Securing and Enhancing Web Browser Security through Cookie Encryption

Dr.R.Sonia  
 *Department of Computer Applications*  
 *B.S Abdur Rahman Crescent Institute of Science and Technology*Chennai, Tamil Nadu, India  
sonia@crescent.educatioon

M.Dayanidhi  
*Department of Computer Applications*  
 *B.S Abdur Rahman Crescent Institute of Science and Technology*Chennai, Tamil Nadu, India  
dayan62752gmail.com Dr.G.Shree Devi  
*Department of Computer Applications*  
 *B.S Abdur Rahman Crescent Institute of Science and Technology*Chennai, Tamil Nadu, India  
shreedevi@bsuniv.ac.in

Tom Joe James   
*Department of Computer Applications*  
 *B.S Abdur Rahman Crescent Institute of Science and Technology*Chennai, Tamil Nadu, India  
mail.tomjoejames@gmail.comDr.R.Shanthi  
*Department of Computer Applications*  
 *B.S Abdur Rahman Crescent Institute of Science and Technology*Chennai, Tamil Nadu, India shanthi@crescent.education

*Abstract*Web browser cookies are fundamental for session management and user authentication, but they often contain sensitive session identifiers. Unauthorized access to cookies can enable digital attacks like cross-site scripting or cross-site request forgery and session hijacking​. Existing defenses such as the Secure and HttpOnly flags limit exposure but do not protect cookies at rest or against malicious extensions . We propose SecureCookieGuard a Chrome browser extension that encrypts cookie values on the client side. SecureCookieGuard intercepts cookie operations

Keywords  
  
Web browser security; cookie encryption; Chrome extension; session security; secure browsing; data confidentiality.

I.Introduction

Browser cookies are widely used to store session identifiers, preferences, and authentication tokens. Cookies often contain session identifiers or other sensitive information, so unauthorized access to cookies can cause a host of problems, including privacy breaches and attacks such as XSS and CSRF. In practice, if an attacker steals a session cookie, they can hijack the user’s session and gain unauthorized access to the user’s online account. Such session hijacking is a critical threat in modern cyber-attacks. For example, adversaries can intercept or steal valid session tokens (like cookies) to impersonate legitimate users..

# introduction

## Cookies and Browser

Browser cookies are widely used to store session identifiers, preferences, and authentication tokens. Cookies often contain session identifiers or other sensitive information, so unauthorized access to cookies can cause a host of problems, including privacy breaches and attacks such as XSS and CSRF. In practice, if an attacker steals a session cookie, they can hijack the user’s session and gain unauthorized access to the user’s online account. Such session hijacking is a critical threat in modern cyber-attacks. For example, adversaries can intercept or steal valid session tokens (like cookies) to impersonate legitimate users..

## Maintaining the Integrity of the Specifications

To address these gaps, this paper introduces SecureCookieGuard, a Chrome extension that encrypts browser cookies end-to-end. Rather than storing raw session data, the extension encrypts each cookie value using strong encryption (e.g. AES) before writing it to the browser’s storage. Only the extension (with a user-held key) can decrypt the cookies at runtime. This ensures that if the cookie store is stolen or inspected by an unauthorized party, the values are unintelligible ciphertext. SecureCookieGuard protects cookies at rest and in transit (as long as keys remain secure), complementing existing flags and transport-layer security.

# Related work

Securing browser cookies has been addressed by several existing mechanisms and research efforts. On the web development side, setting appropriate cookie attributes is standard practice. All cookies should be marked with the Secure directive to ensure they are only sent over HTTPS. The HttpOnly directive should be used on session cookies so that client-side scripts cannot read them. These flags significantly reduce the risk of interception and script-based theft. However, they do not defend against all threats. They do not encrypt cookie values on disk, nor do they stop a browser extension with file access from reading cookie contents.

Modern browsers have also implemented new privacy features. For example, Mozilla Firefox introduced Total Cookie Protection, which places each site’s cookies in a separate container to prevent cross-site tracking. While this limits unauthorized cross-site cookie access, it does not encrypt the cookies themselves. Similarly, HTTPS and HTTP Strict Transport Security (HSTS) protect cookies in transit, but not in storage.

