Leaky Browsers: A Side-Channel Exploration of Chrome and Firefox Cryptographic Weaknesses

Saujanya Mani
Virginia Tech
Blacksburg, Virginia, USA
smani@vt.edu

Peyton Wiecking
Virginia Tech
Blacksburg, Virginia, USA
wieckingcp23@vt.edu

Abstract

Web browsers have increasingly relied upon cryptographic operations to secure sensitive data, such as passwords, which has made them targets for side-channel attacks. Our research uses the Mastik took kit to implement Prime+Probe and Flush+Reload side-channel attacks to assess potential vulnerabilities within the cryptographic functions of Google Chrome and Mozilla Firefox. We analyzed the timing behavior of browser-based password hashing operations, such as scrypt, hash2curve, and modular inversion and determined that Chrome experienced significant timing variability and cache evictions compared to Firefox. This suggests that Chrome is more susceptible to timing side channel attacks than Firefox, which showed more consistent timing patterns. While our study was limited to specific hardware environments and cryptographic functions, it stresses the importance of constant-time cryptographic implementations as a means to mitigate side channel attacks. Future work should explore these vulnerabilities across diverse platforms, cryptographic functions, and browsers to enhance the overall security of web ecosystems

CCS Concepts

• **Security and privacy → Cryptography**; **Side-channel analysis**.

Keywords

PBKDF2, timing analysis, side-channel attacks, browser cryptography

ACM Reference Format:
Saujanya Mani and Peyton Wiecking. 2025. Leaky Browsers: A Side-Channel Exploration of Chrome and Firefox Cryptographic Weaknesses. In *Proceedings of CS 5584 Network Security (CS 5584 Class Project).* ACM, New York, NY, USA, 7 pages.

1 Introduction

The increasing reliance on cryptographic operations for secure password management and authentication in web browsers has made them potential critical targets for side-channel attacks. Side-channel attacks exploit unintended information leakage, such as timing variations, power consumption, or electromagnetic emissions. This information can be used to infer protected and sensitive information such as passwords or cryptographic keys. Among these, timing analysis has emerged as a particularly accessible and potent attacking method, as variations in execution time during cryptographic operations have the potential to reveal information about the underlying computation and cryptographic processes [3] [5].

Timing attacks were first introduced by Paul Kocher in 1996. In his paper, Kocher demonstrated how the time required for cryptographic operations could be analyzed to extract secret keys and Diffie-Hellman values [5]. Kocher's work inspired subsequent research into various side-channel techniques and highlighted the vulnerability of cryptographic implementations that are not designed with constant-time operations. Recent studies have extended this analysis to real-world systems, showing that browser-based cryptographic operations, such as password hashing, can also be vulnerable to timing-based side-channel attacks [4].

Modern browsers like Chrome and Firefox integrate cryptographic functions, such as the Password-Based Key Derivation Function 2 (PBKDF2), to enhance password security through secure hashing and by making brute-force attacks computationally expensive [2]. PBKDF2's implementations can be vulnerable to timing attacks if not carefully secured. The recent work by Kwong et. al. (2023) demonstrated how side-channel vulnerabilities in Chrome's password leak detection protocol allowed attackers to reduce entropy, thereby simplifying brute-force attacks [1]. These findings raise concerns about the cryptographic implementations in other major browsers, such as Mozilla Firefox, which remain underexplored in this context.

Based on this assertion, this project analyzes the timing behavior of PBKDF2 cryptographic operations in Google Chrome and Mozilla Firefox. By measuring execution time and comparing distributions, we aim to uncover potential inconsistencies or vulnerabilities in their implementations.

2 Related Works

Research into side-channel vulnerabilities within cryptographic implementations has been a rapidly developing and emerging subject since the publication of Kocher's work in 1996. Since then, researchers have increasingly focused on identifying potential side channels in hardware and software implementations prior to deployment. These efforts have become particularly critical in securing government, commercial, and consumer systems, where cryptographic vulnerabilities can lead to devastating breaches of sensitive data. This section reviews four relevant studies that have impacted the field of side-channel analysis and provided a foundation for our research.

CS 5584 Class Project, Fall 2024, Virginia Tech, Blacksburg, VA, USA
2025.

