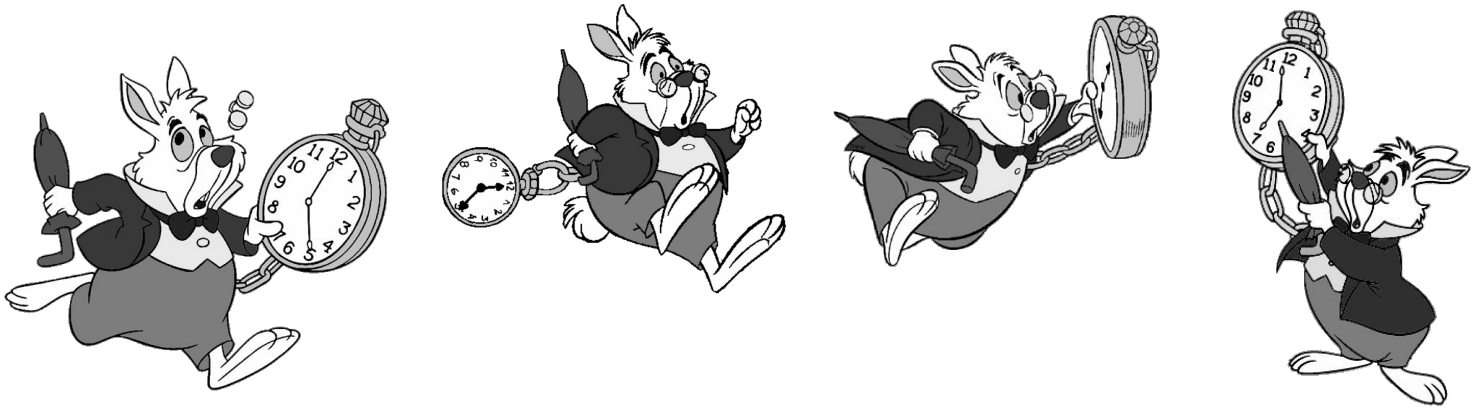
Clock Around the Clock



Time-Based Device Fingerprinting

Iskander Sanchez-Rola, Igor Santos, Davide Balzarotti

Device Fingerprinting

Exploits different features to **uniquely** identify a machine. The entity interested in computing the fingerprint is able to run arbitrary code with user privileges in the machine.

Device Fingerprinting

Exploits different features to **uniquely** identify a machine. The entity interested in computing the fingerprint is able to run arbitrary code with user privileges in the machine.

(i) malicious applications that want this information to perform **selective attacks** against certain victims.

Device Fingerprinting

Exploits different features to **uniquely** identify a machine. The entity interested in computing the fingerprint is able to run arbitrary code with user privileges in the machine.

(i) malicious applications that want this information to perform **selective attacks** against certain victims

(ii) proprietary applications that want to **bind a license** to a single machine.

Web Device Fingerprinting

A website computes a unique identifier for each visitor's machine, **without storing** any information on the client side. The entity interested in computing the fingerprint is able to run arbitrary code with user privileges in the browser.

Web Device Fingerprinting

A website computes a unique identifier for each visitor's machine, **without storing** any information on the client side. The entity interested in computing the fingerprint is able to run arbitrary code with user privileges in the browser.

(i) advertisers or tracking companies can use it to obtain the **browsing history** of users.

Web Device Fingerprinting

A website computes a unique identifier for each visitor's machine, **without storing** any information on the client side. The entity interested in computing the fingerprint is able to run arbitrary code with user privileges in the browser.

(i) advertisers or tracking companies can use it to obtain the **browsing history** of users.

(ii) websites that require strong authentication (e.g., banking and shopping) can use this technique to include an **additional verification** to their process.

Web Device Fingerprinting

Attribute-based
Uses several browser attributes (e.g., installed fonts/plugins, or UserAgent). Unfortunately, these attributes change rapidly, rendering the fingerprint obsolete.

Hardware-based
Uses browser implementations of different APIs to compute the differences between devices that are based in hardware features (e.g., HTML5 Canvas or WebGL).