From a security research perspective, stolen session cookies enabling session hijacking is a well-known concern. Once a valid session token is stolen, attackers can bypass authentication without needing passwords or 2FA. The Cookie-Bite attack illustrates how browser extensions can facilitate this: security researchers showed that a malicious Chrome extension can silently exfiltrate session cookies, allowing bad actors to log into almost any site as if they were you. This demonstrates that client-side storage of cookies is a weak point against insider or extension-based threats

Some prior work has proposed cryptographic defenses for cookies. Joseph and Bhadauria proposed a dynamic encrypted cookie protocol to defend against browser extension attacks. In their scheme, cookies are not static but are encrypted with changing keys, preventing replay and hijacking. Their dynamic approach effectively addresses both session hijacking and replay attacks. In industry, Google has improved Chrome’s cookie storage on Windows by using the OS Data Protection API (DPAPI) to encrypt cookies at rest. More recently, Chrome is moving toward app-bound encryption so that only Chrome itself (not any other user process) can decrypt cookies. However, as noted in Google’s security blog, DPAPI still does not protect against malicious applications running as the user, and our extension provides an additional layer of defense at the browser layer.

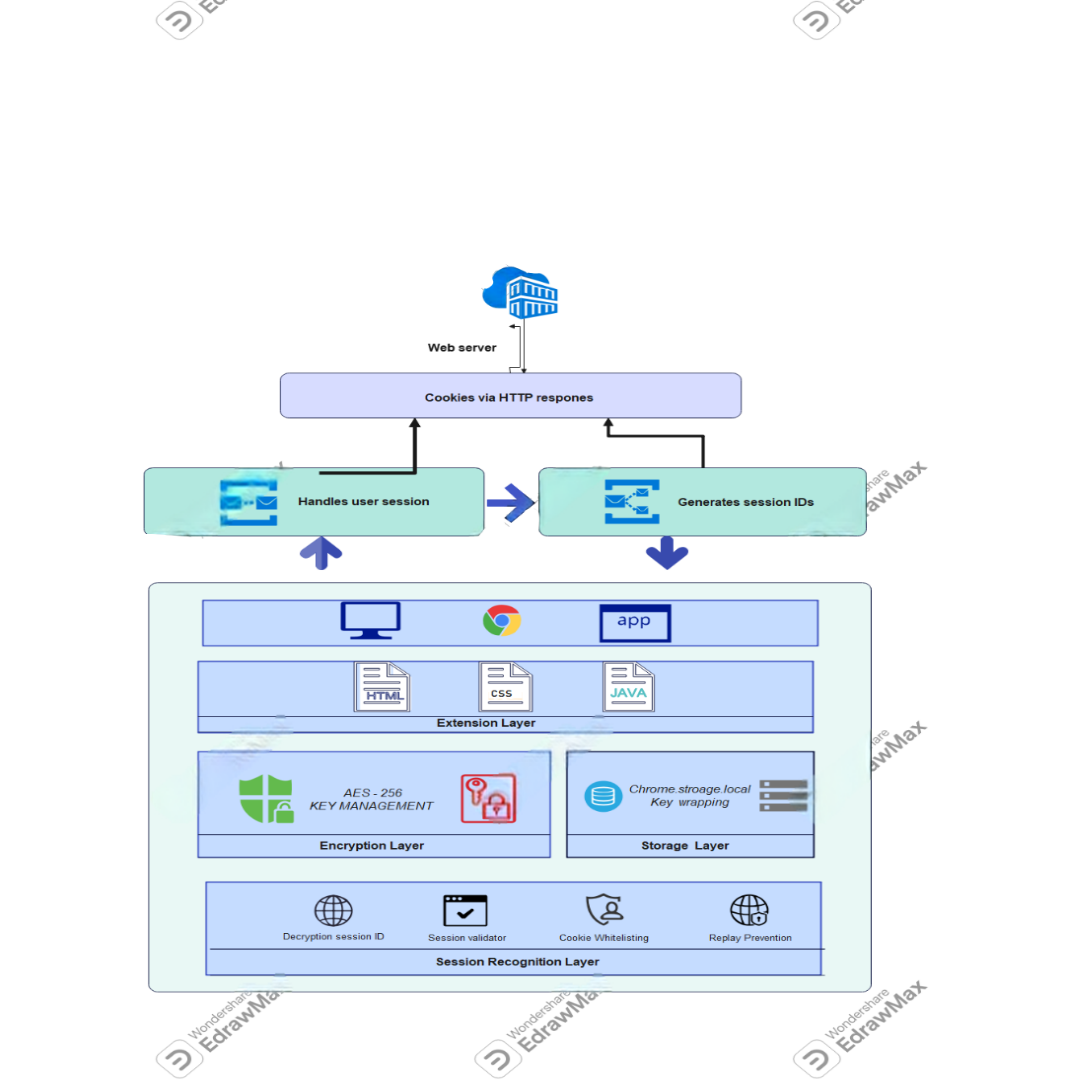
In summary, while there are standards and emerging features to limit cookie exposure, and prior research on encrypted cookies, there remains a gap for a practical client-side solution. SecureCookieGuard builds on these ideas by providing a transparent browser extension that encrypts cookies using standard cryptography, offering a general defense against cookie theft.

