Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Comparative Timing Characterization of Early-Exit and Fixed-Work Secret Comparisons on ESP32 and ESP8266

Abrar Sobhan

abrarsobhan2006@gmail.com

Independent Researcher

Research Article

Keywords: timing side channels, early-exit comparisons, constant-time programming, embedded cryptography, ESP32, ESP8266, CPU-cycle measurements

Posted Date: November 9th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-11639965/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: The authors declare no competing interests.

Comparative Timing Characterization of Early-Exit and Fixed-Work Secret Comparisons on ESP32 and ESP8266

Abrar Sobhan
Independent Researcher, Bangladesh
Preprint — not peer-reviewed

Abstract

An early-exit secret comparison can expose matching-prefix length through execution timing. We quantify this behavior for 16-byte early-exit comparisons and a fixed-work XOR/OR alternative on ESP32 and ESP8266 microcontrollers. Internal CPU-cycle measurements were collected for prefix lengths 0–16 under two interrupt/barrier configurations, yielding 40,800 observations. Across prefix lengths 0–15, early-exit median timing rose by 40 cycles per matching byte on ESP32 and 25 on ESP8266. In the recorded sessions, 63 of 68 early-exit groups contained 300 identical readings, and all 17 group medians were distinct within each configuration. Fixed-work median timing did not vary with prefix length, although occasional elevated observations remained. These measurements characterize implementation-specific on-device timing leakage and the cost of the fixed-work routine; they neither establish binary-level constant-time behavior nor demonstrate a remotely observable attack.

Keywords: timing side channels; early-exit comparisons; constant-time programming; embedded cryptography; ESP32; ESP8266; CPU-cycle measurements; microcontrollers; information leakage; XOR/OR comparison.

1. Introduction

Comparing secret values is a common operation in authentication and cryptographic software. Although a comparison function may return only whether two inputs match, the time required to produce that result can reveal information about how the comparison was performed. In an early-exit implementation, the function stops as soon as it encounters a mismatching byte. Consequently, inputs that match a longer prefix of the secret may require more execution time than those that differ at the beginning. Such input-dependent behaviour is relevant to timing side-channel attacks, which exploit differences in execution time to infer information about otherwise protected operations (Kocher, 1996).

One approach to reducing this type of timing dependence is to process the entire input instead of terminating at the first mismatch. For example, a fixed-work comparison can

combine byte-wise XOR results into an accumulator and return a result only after examining every byte. This removes the early termination present in a conventional byte-by-byte comparison. However, source-level control flow alone does not guarantee constant-time execution, as the generated machine instructions and target architecture can also influence timing behaviour. Previous research has investigated these issues through statistical testing and analysis of compiled implementations (Cinal, 2026; Reparaz et al., 2017).

Timing side channels in embedded systems have also received increasing attention. Van Dijck et al. examined timing vulnerabilities in low-end trusted execution environments, including those associated with comparison functions and compiler-generated instructions. More recently, Cinal investigated compiler- and architecture-related timing leakage on Xtensa-based ESP32 devices. These studies demonstrate that timing behaviour must be considered across different layers of embedded software and hardware (Cinal, 2026; Van Dijck et al., 2025).

Building on this research, the present study examines two 16-byte secret-comparison implementations on ESP32 and ESP8266 microcontrollers. An early-exit implementation and a fixed-work implementation were evaluated by varying the number of initial bytes matching a fixed secret. Execution time was measured using the devices' internal CPU-cycle counters under two firmware configurations: one that temporarily suppressed interrupts during measurement and another that allowed normal interrupt activity. The experiment produced 40,800 recorded measurements across the four device and configuration combinations.

The study examines the relationship between matching-prefix length and execution time, within-group repeatability of early-exit measurements, cross-device differences, and occasional timing variation. Its contribution is a controlled, cross-device dataset of 40,800 on-device CPU-cycle measurements that quantifies early-exit timing slopes, within-session prefix-length distinguishability, fixed-work execution costs, and occasional elevated observations across the four tested configurations. These measurements provide an empirical baseline for the specific unoptimized routines and do not establish a new class of vulnerability, demonstrate secret recovery, or show a practical remote timing attack.

2. Related Work

Timing side-channel attacks have been studied for several decades. Kocher (1996) showed that differences in the execution time of cryptographic operations could reveal information about secret values. Subsequent research has examined how these differences can be identified and mitigated. Reparaz et al. (2017) introduced *dudect*, a tool that uses repeated timing measurements and statistical testing to detect evidence of variable-time execution. Almeida et al. (2016) approached the problem through formal verification, developing a method for checking constant-time properties in software implementations. These studies provide different ways of assessing timing security, ranging from experimental testing to verification of program behaviour (Almeida et al., 2016; Kocher, 1996; Reparaz et al., 2017).

Recent research has also examined how timing observations should be interpreted statistically. Dunsche et al. (2024) developed a testing methodology that controls false-positive findings under noisy measurement conditions. That work addresses a different question from fitting a regression line to group medians: a strong descriptive timing trend does not, on its own, establish how reliably leakage could be distinguished from noise in an external attack (Dunsche et al., 2024). Statistical timing analysis has also been extended beyond elapsed-time measurements for particular threat models. Bhattacharya and Verbauwhede (2021) combined statistical testing with hardware performance-counter observations to investigate microarchitectural leakage on general-purpose processors. Ghaniyoun et al. (2025) developed MicroSampler, which tests for associations between secret inputs and microarchitectural states obtained through cycle-accurate RTL simulation. Neither setting is equivalent to the internal comparison-call measurements in this study, but both illustrate why unchanged median cycle counts alone cannot rule out all leakage mechanisms (Bhattacharya and Verbauwhede, 2021; Ghaniyoun et al., 2025).

Timing behaviour has also been investigated on embedded processors. Dewald et al. (2017) developed a framework for analysing the timing security of assembly programs on 8-bit AVR microcontrollers. Their work demonstrates how processor-specific instruction behaviour can be incorporated into security analysis. More recently, Van Dijck et al. (2025) examined timing side channels in low-end trusted execution environments. They identified vulnerabilities associated with comparison routines, compiler-generated instructions, and hardware behaviour, and demonstrated their security implications. These findings establish that timing vulnerabilities are not limited to general-purpose computers (Dewald et al., 2017; Van Dijck et al., 2025).

Microarchitectural effects remain relevant even on relatively simple embedded processors. Bognár et al. (2023) profiled a DMA-related timing side channel on an openMSP430 processor and extended compiler-based mitigation and binary validation techniques. Their target hardware and observability differ from the internal comparison-call cycle counts studied here, but their findings show why timing behavior cannot always be understood from source code alone (Bognár et al., 2023).

