attack:** the adversary not only can eavesdrop and steal as for the passive approach but he is also capable of injecting information in the channel or within the devices. He can also change the information being sent through the channel or stored within the devices.

Given that our approach makes no security assumptions neither on the *verifiers* nor on the *claimants*, the kind of active adversary able to unleash a persistent attack has not been taken into account. Indeed, for such an adversary, it would be possible to build a perfect clone of the victim's identity [16], thus rendering the *verifier* unable to distinguish between the two devices without leveraging special hardware elements. Throughout the rest of this paper we will then focus on a passive-only adversary with unlimited entry points (i.e. capable of both hacking the involved devices and the communication channel) and with a temporary attack window that is assumed to be finite but that can be as long as needed by the adversary.

4. Our Approach: CoLLIDE

The solution proposed in this paper (named CoLLIDE for CLoud Latency-based IDEntification) provides a software function that is based on the network communication delays between two or more devices. Unlike previous solutions, our latency-based approach does not rely on artificially generated information but rather on the unpredictability of the network latency. The final goal consists then in the generation of a per-device or even a per-application unique characteristic, similar to the PUFs SRAM unique configurations described in Section 2, to uniquely identify the device or each single application. As a toy example, a device or an application could be identified if and only if equation

$$H_{cl}(ClData\_Dx) = H_{sr}(SrData\_Dx) \tag{1}$$

holds, where H_{cl} and H_{sr} are, respectively, hash functions computed on the device and on the remote *verifier*, whilst $ClData\_Dx$ and $SrData\_Dx$ are, respectively, the device Dx meta-data (containing network latency information) shared between the *claimant* and remote *verifier*. If the above equation holds, then the involved devices are sharing the same knowledge on the network latency which affected their past communications. This shared information on the previous delays is then used to build a digital identity which is unpredictable (as shown in our experimental results, see Section 5) and directly related to the status of the communication channel.

The delay leveraged by this solution is computed as the time spent at each interaction between the *claimant* and the *verifier* (in case of different *verifiers*, they are treated separately) in a challenge-response scheme. Indeed, SYN-ACK packets sent by the *verifier* contain, among other things, a challenge that has to be used by the *claimant* in computing the response. Then, if the response received by the *claimant*, and the one locally computed by the *verifier* match (following Equation 1) the *claimant* gets recognized by the *verifier*. As a toy example, in this paper a simple *XOR*

function has been used to derive a single value (i.e. the digital identity) from the network latency values. Nevertheless, any other function accepting the latency as input can be used.

5. Experimental Activity and Evaluation

Yilmatz et al. showed that both on-line and off-line push notifications on top of the Google Cloud Messaging Android platform[17] had a high unpredictable arrival within a specific time window[18]. As a different from such approaches, this present paper does not assume any trusted third party such as the Google servers leveraged above. Hence, in order to prove the same unpredictability in the message arrival between two devices only (the *claimant* and the *verifier*), a new test bed has been designed as described in the remainder of this sections, whereas obtained results are shown in Section 5.

We have developed 2 main applications to test the unpredictability of the CoLLIDE network latency approach. On one hand, an Android 6.0 client application, deployed on the Android Studio emulator and on two Google Nexus 5x devices located in two different countries, respectively France and Italy. On the other hand, a server component has been deployed on a server located in Italy. Then, the client devices have been monitored under different network configurations such as: i) WiFi only, ii) 3G only and iii) WiFi plus 3G. In addition, one of the two client devices was always kept physically located in one place whilst the other one was continuously moving between the two countries.

To test the unpredictability of the network-latency, a three-way handshake protocol has been implemented in which each client first establishes a connection to the server via a *SYN* packet, then it waits a *SYN-ACK* and finally it sends an *ACK* packet. As shown in Figure 1b, the time between *SYN* and *SYN-ACK* has been leveraged to measure the network latency while the number of three-way handshake messages, labeled as burst length, has been leveraged to mimic different client behaviors. Three distinct configurations have been used for the burst length such as i) zero-length (i.e. no bursts) in which each connection to the server is closed after the SYN-ACK-SYNACK and then re-established again for the next message, ii) a random and variable length in which the number of handshakes has been randomly selected and a iii) fixed and static burst length selection. The above three burst-length setups have been chosen to emulate client devices in which the communications with the server are i) rare, ii) randomly generated and iii) pre-defined.