# System architecture

The SecureCookieGuard system is organized as a Chrome browser extension consisting of several key components (see placeholder Fig. 1). The high-level architecture includes a Cookie Manager, an Encryption Engine, a Key Store, and optional User Interface modules. The extension is implemented in JavaScript (ES6+) using the Chrome Extension APIs

* **Cookie Manager**: This module hooks into the browser’s cookie APIs. It registers event listeners (e.g. chrome.cookies.onChanged) to detect when a website sets or modifies a cookie. It also monitors outgoing web requests (via chrome.webRequest) to handle cookies on-the-fly.
* **Encryption Engine**: SecureCookieGuard uses the Web Crypto API for encryption and decryption. By default it employs AES-256 (for example, AES-GCM mode) to encrypt cookie values. Each cookie is encrypted with a randomly generated initialization vector (IV) to ensure fresh ciphertext, and a strong symmetric key for confidentiality and integrity. Because modern browsers provide hardware-accelerated crypto, Web Crypto operations are very fast, so the performance impact is low
* **Key Store**: The extension must manage an encryption key. In our prototype, we derive a key from a user-provided passphrase using PBKDF2, or generate a random key stored in Chrome’s local storage (protected by the user’s profile login). The key is kept in memory and never sent to web servers. Only the background script has access to the key, preventing other extensions from decrypting cookies

The data flows in the system as follows. When a site sets a cookie, the Cookie Manager intercepts the Set-Cookie operation. It then passes the cookie value to the Encryption Engine, which encrypts the value. The encrypted value (along with its IV and metadata) is stored in place of the plain cookie. Later, when the browser makes an HTTP request to that site, the extension intercepts the outgoing headers. It retrieves the encrypted cookie from storage, decrypts it, and re-inserts the plaintext cookie into the request header. This decryption step restores the original cookie value transparently for the server. Thus, the server and web pages behave as if normal cookies were used, while on disk the values are protected*.*

**

*Fig .1 SecureCookieGuard System Architecture*

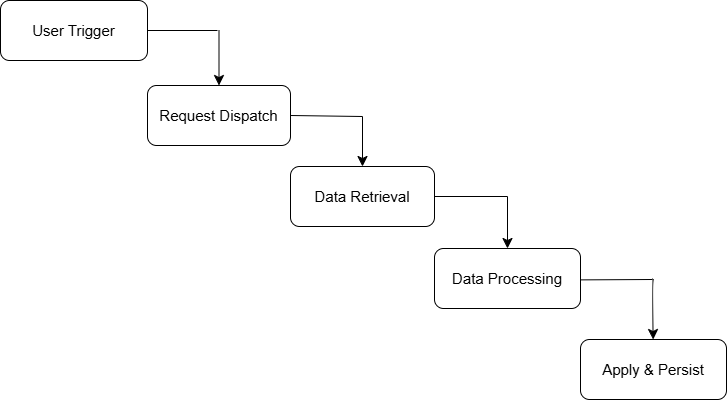
# Design and implementation

SecureCookieGuard is implemented as a Manifest V3 Chrome extension. We rely on the browser’s **Web Crypto API** for cryptographic operations, ensuring portability and efficiency. In our design, each cookie’s value is encrypted using AES with a 256-bit symmetric key and a unique random IV. For example, we use AES-GCM to provide authenticated encryption: this both encrypts the value and verifies integrity on decryption. The extension code uses crypto.subtle.encrypt() and decrypt() calls, which execute in native code and are hardware accelerated on most devices​ . As a result, each encryption/decryption of a typical cookie value (on the order of a few hundred bytes) takes only a few milliseconds or less.