Compiler behaviour is another important consideration. Cinal (2026) investigated timing side channels in cryptographic implementations running on Xtensa-based ESP32 devices. The study showed how compiler-generated instructions could introduce secret-dependent timing behaviour even in code intended to follow constant-time programming practices. The work focused on arithmetic operations in X25519 implementations and demonstrated practical key-recovery attacks. It differs from the present study, which examines controlled matching-prefix variations in two simpler comparison routines rather than a complete cryptographic implementation (Cinal, 2026).

The preservation of constant-time properties through compilation has also been studied directly. Barthe et al. (2018) developed proof methods for determining whether compilation passes preserve constant-time countermeasures. In a large-scale study, Schneider et al. (2025) identified compiler-introduced secret-dependent operations in cryptographic implementations intended to resist timing attacks. These findings are relevant to the present fixed-work comparison: processing every byte in the source code and observing flat group

medians do not amount to a binary-level constant-time guarantee (Barthe et al., 2018; Schneider et al., 2025).

Complementary approaches attempt to verify constant-time properties rather than infer them from a limited set of timing observations. Blazy et al. (2019) used abstract interpretation to analyse potentially secret-dependent flows in C programs. Barthe et al. (2020) formally verified that a C compiler preserves constant-time properties under its model, while Cai et al. (2024) combined taint analysis and program verification in CT-Prover. The present work does not undertake source-level verification or inspect the resulting machine instructions; it instead characterizes observed cycle-count trends in the tested builds (Barthe et al., 2020; Blazy et al., 2019; Cai et al., 2024).

Compiler and hardware countermeasures also highlight the performance cost of timing protection. Van Cleemput et al. (2020) evaluated adaptive compiler strategies that trade timing protection against execution overhead, and Miao et al. (2023) investigated hardware support for constant-time programming. Their designs differ substantially from the two unoptimized comparison routines evaluated here. Accordingly, the additional cycles observed for the fixed-work routine should not be interpreted as the general overhead of constant-time protection in optimized embedded applications (Miao et al., 2023; Van Cleemput et al., 2020).

Other studies have considered how timing-related information can be observed on embedded hardware. Robyns et al. (2020), for example, investigated electromagnetic leakage from an ESP8266-based NodeMCU device to identify cryptographic operations. Winderix et al. (2021) studied interrupt-latency side channels in embedded software and proposed compiler-assisted countermeasures. Although these studies involve different observation methods and threat models, they show that embedded execution behaviour can be examined through several forms of side-channel analysis (Robyns et al., 2020; Winderix et al., 2021).

A different body of work investigates externally observable timing signals. Brumley and Boneh (2005) demonstrated key extraction through timing measurements of a networked OpenSSL service. Van Goethem et al. (2020) later explored concurrency-based techniques that infer relative timing differences from response order rather than absolute network latency. Unlike those attack settings, the present experiment measured CPU cycles within the microcontrollers and did not assess remotely observable timing or secret recovery (Brumley and Boneh, 2005; Van Goethem et al., 2020).

Previous studies have established timing attacks, compiler-induced leakage, and methods for testing or verifying constant-time code. The present study addresses a narrower empirical question: how do early-exit and fixed-work 16-byte comparisons behave, at CPU-cycle resolution, on ESP32 and ESP8266 under the tested firmware configurations? It reports matching-prefix timing slopes, within-session repeatability and distinct early-exit group medians, fixed-work execution costs, and unusual measurement counts across a publicly archived dataset of 40,800 observations. The results offer a comparative baseline for the specified boards and builds.

3. Methods

3.1 Experimental Design

This study examines the execution time of two 16-byte secret-comparison implementations on ESP32 and ESP8266 microcontrollers. The first implementation follows an early-exit approach, stopping the comparison when it finds the first mismatching byte. The second uses a fixed-work approach, where all 16 bytes are processed regardless of whether a mismatch is found.

For both implementations, we varied the number of initial matching bytes between the candidate and the stored secret from 0 to 16. We then measured the execution time in CPU cycles on both microcontrollers under two conditions: with interrupts suppressed and with normal interrupt activity allowed. The overall experimental workflow, from candidate generation to data recording, is illustrated in Figure 1.

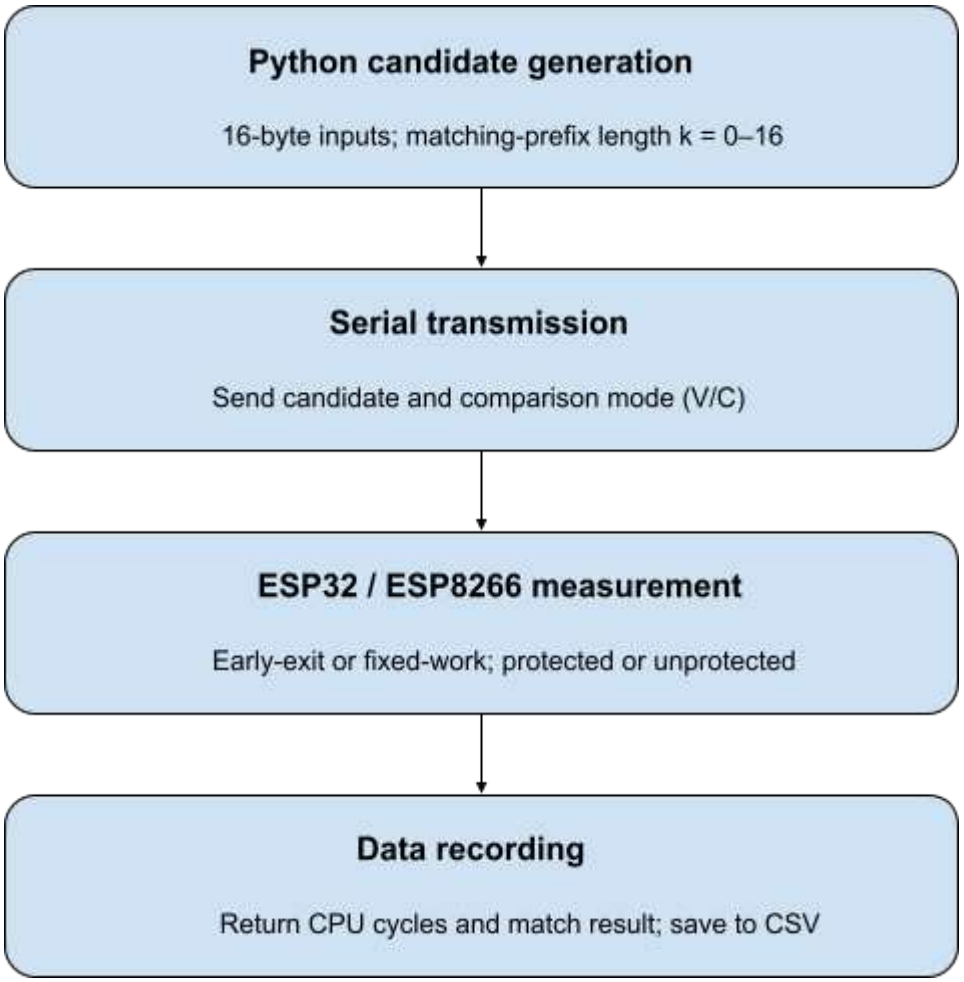


Figure 1. Experimental workflow showing candidate generation, serial communication, on-device comparison timing, and data recording for the ESP32 and ESP8266 microcontrollers.