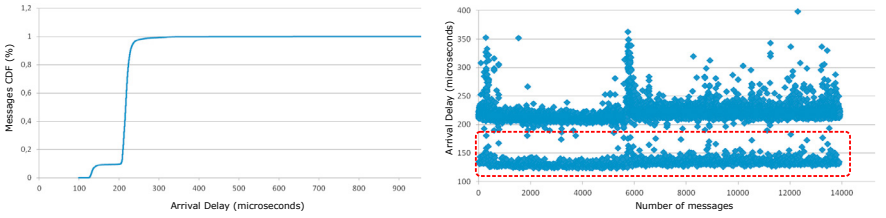
Both the client and the server applications have been implemented in Java and *System.nanoTime()* has been used as the time function as it perfectly fits our delay computing precision requirements. Indeed, *System.nanoTime()* returns the current value of the running Java Virtual Machine's high-resolution time source, in nanoseconds, and can be only used to measure elapsed times as it is not related to any notion of system or wall-clock time but to some fixed and arbitrary origin time which remains the same in all the invocations of the method.

Figure 2a shows the message delays collected in our tests. On the right side we can see that the majority of the message delays (roughly 80%) is comprised between 210 and 230 milliseconds with the messages being used to open the channel at the beginning of each burst spanning between 130 and 150 millisecond. We can see that message delays form a bipartition where messages enclosed in the dotted red box (from 130 to 150 milliseconds) are the same displayed in the dotted red circle in Figure 2b and represent messages being used for the establishment of the communication channel between the *claimant* and the *verifier*.

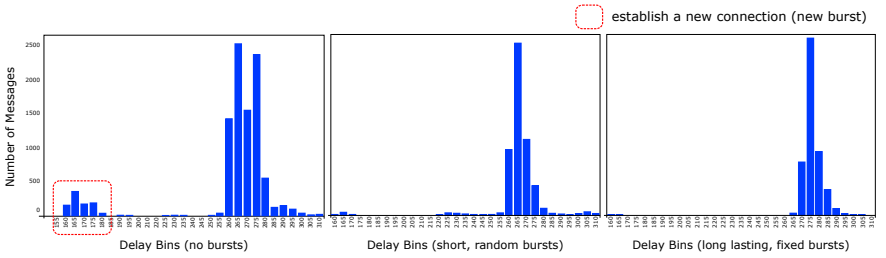
On the left hand side, we can see the cumulative distribution function of the right hand side which shows that delays are uniformly distributed. This result is pretty much in line with the one obtained by Yilmatz et al. on the Google Cloud Messaging Android platform[17], thus confirming the unpredictability of message arrivals even in the case of a direct communication between the *claimant* and the *verifier*. To better prove the initial result, our test has been repeated on three different burst configurations (as explained in Section 5) and the results are shown in Figure 2b. Regardless of the burst-length, message delays proved to be uniformly distributed within a short range of values.

To better describe the nature of the bipartition found in the right side of Figure 2a, each message has been classified as i) starting a new burst, ii) ending an ongoing burst or iii) sitting in the middle of a burst. Results are shown in Figure 2c where it is easy to see that all the messages starting a new burst are slightly faster than the others thus creating the bipartition in the right side of Figure 2a. In fact, by lowering the number of bursts (right-side of Figure 2c) the bipartition disappears (right side of Figure 2b).

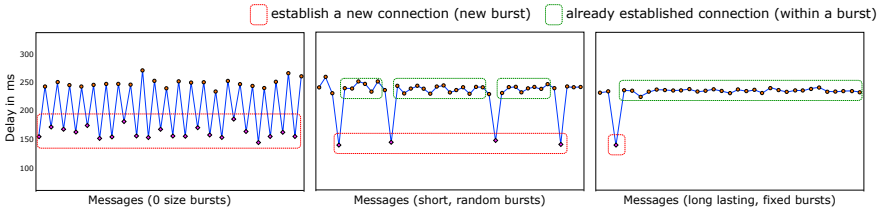
Results depicted in Figure 2b and Figure 2c show that, regardless of the network channel being used (either WiFi only, 3G only or both), the longer the burst (i.e. the longer the interaction between the client and the server), the higher the number of messages being uniformly distributed within a short range. However, this was not enough to



(a) Latency-based message CDF on the left. Absolute message arrival frequency on the right.



