PUSH for Security: A PUF-Based Protocol to Prevent Session Hijacking

Emiliia Geloczi, Nico Mexis, and Stefan Katzenbeisser

Chair of Computer Engineering, University of Passau, Innstr. 41, Passau, Germany
{emiliia.geloczi, nico.mexis, stefan.katzenbeisser}@uni-passau.de

Abstract. Session hijacking attacks still affect thousands of users of web services every year. In this paper, we propose a novel lightweight hardware-binding protocol that associates a web session with a device using a unique fingerprint derived from an SRAM-based Physical Unclonable Function (PUF). The hardware/software co-design of the proposed protocol ensures continuous verification of the session's legitimacy without introducing additional latency. Furthermore, it enables the timely termination of a hijacked session, thereby mitigating the impact of session hijacking attacks.

Keywords: Session Hijacking · PUF · SRAM PUF · Hardware Binding

1 Introduction

According to Statista.com, approximately 67% of the global population use web services for different purposes [40] and have user accounts that store their sensitive data, such as bank card details. Despite many technical measures for account security [39], users still routinely become victims of attacks that lead to account loss and data compromise [19,23,41]. Moreover, they often unintentionally assist adversaries by opening seemingly legitimate but malicious files or links, thereby infecting their devices with malware. Such malware is often designed to obtain session identifiers (IDs) stored in session cookies. Using these session IDs, adversaries can access accounts that are active during the session without further authentication. This process is called a session hijacking [22].

Although it is unlikely to completely prevent session hijacking attacks, their impact can be mitigated, for example, by encrypting cookies [31]. Recent proposals introduce solutions based on the use of temporary cookies [13], sessions [35], or tokens [37], which require re-authentication and increase the load on the server. Some approaches bind the session to a virtual or physical asset, such as a browser [15] or a device [33], hence requiring adversaries not only to possess an active session ID but also to gain control over the associated asset. However, approaches that bind sessions to devices typically require additional hardware, such as a Trusted Platform Module (TPM) [33], which may introduce performance overhead, restrict the range of compatible devices, and enable vendor-imposed limitations, therefore transferring control over the device away from the user. In

This is a Pre-print: The final version will appear in the proceedings of Computer Security – ESORICS 2025, in Volume 16055 of the Lecture Notes in Computer Science by Springer Nature.

addition, all existing approaches rely on the persistent storage of cryptographic keys, introducing further security risks.

Contribution. In this paper, we propose a novel lightweight protocol that associates a web session with a physical device by utilising an SRAM Physical Unclonable Function (PUF). It serves as a natural fingerprint of a device, can be derived from existing hardware components, exhibits properties of uniqueness and unclonability, and eliminates the need to store cryptographic keys [20]. Our protocol, called **PUSH** (**PU**F against **S**ession **H**ijacking), mitigates session hijacking attacks by preventing an adversary from using a session ID without having access to the associated device. PUSH can be seamlessly integrated into existing client/server architectures, provides continuous verification of the session's legitimacy without introducing additional latency, and enables the timely termination of hijacked sessions. Furthermore, to the best of our knowledge, PUSH is the first PUF-based protocol to address the problem of session hijacking in web communication.

Paper Organisation. Section 2 provides the background information on PUFs. Section 3 describes the proposed PUSH protocol, its proof-of-concept implementation, and the potential challenges. The security analysis of PUSH is presented in Section 4. Section 5 provides an overview of related work, followed by their comparison with PUSH in Section 6. Finally, Section 7 concludes the paper and outlines directions for future research.

2 Background: Physical Unclonable Functions

During the manufacturing process of electronic devices, small differences in the characteristics of their components (e.g., transistors) may occur. As a result, even devices from the same product line may have slight variations in their physical characteristics [27]. While these differences are not significant in terms of functionality, they can form the basis of a unique device fingerprint, also known as a Physical Unclonable Function (PUF). As a function embedded in a physical object, PUF produces a response y upon receiving a challenge x [20]. The pair (x, y) is called the challenge-response pair (CRP). To serve as the fingerprint, PUF should be robust, unclonable, unpredictable and tamper-proof [26].

In our protocol, a start-up Static Random Access Memory (SRAM) PUF [16] is used, which is an integral part of almost all devices, not easily accessible externally, provides high entropy [11,12] and stable [43,45]. SRAM cells are initialised with different bit values due to slight variations in transistor characteristics. This bit pattern is unique to each SRAM module and can be considered as its fingerprint, however, it must be observed during module startup. Thus, the challenge is the set of memory addresses ($C \coloneqq \{a_1, a_2, \ldots, a_n\}$), and the response is the set of corresponding bit values ($R \coloneqq \{b_1, b_2, \ldots, b_n\}$). Together, C and R form CRP associated with SRAM PUF. CRP can contain any required number n of address-bit value pairs (a, b) (or (c, r) pairs: $(a, b) \rightarrow (c, r)$). In the remainder of this paper, when referring to PUF initialisation, we specifically mean the identification of the corresponding CRP.

This is a Pre-print: The final version will appear in the proceedings of Computer Security – ESORICS 2025, in Volume 16055 of the Lecture Notes in Computer Science by Springer Nature.

3 Description of the PUSH Protocol

PUSH is a PUF-based protocol that binds a web session to a physical device (dongle) and continuously verifies the session's legitimacy. In this section, we present the system model in which PUSH operates, describe its operational workflow, detail the proof-of-concept implementation, evaluate its performance, and discuss potential challenges.

3.1 System Model

We consider a setting involving the following four legitimate parties in the PUSH protocol (see Figure 1):

- *Dongle* is a Universal Serial Bus (USB) device to which a session is bound. It contains a PUF SRAM module and a crypto module.
  - *Crypto Module* performs initialisation of PUF, generation of cryptographic keys, and digital signing.
- *User's Device* is a device used to access web services, which hosts the following components:
  - *PUFMAN* (PUF MANager) is a service that communicates with the dongle via a USB port and provides an Application Programming Interface (API) for the browser.
  - *Browser* is a software application that allows a user to send requests to web servers and receive responses.
- *Server* is a system that processes user requests and provides responses.