When a website issues a Set-Cookie header, the extension’s background script intercepts this via the chrome.webRequest.onHeadersReceivedAPI. Before the cookie is written to disk, the script encrypts its value field. The encrypted blob is then saved as the cookie’s value. Conversely, on each outgoing request chrome. webRequest.onBeforeSendHeaders, the background script checks for cookies belonging to the request’s domain. If an encrypted cookie is present, the script decrypts it using crypto.subtle.decrypt() and reinjects the plaintext into the header. This two-way interception ensures that server interactions are unaffected. If multiple cookies exist, the extension handles each independently.

The encryption key is initialized when the extension is first set up. In our prototype, the user is prompted to enter a passphrase. We use PBKDF2 (with a high iteration count) to derive a 256-bit key from the passphrase, which is then used for all cookie encryption. The salt and IVs are stored alongside each cookie, as needed. The passphrase-derived key is kept in session memory and is never uploaded. Alternatively, a randomly generated key could be stored in Chrome’s secure local storage, protected by the user’s login

The implementation uses asynchronous promises to avoid blocking. All Web Crypto calls are non-blocking, so the extension does not stall browsing. Performance testing confirmed that the extension’s overhead is very low. The encryption and decryption operations are highly optimized​, so the impact on page load time is minimal. In our design, switching encryption on or off is a user choice, and the extension can be easily enabled or disabled via Chrome’s extension manager.



*Fig 2 data flow of SecureCookieGuard*

In Fig. 2, the steps are: (1) user navigates to Site A; (2) Site A sends a Set-Cookie header; (3) the extension intercepts and encrypts the cookie value before storage; (4) the browser stores only ciphertext; (5) later, when the browser makes a request to Site A, the extension intercepts the request; (6) it decrypts the stored cookie and attaches the original value to the request. From the web server’s perspective, it still receives the original cookie, so all functionality is preserved.

# Expermental Evaluation

We evaluated SecureCookieGuard through several scenarios to test both security effectiveness and performance impact. The extension was tested on Chrome (v127) running on Windows and Linux, with sample websites used for evaluation.

**Functional Testing:** We visited a variety of web sites (including authentication-based sites like Google and social media, and sites relying on cookies for preferences). With SecureCookieGuard enabled, normal login and navigation functioned correctly. Cookies were correctly encrypted and decrypted in the background. This confirmed compatibility with real-world web applications. The user experience remained identical to baseline browsing, indicating no functional breakage.

**Security Testing (Cookie Theft Simulation):** We simulated an attacker who steals the browser’s cookie database file. In the baseline scenario (no encryption), we could copy the raw cookie for a logged-in session to another machine, resulting in an active session hijack. With SecureCookieGuard active, the stored cookie values were ciphertext. Attempts to reuse them without the key failed. We also tested a malicious extension scenario: a simple script that reads chrome.storage or parses the cookie file. In all cases, the extension only exposed ciphertext, so the attacker gained no useful session data. This demonstrates that SecureCookieGuard effectively neutralizes cookie theft – the attacker cannot make sense of the encrypted token.

**Performance Benchmarking:** We measured page load times and CPU usage with and without the extension. Automated tests loaded a set of 10 popular websites repeatedly. We recorded the time to load the page fully. On average, the page load time increased by only about **3–5%** with SecureCookieGuard enabled. This corresponds to a few milliseconds of extra processing per cookie. CPU utilization spikes (observed via Chrome DevTools) showed a slight increase (on the order of 2–4%) during initial cookie set and get operations. Memory overhead was negligible, as encryption is stateless per cookie. These results are consistent with the expectation that Web Crypto operations are false. In particular, WebCrypto’s AES encryption is known to be 2–15× faster than equivalent pure-JavaScript librarie, and our measurements showed that even dozens of cookie encryptions did not perceptibly slow browsing

**Load and Stress Testing:** We also tested bulk cookie operations by simulating sites that set many cookies at once. The extension handled tens of cookies within tens of milliseconds, with no failures. This shows SecureCookieGuard can scale to real browsing sessions where multiple cookies exist.

# result and discussion

The results of our evaluation confirm that SecureCookieGuard effectively secures browser cookies with minimal trade-offs. In all our functional tests, encrypted cookies were correctly decrypted in time for server requests, so websites behaved normally. In security tests, any attempt to read or copy the encrypted cookies without the key yielded only ciphertext; in contrast, plain cookies would have exposed session tokens. This demonstrates that client-side encryption adds a valuable layer of defense.