Hardware-based Identification

We notice that the **execution time** of certain functions was uniquely correlated to the machine where it was running, and could be used to differentiate even among identical devices.

Hardware-based Identification

We notice that the **execution time** of certain functions was uniquely correlated to the machine where it was running, and could be used to differentiate even among identical devices

What is the **reason** behind this?

Clock Imperfections I

Computers can be fingerprinted by comparing **different clocks**.

Salo proposed to compare the CPU clock cycles of ticks in the clock with the cycles needed for the digitalization of an analog signal using a **sound card**. Afterwards, the author computed different statistical tests to distinguish among different machines.

Timothy J Salo. 2007. Multi-Factor Fingerprints for Personal Computer Hardware.
In Proceedings of the Military Communications Conference (MILCOM). IEEE.

Clock Imperfections II

Several factors play a crucial role for this technique to work:

Clock Imperfections II

Several factors play a crucial role for this technique to work:

(1) The program needs to have access to the CPU **clock cycles**, which is not a common option in high-level languages.

Clock Imperfections II

Several factors play a crucial role for this technique to work:

(1) The program needs to have access to the CPU **clock cycles**, which is not a common option in high-level languages.

(2) The sound card used must not rely on the CPU clock and should use an **independent crystal**-controlled oscillator.

Clock Imperfections II

Several factors play a crucial role for this technique to work:

(1) The program needs to have access to the CPU **clock cycles**, which is not a common option in high-level languages.

(2) The sound card used must not rely on the CPU clock and should use an **independent crystal**-controlled oscillator.

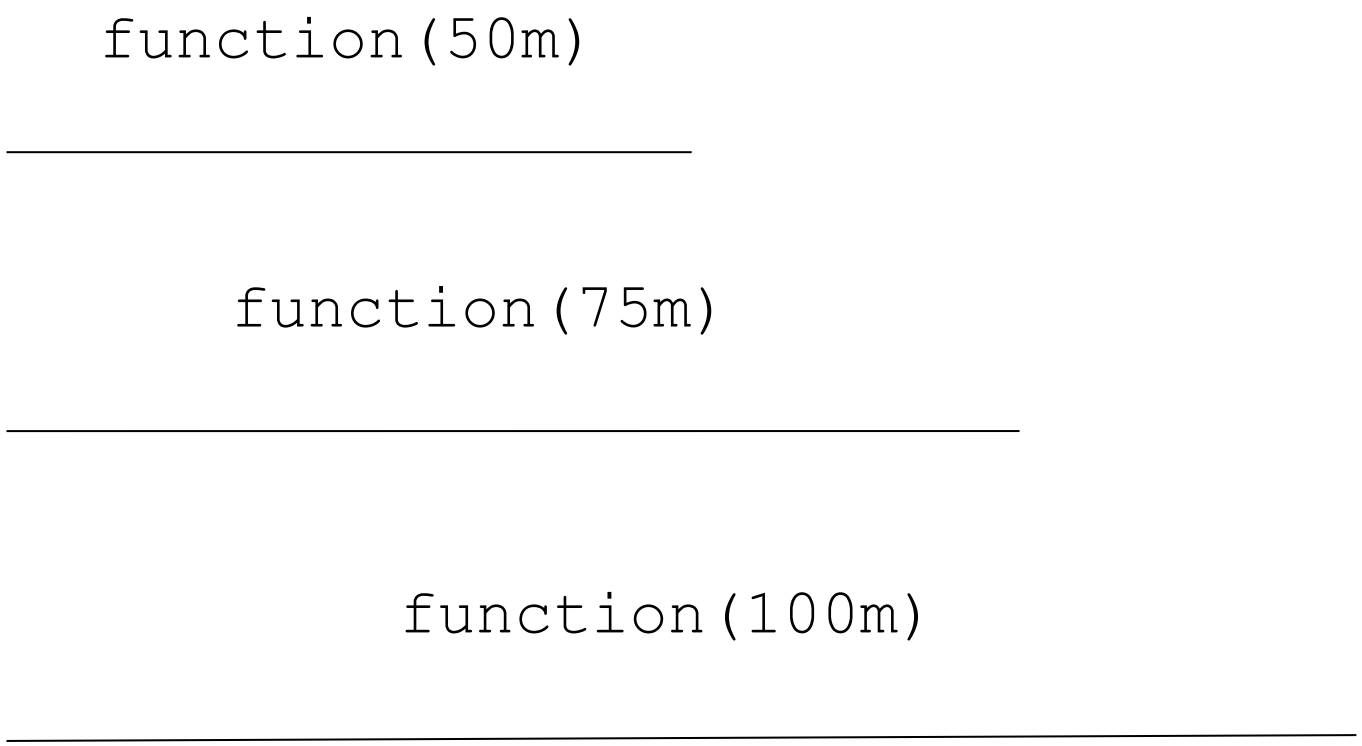
(3) The experiment needed to run for approximately **one hour**.