3.2 Hardware and Software

The experiment was conducted using ESP32 and ESP8266 development boards connected to a Windows computer. The ESP32 was configured using the ESP32 Dev Module board profile in Arduino IDE and operated at a CPU frequency of 240 MHz. The ESP8266 was a NodeMCU-style development board configured to run at 80 MHz. The development boards used in the experiments are shown in Figure 2.

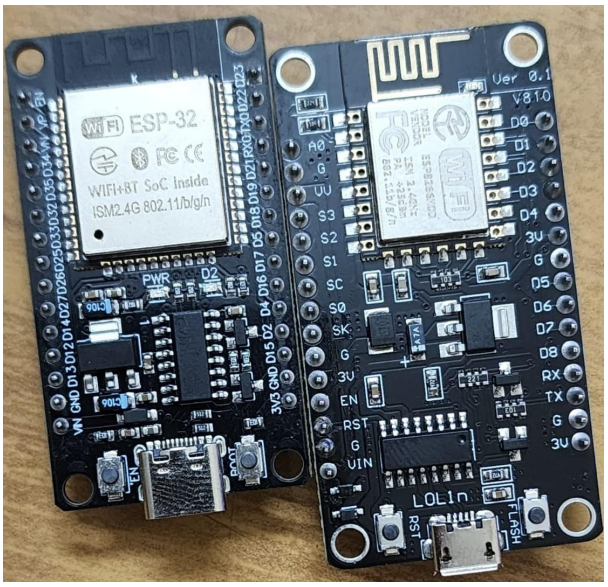


Figure 2. ESP32 (left) and ESP8266 NodeMCU-style (right) development boards used in the timing measurements. Images by the author.

The two microcontrollers were tested separately. We first connected the ESP32 to the computer using a USB Type-C cable and collected its measurements. After completing the ESP32 tests, we disconnected it and connected the ESP8266 using a USB cable to carry out the remaining measurements.

The firmware was uploaded using Arduino IDE, while Python scripts were used to control data collection. The scripts were run through the Windows command-line environment and communicated with each microcontroller over a serial connection at 115200 baud. The Arduino Serial Monitor was kept closed during data collection to prevent conflicts with the Python scripts.

The experimental firmware was built using Arduino IDE 2.3.10. For the ESP32, the firmware was compiled for the ESP32 Dev Module profile (Arduino-ESP32 core 3.3.11, 240 MHz) using the Xtensa toolchain package esp-x32 (version 2601) with Xtensa GCC 14.2.0, as documented in the retained ESP32 protected sketch build log. For the ESP8266, the firmware was compiled for the NodeMCU 1.0 profile (ESP-12E Module; Arduino-ESP8266 core 3.1.2, 80 MHz) using the Xtensa LX106 compiler package 3.1.0-gcc10.3-e5f9fec with GCC 10.3.0, as documented in the retained ESP8266 unprotected sketch build log. Both build logs specify the general -Os optimization setting, while the comparison functions were isolated with noinline and optimize("O0") function attributes to request unoptimized compilation and prevent function inlining.

Execution time was measured using the internal CPU-cycle counters of the microcontrollers. The ESP32 used esp_cpu_get_cycle_count(), while the ESP8266 used ESP.getCycleCount(). The measurements were saved in CSV files for later analysis.

3.3 Comparison Implementations

Two comparison functions were implemented to examine how the position of a mismatching byte affects execution time. Both functions compared a 16-byte candidate with a fixed 16-byte secret, but they differed in how they handled mismatches.

The first function used an early-exit approach. It compared the candidate and secret byte by byte, starting from the first byte. When a mismatch was found, the function immediately returned `false` without checking the remaining bytes. If all 16 bytes matched, it returned `true`. Therefore, the number of bytes examined depended on the position of the first mismatch.

Algorithm 1. Early-exit comparison

```
__attribute__((noinline, optimize("O0")))
bool verify_vulnerable(const uint8_t *input,
                       const uint8_t *secret,
                       size_t len) {
    for (size_t i = 0; i < len; i++) {
        if (input[i] != secret[i]) {
            return false;
        }
    }
    return true;
}
```

The second function used a fixed-work approach. Instead of stopping when a mismatch occurred, it continued comparing bytes until the entire input had been processed. It applied a bitwise XOR operation (`^`) to each pair of bytes and combined the results using a bitwise OR accumulator (`|=`). After processing all 16 bytes, the function returned `true` if the accumulated value was zero, indicating a complete match. Otherwise, it returned `false`.

Algorithm 2. Fixed-work comparison

```
__attribute__((noinline, optimize("O0")))
bool verify_constant_time(const uint8_t *input,
                          const uint8_t *secret,
                          size_t len) {
    uint8_t accumulator = 0;

    for (size_t i = 0; i < len; i++) {
        accumulator |= (input[i] ^ secret[i]);
    }

    return (accumulator == 0);
}
```

Both functions were marked with `noinline` and `optimize("O0")` attributes to prevent function inlining and request unoptimized compilation. The listings present the shared comparison logic used on both microcontrollers. In the original ESP8266 firmware, the noinline and optimize("O0") attributes were written separately, rather than in the combined

form shown above. The fixed-work implementation was designed to avoid the early termination present in the vulnerable function. However, its timing behaviour was evaluated only under the hardware and compilation settings used in this experiment.

3.4 Candidate Generation and Matching-Prefix Length

The input candidates were generated using Python to examine how execution time changes with the number of matching bytes at the beginning of a comparison. Each candidate was 16 bytes long and was constructed using a fixed 16-byte test secret (hexadecimal: 2B 7E 15 16 28 AE D2 A6 AB F7 15 88 09 CF 4F 3C). The number of consecutive matching bytes from the beginning was represented by the matching-prefix length k, which ranged from 0 to 16.

For each value of k between 0 and 15, the first k bytes of the candidate were copied from the secret, while the following byte was deliberately changed to ensure a mismatch at that position. The remaining bytes were generated using Python's random.randint() function. The collectors did not save the generated candidate byte strings or the pseudorandom seed, so the exact candidate sequence cannot be reconstructed from the CSV files. When k was 16, the candidate was identical to the secret, representing a complete match.

This approach allowed us to control the position of the first mismatch while keeping the input length constant across all trials.

3.5 Measurement Conditions: Interrupt Masking and Barriers

We measured two configurations, labelled 'protected' and 'unprotected' in the recorded datasets and figures. These terms describe firmware measurement conditions, not verified security properties. In the protected configuration, interrupts were temporarily suppressed during the comparison; the unprotected configuration allowed normal interrupt activity. As detailed below, the two configurations also differed in compiler memory-barrier use.

