CPU Temperature as a Device Biometric

Evan Smith   
*School of Engineering and Computer Science*  
*Syracuse University*Syracuse, NY  
esmith15@syr.edu

*Abstract*—This paper will explore the possibility of using measured characteristics of devices as an input to a classifier that can identify individual devices. The characteristic of CPU temperature will be implemented as a proof-of-concept.

Keywords—biometric, classifier, device security, temperature measurement, verification, identification

# Introduction

Device identification is increasingly important as the volume of devices on networks rises. The standard methods of identifying devices are susceptible to spoofing, as they are self-reported and not fundamentally linked to the device itself. We will consider some of these existing identifiers and show that for some operations where device identification is important, a more advanced process may be needed. There are, however, some identifying features that a device can provide which are fundamentally linked with that particular set of hardware and chipset. The existing identifiers in this category are still static, however, and are therefore able to be faked with relative ease.

This paper proposes and tests a paradigm that authenticates devices using complex characteristics inherent to a specific device using a classifier. This is closer to a biometric such as a fingerprint than a declared single value like an IP address. For that reason, we will explore various classification techniques that could be effective for use in a device biometric scenario. Additionally, the paper will use a single metric as the input to the classifier, but the concepts should be extendable to other inherent measurements of a device. We choose CPU temperature as a proof of concept, since there is a fairly wide range of manufacturing consistency as far as heat sinks, thermal past application, ventilation, and more that can cause measurable discrepancies even between identical device hardware. Additionally, operating systems are configured to have easy access to CPU temperature from code, allowing for a simpler integration into a hypothetical security check.

# Existing Identifiers

IP addresses are often used to identify devices during network communications. While IP addresses are simple to select and are convenient for server-side authentication or rejection, they are also exceedingly easy to spoof. This allows for trivial circumvention of security measures intended to block access by unauthorized or offending devices by injecting a false IP address into HTTP headers and the like.

MAC addresses are better, since their value is hard coded into the network interface, but sniffing can still expose this information for later spoofing. Additionally, it is still possible to fake the MAC address through drivers and other software which nullifies the initial draw of the approach. With a spoofing setup and a given address, it is possible to produce a fleet of devices which all appear to have identical addresses and are superficially indistinguishable to a server.

In a similar vein as MAC addresses, mobile devices have an IMEI, which is analogous to the VIM number on a vehicle (1). While these are also hard coded into the device, it is possible to flash a new, arbitrary IMEI onto an existing device using known technology. This practice is illegal in many countries but is currently still legal in the United States (2).

The weaknesses of IP and MAC addresses are widely known, and there are many ways of attempting to validate the provided address beyond taking it on good faith. One example uses Layer Three Switches to validate source IPs within a LAN (3). While possible to reduce these weaknesses using defensive hardware or specifics as described, reliance on a single, static value is inherently vulnerable, and pursuing a more dynamic identifier that is more difficult to spoof would be beneficial for high-security situations. Device biometrics slot into this niche. We propose that such biometrics would require submission of several benchmarked values from the device which are then used to positively or negatively identify the device based on the profile for that trait stored for their MAC address. This approach combines the benefit of the hard-coded single identifier (MAC) with a validation step against spoofing or driver manipulation.

# Classification Techniques

For this use case, we will be classifying inputs consisting of several temperature readings over time and resolving to a single device identifier class. For this purpose, we will utilize many Python libraries, such as SciKit (4), NumPy (5) and SciPy (6). These libraries are widely used and extremely well documented. They also provide the ability to both make custom implementations of classification techniques as well as use some classifiers out-of-the-box.

Given the small amount of training data that will be provided, and the desire to have more visibility to the decision-making process, we will explore Principal Component Analysis (PCA) and other Singular Value Decomposition (SVD) approaches as the overall classifier type. Between these tools there are multiple PCA variations, such as PCA and SparsePCA. These can extract the key features in the temperature/time series and may allow for exploration of what components of each device’s measurements are the most distinct, which will be interesting to compare between devices of the same type. We can then more clearly see whether this device biometric is able to successfully distinguish between the same chipset on different devices.

An open question is the distinction between using a multi-class and binary classifier for this task. In our use case, multi-class classification would equate to identification, that is, given a CPU temperature profile, determine which device it is from. A binary classification would take that CPU profile as well as the declared device identifier and determine whether the profile matches the identifier.

It is likely that the latter will be more accurate, given our smaller sample size, but for the purposes of this paper and to assess the viability of the biometric more completely, we will look primary to the identification case. The results from identification will show the devices that are the best fits to the given measurements and can allow us to see what sort of devices are mistaken for each other, rather than a pure binary of answers from the authorization case. Our PCA approach will allow for both identification and authentication if effective, as the most likely identified class can simply be selected as the authorized class and compared against the declared class identity.