Our Observation

When a small function is repeated a sufficient number of times, it can be used to **amplify the small differences** between the CPU and the timer clocks. By measuring the execution time by using the datetime API, we can use this information to remotely fingerprint a machine.

Our algorithm is divided into **two different phases**: the generation of the fingerprint, and the comparison phase.

Fingerprint Generation



Fingerprint Generation

`function(75m)`

`function(100m)`

Fingerprint Generation

0.6s

function(75m)

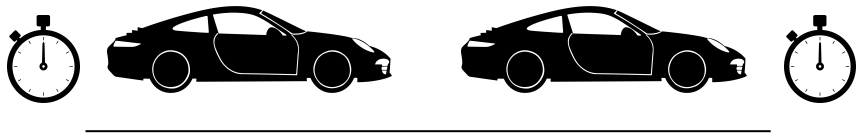
function(100m)

Fingerprint Generation

0.6s

`function(100m)`

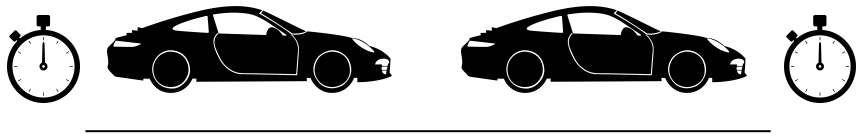
Fingerprint Generation



0.6s

`function(100m)`

Fingerprint Generation

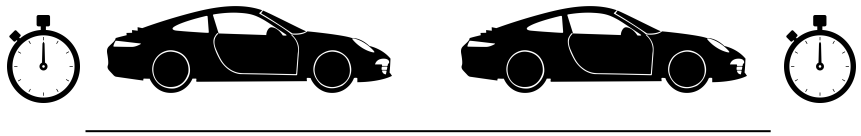


0.6s

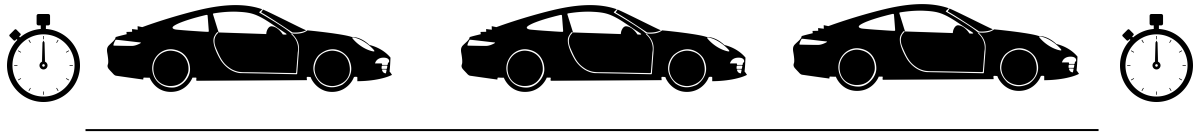
0.94s

function(100m)