(b) Latency-based message delays distributions.



(c) Inter and intra burst delay distribution.

Fig. 2: Experimental Results

prove the unpredictability of the network latency. Hence, the sequence of message delays have been compared to different sequences of pseudo random values generated by well known PRNGs such as the ones used by PHP, Python, Ruby and Javascript languages. Furthermore, even on-line PRNG services have been used in order to compare our latency-based approach with PRNGs which are claimed to be of a high quality.

Results listed in Table 1 have been obtained by running the *DIEHARD* [19] test suite, used to measure the quality of random number generators. More in detail, Table 1 lists the results produced by *DIEHARD*. Results show that our latency-based approach performed on par with the other well known RNGs by failing only one test over 27.

In fact, even though the results we obtained with the *ENT* test suite were slightly worse than the other ones, the big novelty and major advantage of the proposed approach is the fact that it is intrinsically present in the network channel and it is not not artificially computed by one or more of the involved devices. Furthermore, our tests leveraged a simple client/server architecture whereas in a real cloud entropy is increased due to the transparent usage of a number of instances per each service. In fact, such instances have potentially different network latency and response time that can (and will in future work) be used to further increase entropy for our solution. A toy example is the Java *math().random()* function that has been used in our test. Even though results in Table 2 show that Java performed better than our latency-based approach, it has been proved [3] that it is possible to predict the random number generated by this function with a high degree of accuracy. This, could make an adversary capable of generating the secret values shared between the *claimant* and the server in advance, thus being able to fake the identity of his victim. On the other side, as our solution is not only based on victims' behaviors (which can be predicted by attackers) but also on the behavior of all those users which are using one or more hops in the path between victims and their servers.

[3] http://ib.compscihub.net/wp-content/uploads/2016/07/CS-EE-a-29-A.pdf

Table 1: Comparative analysis between our latency-based approach and common random number generators (*DIEHARD* test suite).

Test	**Our solution**	/dev/random	PHP	Python	Ruby	Shuf	Unix Rand	/dev/urandom	Random.org	Java	JavaSecure	JavaScript
Birthdays	✓	✓	✓	✓	–	✓	✓	✓	✓	✓	✓	✓
Operm5	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Rank 32x32	✓	✓	–	✓	✓	✓	✓	✓	✓	✓	✓	✓
Rank 6x8	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
BitStream	✓	✓	–	✓	✓	✓	✓	✓	✓	✓	✓	✓
Opso	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Oqso	✓	✓	✓	–	✓	✓	✓	✓	✓	✓	✓	✓
Dna	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Count 1 string	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Count 1 byte	✓	✓	✓	✓	✓	✓	–	✓	✓	✓	✓	✓
Parking Lot	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
2D Sphere	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
3D Sphere	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Squeeze	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Sums	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	✓
Craps	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Marsaglia T.Gcd	✓	–	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Sts Monobit	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Sts Runs	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	✓
Sts Serial	–	✓	✓	✓	✓	–	–	✓	–	✓	✓	✓
Rgb BitDist	✓	✓	✓	–	✓	–	✓	✓	–	✓	–	✓
Rgb Min Dist	✓	✓	–	✓	✓	✓	✓	✓	✓	✓	✓	✓
Rgb Perms	✓	✓	✓	✓	✓	✓	✓	–	✓	–	✓	–
Rgb Lagged Sum	✓	✓	–	–	–	–	–	✓	–	–	–	✓
Rgb KSTest	✓	✓	✓	–	✓	✓	✓	✓	✓	✓	✓	✓
Dab FillTree	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Dab Monobit2	✓	✓	✓	✓	✓	–	✓	✓	✓	✓	✓	✓
Failed Tests	1	1	4	4	2	4	3	1	3	2	4	1

Table 2: Comparative analysis between our latency based approach and common random number generators (*ENT* test suite).