On the ESP32, the protected firmware placed the timed comparison inside a critical section using portENTER_CRITICAL() and portEXIT_CRITICAL(). This suppresses interrupts on the executing core, but does not necessarily stop activity on the other core. The ESP8266 used `noInterrupts()` and `interrupts()` for the same purpose. Both protected implementations also included compiler memory barriers around the cycle-counter measurements. These barriers were not present in the unprotected versions.

For each comparison, the firmware recorded the CPU-cycle counter immediately before and after the function call. The difference between these readings was used as the comparison-call cycle measurement. This interval includes function-call and cycle-counter overhead. Input reception and serial communication took place outside the timed region, so their duration was not included in the recorded cycle counts.

Both early-exit and fixed-work comparisons were tested under each configuration. Because the protected and unprotected firmware also differed in their use of compiler memory barriers, any differences between the configurations cannot be attributed solely to interrupt suppression.

3.6 Data Collection

Data were collected using Python scripts that communicated with the microcontrollers through a serial connection. For each trial, the script generated a 16-byte candidate and sent it to the selected microcontroller along with a character indicating the comparison mode. The firmware performed the requested comparison and returned the match result and the measured CPU-cycle count. These values were then recorded by the Python script.

Measurements were collected separately for each device under protected and unprotected conditions, producing four datasets in total. For each dataset, the early-exit implementation was tested first, followed by the fixed-work implementation. In both cases, the matching-prefix length was increased sequentially from 0 to 16, with 300 trials performed at each value.

Each dataset contained 10,200 measurements, giving a total of 40,800 observations across the four configurations. The results were stored in CSV files containing the trial ID, comparison mode, matching-prefix length, match result, and CPU-cycle count.

The measurements were collected in a fixed sequence rather than a randomized order. This collection order was retained in the dataset and is considered when interpreting possible timing variations across trials.

3.7 Data Analysis

The four CSV datasets were analyzed to examine how execution time varied with matching-prefix length, comparison method, and measurement condition. For each device and configuration, the measurements were grouped by comparison mode and matching-prefix length. The median, mean, and standard deviation of the CPU-cycle counts were calculated from the 300 trials in each group. The median was used to represent the main timing trends because some individual measurements were considerably higher than the typical values.

For the early-exit implementation, linear regression was applied to the median CPU-cycle counts for matching-prefix lengths from 0 to 15. This allowed us to examine whether execution time increased consistently as more initial bytes matched the secret. The relationship was modeled using the following equation:

$$T(k) = \alpha + \beta k \quad (1)$$

Here, T(k) represents the estimated median execution time in CPU cycles at matching-prefix length k, α is the intercept, and β is the regression slope, expressed in cycles per additional matching byte. The coefficient of determination (R^2) was calculated to assess how closely the observed median values followed the fitted linear relationship. Separate regression models were fitted for each device and measurement condition.

The reported regression goodness-of-fit describes the relationship among the aggregated median points. It was not used as a statistical test for timing leakage, nor does it estimate whether an external observer could reliably distinguish the timing differences.

The complete-match condition (k = 16) was examined separately because the early-exit function reaches the end of the comparison and returns through a different path from the mismatching inputs. For the fixed-work implementation, median CPU-cycle counts were compared across all matching-prefix lengths to determine whether execution time changed with the position of the first mismatch.

Timing variability was characterized within each 300-trial matching-prefix group using the mean, sample standard deviation (SD), median, and readings above the group median. All observations, including initial trials and elevated measurements, were retained. For early-exit comparisons, the number of distinct cycle-count values within each 300-trial group was counted, and the 17 group medians (k = 0–16) were checked for uniqueness within each configuration. For fixed-work comparisons, group means ±1 SD are plotted across matching-prefix lengths in Figure 5. A fixed-versus-random dudect-style test was not performed: the experiment collected sequential prefix-length groups rather than randomized fixed and random input classes. The near-degenerate distributions and non-randomized collection order also limit the interpretation of conventional hypothesis tests across groups.

For comparisons between the two microcontrollers, CPU-cycle counts were also converted into execution time using the reported processor frequencies. This made it possible to compare the timing measurements in both processor cycles and absolute time.

4. Results

4.1 Effect of Matching-Prefix Length on Execution Time

Figures 3 and 4 show how median execution time changed with matching-prefix length on ESP32 and ESP8266. The reported R^2 values describe regressions fitted to 16 group medians (k = 0–15); they do not quantify within-group variability or the feasibility of an externally observed attack.

On the ESP32, the early-exit implementation showed a steady increase in execution time as more initial bytes matched the secret. Under the protected configuration, the median increased from 67 cycles at k = 0 to 667 cycles at k = 15. The corresponding values in the unprotected configuration were 64 and 664 cycles. In both configurations, each additional matching byte increased the median execution time by 40 cycles. Linear regression of the median values for k = 0 through k = 15 produced an R^2 value of 1.000.