Fingerprint Generation

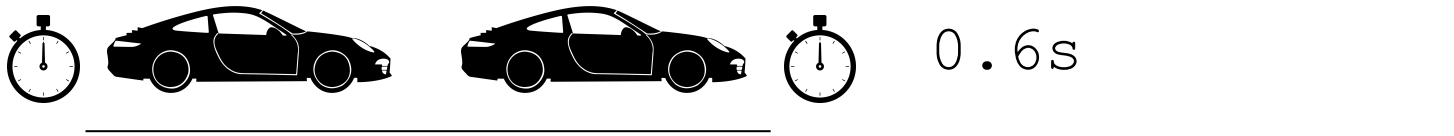


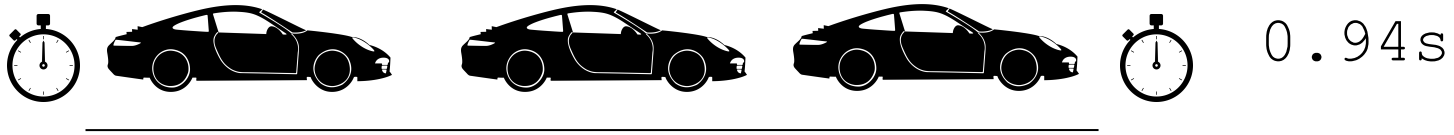
0.6s

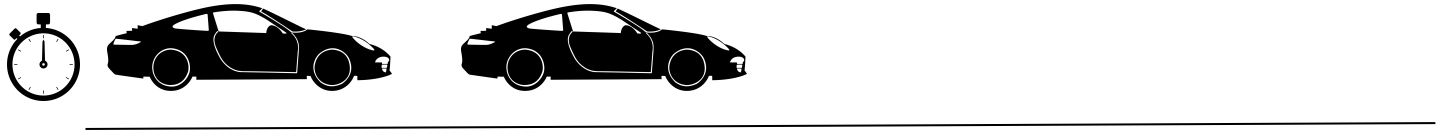


0.94s

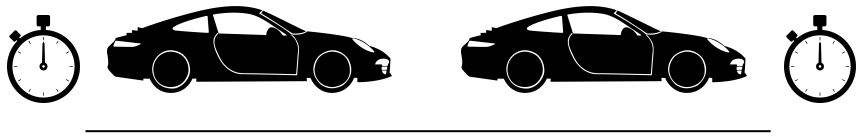
Fingerprint Generation



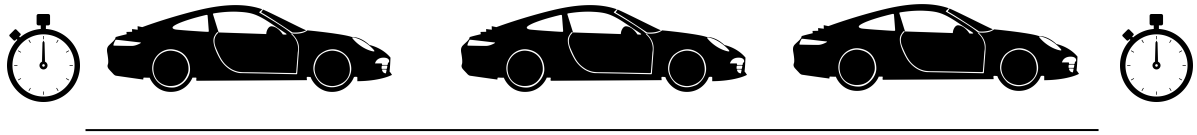




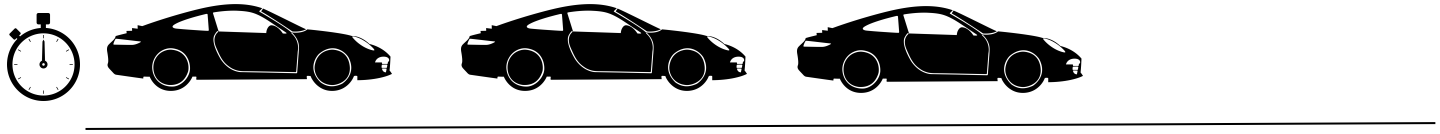
Fingerprint Generation



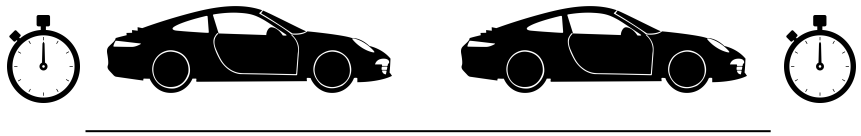
0.6s



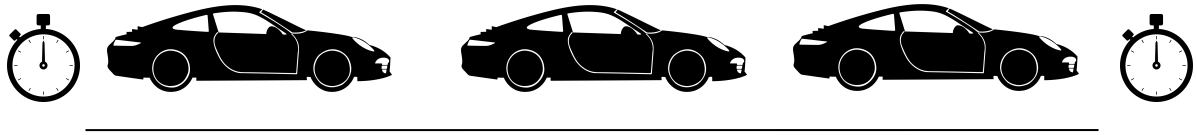
0.94s



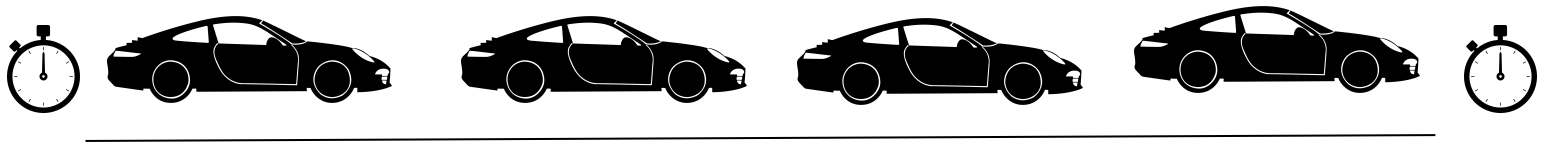
Fingerprint Generation



0.6s



0.94s



1.3s

Fingerprint Generation

0.6s 0.94s 1.3s

Fingerprint Generation

0.6s	0.94s	1.3s
0.6s	0.94s	1.4s

Fingerprint Generation

0.6s	0.94s	1.3s
0.6s	0.94s	1.4s
0.6s	0.94s	1.4s

Fingerprint Comparison

```
fp1=[{0.6,0.94,1.3},{0.6,0.94,1.4},{0.6,0.94,1.4}]

fp2=[{0.6,0.94,1.4},{0.7,0.94,1.3},{0.6,0.94,1.3}]
```

Fingerprint Comparison

```
fp1=[{0.6,0.94,1.3},{0.6,0.94,1.4},{0.6,0.94,1.4}]

fp2=[{0.6,0.94,1.4},{0.7,0.94,1.3},{0.6,0.94,1.3}]
```

```
mode_fp1={}
```

Fingerprint Comparison

```
fp1=[{0.6,0.94,1.3},{0.6,0.94,1.4},{0.6,0.94,1.4}]