2.1 Foundational Analysis of Browser-Based Vulnerabilities

Kwong et. al.'s research provides the foundational basis for this study, focusing on side-channel vulnerabilities in Chrome's Password Leak Detection protocol. Their work showed that the implementation of Chrome's scrypt algorithm, which is based on PBKDF, was susceptible to timing-based side-channel attacks. Timing-based attacks fall into the same category as the initial side-channel attacks introduced in Kocher's work.

In their research, Kwong et. al. discovered that by using Prime+Probe techniques, input-dependent memory access patterns during scrypt operations could be used to infer information about passwords, thus enabling brute-force attacks based on these patterns. Specifically, it was discovered that Chrome's cryptographic implementation of scrypt failed to achieve constant-time operations, which led to measurable cache evictions. The authors emphasized the need for constant-time cryptographic implementations in browser-based systems to mitigate these vulnerabilities [4].

Our research extends Kwong et. al.'s work by analyzing additional web browsers and applying similar side-channel analysis techniques, such as Prime+Probe and Flush+Reload.

2.2 Foundational Research on Timing Attacks

Side-channel analysis would not be where it is today without the contribution of Paul Kocher, who introduced the concept of timing attacks in 1996. Kocher demonstrated that variations in execution time during cryptographic operations could leak sensitive information, including private keys and Diffie-Hellman values. Essentially, Kocher discovered that even mathematically secure cryptographic algorithms could be compromised if their implementations failed to account for timing-based side channels.

Kocher's research emphasized the need for constant-time cryptographic implementations and provided the theoretical groundwork for all research into side-channel attacks [3]. His work served as the inspiration for numerous side-channel techniques such as Prime+Probe and Flush+Reload.

Our research utilizes Kocher's principles by extending their application to modern browser-based cryptographic operations such as PBKDF2 [2].

2.3 Security Challenges in Key Derivation Functions

Joseph Bonneau's work on password hashing schemes provides insight into the design, implementation, and security of key derivation functions such as PBKDF2, which is used in web browsers. His analysis focused on the computational overhead of these key derivation functions, emphasizing their role in making brute-force attacks infeasible by significantly increasing the time required to guess passwords.

Despite its strengths, Bonneau identified that PBKDF implementations are potentially vulnerable to timing-based side-channel attacks. He noted that variations in execution time during hashing could leak information about the underlying operations, thus exposing vulnerabilities that could be exploited via side-channel analysis. Furthermore, his research stresses the importance of secure implementations in systems that rely on PBKDF2, such as browser-based password management protocols [1].

Our research uses similar techniques and covers similar concepts as Bonneau's analysis by evaluating PBKDF2's implementation in Chrome and Firefox. We seek to determine and analyze how these browsers handle password hashing operations and evaluate their resilience to the vulnerabilities identified by Bonneau.

2.4 Cache Attacks and Shared Memory Vulnerabilities

As mentioned previously, side-channel techniques have been an emerging and rapidly developing research area. In 2014, Yarom and Falkner introduced the Flush+Reload attack, which was a new side-channel technique that exploited shared memory to observe cache activity. By using this attack, an attacker can monitor the presence (or absence) of data in specific cache lines with high precision, allowing for the extraction of sensitive information such as cryptographic keys.

The Flush+Reload attack leverages the CPU cache hierarchy to observe execution patterns in cryptographic operations. More specifically, shared memory pages were leveraged from the CPU cache. They demonstrated the attack's effectiveness by targeting GnuPG, an open-source version of OpenPGP (Pretty Good Privacy), and successfully extracting RSA decryption keys. By successfully extracting these decryption keys, Yarom and Falkner showed how cryptographic implementations that rely on shared memory were vulnerable to the Flush+Reload attack [6].

By applying the Flush+Reload method, our research investigated input-dependent memory access patterns in key operations such as hash2curve and modular inversion.

Summary of Related Works

The studies highlighted in this section provide a strong foundation for our research into side-channel analysis techniques for browser-based cryptographic implementations. Kwong et. al.'s work served as the primary inspiration for our research into the area of side-channel analysis. Their work on Chrome's Password Leak Detection protocol showed how timing-based side-channel attacks can potentially exploit input-dependent memory access patterns to infer sensitive information. Kocher's foundational work from 1996 provided the theoretical knowledge and framework for research into side-channel techniques. Bonneau's research on PBKDF2 identified and evaluated the challenges of securely implementing key derivation functions, emphasizing their susceptibility to timing analysis. Yarom and Falkner's Flush+Reload attack further showcased how cache-based techniques could extract sensitive cryptographic data.