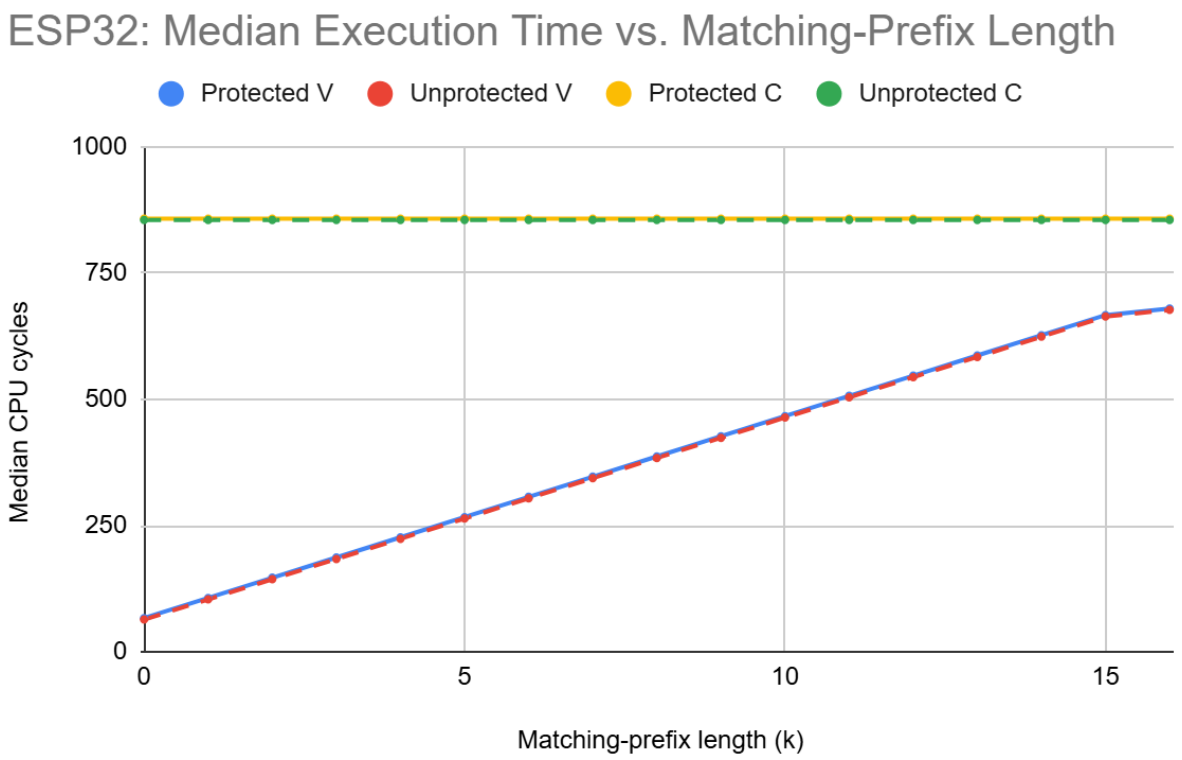


Figure 3. Median execution time across matching-prefix lengths on ESP32 (300 trials per point). In the legend, V denotes early-exit and C denotes fixed-work; protected denotes interrupts masked with compiler memory barriers, and unprotected denotes interrupts enabled without these barriers. The paired traces nearly overlap at this scale.

A similar pattern was observed on the ESP8266, although the increase per matching byte was smaller. In the protected configuration, the early-exit median rose from 51 cycles at k = 0 to 426 cycles at k = 15. In the unprotected configuration, it increased from 52 to 427 cycles. Both configurations showed an increase of 25 cycles for each additional matching byte, with an R^2 value of 1.000 for the regression based on group medians.

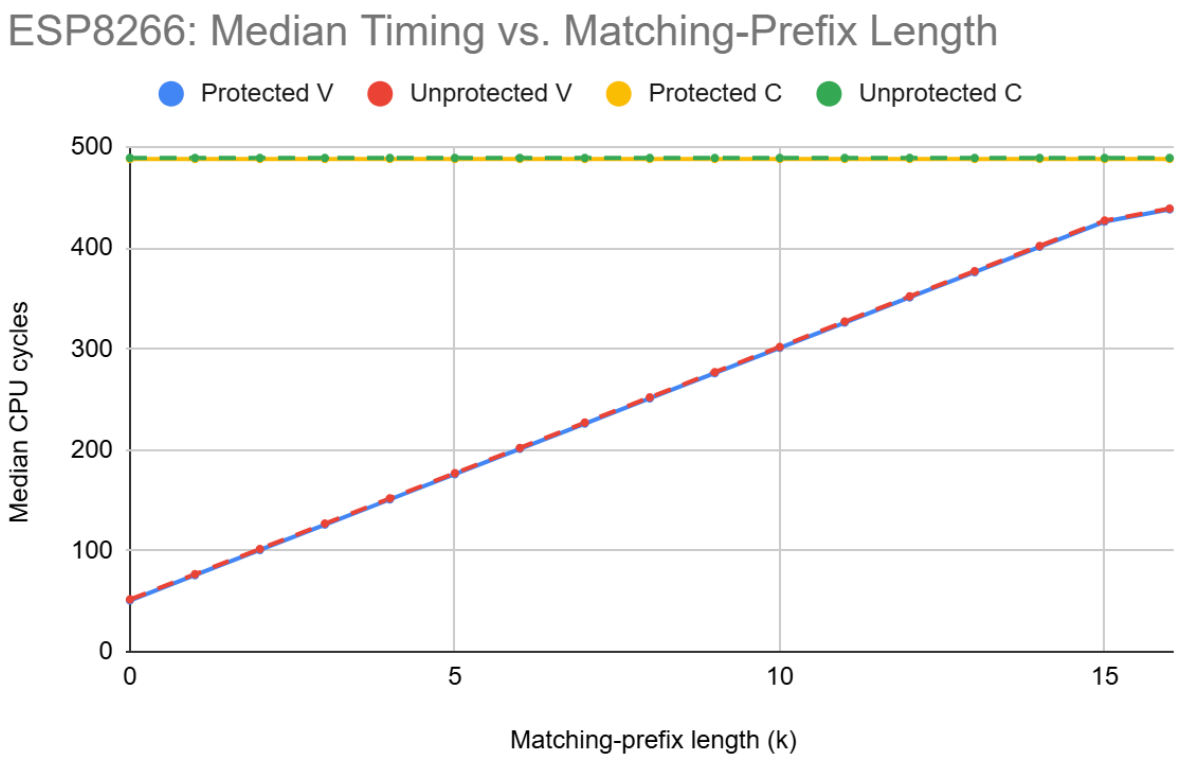


Figure 4. Median execution time across matching-prefix lengths on ESP8266 (300 trials per point). In the legend, V denotes early-exit and C denotes fixed-work; protected denotes interrupts masked with compiler memory barriers, and unprotected denotes interrupts enabled without these barriers. The paired traces nearly overlap at this scale.

The early-exit trial distributions showed exactly one recorded cycle count across all 300 readings in 17/17 ESP32 protected groups, 16/17 ESP32 unprotected groups, 16/17 ESP8266 protected groups, and 14/17 ESP8266 unprotected groups. Within each configuration, the 17 group medians for k = 0–16 were all distinct. Across these recorded on-device sessions, 20,395 of 20,400 early-exit readings (99.975%) equalled their own group median. Accordingly, in the ESP32 protected recording, a single internal CPU-cycle reading identified the tested matching-prefix length. This is a result for that particular on-device run, not evidence of reliable classification across sessions or secret recovery through an external interface.

For the fixed-work implementation, median execution time did not change across the tested matching-prefix lengths. On the ESP32, the median remained at 858 cycles in the protected configuration and 855 cycles in the unprotected configuration. The corresponding values on the ESP8266 were 488 and 489 cycles. These results show that, under the tested configurations, the fixed-work implementation did not exhibit the stepwise median timing increase observed in the early-exit implementation.

4.2 Cross-Device Comparison

The timing results were compared across ESP32 and ESP8266 to examine how the two microcontrollers differed in their execution characteristics. Although both devices showed a linear increase in median execution time for the early-exit implementation, the number of additional CPU cycles per matching byte was different.

As shown in Table 1, the ESP32 exhibited an increase of 40 cycles per additional matching byte under both measurement configurations, compared with 25 cycles on the ESP8266. However, the two microcontrollers operated at different CPU frequencies. Using the reported frequencies of 240 MHz for ESP32 and 80 MHz for ESP8266, these slopes corresponded to approximately 0.167 µs and 0.313 µs per matching byte, respectively.

Table 1. Comparison of timing characteristics across the four experimental configurations.