fp2=[{0.6,0.94,1.4},{0.7,0.94,1.3},{0.6,0.94,1.3}]
```

```
mode_fp1={}
```

Fingerprint Comparison

fp1=[{**0.6**,0.94,1.3},{**0.6**,0.94,1.4},{**0.6**,0.94,1.4}]

fp2=[{0.6,0.94,1.4},{0.7,0.94,1.3},{0.6,0.94,1.3}]

mode_fp1={0.6}

Fingerprint Comparison

fp1=[{0.6,**0.94**,1.3},{0.6,**0.94**,1.4},{0.6,**0.94**,1.4}]

fp2=[{0.6,0.94,1.4},{0.7,0.94,1.3},{0.6,0.94,1.3}]

mode_fp1={0.6}

Fingerprint Comparison

fp1=[{0.6,**0.94**,1.3},{0.6,**0.94**,1.4},{0.6,**0.94**,1.4}]

fp2=[{0.6,0.94,1.4},{0.7,0.94,1.3},{0.6,0.94,1.3}]

mode_fp1={0.6,0.94}

Fingerprint Comparison

fp1=[{0.6,0.94,1.3},{0.6,0.94,**1.4**},{0.6,0.94,**1.4**}]

fp2=[{0.6,0.94,1.4},{0.7,0.94,1.3},{0.6,0.94,1.3}]

mode_fp1={0.6,0.94}

Fingerprint Comparison

fp1=[{0.6,0.94,1.3},{0.6,0.94,**1.4**},{0.6,0.94,**1.4**}]

fp2=[{0.6,0.94,1.4},{0.7,0.94,1.3},{0.6,0.94,1.3}]

mode_fp1={0.6,0.94,1.4}

Fingerprint Comparison

```
fp1=[{0.6,0.94,1.3},{0.6,0.94,1.4},{0.6,0.94,1.4}]

fp2=[{0.6,0.94,1.4},{0.7,0.94,1.3},{0.6,0.94,1.3}]
```

```
mode_fp1={0.6,0.94,1.4}
```