The dongle, the user's device, PUFMAN, and the server are assumed to be trusted; however, the browser can be exposed to potential malicious extensions during the operation.

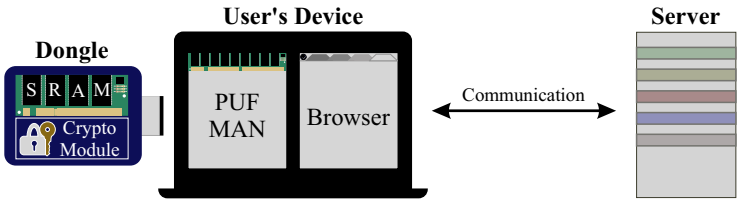


Fig. 1. PUSH Protocol Parties.

PUSH operates at the application layer and runs on top of the Hypertext Transfer Protocol Secure (HTTPS), ensuring secure communication between the browser, the server, and PUFMAN. Moreover, since PUSH is designed to integrate with existing web communication, it is assumed that session cookies are encrypted and stored securely using the default existing mechanisms, such as the Data Protection API (DPAPI) [31], which are not detailed in this paper.

In the protocol description, we use the abstract cryptographic building blocks such as $KeyGen(\cdot)$ and $Fuzzy(\cdot)$ to represent key generation and fuzzy extractor algorithms, respectively.

This is a Pre-print: The final version will appear in the proceedings of Computer Security – ESORICS 2025, in Volume 16055 of the Lecture Notes in Computer Science by Springer Nature.

3.2 Operation Workflow

PUSH consists of three phases. The first phase (I) is dedicated to the initialisation of PUF on the dongle. During the second phase (II), the browser initiates its first communication with the server, performing enrolment. During the third phase (III), communication between the browser and the server takes place. The generalised workflow of the PUSH operation is illustrated in Figure 2, and each phase is described in detail in the following sections.

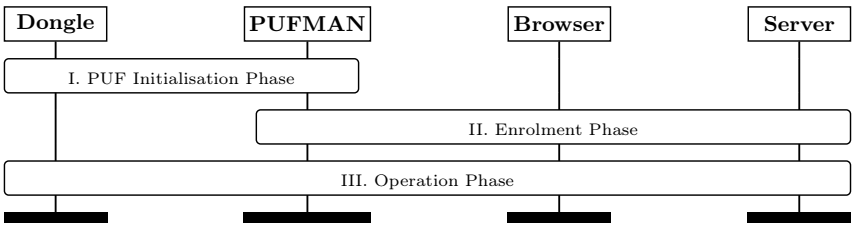


Fig. 2. Operation workflow of the PUSH protocol.

In hardware binding, it is critical that protocol parties can verify the existence of a dongle and confirm that it is not being simulated at the very first contact. For this purpose, our protocol includes a manufacturer-issued cryptographic key pair: a secret key SK_D, embedded in the dongle, and a corresponding public key PK_D, provided to the user. This key pair is used exclusively during the enrolment phase (II) to confirm the dongle's authenticity.

I. PUF Initialisation Phase
In this phase, a unique fingerprint of the dongle is identified and used to generate a pair of cryptographic keys: a secret key SK and a public key PK. This process is performed only once, unless the user explicitly requests re-initialisation. During this phase, PUSH operates according to the following algorithm (see Figure 3):

First, PUFMAN sends a request to the Dongle to start the PUF initialisation phase (1). Inside the Dongle, the Crypto Module triggers the power-cycling of the SRAM module (2,3) and subsequently reads its content, obtaining a set of n memory values $MV_r \coloneqq \{b_i \mid i = 0, \ldots, n\}$, where r is the reading iteration number, and b_i is a bit value in the memory cell at address i (4). Steps 2–4 are repeated k times to collect data for an analysis. Next, the Crypto Module computes the average Hamming Distance and Hamming Weight for each memory cell to identify a set of memory addresses (challenge) $C \coloneqq \{c_i\}$ and their corresponding bit values (response) $R \coloneqq \{r_i\}$, where $i = 0, \ldots, m$, forming $CRP \coloneqq (C, R)$, which serves as a unique fingerprint of the Dongle (5). The response R is then processed using a fuzzy extractor $Fuzzy()$, and the result is used as a seed for a key generation function $KeyGen()$, producing a key pair (SK, PK) (6). Finally, the Crypto Module signs both C and PK using the manufacturer-issued secret key SK_D (7), and sends the challenge C, the public key PK, and their corresponding digital signatures $((C)_{SK_D}, (PK)_{SK_D})$ to PUFMAN (8).

After this phase is completed, the SRAM module is powered off, and the keys SK and PK are deleted. PUFMAN keeps C, PK, $(C)_{SK_D}$, and $(PK)_{SK_D}$.