	Entropy (bits/byte)	Arithmetic Mean (127.5=random)	Serial Correlation (0.0=uncorrelated)
Our solution	**1.713949**	**41.1854**	**0.368578**
/dev/random	1.354527	44.8819	-0.116905
PHP	1.354577	44.8818	-0.117224
Python	1.354487	44.8821	-0.117022
Ruby	1.354803	44.8772	-0.117025
Shuf	1.354449	44.8803	-0.116899

	Entropy (bits/byte)	Arithmetic Mean (127.5=random)	Serial Correlation (0.0=uncorrelated)
Unix Rand	1.354240	44.8812	-0.116791
/dev/urandom	1.353966	44.8863	-0.116498
Random.org	1.299439	45.4696	-0.07319
Java	1.299626	45.4700	-0.073120
JavaSecure	1.299538	45.4698	-0.073122
JavaScript	1.299492	45.4697	-0.073392

In conclusion, whilst with our approach it is still possible for an adversary to eavesdrop the delay information from the channel or to extract them from the *claimant* device, it is not possible to hack all the elements involved in the identification system (as it has been done for the Java *math().random()* function). As such, the adversary is required to listen the channel at run-time (i.e. while the attack is in execution), and for the entire duration of the attack.

5.1. Security Evaluation

Compared to hardware approaches such as PUFs and to software approaches such as device fingerprinting, our solution (CoLLIDE) has the advantages of being unpredictable and unclonable, while not relying on any physical component of the devices. As our solution is directly derived from the communication channel, and not based neither on special hardware elements nor on ad-hoc protocols, it cannot be cloned and re-played by the adversary, thus being unclonable. Furthermore, our digital identities are based on the network latency which does not only depend on the device network activities but also on the activities of all the other devices connected throughout the path from the *claimant* to the *verifier*. It is then hard (as shown in Section 5) for an adversary to predict the following delays.

As already mentioned in Section 3, even though we foresee devices in the Internet of Things to be always connected, they might have off-line windows. This might seem to jeopardize our latency-based identities since they are apparently based on the continuity of network interactions. However, the only way for an adversary to fully succeed in the stealing

of a network latency based digital identity is to: **i) Steal** all past latency information collected by the victim and used to be identified by the remote device; **ii) Mimic** the victim's behavior thus starting an interaction with the remote device staying close to the same latency values being used so far by the victim. Indeed, the remote device can profile the user over time and if the latency values drastically changes, it can decide to raise a warning and to evenyually disconnect the *claimant*; **iii) Prevent** the victim from connecting back to the remote device. Indeed, if two identification requests are received at the same time with slightly different latency values (thus different digital identities) but both claiming to be the same device, the remote *verifier* can decide to raise a warning and eventually disconnect such devices.

It is important to highlight that, as described in Section 3, in this paper we only assume the presence of a passive adversary. Such an adversary can not tamper with the channel and obtain a latency close to the victim's one. Further, the adversary can not prevent the victim from connecting with the remote *verifier*. It is then not possible for the adversary to accomplish neither the 2^{nd} nor the 3^{rd} of the above points. As such, CoLLIDE is resilient both for always-connected devices, and for devices which might have off-line time windows. It is worth mentioning that our solution offers better security when the adversary is located far from the victim. In fact, the physically closer the adversary, the more similar the network latency behavior he can potentially experience.

6. Conclusion

In this paper we have introduced CoLLIDE, an approach for fine-grained device/application identification in the Cloud. CoLLIDE allows online devices to share unpredictable values derived solely from the client-server network latency, on top of which it is possible to build unclonable digital identities. The proposed latency-based solution is fully novel and it does not require neither special hardware elements nor strong security assumptions. Presented results, obtained from a prototypal implementation of a latency-based handshake protocol, show promising properties on CoLLIDE unclonability and unpredictability, close to the actual standards. As further work we will implement more advanced protocols leveraging our main idea, and extend the experimental results involving multiple devices.