Fingerprint Comparison

fp1=[{0.6,0.94,1.3},{0.6,0.94,1.4},{0.6,0.94,1.4}]

fp2=[{**0.6**,0.94,1.4},{0.7,0.94,1.3},{**0.6**,0.94,1.3}]

mode_fp1={0.6,0.94,1.4}

Fingerprint Comparison

fp1=[{0.6,0.94,1.3},{0.6,0.94,1.4},{0.6,0.94,1.4}]

fp2=[{0.6,**0.94**,1.4},{0.7,**0.94**,1.3},{0.6,**0.94**,1.3}]

mode_fp1={0.6,0.94,1.4}

Fingerprint Comparison

fp1=[{0.6,0.94,1.3},{0.6,0.94,1.4},{0.6,0.94,1.4}]

fp2=[{0.6,0.94,**1.4**},{0.7,0.94,1.3},{0.6,0.94,1.3}]

mode_fp1={0.6,0.94,1.4}

Fingerprint Comparison

This process is then repeated, inverting the order and checking the most common values in the second one (`fp2`) with all the values from the first one (`fp1`).

In this case, the percentage of similarity would have been 100%, which is a **perfect match**. Our method would have determined that both fingerprints belonged to the same computer.

Function Selection

We decided to perform a preliminary set of tests to **assess** the different candidate functions, such as string::compare or crypt.

According to our results, different candidates provided good results, but one important point is that our method needs to use a function not often interrupted by the **scheduler** because, otherwise, the timing values would obviously be polluted.

Stability Tests

CPU Temperature
We stressed the CPU for 20 minutes at 100% load, successfully doubling the internal temperature.

While the increase in temperature can impacts clock-based measurements, we did not observe any variations or errors in our fingerprint identification.

A possible explanation is that as the two clock are physically located in the same machine, temperature would affect similarly both of them.

CryptoFP

Since this clock-based fingerprinting method works with virtually any simple function, we selected one based on its general availability and on the **possibility to generalize** our results and compare our native and web-based approaches.

We decided to implement our prototype by timing the execution of the **pseudo-random generator**, as it is available also in JavaScript, called by a wrapper in this scripting language.

Native Device Fingerprinting

In order to provide **homogeneity** and test our fingerprinting technique with the same type of computers rather than with different computers, we performed our experiments using two groups of machines with perfectly identical software (installed through a disk image) and hardware components.

The groups included **176** and **89** computers, respectively.

Native Device Fingerprinting

CryptoFP was able to distinguish every computer in each group. In other words, the uniqueness of our method in both tests is **100%**, even when both hardware and software are identical.

This shows that it is capable of detecting clock imperfections in order to **accurately distinguish** machines.

Web Device Fingerprinting

We used the **HTML5 Web Cryptography** API, that provides a wrapper that allows system-level cryptographic operations such as hashing, encryption, or generating random numbers.

In this case, we compare it with the other two state-of- the-art web hardware-level techniques: **canvasFP** and **WebGL fingerprinting**.

Methodology I

Since our fingerprinting does not produce a hash but it needs a comparison phase and is not transitive, we adapted the anonymity set measurement to an **identical comparison sets** that translates the idea behind anonymity sets to the comparisons performed by our method.

The size is no longer the number of computers with the same fingerprint, but the number of computers with the same number of **positive matches** with other computers.

Methodology II

Imagine four different machines: A, B, C and D.

Methodology II

Imagine four different machines: A, B, C and D.

– A matches B

Methodology II

Imagine four different machines: A, B, C and D.

– A matches B
– B matches A and D

Methodology II

Imagine four different machines: A, B, C and D.

– A matches B
– B matches A and D
– C matches D

Methodology II

Imagine four different machines: A, B, C and D.

– A matches B
– B matches A and D
– C matches D
– D matches C and B

Methodology II

Imagine four different machines: A, B, C and D.

– **A matches B**
– B matches A and D
– **C matches D**
– D matches C and B

```
sizes={1:2}
```

Methodology II

Imagine four different machines: A, B, C and D.