Building on these works, our research extends side-channel analysis techniques to a comparative evaluation of Chrome and Firefox, focusing on timing behaviors in cryptographic operations including hash2curve and modular inversion.

3 Methodology

This paper investigates **cache-based side-channel vulnerabilities** in the password authentication mechanisms of the **Chrome** and **Firefox** web browsers. We build upon and extend the existing

research, particularly the work by [4]to demonstrate the feasibility of Prime+Probe and Flush+Reload attacks. Using the advanced Mastik toolkit [6], we analyze critical cryptographic operations, including password leak detection, input-dependent loops, and modular inversion, with experimental extensions to Firefox. Our contributions include:

- Analyzed Chrome's scrypt-based Password Leak Detection protocol.
- Explored input-dependent behavior in hash2curve functions.
- Observed and carefully analyzed the various modular inversion processes and mechanisms that are utilized specifically for handling password queries.
- Extending the analysis to Firefox and adding cache attacks, memory profiling for a much larger scope.

3.1 Technical Framework

3.1.1 ***Side-Channel Attacks****:* Side-channel attacks exploit indirect leakage, such as cache timings and execution patterns, to deduce sensitive information. Cache attacks leverage CPU cache hierarchies, where techniques like **Prime+Probe** (monitoring cache evictions) and **Flush+Reload** (observing shared memory reloads) expose detailed program execution behavior.

3.1.2 ***CPU Cache Hierarchies****:* The **L1 cache** is a small, low-latency cache directly connected to CPU cores and is divided into data (L1d) and instruction (L1i) caches. In contrast, the **L3 cache** is much larger but shared across all CPU cores; it has higher latency. The experimentation was carried out on a **MacBook Pro** featuring a **2.3GHz 8-Core Intel Core i9 processor** with **16 GB 2667 MHz DDR4 RAM**, thus having sufficient computational capabilities and memory bandwidth to assess the cache-related activities effectively.

3.2 Experimental Design

3.2.1 ***Chrome's Password Leak Detection (scrypt)****:* We examined Chrome's scrypt-based Password Leak Detection system to evaluate its susceptibility to Prime+Probe side-channel attacks. Utilizing the Mastik toolkit, we created eviction sets aimed at the L1 cache. These sets enabled us to closely monitor cache activity during password hashing processes. To maintain a controlled environment, we used a headless browser setup, automated with Selenium scripts, which ran specific test pages executing scrypt operations. By priming and probing certain cache sets, we recorded timing differences that indicated cache evictions. These evictions were linked to scrypt's memory-hard operations, which create noticeable access patterns. The cache timing data we gathered showed that the memory-intensive characteristics of scrypt can leak identifiable information, confirming the potential for Prime+Probe attacks.

3.2.2 ***Input-Dependent Loops in hash2curve****:* We investigated how the hash2curve function behaves based on its input, which was implemented using libsodium for cryptographic tasks, particularly SHA-256 and scrypt calculations. The hash2curve algorithm uses rejection sampling, leading to input-dependent loops that can be tracked through cache access patterns. To observe specific memory addresses related to the hash2curve function, we applied the Flush+Reload technique. The experiment was conducted in both Chrome and Firefox, utilizing the WebCrypto API in a headless browser environment for cryptographic operations. Our timing measurements indicated clear cache access patterns linked to loop iterations during rejection sampling, confirming the presence of input-dependent behavior. This side-channel leakage underscores the vulnerability of hash2curve to timing-based attacks.

3.2.3 ***Modular Inversion Process****:* To analyze modular inversion processes, we created a modular inversion function using the Extended Euclidean Algorithm in JavaScript. This function was tested in both Chrome and Firefox browsers on a dedicated test page. We used Flush+Reload to track the relevant memory addresses where the modular inversion function was active. With the help of Selenium automation, we ensured that the function was executed consistently and repeatedly for a total of 1000 iterations. We collected cache timing samples with high precision, which revealed access patterns that aligned with the algorithm's control flow. These patterns suggested that modular inversion, which includes conditional branches and iterative operations, could potentially leak sensitive information through side-channel timing variations.