# Experiment Design

The dataset for this experiment is a small set of Arduino microcontrollers from two different product families. The Arduino Nano Every features the ATMega4809 processor, while the Arduino Uno Rev3 has the ATmega328P. In addition to the difference in processors, each device has a different form factor. The Every is very slim and compact with minimal additional circuit components and the Uno has a fuller feature set in a larger board. Combined, we expect that these boards will demonstrate a sufficient difference in device biometrics, especially CPU temperature during certain operations.

We consider the baseline level of success for this paper to be able to distinguish between two devices of different types based on our device biometric. The next level of success is to distinguish between two individual builds of the same device chipset. For this purpose, several Every boards will be used to try and detect differences in manufacturing steps, such as thermal paste application and soldering. Some boards will also be enclosed in a container to change the ambient environment.

It is expected that the variations between devices of the same type will be substantially less than the difference between different device types, but we hope to mitigate that large gap by structuring the identification as a verification rather than identification. That is, the device will provide a MAC and CPU temperature data while the server will check the provided biometric information against the benchmark on file for that MAC. This will be helpful given the necessarily small set of devices available for training.

Another factor that could disrupt the experiment is the ambient temperature. It is not desirable for a device to fail authentication if the temperature around it is different from its benchmark measurement run, and we should plan for our method of detection to be relatively robust in the face of environmental changes. To combat this, we will have the devices report both their absolute temperatures and the deviation of their CPU temperature from their initial measurement. This should help to show whether standardizing the input to our classifier will improve classification across different ambient temperatures.

# Data Collection

Data collection was done using a custom CPU soaking program written on the Arduino stack. This program ran a parabolic set of loops, each checking the RX pin of the controller, which is among the most instruction-intensive tasks for the microcontrollers (7). After each loop, the temperature of the CPU was recorded using the on-chip temperature-sensitive resistor provided for that purpose.

The loops were parabolic due to the oscillating number of iterations done in each cycle, with the intention of exposing a clearer relationship between CPU usage and internal temperature. If the iterations were constant or purely increasing, the concern was that the data collection would simply correlate with the amount of time that the processor was running rather than the CPU effort. As discussed later, the data collection proved to correlate more strongly to overall soak time rather than the soak intensity at any given point.

A single unit for the purposes of identification for the experiment was defined as a set of 100 temperature measurements, one taken directly before and after 50 CPU soak cycles with a one second cooldown between cycles. These measurement units took roughly two minutes each, which is substantially longer than most biological biometric measurements, but is within the realm of possibility for device verification. It is not difficult to imagine integrating such a measurement process into the boot routine of a device, or on the launch of a particularly sensitive application or network connection.

The experiment utilized four total devices: three of the Arduino Every boards, and one Arduino Uno board. Each device took 40 measurement units at the same ambient temperature, with each starting from a powered-off state (fully cooled) and running 20 measurements at a time. The data was stored as a combined .csv file labelled by the designated ID of the device, at which point post-processing was performed to produce a standardized dataset using the initial temperature from each unit measurement as a baseline. Both datasets were made available for the classification training and testing in the next stage.

| Device ID | Device Type |
| --- | --- |
| 1 | Arduino Every |
| 2 | Arduino Every |
| 3 | Arduino Every |
| 4 | Arduino Uno |

1. List of devices used for data gathering and their type.

# Classification Results

Temperature measurement units from data gathering were then examined in Python to attempt classification. Using the key tools from the SciPy and SciKit stacks, the data was split into two for this process with 30 measurements per device in the training set and 10 measurements in the test set. Work was conducted in a Jupyter notebook to allow for rapid prototyping and assessment of progress in the classifiers.

The first attempt to generate a classifier was done using the same steps described below but using the normalized temperature dataset. This had an unsatisfactory outcome, with SVD struggling to pull out meaningful singular values that classified the data properly. While there was some clustering in singular vectors within the same device types, even that result was highly variable based on how the training and test data was divided. This pointed towards an overtrained or misleading classifier at best using the standardized data, and there is low confidence that this approach would successfully disambiguate between devices of the same type. For those reasons, the selected dataset was unnormalized in that the values were absolute temperatures. However, the ambient temperature conditions during data collection were identical, so mismatching the scale of the values was not a concern.

To have a more fine-grain control and understanding of the overall PCA process, SVD was used to pull out all singular values and assess how many of the values contain useful information. The overall sigma vector was largely flat, which was a good sign that most of the information distinguishing the devices was contained in the initial values.