References

1. Maes, R.. *Physically Unclonable Functions*. Springer Science + Business Media; 2013. doi:10.1007/978-3-642-41395-7.
2. Eckersley, P.. How unique is your web browser? In: *Proc. 10th Intl. Conf. on Privacy Enhancing Technologies*; PETS'10. Springer-Verlag. ISBN 3-642-14526-4, 978-3-642-14526-1; 2010, p. 1–18.
3. Dodis, Y., Pointcheval, D., Ruhault, S., Vergniaud, D., Wichs, D.. Security analysis of pseudo-random number generators with input: /dev/random is not robust. In: *Proc. ACM CCS*. New York, NY, USA: ACM; 2013, p. 647–658.
4. Dorrendorf, L., Gutterman, Z., Pinkas, B.. Cryptanalysis of the random number generator of the windows operating system. *ACM Trans Inf Syst Secur* 2009;**13**(1):10:1–10:32.
5. Kumari, R., Alimomeni, M., Safavi-Naini, R.. Performance analysis of linux rng in virtualized environments. In: *Proc. ACM CCSW*. New York, NY, USA: ACM; 2015, p. 29–39.
6. Ravikanth, P.S.. *Physical One-way Functions*. Ph.D. thesis; Massachusetts Institute of Technology; 2001.
7. Guajardo, J., Kumar, S.S., Schrijen, G.J., Tuyls, P.. FPGA Intrinsic PUFs and Their Use for IP Protection. In: *Proc. Intl. Workshop on Cryptographic Hardware and Embedded Systems*. Springer-Verlag; 2007, p. 63–80.
8. Moon, S., Skelly, P., Towsley, D.. Estimation and removal of clock skew from network delay measurements. In: *Proc. Joint Conf. of IEEE Comp. and Comm. Societies*; vol. 1. 1999, p. 227–234.
9. Nguyen, N.T., Zheng, G., Han, Z., Zheng, R.. Device fingerprinting to enhance wireless security using nonparametric bayesian method. In: *Proc. IEEE Intl. Conf. on Computer Communications*. 2011, p. 1404–1412.
10. Sanorita, D., Nirupam, R., Wenyuan, X., Romit, R.C., Srihari, N.. Accelprint: Imperfections of accelerometers make smartphones trackable. In: *Proceedings of the 21st Annual Network and Distributed System Security Symposium*. 2014, .
11. Linnartz, J.P., Tuyls, P.. New shielding functions to enhance privacy and prevent misuse of biometric templates. In: *Lecture Notes in Computer Science*. Springer Nature; 2003, p. 393–402. doi:10.1007/3-540-44887-x_47.
12. Boyen, X.. Reusable cryptographic fuzzy extractors. In: *Proc. ACM CCS*. 2004, .
13. Helfmeier, C., Boit, C., Nedospasov, D., Seifert, J.P.. Cloning physically unclonable functions. In: *Proc. IEEE Intl. Symposium on Hardware-Oriented Security and Trust*. 2013, p. 1–6.
14. Kelsey, J., Schneier, B., Wagner, D., Hall, C.. Cryptanalytic attacks on pseudorandom number generators. In: *Fast Software Encryption*. Springer Nature; 1998, p. 168–188. doi:10.1007/3-540-69710-1_12.
15. Nyo, A.M., Oo, M.M., Phyo, Z.L., Soe, C.Y., Thwin, T.T., Thida, A., et al. Quality analysis of pseudorandom number generator using rough sets. In: *Intl. Conf. on Education Technology and Computer*; vol. 2. IEEE; 2010, p. V2–338–V2–342.
16. Cole, E.. *Advanced Persistent Threat: Understanding the Danger and How to Protect Your Organization*. Syngress Publishing; 1st ed.; 2013. ISBN 9781597499491, 9781597499552.
17. Google, . Cloud messaging; 2015. Online; URL `https://developers.google.com/cloud-messaging`.
18. Yilmaz, Y.S., Aydin, B.I., Demirbas, M.. Google Cloud Messaging (GCM): an evaluation. In: *IEEE Global Communications Conference*. 2014, p. 2807–2812.
19. Garpman, S., Randrup, J.. Statistical test for pseudo-random number generators. *Computer Physics Communications* 1978;**15**(1-2):5–13.