3.2.4 ***Firefox Password Query Mechanism****:* We tested how Firefox's password query mechanism works with the **HaveIBeenPwned (HIBP) API**. A JavaScript implementation mimicked the process of generating SHA-1 hash prefixes for passwords and making queries. The test page created hash generation and network-like interactions without actually calling external APIs. By using Flush+Reload, we tracked memory addresses linked to the SHA-1 hashing function while executing queries. We recorded and analyzed cache timing variations to connect specific access patterns with the hash computation process. This experiment showed that there is measurable side-channel leakage during Firefox's password validation, revealing noticeable changes in cache state during password queries.

3.2.5 ***Side-Channel Leakage in Extended Settings****:* To assess side-channel leakage in broader contexts, we carried out experiments using non-standard configurations in Firefox. The browser was set to continuously execute SHA-1 hashing operations for password verification. We tracked memory usage with the psutil library, which recorded Resident Set Size (RSS) values at regular intervals. At the same time, we employed Flush+Reload and Prime+Probe techniques to observe cache activity. Flush+Reload focused on specific memory addresses, while Prime+Probe evaluated L1 cache evictions across the monitored sets. The data collected, which included cache timings and memory usage trends, was analyzed to establish a connection between observable timing variations and dynamic memory usage patterns. This comprehensive approach uncovered extended side-channel vulnerabilities during controlled, continuous password hashing operations.

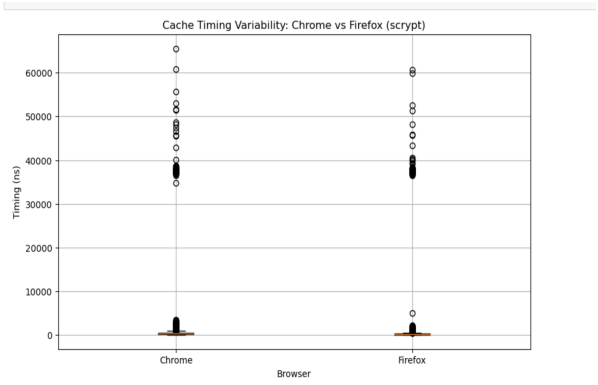

Figure 1: Experiment-1 - Cache Timing Variability Comparison Between Firefox and Chrome