– A matches B
– **B matches A and D**
– C matches D
– **D matches C and B**

```
sizes={1:2,2:2}
```

Evaluation

Homogeneous
Both companies and universities often rely on large numbers of identical machines, which can greatly complicate fingerprinting (265 machines).

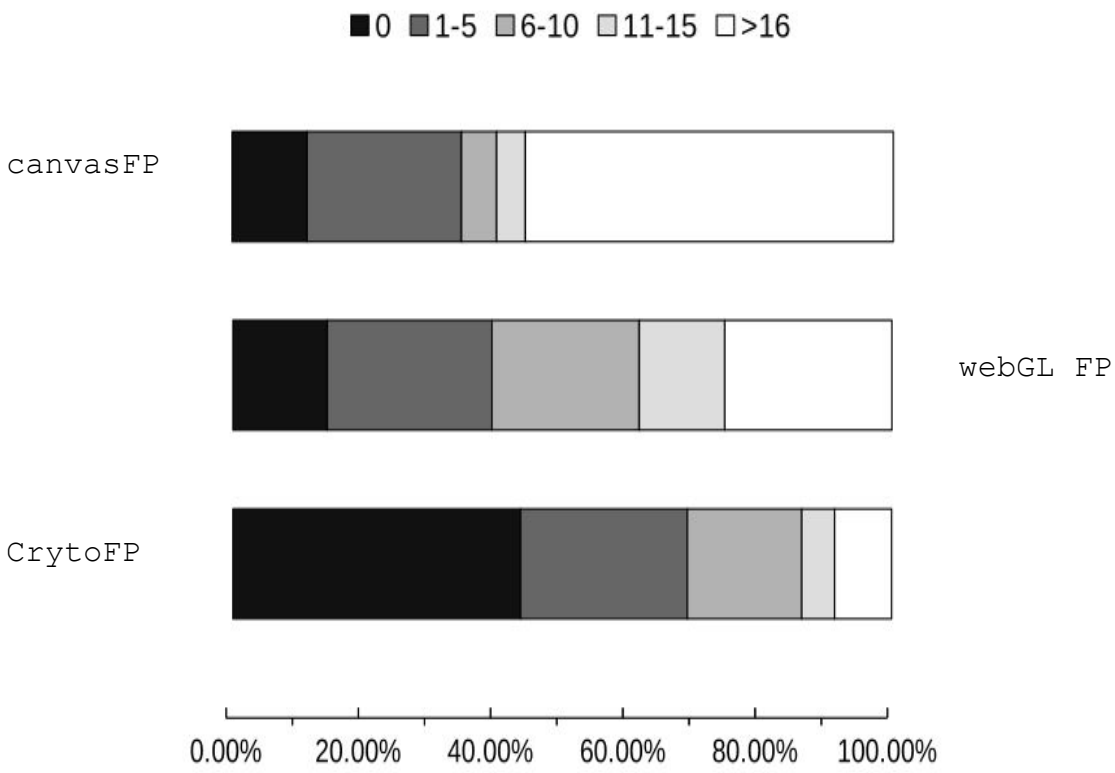
Heterogeneous
A classical web evaluation where users were asked to visit a website that performed all the techniques. Users where using their own machines and had no restriction on the computer
(300 participants).