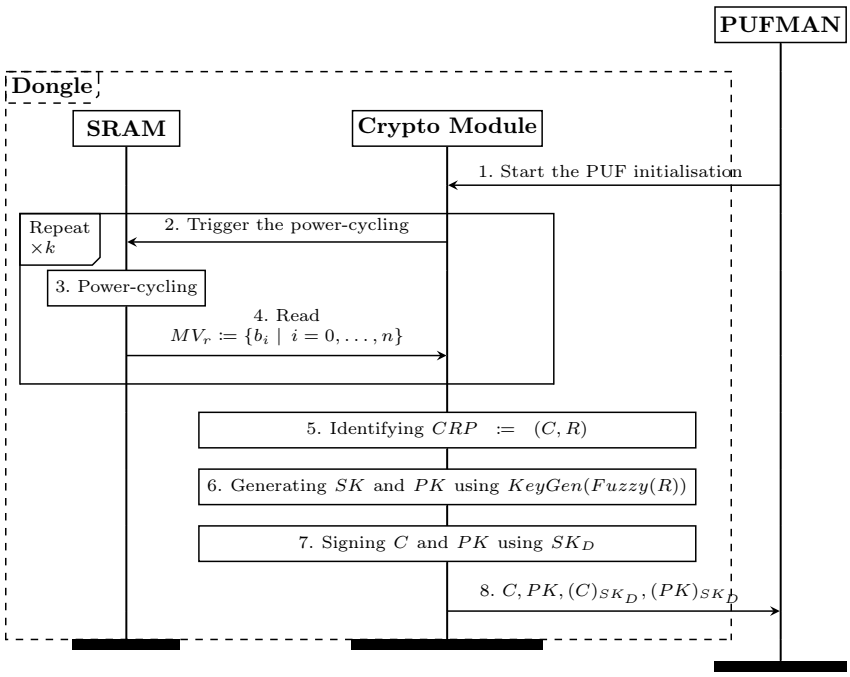


Fig. 3. Workflow of PUSH during the PUF Initialisation Phase (I).

II. Enrolment Phase

During this phase, the browser initiates its first connection with the server to perform enrolment. This process is executed only once unless the user explicitly requests re-enrolment. The phase follows the steps outlined in the algorithm presented in Figure 4:

First, the User provides the Browser with the public key PK_D (1). The Browser then requests from PUFMAN the enrolment data associated with the Dongle (C, PK, $(C)_{SK_D}$ and $(PK)_{SK_D}$) (2), and receives the corresponding response (3). The Browser validates the received signatures $(C)_{SK_D}$ and $(PK)_{SK_D}$, using previously obtained from the User PK_D (4). If the validation is successful, the enrolment is continued (5); otherwise, it is terminated. Subsequently, the Server performs the enrolment procedure for the User: creates a session, assigns a $SessionID$, and stores it in the session $Cookies$ (6). Finally, the Server sends $Cookies$ to the Browser (7).

At the end of this phase, the User's session is established, and the Browser holds $Cookies$ containing $SessionID$.

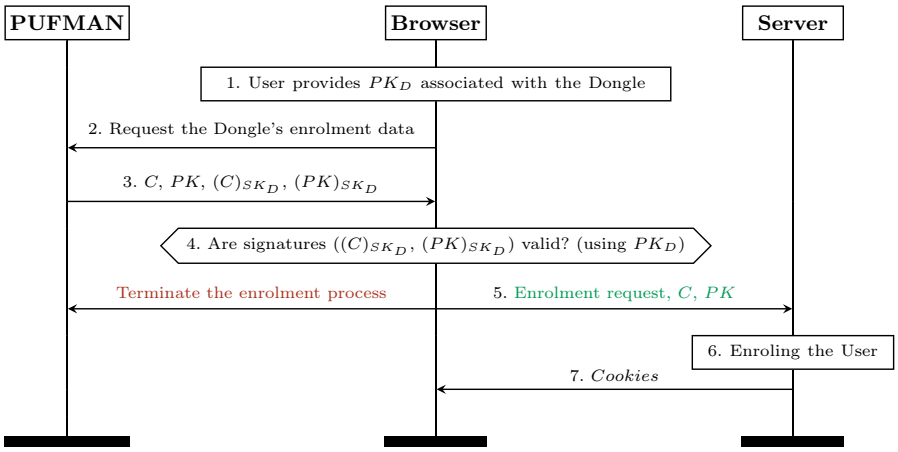


Fig. 4. Workflow of PUSH during the Enrolment Phase (II).

III. Operation Phase

After the enrolment phase is completed, PUSH proceeds to the operation phase, during which subsequent communication between the protocol parties is carried out according to the algorithm presented in Figure 5:

The Browser requests a service from the Server and sends $Cookies$ (1). The Server responds with the challenge C, a random nonce N and a request to sign C, N, and $Cookies$ (2). This request is then forwarded to the Dongle via the Browser and PUFMAN (3).

Upon receiving the request, the Dongle initiates the signing process (4) (see Figure 6): the Crypto Module triggers the power-up of the SRAM module, and reads values r_i for each address $c_i \in C$, forming the response $R \coloneqq \{r_i\}$, where $i = 0, \ldots, m$ (4.1-4.2); then, it generates SK using $KeyGen(Fuzzy(R))$ (4.3) and signs N and $Cookies$ with this SK (4.4).

After signing, the Dongle sends N, $Cookies$, and the signatures, $(N)_{SK}$ and $(Cookies)_{SK}$, to the Server via PUFMAN and the Browser (5). The Server then validates N and the signatures (using PK). If all items are valid, the session is continued; otherwise, it is terminated (6).

The signing procedure can be requested by the server on every interaction. However, this may introduce additional latency (see Section 3.5). To mitigate this, PUSH can be configured so that the server includes the signing request in one of its responses to the browser, and then receives the updated signature in one of the browser's subsequent requests. In the meantime, routine request-response operations continue uninterrupted (see Figure 5 (- -›)).

Phases I–III describe the operation of PUSH under normal conditions. However, exceptions may arise that require special handling by the protocol. These exceptions and the corresponding PUSH behaviour are discussed in Section E.

This is a Pre-print: The final version will appear in the proceedings of Computer Security – ESORICS 2025, in Volume 16055 of the Lecture Notes in Computer Science by Springer Nature.