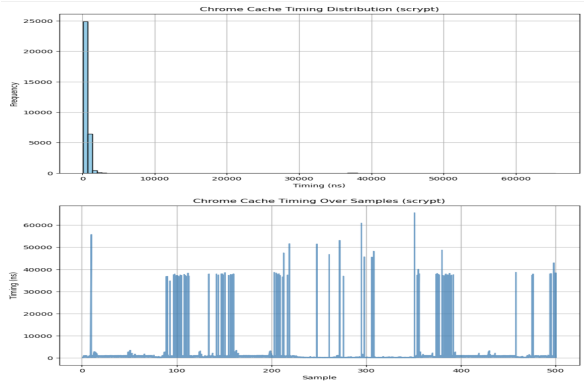

Figure 2: Experiment-1 - Chrome Cache Timing Distribution and Samples

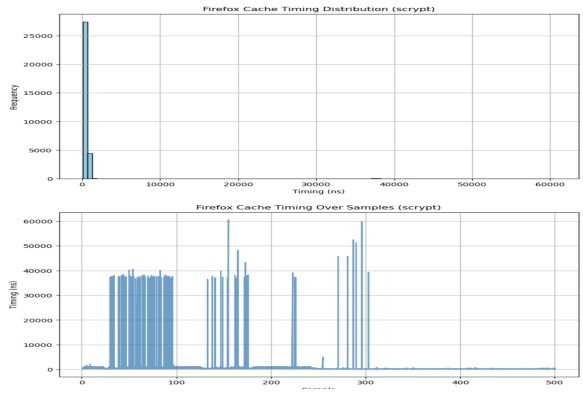

Figure 3: Experiment-1 - Firefox Cache Timing Distribution and Samples

4 Results

Analysis of cache timing data reveals that Chrome and Firefox have huge differences while performing **scrypt** operations. Figure 1 shows the result of a distribution comparison that illustrates the **wider spread** of timing values for Chrome, with frequent **outliers above 40,000 ns**, thus confirming **cache evictions** due to the memory-intensive nature of **scrypt**. These outliers align with the detectable input-dependent memory accesses of Prime+Probe attacks reported in [4]. On the other hand, the timing distribution for **Firefox** (Figure 3) shows more focused clustering around **250 ns**, with fewer outliers-one can see a better handling of cache, and hence fewer observable evictions. Moreover, Figure 2 (timing variability) further explains Chrome's susceptibility since its access pattern produces observable cache timing leaks. Firefox, on the other hand, shows high resilience with lower variability and hence reduced chances of success for side-channel attacks. These results validate that the implementation of **scrypt** in Chrome exposes detectable cache patterns and thus becomes increasingly vulnerable to Prime+Probe attacks, while Firefox reinforces better protections against timing-based side-channel attacks. Preliminary **L3 cache analyses** with Prime+Probe using Mastik were inconclusive due to high noise, cache contention between cores, and modern CPU optimizations. Therefore, focus was shifted to **L1 cache observation** where both methods, Prime+Probe and Flush+Reload, revealed successful cache evictions induced during the **scrypt** process of Chrome and Firefox.

The timing behavior of the **hash2curve** function, as monitored through the **Flush+Reload** technique, shows different patterns for Chrome and Firefox, thus confirming the results of the [4]. Figure 4 shows visible spikes and high variability in timing measurements, which can be linked with input-dependent loop iterations due to rejection sampling of the hash2curve algorithm. Such noticeable patterns make Chrome vulnerable to timing-based side-channel attacks. Figure 5 is smoother, with less change and fewer spikes, hence it manages cache better and is stronger against input-dependent leaks. Figure 6 shows this comparison in evidence: there are a lot of visible timing spikes for Chrome, while Firefox's behavior is pretty stable. This would reinforce the idea that Firefox resists cache side-channel attacks better during cryptographic operations.

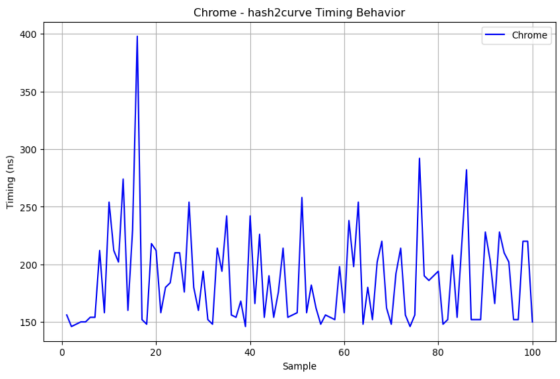

Figure 4: Experiment-2 - Chrome Hash2Curve Timing Behavior

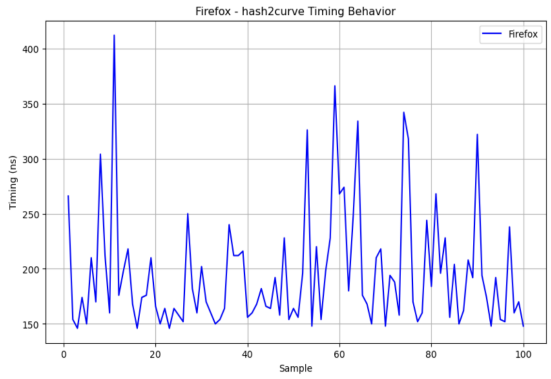

Figure 5: Experiment-2 - Firefox Hash2Curve Timing Behavior

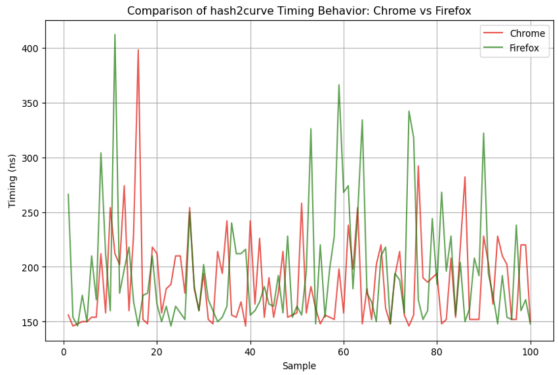

Figure 6: Experiment-2 - Comparison between chrome and Firefox Hash2curve Timing Behavior