Configuration	Early-exit slope (cycles/byte)	Slope (µs/byte)	Fixed-work median (cycles)
ESP32 protected	40	0.167	858
ESP32 unprotected	40	0.167	855
ESP8266 protected	25	0.313	488
ESP8266 unprotected	25	0.313	489

The ESP8266 therefore showed a greater timing increase per matching byte when the results were expressed in absolute time, despite having a smaller cycle-count slope. A similar distinction was observed for the fixed-work implementation. Its median execution time was approximately 3.56–3.58 µs on ESP32 and 6.10–6.11 µs on ESP8266.

The measured differences describe the behavior of the comparison routines under the tested hardware and compilation configurations.

4.3 Timing Variability and Unusual Measurements

Across all four datasets, 30 of 40,800 observations differed from the median of their corresponding 300-trial group; all 30 exceeded their group median and no observations fell below it (five from early-exit and 25 from fixed-work comparisons; Table 2). The ESP32 protected dataset had none. ESP32 unprotected had 22 elevated observations, 21 during fixed-work comparison. The ESP8266 protected and unprotected datasets had two and six elevated observations, respectively.

Table 2. Recorded measurements exceeding the median of their corresponding comparison-mode and matching-prefix group (no observations fell below the group median).

Configuration	Early-exit	Fixed-work	Total
ESP32 protected	0	0	0
ESP32 unprotected	1	21	22
ESP8266 protected	1	1	2
ESP8266 unprotected	3	3	6
Total	5	25	30

Some of the largest measurements occurred at the beginning of a collection sequence. For example, in the ESP32 unprotected early-exit dataset at k = 0, the first measurement was 2047 cycles, whereas the remaining 299 measurements were 64 cycles. As a result, the group median remained at 64 cycles, while the mean increased to approximately 70.61 cycles.

Elevated first-trial measurements were also observed at the beginning of both comparison modes in the ESP8266 protected and unprotected datasets. Additional higher readings appeared later in the ESP32 unprotected fixed-work measurements. These observations show that occasional timing increases were present even when the typical execution time remained unchanged.

All measurements were retained in the analysis. The elevated readings affected means and standard deviations but did not change the median trends in Figures 3 and 4. Of the 21 elevated ESP32 unprotected fixed-work readings, 20 occurred at k = 8–16, the later portion of the sequential collection run. Thus this clustering cannot be attributed to matching-prefix length independently of collection time. The data do not identify whether interrupts, barriers, transient system activity, or other factors caused the excursions.

For the fixed-work routine (5,100 observations per configuration), all ESP32 protected measurements were 858 cycles (mean 858.00, SD 0.00). The ESP32 unprotected mean was 858.91 cycles (SD 60.93; median 855; maximum 1,985). The corresponding ESP8266 protected and unprotected means were 488.19 (SD 13.22; median 488; maximum 1,432) and 489.29 (SD 14.32; median 489; maximum 1,434) cycles. These overall summaries pool 17 prefix groups; Figure 5 displays the group-level means and standard deviations.

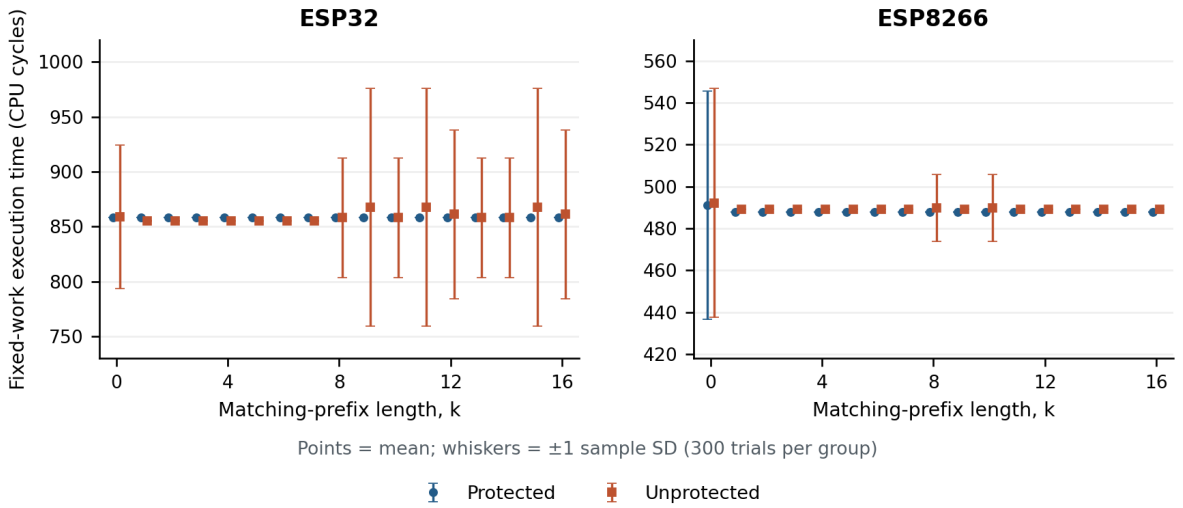


Figure 5. Mean fixed-work execution time ±1 sample SD by matching-prefix length (300 trials per point). Protected = interrupts masked with compiler memory barriers; unprotected = interrupts enabled without those barriers. Whiskers do not show maxima.

4.4 Execution-Time Cost of Fixed-Work Comparison

The fixed-work implementation required more CPU cycles than the early-exit implementation across all tested matching-prefix lengths. However, the difference between the two implementations depended on how many initial bytes matched the secret.

On the ESP32, the protected fixed-work implementation had a median execution time of 858 cycles, compared with 67 cycles for the early-exit implementation at k = 0. This represented a difference of 791 cycles. At k = 15, the difference decreased to 191 cycles. The unprotected configuration showed the same differences, with fixed-work and early-exit medians of 855 and 64 cycles at k = 0, and 855 and 664 cycles at k = 15.

A similar pattern was observed on the ESP8266. At k = 0, the fixed-work implementation required 437 additional cycles relative to the early-exit median in both configurations. This difference decreased to 62 cycles at k = 15. For the complete-match condition k = 16, the difference was 178 cycles on ESP32 and 50 cycles on ESP8266.

Table 3. Difference between fixed-work and early-exit median execution times at selected matching-prefix lengths (CPU cycles).

Configuration	k=0	k=15	k=16
ESP32 protected	791	191	178
ESP32 unprotected	791	191	178
ESP8266 protected	437	62	50
ESP8266 unprotected	437	62	50

The largest difference was observed when the candidate failed at the first byte, while the difference became smaller as the matching-prefix length increased. These values represent differences between group medians rather than paired measurements. They describe the execution-time cost of the fixed-work implementation relative to the early-exit implementation under the hardware and compilation settings used in this experiment.