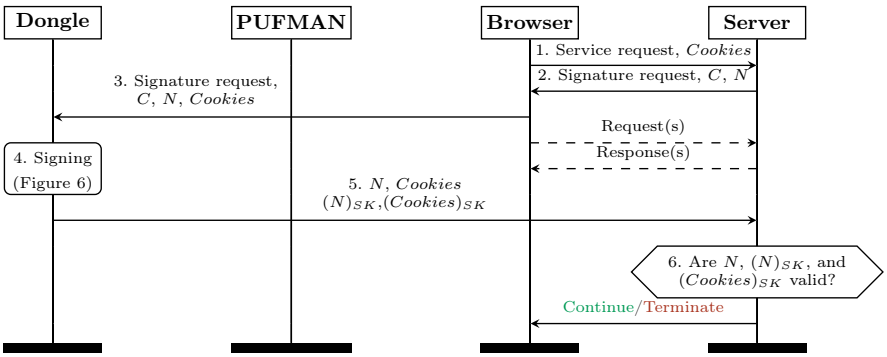


Fig. 5. Workflow of PUSH during the Operation Phase (III).

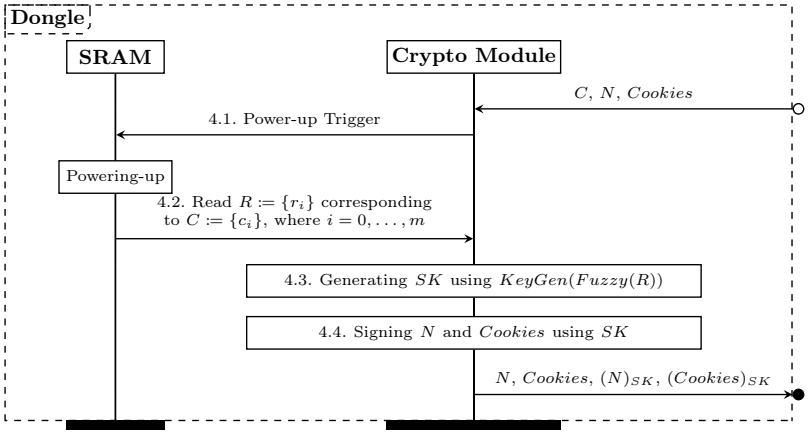


Fig. 6. Workflow of the PUSH protocol during the Signing Sub-phase.

E. Exception Cases Handling

Consider two scenarios where a user needs to replace a dongle or associate an additional dongle with the same account. In both cases, PUSH performs two steps: a common *preparation* phase, followed by either a dongle *replacement* or *association*.

Preparation: The user requests the server to generate a set of recovery keys $RKeys := (RK_1, \ldots, RK_n)$ and stores it securely. The dongle replacement or new association can only be completed if the user provides RK_i to the server, thereby preventing a potential unauthorised process initiated by an adversary. This step should be completed before the need for a replacement or new association, ideally, immediately after the initial enrolment with the server.

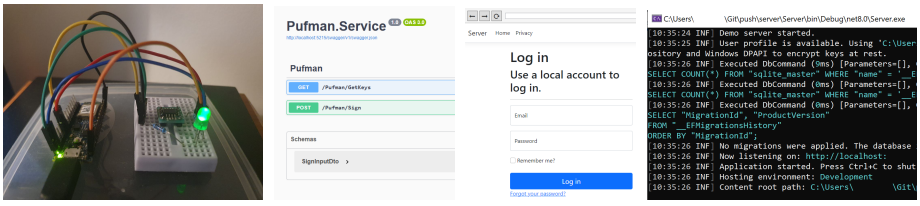
Dongle Replacement: The user requests PUFMAN to initiate the dongle replacement procedure. First, the PUF initialisation (I) for the new dongle is performed. Then, PUSH proceeds with the enrolment phase (II), but instead of the enrol-

ment request (II.5), the browser sends a replacement request to the server. This request contains the user's recovery key RK_i and the new dongle's data obtained during the PUF initialisation phase. This allows the server to authenticate the user and associate the session with the new dongle.

Association of an Additional Dongle with an Existing Account: The user requests PUFMAN to associate an additional dongle with the existing account. First, the PUF initialisation (I) for the new dongle is performed, during which PUFMAN receives a challenge C_{new}, a public key PK_{new}, and their signed values $(C_{new})_{SK_{new_D}}$ and $(PK_{new})_{SK_{new_D}}$, where the secret key SK_{new_D} and the public key PK_{new_D} are issued by the manufacturer of the new dongle. Next, PUFMAN requests the old dongle to sign C_{new} and PK_{new}, receiving $(C_{new})_{SK_{old}}$ and $(PK_{new})_{SK_{old}}$. PUFMAN then passes all the data it has collected to the browser. The browser asks the user to enter PK_{new_D} and verifies items signed with SK_{new_D} to confirm the existence of the new dongle. Upon successful verification, the browser forms an additional association request to the server, including RK_i, C_{new}, PK_{new}, $(C_{new})_{SK_{old}}$ and $(PK_{new})_{SK_{old}}$, allowing the server to verify the source of the request, confirm the presence of both dongles, and associate the new dongle with the account.

3.3 Proof-of-Concept Implementation

Protocol Parties: For the implementation of the dongle, the Arduino Nano 33 BLE board [5], equipped with the external 512 Kb SRAM module [30], was selected (see Figure 7 (a)). The external SRAM can be power-cycled independently of the dongle, enabling PUF extraction at module startup without the need to reboot the dongle itself, simplifying implementation. PUFMAN is implemented as a service using the Microsoft ASP.NET Core Web Service Framework and provides the browser with an API to interact with the USB dongle (see Figure 7 (b)). To simulate web communication, we developed a demo browser (see Figure 7 (c)) and a demo server (see Figure 7 (d)) using the Microsoft .NET WPF and Microsoft ASP.NET Core 8.0 frameworks, respectively. Communication between the dongle and PUFMAN was carried out via the USB port, while all others are done via HTTPS. Alternatively, a memory pipe [32] could have been used for data transfer between PUFMAN and the browser.