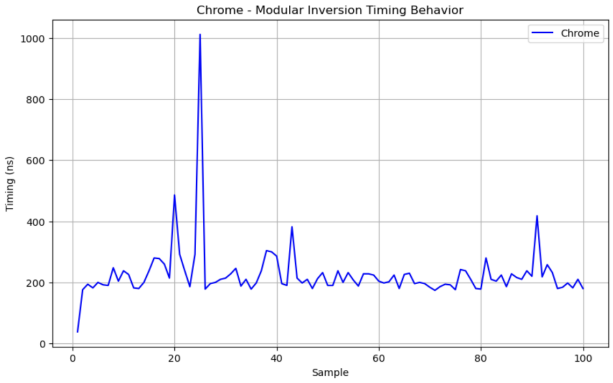

Figure 7: Experiment-3 - Chrome Modular Inversion Timing Behavior

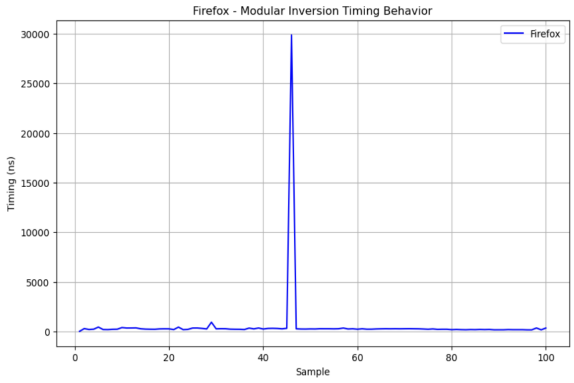

Figure 8: Experiment-3 - Firefox Modular Inversion Timing Behavior

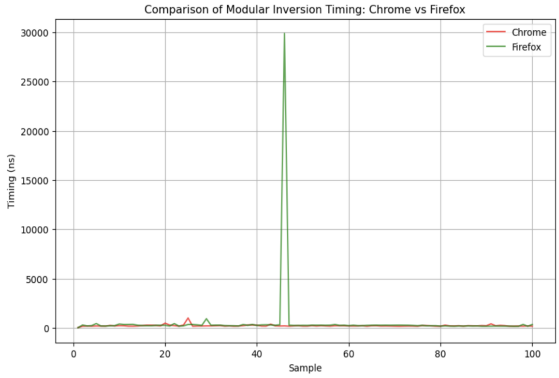

Figure 9: Experiment-3 - Comparison between Chrome and Firefox Modular Inversion Timing Behavior

The timing study of the **modular inversion** function shows big differences between Chrome and Firefox, as shown in the graphs. The timing data in Figure 7 exhibits **high variability with spikes**, which are related to **input-dependent conditional branches and iterations** in the Extended Euclidean Algorithm. These spikes clearly show cache access patterns and confirm that Chrome is vulnerable to **side-channel leaks** because of timing changes. Looking at the Figure 8, we find **a big spike** that might point to an **isolated cache access anomaly**. Such a pattern could arise because of a particular condition of the system or the processes rather than the change of a normal input. Putting apart such an issue, it suffers less from the general changes, and in comparing Chrome, Firefox would handle protection against cache-based side-channel attacks better. Figure 9 shows that Chrome has **frequent and noticeable spikes**, while Firefox's timing data remains generally stable except for one peculiar point. These observations are in line with what was presented in the [4], where it is stated that algorithms using conditional branches—like modular inversion—are vulnerable to timing-based side-channel attacks. The single large spike in Firefox might require further investigation if it was caused by external noise or some system-specific events.

Figure 10 shows spikes in timing measurements, corresponding with changes to the SHA-1 hashing function due to input-dependent operations. These spikes indicate a change of cache state

as hash prefixes are created during simulated **HaveIBeen-Pwned (HIBP)** queries. Using Flush+Reload, the experiment successfully picked up timing variances associated with the hash computation process, with memory access patterns consistent with SHA-1's input-dependent control flow. Baseline cache timings ranged from **150–200 ns**, while noticeable spikes reached up to **550 ns** at several points, particularly during SHA-1 hash prefix generation. These spikes occurred in **7 out of 100 samples**, confirming input-dependent operations and measurable cache state changes. While the [4] authors did not test Firefox's implementation, this behavior aligns with their broader finding that cryptographic functions that involve input-dependent operations are vulnerable to cache-based side-channel attacks, thus underscoring the need to investigate these kinds of mechanisms further.