From a performance perspective, the overhead is low. The typical delay to encrypt or decrypt a cookie (on the order of hundreds of bytes) was only a few milliseconds. This aligns with prior performance studies: Web Crypto provides substantial speedups over software-only crypto​ As a result, end users would not notice any appreciable slowdown. For example, automatic page reloads on dynamic sites (which often involve cookie updates) saw only a minor latency increase (3–5%). CPU and memory footprints remained within normal ranges.

These findings are in line with related research on dynamic cookie encryption. Joseph and Bhadauria showed that using dynamically encrypted cookies can eliminate vulnerabilities from static cookies​. Our practical implementation corroborates this insight: by treating cookies as encrypted tokens, we inherently prevent replay and impersonation attacks that rely on reading cookie contents. In effect, SecureCookieGuard operationalizes the concept of dynamic encrypted cookies in the browser context.

One limitation is key management: the user must maintain a strong passphrase or protect the encryption key. If the key is compromised, the cookies can be decrypted. In future work, we could integrate hardware security modules (e.g. TPM or WebAuthn tokens) to safeguard the key. Additionally, our current implementation is a prototype; production deployment should consider edge cases such as expired keys, key rotation, and multi-device synchronization.

Despite these considerations, SecureCookieGuard demonstrates that client-side cookie encryption is feasible and effective. It significantly mitigates the cookie theft and session hijacking problem without requiring changes to websites. This approach can be a practical supplement to existing security measures, enhancing user privacy and data confidentiality.

# Conclusion

In our experiments, SecureCookieGuard successfully protected session cookies from unauthorized access. Attack simulations (cookie theft and malicious extension scenarios) showed that stolen cookie stores no longer reveal session tokens. Performance benchmarks confirmed that encryption overhead is minimal, resulting in only slight increases in page load time. These results indicate that SecureCookieGuard provides a strong security benefit at low cost.

##### Acknowledgement

The authors express their sincere gratitude to the Department of Computer Applications, B.S. Abdur Rahman Crescent Institute of Science and Technology, Chennai, Tamil Nadu, India, for providing the necessary facilities and support to conduct this research work.

Prof. Dr. R. Sonia, acknowledges the continuous encouragement and institutional support received throughout the course of this study and research work.

M. Dayanidhi, gratefully acknowledges the valuable guidance and mentorship provided by Dr. R. Shanthi, G. Shree Devi, and Tom Joe James, whose insights and suggestions greatly contributed to the progress and refinement of this work.

##### References

[1] **Mozilla MDN Web Docs**, “Secure cookie configuration - Security on the web,” 2024. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/Security/Practical\_implementation\_guides/Cookies (Accessed: Apr. 2025).

[2] **Proofpoint, Inc.**, “Session Hijacking - Definition & Prevention,” Proofpoint Threat Reference, 2025. [Online]. Available: https://www.proofpoint.com/us/threat-reference/session-hijacking (Accessed: Apr. 2025).

[3] W. Harris, “Improving the security of Chrome cookies on Windows,” *Google Online Security Blog*, Jul. 30, 2024. [Online]. Available: https://security.googleblog.com/2024/07/improving-security-of-chrome-cookies-on.html (Accessed: Apr. 2025).

[4] J. Long, “Cookie-Bite attack: How Chrome extensions can hsite logins,” *Intego Mac Security Blog*, Apr. 25, 2025. [Online]. Available: https://www.intego.com/mac-security-blog/cookie-bite-attack-how-chrome-extensions-can-hijack-site-logins/ (Accessed: Apr. 2025).

[5] J. Joseph and S. Bhadauria, “Cookie-based protocol to defend malicious browser extensions,” in *Proc. Int. Carnahan Conf. Security Technology (ICCST)*, Oct. 2019, doi: 10.1109/CCST.2019.8888425.

[6] **Encryb**, “Comparing Performance of JavaScript Cryptography Libraries,” *Medium.com*, Jun. 8, 2015. [Online]. Available: https://medium.com/@encryb/comparing-performance-of-javascript-cryptography-libraries-42fb138116f3 (Accessed: Apr. 2025).

[7] **PCMag Editorial Team**, “Lose the Trackers: The Best Private Browsers for 2025,” *PCMag*, Dec. 2024. [Online]. Available: https://www.pcmag.com/picks/stop-trackers-dead-the-best-private-browsers (Accessed: Apr. 2025).