(a) (b) (c) (d)

Fig. 7. PUSH prototype: (a) the dongle, (b) PUFMAN, (c) the browser, (d) the server.

PUF Handling and Key Generation: During PUF initialisation (I), the SRAM was power-cycled three times ($k = 3$) and its response sets MV_1, MV_2, MV_3 were analysed to identify CRP (16 B), consisting of values closest to the ideal in terms of both Hamming Distance ($= 0$) and Hamming Weight ($= 0.5$). Then, the response R was processed using the fuzzy extractor [29] and passed to the cryptographic algorithm for key generation. Any suitable asymmetric cryptosystem supporting key generation and digital signing can be selected. In our prototype, Curve25519 elliptic curve cryptography was used, which is well-suited to Arduino boards due to its lightweight nature. Implementation was performed using the Ed25519 class from the Arduino Cryptography Library [44].

3.4 Performance Evaluation

To evaluate the overhead introduced by PUSH, the following aspects were measured: execution time, RAM usage, required persistent storage, network payload, and power consumption (see Table 1). The protocol involves several data contributing to this overhead, including memory values MV (48 B); the challenge C and response R (16 B); cryptographic keys SK, PK, SK_D, PK_D, and signatures $(X)_Y$ (each 32 B); a nonce N (8 B). Furthermore, the following fine-consuming operations are performed: power-cycling, CRP identification, $KeyGen(Fuzzy(R))$, digital signing, and signature validation.

Table 1. Resource overhead introduced by the PUSH protocol.

Phase	Time	RAM (KB)			Storage (B)			Network			Power (mW)		
		D	UD	S	D	UD	S	D↔P	P↔Br	Br↔S	D	UD	S
I. PUF Initialisation	11 sec	160	–		36	176	–	176 B	–		154	–	–
II. Enrolment	2 ms	–	2	–	–	32	48	–	176 B	48 B	154	–	–
III. Operation	419 ms	4	–	3	–		16	2.16 KB		176 B	154	–	–

D: Dongle; **UD**: User's Device (hosts Browser **Br** and PUFMAN **P**); **S**: Server;
↔: communication between protocol parties; –: no overhead introduced.

Based on Table 1, the PUF initialisation (I) is the most resource-intensive phase in terms of execution time and memory requirements. On average, this phase took 11 seconds (sec), with around 9 sec consumed by three power-cycling operations of the SRAM, as it is necessary to wait roughly 3 sec between reboots to allow the memory to be fully reset. The remaining time was spent on CRP identification and key generation (see Figure 8). Although 11 sec is relatively long, it does not introduce communication latency, as this phase only runs once and precedes the operational phase. In terms of memory consumption, the dongle must allocate a significant amount of its RAM (160 KB out of 256 KB available in our implementation). However, the data that needs to be stored during operation is relatively small: 36 B on the dongle (SK_D and helper data for $Fuzzy(R)$), and 176 B on the user's device (C, PK, $(C)_{SK_D}$, and $(PK)_{SK_D}$).

The enrolment phase (II), by contrast, introduces minimal overhead. In terms of execution time, only 2 ms are required, which is deemed negligible. Moreover, the user's device is only required to store PK_D (32 B) for dongle verification while the server additionally stores C and PK (48 B in total). The network overhead is also minor, especially when compared to the typical volume of data transferred during web communication [42].

This is a Pre-print: The final version will appear in the proceedings of Computer Security – ESORICS 2025, in Volume 16055 of the Lecture Notes in Computer Science by Springer Nature.

In the operation phase (III), the primary concern is the execution time, as the PUSH protocol potentially can introduce latency that would negate its lightweight nature. The most time-consuming operation here is the cookie signing. Our measurements showed that signing 1 KB cookies takes an average of 419 ms (see Figure 8), which would lead to noticeable delays if performed on every request. To address this, cookie signatures are updated every 4 minutes, with a validity period of 5 minutes (see Section 3.5). However, in this case, a trade-off between security and latency is inevitable. More frequent signature updates increase security but also introduce higher latency, whereas less frequent updates minimise latency at the cost of reduced security.

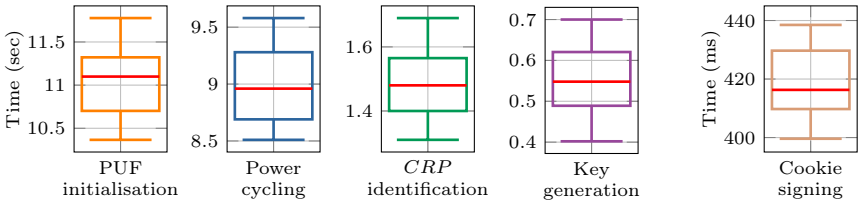


Fig. 8. Time required to complete the PUF initialisation fully, its individual steps (power cycling, CRP identification, key generation), and the signing of 1 KB cookies.

The power overhead introduced by the dongle remains constant across all phases, as the dongle (Arduino) operates in a constant power mode; this consumption is minimal and can be considered negligible. The power overhead from operations on the user's device and the server is similarly low. It was undetectable during measurements and thus considered negligible.

In summary, PUSH introduces very low overhead. Its modest memory requirements make it suitable for resource-constrained devices (e.g., Arduino). However, using larger CRP sets, longer keys, or alternative cryptographic algorithms may require more powerful hardware. Notably, despite two relatively time-consuming phases, PUSH adds no latency during regular communication.

3.5 Challenges

The following three key areas of concern can be identified in the implementation and deployment of PUSH: latency, scalability, and applicability.