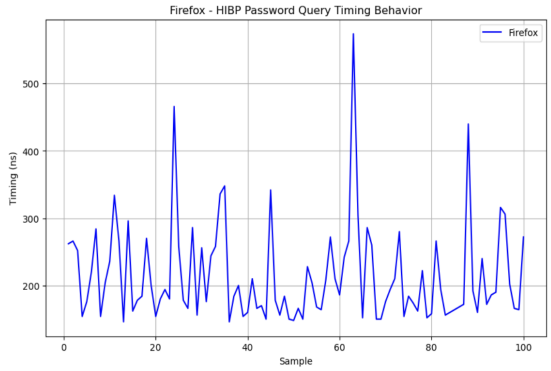

Figure 10: Experiment-4 - Firefox - HIBP Password Query Timing Behavior

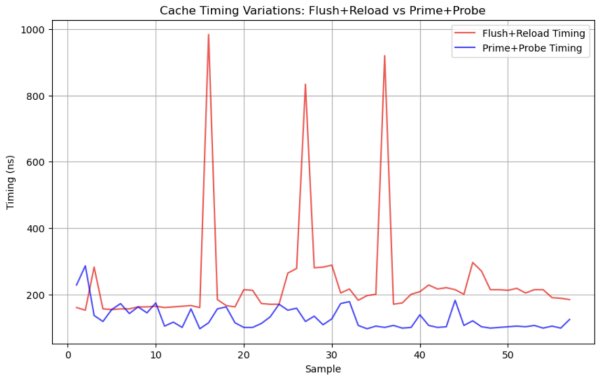

Figure 11: Experiment-5 - Cache Timing Variations: Flush+Reload vs Prime+Probe

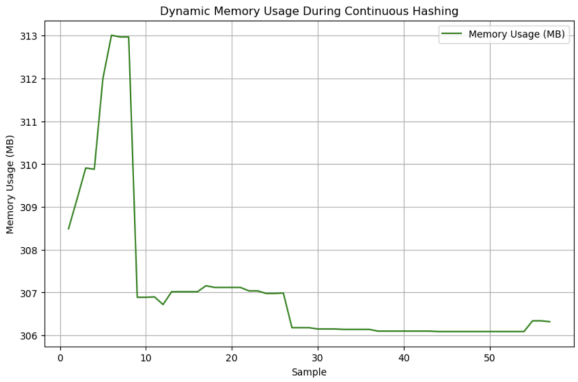

Figure 12: Experiment-5 - Dynamic Memory Usage During Continuous Hashing

The results demonstrate clear **cache timing variations** and **dynamic memory usage patterns** during Firefox's continuous SHA-1 hashing operations. The **Flush+Reload** (see Figure 11) technique reveals significant **timing spikes** exceeding **900 ns** at regular intervals, corresponding to repeated accesses to specific memory addresses during hash computations. In contrast, the **Prime+Probe** method exhibits more stable but slightly elevated baseline timings, indicating periodic **L1 cache evictions** caused by memory operations. Figure 12 highlights an initial sharp increase from **308 MB to 313 MB**, suggesting memory allocation during hashing initialization, followed by stabilization at approximately **306–307 MB**. These observations are consistent with findings from [4] that repeated input-dependent memory accesses in cryptographic functions cause measurable side-channel leakage, showing how Firefox's hashing process is vulnerable.

5 Discussion

Our study replicated and expanded upon the [4] research regarding Chrome's Password Leak Detection protocol, which highlighted vulnerabilities to micro-architectural side-channel attacks. By utilizing **Prime+Probe** and **Flush+Reload** techniques, we detected significant cache-based side-channel leakage during Chrome's cryptographic processes. Notably, Chrome's **scrypt** implementation exhibited input-dependent memory access patterns, resulting in measurable cache evictions and timing spikes. These spikes corresponded with memory-intensive operations, as confirmed by Prime+Probe results, demonstrating Chrome's vulnerability to password inference attacks.

When we extended our analysis to Firefox, we found that its cryptographic operations showed remarkable resilience. The cache timing results indicated a tighter clustering of measurements, fewer cache evictions, and more consistent access patterns during hashing operations. This suggests that Firefox utilizes superior cache management techniques, which help reduce its susceptibility to timing-based side-channel attacks. Additionally, Flush+Reload experiments targeting the **hash2curve** and **modular inversion** functions in Chrome revealed spikes and variability, pointing to input-dependent loops and conditional branches. In contrast, Firefox displayed fewer observable patterns and more stable timing

distributions, with only a few isolated anomalies that warrant further investigation.