5. Discussion

The results show a clear relationship between matching-prefix length and execution time in the early-exit comparison. On both microcontrollers, the median number of CPU cycles increased as more initial bytes matched the secret. This behaviour is consistent with the function's control flow, since a later mismatch requires additional iterations of the comparison loop. The linear relationship observed for matching-prefix lengths from 0 to 15 indicates that the position of the first mismatch influenced execution time under the tested conditions. Such input-dependent timing is relevant to the broader timing side-channel problem described in earlier cryptographic research (Kocher, 1996). The within-group check additionally showed that 63 of 68 early-exit prefix-length groups contained 300 identical cycle counts and that all 17 group medians were distinct in every configuration. This supports within-session distinguishability using internal measurements, but not repeatable classification across sessions or through an external interface. The cause of the high within-group repeatability was not independently investigated.

The fixed-work implementation showed a different pattern. Its median execution time remained unchanged across matching-prefix lengths on both ESP32 and ESP8266. Unlike the early-exit function, this implementation processed every byte before returning a result, avoiding termination at the first mismatch. Although the median values showed no dependence on matching-prefix length, occasional higher measurements were still recorded. The findings therefore support the conclusion that the fixed-work implementation avoided the systematic median timing increase observed in the early-exit comparison. Previous research has addressed this distinction through statistical timing-leakage testing and formal verification of constant-time implementations (Almeida et al., 2016; Reparaz et al., 2017).

The comparison also revealed a performance trade-off. The fixed-work implementation required more CPU cycles than the early-exit implementation for every tested matching-prefix length. This difference was greatest when the candidate failed at the first byte and became smaller as the matching prefix increased. The additional execution cost

reflects the different amounts of work performed by the two routines in this experiment. However, these measurements were obtained with unoptimized comparison functions, so the observed costs should not be assumed to apply to optimized implementations.

Another finding was the difference between the two microcontrollers. The early-exit timing increased by 40 cycles per matching byte on ESP32 and 25 cycles on ESP8266. When these values were converted using the reported CPU frequencies, the ESP8266 showed a larger increase in absolute execution time per matching byte. This illustrates why cycle counts alone are insufficient for comparing timing behaviour across devices operating at different clock frequencies. The differences may reflect a combination of processor characteristics, compiler output, and the particular firmware configurations used in the experiment. The present measurements cannot isolate their individual contributions.

The repeated measurements also showed that timing behaviour was not entirely uniform. Most observations matched their corresponding group medians, but a small number of substantially higher readings occurred, particularly in the ESP32 unprotected configuration. These observations suggest that the measurement environment can affect individual timing results even when the median trend remains stable. However, the protected and unprotected firmware differed in their interrupt handling and compiler memory barriers, while the trials were collected in a fixed sequence. Consequently, the higher measurements cannot be attributed to interrupt activity alone.

Finally, the experiment measured execution time directly on the microcontrollers using internal CPU-cycle counters. It did not evaluate whether the observed differences could be detected through external communication or used to recover a secret. Previous research on remote timing attacks has shown that network delays and measurement noise are important considerations when assessing attacks based on externally observed response times (Crosby et al., 2009).

6. Limitations

First, the experiments were conducted on individual ESP32 and ESP8266 development boards under specific hardware and compilation settings. Both comparison functions were compiled using noinline and optimize("O0") attributes, intended to prevent inlining and request unoptimized compilation within an otherwise -Os build. Consequently, the measured execution times may differ from those obtained with other compiler versions, general optimization levels, or alternative hardware implementations. Furthermore, the compiled machine instructions were not independently disassembled or analyzed, so the exact instruction sequences and pipeline interactions were not directly verified.

Second, the protected and unprotected configurations were not identical apart from interrupt handling. The protected firmware temporarily suppressed interrupts during the timed comparison and included compiler memory barriers, whereas the unprotected firmware allowed normal interrupt activity and did not include the same barriers. Although differences in timing variability were observed between these configurations, the experiment cannot determine how much of the variation was caused specifically by interrupts.

Another limitation concerns the order of data collection. The measurements were collected sequentially, beginning with the early-exit implementation and continuing with the fixed-work implementation. Within each mode, the matching-prefix length increased from 0 to 16. Because the trial order was not randomized, temporary changes in device behaviour or measurement conditions may have coincided with particular groups of trials. This is especially relevant when interpreting the unusually high measurements recorded in some datasets.

The experiment also measured execution time directly on the microcontrollers using their internal CPU-cycle counters. Serial communication and other external delays were excluded from the timed region. Consequently, the recorded differences describe the execution behaviour of the comparison functions but do not establish whether the same differences would be observable through an external interface. No attempt was made to recover the stored secret or evaluate a practical timing attack.

Finally, the investigation was limited to two comparison implementations operating on a fixed 16-byte secret. Although the fixed-work implementation showed stable median execution times across matching-prefix lengths, this does not constitute formal verification of constant-time execution or exclude differences in the tails of the timing distributions. The findings should therefore be interpreted as an empirical characterization of the tested implementations rather than a general conclusion about all embedded comparison routines. Further work could examine optimized firmware, randomized measurement sequences, additional hardware samples, and externally observed timing behaviour.

7. Conclusion

This study examined the timing behaviour of early-exit and fixed-work secret-comparison implementations on ESP32 and ESP8266 microcontrollers. The early-exit implementation showed a consistent increase in median execution time as the position of the first mismatching byte moved further into the 16-byte input. For matching-prefix lengths from 0 to 15, the median increased by 40 CPU cycles per additional matching byte on ESP32 and 25 cycles on ESP8266, under both protected and unprotected measurement configurations. Across all four early-exit datasets, 63 of 68 matching-prefix groups had no observed within-group variation, and the 17 group medians were distinct within every configuration. For the ESP32 protected recording, a single internal cycle-count measurement identified the matching-prefix group in that session.

In contrast, the fixed-work implementation maintained the same median execution time across all matching-prefix lengths on both devices. This removed the systematic increase observed in the early-exit implementation, although the fixed-work approach required more cycles, particularly when the early-exit comparison could terminate after the first byte. A small number of unusually high measurements were also recorded, showing that individual execution times were not always identical even when the group medians remained stable.

Overall, the findings demonstrate how the structure of a comparison routine can affect its measured execution time on resource-constrained microcontrollers. However, the results are limited to the tested hardware, compilation settings, and internal timing measurements. They do not establish that the fixed-work implementation is universally constant-time or demonstrate a practical attack against either device. Future work could investigate optimized implementations, randomized measurement sequences, additional hardware samples, and timing measurements obtained through external interfaces.

Data and Code Availability

The raw CPU-cycle timing measurements comprising 40,800 observations across four ESP32 and ESP8266 datasets are publicly archived on Zenodo (version 1.0.0; DOI: 10.5281/zenodo.23734527) under a Creative Commons Attribution 4.0 International (CC BY 4.0) licence. The experimental firmware and data-collection scripts are not publicly released.

Declarations

OpenAI ChatGPT was used to enhance the language and clarity of the manuscript and to assist in generating and refining experimental code. The author designed and conducted the study, collected the measurements, reviewed the AI-assisted material and takes responsibility for the accuracy of the code, data, analysis and manuscript.

Funding: The research was conducted without external funding.