Latency: The evaluation results indicate that the operation phase may introduce latency if cookies and nonce signing are performed on every request. To address this, the following signature update strategy is used: Assume that the signature remains valid and the session is legitimate for a predefined time interval $T_{validity}$. When the signature expires, the server terminates the session. Thus, the signature must be updated before its expiration, i.e., $T_{update} < T_{validity}$. For example, one minute before expiration, the server includes a signature request in a response to a user's request. The updated signature is then sent to the server with a subsequent request. This process runs in the background, without user interaction or disruption to ongoing communication. As mentioned earlier, there is a trade-off between security and latency. If $T_{validity}$, and consequently

T_{update}, are set very short, a higher security level can be achieved, but at the cost of increased delays. Conversely, longer $T_{validity}$ and T_{update} reduce latency but also decrease the security level.

Scalability: The scalability of PUSH refers to the number of concurrent sessions that can be associated with a single dongle. Assume that each session (originating from a different server) is linked to a unique and independent (i.e., non-overlapping) CRP. Consequently, the total number of such $CRPs$ is the maximum number of supported sessions. In our prototype, 976 $CRPs$ (16 B) can be derived from SRAM. Increasing CRP's length to 32 B would enhance security but reduce the number of usable $CRPs$ to 448. This limitation can be addressed by using a larger SRAM or adding an extra SRAM module, both of which would increase monetary costs; or by using an alternative PUF (e.g., arbiter PUF [17]), which may require protocol modifications.

Applicability: In this paper, we describe the session binding to a USB dongle, which ensures cross-platform compatibility and facilitates development. However, the dongle is an external device and is inherently resource-constrained due to its compact form factor. Moreover, it could potentially be stolen and used for future attacks, which may not be immediately noticed by the user. To address these limitations, we see the potential for implementing PUSH without relying on a dongle. For example, the session could be bound directly to the SRAM module of the user's device, e.g., a laptop. This approach would eliminate the need for additional hardware and reduce resource constraints while maintaining cross-platform compatibility. Nevertheless, such an implementation would likely require cooperation from device manufacturers, as SRAM is generally not externally accessible.

4 Security Analysis

To analyse the security of PUSH, we define an adversary model that operates within the system model described in Section 3.1. Subsequently, we discuss the protocol's resistance to potential attacks carried out by the defined adversary.

4.1 Adversary Model

The adversary $\mathcal{A}$ aims to disrupt the correct operation of PUSH by performing various attacks within their capabilities and limitations [14]. It is assumed that $\mathcal{A}$ can communicate with the PUSH parties; intercept, modify, replay, or drop protocol messages; and compromise the browser (e.g., through malicious browser extensions). However, $\mathcal{A}$ cannot physically access or tamper with the dongle or its SRAM; generate the secret key (SK) without the dongle's PUF response; forge or guess cryptographic primitives (e.g., digital signatures, nonces).

4.2 Resistance of the Protocol to Attacks

We evaluate the resistance of PUSH not only against session hijacking attacks [10,22], but also against other potential attacks that $\mathcal{A}$ may attempt to disrupt the correct operation of the protocol [21,27].

Sniffing Attack: During passive monitoring of traffic between protocol parties, $\mathcal{A}$ may obtain public data (C, PK), $Cookies$, N, and signed values (N_{SK}, $Cookies_{SK}$). Even if this data is transmitted in plain text (which is not the case, as PUSH operates on top of HTTPS) and the session ID is exposed, it would not enable successful future attacks. This is because each authentication requires a fresh server-generated nonce N and its valid signature, both of which $\mathcal{A}$ cannot forge according to the assumed adversary model.

Cross-Site Scripting (XSS) Attack: $\mathcal{A}$ injects a malicious script (typically written in JavaScript) into a trusted website. When the user visits this website, the script is triggered to send cookies that are not marked as HTTP-only to $\mathcal{A}$. However, even in this scenario, PUSH remains secure because successful session hijacking also requires a fresh, signed nonce N, which $\mathcal{A}$ cannot forge (see Section 4.1).

Reply Attack: PUSH is resistant to replay attacks, because even if $\mathcal{A}$ intercepts messages over time, they can only obtain PK, C, N, $Cookies$ and their signatures. However, this data is not reusable, as the nonce N is fresh, and the server detects and rejects any attempt to replay the old one.

Man-in-the-Middle (MitM) Attack: Although $\mathcal{A}$ may attempt to inject or modify C, N or signatures, as allowed by the adversary model (see Section 4.1), the periodic requirement for fresh digital signatures, their validation, and the use of HTTPS ensure data integrity and protect against potential MitM attacks.

Side-Channel Attack, Device Capturing or Cloning: In our analysis, we focus on an online $\mathcal{A}$, typical in web communication, who cannot physically access the dongle and thus cannot perform side-channel attacks, theft, or cloning. As a result, physical protection of the dongle is beyond the scope of this paper but can be addressed in future work through hardware-level countermeasures [28]. Nevertheless, such attacks remain relevant in broader security contexts. Although PUSH does not inherently protect against them, the absence of permanent storage of secret data in the dongle, combined with the unclonability and the unpredictability of PUFs, prevents the disclosure of confidential information, cloning, or long-term simulation. Moreover, the user would likely detect the dongle's absence before its reverse engineering is successfully completed.

Brute Force or Phishing Attack: Assume that $\mathcal{A}$ obtains a session ID, either by guessing it or by sending a seemingly trustworthy phishing email containing malicious links or malware, and the user falls for the trick. Using this session ID, $\mathcal{A}$ can hijack the session. However, PUSH introduces an additional layer of security by requiring the session cookies and a freshly generated nonce N to be periodically signed using the dongle. $\mathcal{A}$ cannot derive SK from the dongle, cannot clone PUF, and cannot forge signatures. Therefore, even if the session is hijacked, it will be terminated once the valid signature expires.