Shape, square

Description automatically generated

1. Graph of the sigma vector of the training set

Shape

Description automatically generated

1. Graph of the first 5 sigma values of the training set

It was clear after zooming into the first few values that the first sigma from the SVD was the most important for classification. The first three values were the only ones that were visibly important and should contain most of the total information. The first three singular vectors were then examined to verify that they showed significant trends.

Chart, line chart, histogram

Description automatically generated

1. Graph of the first 3 singluar vectors of the training set

The singular vectors were the first indicators that the data collected seemed to lead in unexpected directions. The trend is largely positive over time, especially for the second vector. The expected result was to see a wave corresponding to the oscillating CPU soak program being run on the devices while the measurements were taken. Regardless of the expectations, the trends in the top two singular vectors were so strong that confidence was built that a classifier trained on the data could be make accurate identifications.

To convert the SVD results into a form that allows for testing using our input data, we need to create a “fingerprint” for each known device. A fingerprint process concept allows for comparing an arbitrary measurement unit to the known device measurements and facilitated classification.

The fingerprints were calculated using the inner product of the measurement unit and some subset of the singular vectors. At first, the promise of this technique was demonstrated by using only one singular value to generate fingerprints. This produced a set of scalars, one per training measurement unit, which could be graphed as a set of ranges.

Chart

Description automatically generated

1. SVD-generated fingerprints using one singular value

The ranges seen here already are very positive results for classifying a sample by device type. The deviation of fingerprint values for device 4 (Arduino Uno) is very tight and relatively distinct from devices 1-3 (Arduino Every).

The next goal was to increase the dimensionality of the fingerprints through projecting onto another singular value and assess whether that improved the classification potential. These results are best visualized as a scatterplot, where the classification is to subdivide the space such that test data can be converted to a fingerprint pair and classified based on which space it falls into.

Chart, scatter chart

Description automatically generated

1. SVD-generated fingerprints using two singular values

These results significantly reduced the overlap of the aggregate fingerprint with the addition of another dimension. The separation between device types is even more pronounced and is nearly perfectly subdivided. While the specific devices are more difficult to distinguish, there are clear bands which give at the very least a preferred classification for virtually any random point in the space.

While these results were achieved using SVD to maintain a closer degree of control over the outcome, the ultimate output we have created and the use case we describe makes a PCA solution roughly equivalent in theory. To verify that our approach was equally successful to an out-of-the-box function in our Python libraries, we ran the same test data through SKLearn’s PCA function for only 2 principal components.

Chart, scatter chart

Description automatically generated

1. Built-in PCA-generated fingerprints using two principal components

These results built high confidence that the SVD-based approach for classification was both nearly identical to a built-in PCA, but was also less of a black box concept since we can more easily examine the singular vectors themselves.

The final step for classification is to run the test set through the same analysis and compare the generated fingerprints to the divided space. For that comparison step, we model the set of training fingerprints as a multivariate Gaussian distribution, then select the most probable device that each test fingerprint belongs to by simply taking the max probability.

The last question was to resolve what number of singular values classified the test data most accurately. While the previous analysis suggested that only the top two values would be helpful, it was found that three values was the more accurate. There are some non-random patterns to the third vector (as shown in Fig 4), so it’s not out of the question that this fingerprint would help distinguish certain cases.

| Device ID | Classification Accuracy | |
| --- | --- | --- |
| 2 Singular Values | 3 Singular Values |
| 1 | 90% | 90% |
| 2 | 10% | 60% |
| 3 | 100% | 100% |
| 4 | 80% | 80% |
| *Overall* | 70% | 85% |

1. Classification accuracy per device for two vs. three singular values

A picture containing square

Description automatically generated

1. Confusion matrix for three singular value classification

# Observations and Improvements

The overall takeaway from this proof of concept is that CPU temperature patterns can absolutely be used as a biometric of some sort. That is not to say that the techniques for measurement and classification used here would be the best way to implement such a biometric in an industry application – there are many areas for improvement and further research.

To start, the data gathering for this paper had some fundamental issues. One of these was the devices selected and the situation in which the measurements were run. It was unrealistic that all devices in the training set had a single core. This makes measurement of CPU work both easier, since running any CPU soak program is guaranteed to be the only executing instructions and helps to standardize computations across devices.

However, a more complete take on the meaning of CPU temperature is to detect temperature differences due to both how the physical hardware was fabricated given exact computational load match but also due to variety in the typical background processing load. We are unable to meaningfully measure varying computation load as a biometric with the experiment design described in this paper, as the data gathering code is a stand-alone program and cannot run alongside the nominal tasks of the devices given the restrictions of the Arduino system as well as the single core. With the possibility of the soak and measurement program running separately from the nominal work of the device, there is access to much more distinctive and interesting biometric input data that could help to classify a device.