Conflicts of Interest: The author declares no competing interests.

References

Almeida, J.B., Barbosa, M., Barthe, G., Dupressoir, F. and Emmi, M. (2016) 'Verifying constant-time implementations', in Proceedings of the 25th USENIX Security Symposium, pp. 53–70. Available at: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/almeida (accessed 22 October 2026).

Barthe, G., Grégoire, B. and Laporte, V. (2018) 'Secure compilation of side-channel countermeasures: the case of cryptographic “constant-time”', in 2018 IEEE 31st Computer Security Foundations Symposium (CSF), IEEE, Oxford, pp. 328–343. https://doi.org/10.1109/CSF.2018.00031.

Barthe, G., Blazy, S., Grégoire, B., Hutin, R., Laporte, V., Pichardie, D. and Trieu, A. (2020) 'Formal verification of a constant-time preserving C compiler', Proceedings of the ACM on Programming Languages, Vol. 4, POPL, pp. 1–30. https://doi.org/10.1145/3371075.

Bhattacharya, S. and Verbauwhede, I. (2021) 'Exploring micro-architectural side-channel leakages through statistical testing', in 2021 Design, Automation & Test in Europe Conference & Exhibition (DATE), IEEE, Grenoble, France, pp. 633–636. https://doi.org/10.23919/DATE51398.2021.9474091.

Blazy, S., Pichardie, D. and Trieu, A. (2019) 'Verifying constant-time implementations by abstract interpretation', Journal of Computer Security, Vol. 27, No. 1, pp. 137–163. https://doi.org/10.3233/JCS-181136.

Bognár, M., Winderix, H., Van Bulck, J. and Piessens, F. (2023) 'MicroProfiler: principled side-channel mitigation through microarchitectural profiling', in 2023 IEEE 8th European Symposium on Security and Privacy (EuroS&P), IEEE, Delft, Netherlands, pp. 651–670. https://doi.org/10.1109/EuroSP57164.2023.00045.

Brumley, D. and Boneh, D. (2005) 'Remote timing attacks are practical', Computer Networks, Vol. 48, No. 5, pp. 701–716. https://doi.org/10.1016/j.comnet.2005.01.010.

Cai, L., Song, F. and Chen, T. (2024) 'Towards efficient verification of constant-time cryptographic implementations', Proceedings of the ACM on Software Engineering, Vol. 1, FSE, pp. 1019–1042. https://doi.org/10.1145/3643772.

Cinal, A. (2026) 'On timing side channels in “constant-time implementations”', Journal of Cryptographic Engineering, Vol. 16, Article 9. https://doi.org/10.1007/s13389-026-00394-y.

Crosby, S.A., Wallach, D.S. and Riedi, R.H. (2009) 'Opportunities and limits of remote timing attacks', ACM Transactions on Information and System Security, Vol. 12, No. 3, Article 17. https://doi.org/10.1145/1455526.1455530.

Dewald, F., Mantel, H. and Weber, A. (2017) 'AVR processors as a platform for language-based security', in Foley, S.N., Gollmann, D. and Snekkenes, E. (eds) Computer Security – ESORICS 2017, Lecture Notes in Computer Science, Vol. 10492, Springer, Cham, pp. 427–445. https://doi.org/10.1007/978-3-319-66402-6_25.

Dunsche, M., Maehren, M., Erinola, N., Merget, R., Bissantz, N., Somorovsky, J. and Schwenk, J. (2024) 'With great power come great side channels: statistical timing side-channel analyses with bounded type-1 errors', in Proceedings of the 33rd USENIX Security Symposium, pp. 6687–6704. Available at: https://www.usenix.org/conference/usenixsecurity24/presentation/dunsche (accessed 27 October 2026).

Ghaniyoun, M., Barber, K., Zhang, Y. and Teodorescu, R. (2025) 'MicroSampler: a framework for microarchitecture-level leakage detection in constant time execution', in 2025 55th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), IEEE, Naples, Italy, pp. 562–575. https://doi.org/10.1109/DSN64029.2025.00059.

Kocher, P.C. (1996) 'Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems', in Koblitz, N. (ed.) Advances in Cryptology – CRYPTO '96, Springer, Berlin and Heidelberg, pp. 104–113. https://doi.org/10.1007/3-540-68697-5_9.

Miao, Y., Kandemir, M.T., Zhang, D., Zhang, Y., Tan, G. and Wu, D. (2023) 'Hardware support for constant-time programming', in Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture, ACM, Toronto, Canada, pp. 856–870. https://doi.org/10.1145/3613424.3623796.

Reparaz, O., Balasch, J. and Verbauwhede, I. (2017) 'Dude, is my code constant time?', in Design, Automation & Test in Europe Conference & Exhibition (DATE), IEEE, Lausanne, Switzerland, pp. 1697–1702. https://doi.org/10.23919/DATE.2017.7927267.

Robyns, P., Di Martino, M., Giese, D., Lamotte, W., Quax, P. and Noubir, G. (2020) 'Practical operation extraction from electromagnetic leakage for side-channel analysis and reverse engineering', in Proceedings of the 13th ACM Conference on Security and Privacy in Wireless and Mobile Networks, ACM, Linz, Austria, pp. 161–172. https://doi.org/10.1145/3395351.3399362.

Schneider, M., Lain, D., Puddu, I., Dutly, N. and Capkun, S. (2025) 'Breaking bad: how compilers break constant-time implementations', in Proceedings of the 20th ACM Asia Conference on Computer and Communications Security, ACM, Hanoi, Vietnam, pp. 1690–1706. https://doi.org/10.1145/3708821.3733909.

Van Cleemput, J., De Sutter, B. and De Bosschere, K. (2020) 'Adaptive compiler strategies for mitigating timing side channel attacks', IEEE Transactions on Dependable and Secure Computing, Vol. 17, No. 1, pp. 35–49. https://doi.org/10.1109/TDSC.2017.2729549.

Van Dijck, R., Bognár, M. and Van Bulck, J. (2025) 'Wait a cycle: eroding cryptographic trust in low-end TEEs via timing side channels', in 2025 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW), IEEE, Venice, Italy, pp. 462–470. https://doi.org/10.1109/EuroSPW67616.2025.00059.

Van Goethem, T., Pöpper, C., Joosen, W. and Vanhoef, M. (2020) 'Timeless timing attacks: exploiting concurrency to leak secrets over remote connections', in Proceedings of the 29th USENIX Security Symposium, pp. 1985–2002. Available at: https://www.usenix.org/conference/usenixsecurity20/presentation/van-goethem (accessed 27 October 2026).

Winderix, H., Mühlberg, J.T. and Piessens, F. (2021) 'Compiler-assisted hardening of embedded software against interrupt latency side-channel attacks', in 2021 IEEE European Symposium on Security and Privacy (EuroS&P), IEEE, Vienna, Austria, pp. 667–682. https://doi.org/10.1109/EuroSP51992.2021.00050.