Impersonation Attack: To impersonate the user, $\mathcal{A}$ must be able to sign cookies and nonces in response to the server's requests. This would require collusion with one of the protocol parties, which is excluded under the system model. Alternatively, $\mathcal{A}$ would need to replicate the dongle's SRAM to reconstruct SK (see Section 4.1), which is infeasible due to the inherent properties of PUF.

This is a Pre-print: The final version will appear in the proceedings of Computer Security – ESORICS 2025, in Volume 16055 of the Lecture Notes in Computer Science by Springer Nature.

Denial-of-Service (DoS) Attack: All parties in the PUSH protocol can be potential targets of DoS attacks. A direct attack on the dongle is only possible if PUFMAN is compromised, which is excluded by the system model (see Section 3.1). However, flooding PUFMAN with requests can indirectly block access to the dongle. The browser and the server may also be attacked by a large number of requests, consuming resources and disrupting their operation. While PUSH does not inherently protect against DoS attacks, it allows the integration of common countermeasures, e.g, rate limiting, API access controls, etc.

Browser Compromise: Since the browser may be exposed to malicious extensions (see Section 3.1), we evaluate the impact of its compromise. In general, compromising the browser is ineffective, as it has no access to secret data, and previously signed cookies cannot be reused due to the requirement for a fresh nonce. However, if the browser is compromised before enrolment, $\mathcal{A}$ could potentially bind the session to their dongle, thereby performing a session fixation attack. Thus, even without revealing secret information, the session could still be compromised.

For all the attacks discussed above in which $\mathcal{A}$ may obtain cookies, such as sniffing, it is accurate to state that the cookies could potentially be used until the dongle's signature expires. However, we believe this poses minimal risk and is unlikely to enable $\mathcal{A}$ to cause significant damage, as the signature validity period can be flexibly configured to limit the time during which the cookies remain usable. This period can be set quite short without introducing noticeable delays (e.g., four minutes in our implementation).

5 Related Work

In this section, we review existing works on countermeasures against session hijacking, which are considered to be most relevant to our study.

Non-PUF-based approaches aim to protect web communication and typically involve binding a unique user or session ID to device-specific attributes, e.g., geolocation [1] or hardware characteristics [18]. Moreover, protection is often implemented through the use of expiration limits applied to sessions [35], cryptographic keys [36], or tokens [13,37]. Some methods are also designed to provide continuous authentication [2,33], while others aim to obfuscate session IDs [34]. The disadvantages of these approaches include their often non-lightweight and complex design, which leads to delays, system overloads, and maintenance challenges; as well as reliance on additional hardware and the need to store secret keys, both of which broaden the attack surface and increase security risks.

Considering *PUF-based approaches*, it can be observed that none of them are applied to web communication. Instead, they primarily target the IoT domain, where PUFs are used to establish authentication between devices, typically serving as a source of unique identifiers [3,25], cryptographic keys [24,38], or random numbers [4,6]. Similar to non-PUF-based approaches, PUF-based ones often require additional hardware and storage of secret cryptographic primitives, in-

creasing security risks. Additionally, some solutions rely on time synchronisation, which introduces further challenges in terms of reliability and implementation.

6 Discussion

This section presents the comparative analysis of PUSH and the approaches discussed in Section 5. Since existing solutions have been evaluated from different perspectives by their authors, a unified framework is adopted for the analysis. Specifically, each approach is assessed according to nine characteristics and resistance to eleven common attack types. For objectivity, the comparison relies on assumptions and measurements reported by the original authors, aligned with our threat model. The consolidated results are presented in Table 2.

Table 2. Comparison of PUSH with the existing approaches against session hijacking.

Authors	🌐	🫆	⚖	Advantages ▩	Advantages ↔	Advantages 🔑	Limitations ❐	Limitations ?	Limitations ⏱	SA	XSS	RA	MM	SC	BF	PA	IA	DoS	DCC	BC
Non-PUF-based Approaches																				
Ahmed et al. [1]	✓	-	✓	✓	✓	✗	✓	✓	✓	○	●	●	●	○	●	●	●	○	●	○
Al-Sadi et al. [2]	✓	-	✓	✗	✗	✓	✗	✗	✓	●	○	●	●	○	●	●	●	○	-	○
Dacosta et al. [13]	✓	-	✓	✓	✗	✓	✗	✗	✗	●	●	●	●	●	○	○	○	○	○	○
Hwang et al. [18]	✓	-	✓	✓	✓	✗	✗	✓	✗	●	●	○	○	○	○	●	●	○	-	○
Monsen et al. [33]	✓	-	✗	✓	✓	✗	✓	✗	✗	●	●	●	●	○	●	●	●	○	○	○
Nikiforakis et al. [34]	✓	-	✓	✓	✗	✗	✗	✗	✗	○	●	●	●	○	○	○	○	○	○	●
Ogundele et al. [35]	✓	-	✗	✗	✗	✗	✓	✗	✗	●	●	○	●	○	○	●	○	●	-	-
Pothumarti et al. [36]	✓	-	✓	✓	✓	✓	✓	✗	✗	●	●	●	○	○	●	●	○	○	●	-
Prapty et al. [37]	✓	-	✗	✓	✗	✓	✓	✗	✗	●	●	●	●	○	○	●	●	○	○	○
PUF-based Approaches																				
Aljrees et al. [3]	✗	n	✓	✓	✗	✗	✗	✗	✓	●	-	●	●	○	○	○	○	●	○	-
Alshaeri et al. [4]	✗	s	✓	✓	✗	✗	✓	✓	✗	●	-	●	●	○	●	○	●	○	○	-
Badshah et al. [6]	✗	n	✓	✗	✓	✓	✗	✓	✓	○	-	●	●	○	●	○	●	●	●	-
Lo et al. [24]	✗	n	✓	✓	✓	✓	✓	✗	✗	●	-	●	●	○	○	○	●	○	○	-
Lounis et al. [25]	✗	n	✓	✓	✓	✓	✓	✗	✗	●	-	●	○	○	●	○	●	○	○	-
Siddiqui et al. [38]	✗	n	✓	✓	✓	✓	✓	✓	✗	○	-	●	●	○	○	○	●	○	●	-
PUSH	✓	w	✓	✓	✓	✓	✗	✗	✗	●	●	●	●	●	●	●	◐	⊙	●	◐