Homogeneous Scenario

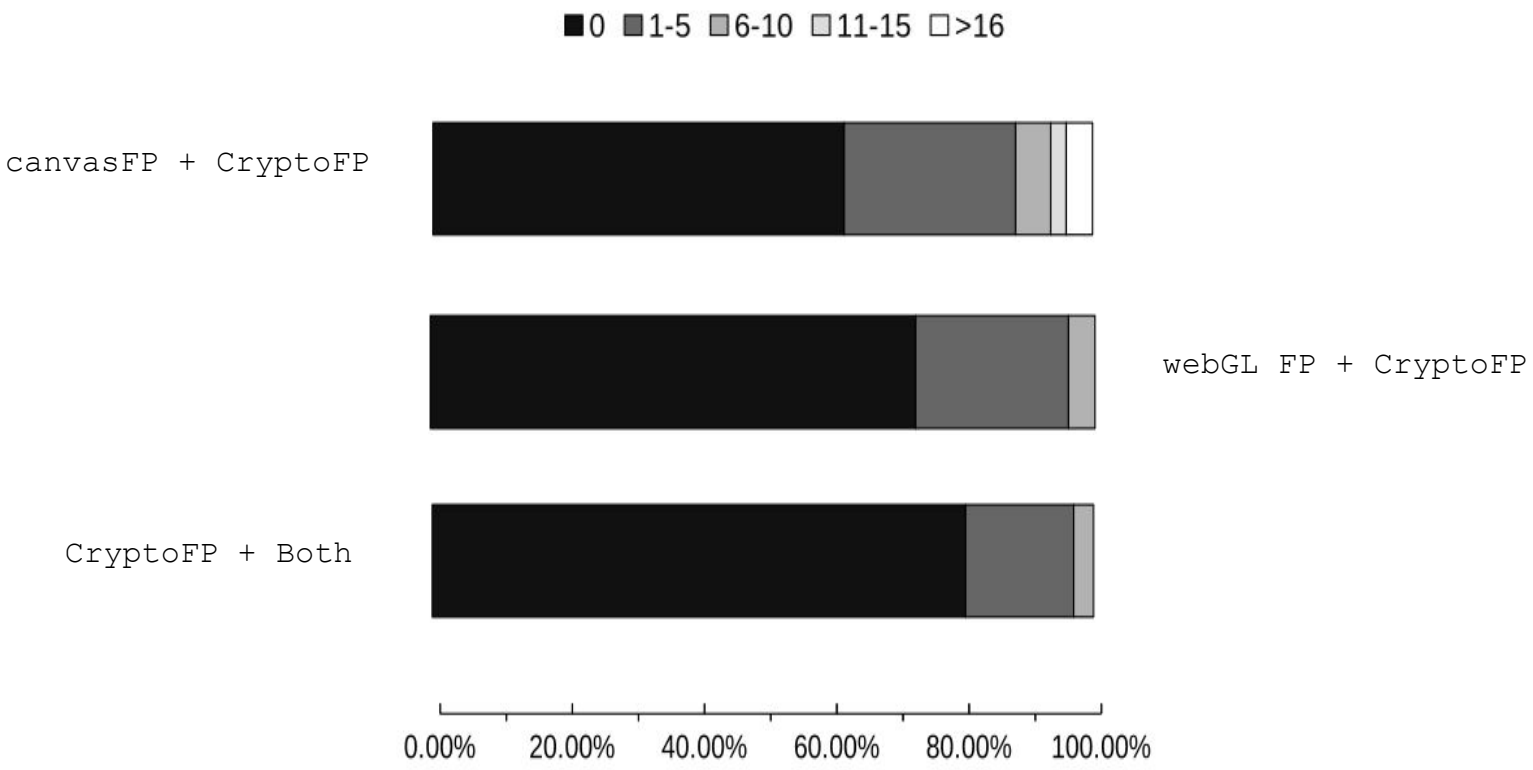
Existing techniques could **not differentiate any** of the computers in none of the two homogeneous groups, resulting in the same fingerprint for all computers.

CryptoFP was able to cover around 18% of the computers with the two first equally divided sets, **outperforming** previous techniques. Is less precise than the native implementation, due to a more coarse-grain precision offered by the HTML5 API.

Heterogeneous Scenario



Heterogeneous Scenario



Conclusions

We showed that a timing side-channel present in all modern computers can be used to **uniquely identify** a machine, tackling the main limitations of previous approaches.

We ported our technique to the web and showed that it **overcomes state-of-the-art** device fingerprinting techniques both in a homogeneous scenario and in a real-world web fingerprinting experiment.

People used to Rock Around the Clock, we prefer to

Clock Around the Clock

iskander.sanchez@deusto.es iskander-sanchez-rola.github.io