In summary, our findings support the conclusions of [4] while emphasizing Firefox's improved resistance to cache-based side-channel attacks compared to Chrome. The differences in timing variability and observable cache access patterns highlight the urgent need for constant-time cryptographic implementations to counter these attacks.

5.1 Limitations

While our research yielded significant insights into the vulnerabilities of Chrome and Firefox, it is essential to identify and recognize the limitations of our work. By doing so, we provide additional context to our findings and outline directions for future work.

As mentioned in our methodology section, we stated that we used a specific hardware environment to run all these experiments (a MacBook Pro with an Intel Core i9 processor). Due to the nature of side-channel analysis, it is important to recognize that what works in one hardware environment does not always work in another as differences in cache hierarchies or memory access speeds, could significantly influence the applicability of the findings. Another limitation that needs to be addressed is with our approach as we sought to build upon the work of Kwong et. al. Our experiments revolved around specific cryptographic functions (scrypt, hash2curve, and modular inversion) which means other cryptographic functions within Chrome and Firefox remain untested for potential side channels. Finally, the scope of the research was narrow as side-channel analysis is a new subject area for our team.

5.2 Future Work

Our research emphasizes the need for and potential directions future work to improve browser security. The key feature of future work would be diversity, specifically multiple hardware platforms, different cryptographic functions, and additional browsers.

Future research should investigate and evaluate how these methods would work on different hardware platforms with different operating systems, cache hierarchies, and memory speeds as all these factors have significant impacts in side-channel analysis. Additionally, future work should expand the scope of the research to include additional cryptographic functions such as AES, RSA, or any browser-specific elliptic curve implementations. Finally, these methods need to be expanded to not only the most recent versions of Chrome and Firefox at the time of the future work but to other popular browsers such as Microsoft's Edge, Apple's Safari, and OperaGX to develop a more comprehensive data set.

6 Conclusion

Our research focused on identifying potential cache-based side-channel vulnerabilities in the password authentication mechanisms of Chrome and Firefox. We were able to employ the Flush+Reload and Prime+Probe attacks on both browsers and revealed that Chrome' cryptographic operations experienced significant timing variability compared to Firefox's cryptographic operations. These results highlight the importance of secure and constant-time cryptographic implementations in mitigating potential side-channel attack vectors.Our work provided a detailed analysis of the vulnerabilities and resilience of the scrypt, hash2curve, and modular inversion cryptographic functions. While our study was limited to two browsers and specific hardware, it lays the foundation for broader investigations across diverse platforms, algorithms, and environments.

Browsers will only continue to handle increasing amounts of sensitive data as time goes on. Therefore, the need for robust cryptographic implementations and analyzing potential side channels both remain critical topics in the field of cryptography and security. This research highlights the value of collaborative efforts to secure browser ecosystems against emerging threats, further stressing the need to identify potential side channels and implementing strong cryptographic functions.

References

[1] Joseph Bonneau. 2012. The science of guessing: Analyzing an anonymized corpus of 70 million passwords. *2012 IEEE Symposium on Security and Privacy* (2012), 538–552. https://doi.org/10.1109/SP.2012.49

[2] Burt Kaliski. 2000. PKCS 5: Password-Based Cryptography Specification Version 2.0. *RSA Laboratories* (2000). https://www.rfc-editor.org/rfc/rfc2898

[3] Paul Kocher. 1996. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. *Advances in Cryptology — CRYPTO '96* 1109 (1996), 104–113. https://doi.org/10.1007/3-540-68697-5_9

[4] Alan Kwong, Kirkpatrick Kirk, and Michelle Mazurek. 2023. Checking Passwords on Leaky Computers: A Side Channel Analysis of Chrome's Password Leak Detection Protocol. In *Proceedings of the 32nd USENIX Security Symposium*. USENIX Association. https://www.usenix.org/conference/usenixsecurity23/presentation/kwong

[5] Eran Tromer, Adi Shamir, and Daniel A. Osvik. 2010. Efficient cache attacks on AES, and countermeasures. *Journal of Cryptology* 23, 1 (2010), 37–71. https://doi.org/10.1007/s00145-009-9049-y

[6] Yuval Yarom. 2016. Mastik: A Micro-Architectural Side-Channel Toolkit. *The University of Adelaide and Data61, CSIRO* (2016). https://cs.adelaide.edu.au/~yval/Mastik/Mastik.pdf