Another issue seen in data collection is that the oscillating soak program failed to really produce an oscillation of CPU effort or CPU temperature. A potential and simple resolution here would be to have even identical chipsets used in different applications such that the load on the chip and the form factor were less homogenous. A case, nearby resistors, or number of pinouts could all create a more clear temperature measurement pattern that aligns with CPU soak level rather than simply the amount of time that the device has been running the soak.

While the classification step described in this paper would certainly need to be adapted for a broader application, the usage of SVD and/or PCA worked very well for this data input type. With a more varied input set, more of the singular values should contain meaningful information. This would allow the classifier to disambiguate a larger pool of possible labels. In such a case, there should also be changes made to the assessment of correctness used in this experiment.

Since classification for SVD and/or PCA is a more open book, there is access to the normalized percentage chance that a given fingerprint corresponds to any given label. In a pure identification context, the approach used in this paper to simply take the maximum percentage would be the only possible solution. In this case, it can be expected that as the total set of devices grows it will be more difficult to clearly identify them especially within the same chipset.

If we shift the use case to authentication, it is likely that the accuracy would increase. This would be analogous to a device sending its unit CPU temperature measurements along with a declared MAC address to a server for authentication. The multi-label classification would then be transformed into a binary classification: does the fingerprint provided most closely match the declared identity, or not?

There is likely some subtlety here, such as potentially tweaking the sensitivity to authenticate if the identity provided is at least the second or third choice for the fingerprint. This approach would be quite flexible given varying business rules. The false positive vs false negative rate could be tuned using a parameter governing how close to the top label a classification must be to be authentic. For a high-security use case, it may be required that the classification selects the provided device as the top choice. This would likely require accepting multiple attempts for authentication due to false negatives, which is a common concession when using human-based biometrics in similar use cases. A more consumer-level solution would likely allow more slack into the classification decisions, as inconveniencing the user might be more harmful than allowing an occasional fake device.

# Conclusion

Even with the tiny sample size of devices and lifetime measurements available for this experiment, alongside the very homogenous form factor and uses for all devices involved, this proof of concept was a success. An overall accuracy of 85% identification given the data collection issues that arose is exciting. Not only was the classifier able to distinguish different chipsets apart, but it also proved to be capable of correctly labelling devices of the same type on average.

For use cases where the expectation is to validate the actual hardware being used, a physical device biometric like CPU temperature under load appears to be a valid approach to authentication. CPU temperature was selected for this paper mainly because of the relative ease of measurement. Some other potential attributes that could be used as biometrics are instructions per second, transmission times for pings, or processing time for a specific computationally intensive task. All these alternatives would be too variable to trust a single measurement in time. For that reason, the established pattern of using a time series as input and transforming it into a fingerprint value should continue to be effective. Following the pattern of the Viola-Jones face recognition approach, combining many of these biometrics would improve the net confidence in authentication or identification correctness. Furthermore, data submitted for authentication could be used to grow the training set, improving accuracy over time.

Given the experiment conducted in this paper, it seems that pursuing device biometrics using time series data collection and an SVD/PCA fingerprinting scheme is a viable path to authenticating or identifying specific devices based on physical characteristics.

##### References

1. “What Is IMEI Number?” *IMEI.Info*, 4 Feb. 2021, www.imei.info/faq-what-is-IMEI.
2. "Text - S.3186 - 112th Congress (2011-2012): Mobile Device Theft Deterrence Act of 2012." Congress.gov, Library of Congress, 15 May 2012, <https://www.congress.gov/bill/112th-congress/senate-bill/3186/text/is>.
3. H. Lema, F. Simba and A. Ally, "Preventing Utilization of Shared Network Resources by Detecting IP Spoofing Attacks through Validation of source IP Address," 2018 IST-Africa Week Conference (IST-Africa), 2018, pp. Page 1 of 8-Page 8 of 8.
4. “Scikit-learn: Machine Learning in Python”, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.
5. Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke & Travis E. Oliphant. Array programming with NumPy, Nature, 585, 357–362 (2020), DOI:10.1038/s41586-020-2649-2
6. Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, CJ Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E.A. Quintero, Charles R Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. (2020) SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. Nature Methods, 17(3), 261-272.
7. “Why is the digital I/O in Arduino slow and what can be done about it?” *codeproject.*com, 16 May. 2014, https://www.codeproject.com/Articles/589299/Why-is-the-digital-I-O-in-Arduino-slow-and-what-ca
8. “Scikit-learn: Machine Learning in Python”, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.