🌐: applicability in web communication; 🫆: PUF type (**n**ot specified, **s**trong, **w**eak); ⚖: lightweightness; ▩: obfuscation of data; ↔: prior mutual authentication; 🔑: independent cryptographic primitives; ❐: need for additional hardware; ?: security level highly depends on implementation; ⏱: time synchronisation requirement.
Attacks: SA: Sniffing; **XSS**: Cross-Site-Scripting; **RA**: Reply; **MM**: Man-in-the-Middle; **SC**: Side-Channel; **BF**: Brute Force; **PA**: Phishing; **IA**: Impersonation; **DoS**: Denial-of-Service; **DCC**: Device Capturing & Cloning; **BC**: Browser Compromise.
✓/✗: yes/no; ●/◐/○: secure/partially secure/not secure; ⊙: can be integrated; −: not applicable.

Applicability in web communication (🌐): As mentioned in Section 5, the non-PUF-based approaches primarily target session hijacking in web communication, whereas PUF-based solutions focus on the IoT domain. This distinction highlights the novelty of PUSH, which is the first protocol to employ PUFs specifically for securing web sessions.

PUF type (🫆) applies exclusively to PUF-based approaches. In most works, the type is unspecified and treated generically. Only one solution relies on a

strong PUF with many $CRPs$ [4], while PUSH supports even weak PUFs as SRAM-based, which can provide only a limited number of $CRPs$.

Lightweightness (⚖): All PUF-based solutions are lightweight in terms of resources, whilst this applies to only about half of the non-PUF-based approaches.

Obfuscation (▩): Most approaches, with a few exceptions [2,6,9,35], apply techniques to conceal sensitive data, e.g., CRPs, to ensure confidentiality and prevent reuse. In PUSH, vulnerable data (e.g., cookies) is encrypted both at rest (using DPAPI) and in transit (via HTTPS). Furthermore, in PUSH, the secret key SK is neither stored nor transmitted, because it is generated on demand.

Prior mutual authentication (↔) is incorporated in nearly half of the reviewed approaches (e.g., [18,36]), reducing the risk of potential attacks and establishing a secure foundation for session setup. In PUSH, mutual authentication is achieved using manufacturer-issued keys: the secret key SK_D is embedded in the dongle, while the corresponding public key PK_D is provided to the user.

Independent cryptographic primitives (🔑) are featured in eight of the reviewed approaches (e.g., [2,13]). Using entirely different primitives enhances resistance to cryptanalysis and side-channel attacks. PUSH leverages this principle in several ways: a secret key SK is derived from a unique SRAM PUF; each distinct CRP enables a unique SK; and a fresh nonce N is included in signature requests.

Need for additional hardware or protocol parties (🗐) such as a Trusted Platform Module (TPM) [33], is a common limitation among the existing approaches. PUSH employs a dongle for binding; however, this could be replaced by the user device's SRAM (see Section 3.5). Although this alternative would require manufacturer support, it would eliminate the need for external hardware.

Security level highly depends on selected mechanisms (❓): Most of the approaches reviewed emphasise the need for careful selection of methods used (e.g., hashing) to ensure security. Our analysis (see Section 4) has shown that PUSH provides strong protection by design, with minimal dependence on specific techniques, though security recommendations, e.g., regarding key lengths, should still be followed [7,8].

Time synchronisation (⏱) is required for some approaches [1,2,3,6]. Although this is a reasonably robust and common way to mitigate some attacks, reliance on synchronisation increases implementation and deployment complexity. In contrast, PUSH does not rely on time synchronisation, but uses a random nonce N to ensure request freshness.

Resistance to Attacks: To evaluate the resistance of each approach to the attacks, we relied on the authors' claims, which typically covered only mitigated ones. If an attack was not mentioned, we attempted to assess the resistance of the presented approach based on available details. When this was infeasible, the approach was marked as not secure (○). The detailed security analysis of PUSH can be found in Section 4. While some attacks, e.g., DoS, are not addressed, and protection against impersonation and browser compromise is only partial, PUSH demonstrates resistance to a broader range of attacks compared to other approaches under the defined adversary model (see Section 4.1).

This is a Pre-print: The final version will appear in the proceedings of Computer Security – ESORICS 2025, in Volume 16055 of the Lecture Notes in Computer Science by Springer Nature.

Among the reviewed approaches, Device Bound Session Credentials (DBSC) is most similar to PUSH and merits a more detailed comparison [33]. Both aim to enable continuous session verification, yet differ in implementation and compatibility. DBSC relies on a TPM, which is slow and not universally supported across devices. For instance, while a TPM is mandatory on Windows 11, users of other operating systems (OS) may lack TPM support and thus cannot use DBSC. In contrast, PUSH uses a USB-connected dongle that is OS-independent, as USB is universally supported. Alternatively, PUSH could leverage internal SRAM (see Section 3.5), removing the need for external hardware altogether. Another difference lies in the introduced latency. DBSC requires an additional server port to avoid delays, while PUSH does not introduce any latency issues.

In summary, PUSH is a secure and efficient protocol for binding web sessions to physical devices using an SRAM PUF. It is significantly distinct from existing approaches and represents the first use of PUF for this purpose. PUSH ensures continuous verification of session legitimacy, providing strong protection against session hijacking and other attacks. Notably, PUSH avoids persistent storage of cryptographic keys, thereby also enhancing resistance to physical and memory-based attacks. Designed for lightweight operation, PUSH is well-suited to resource-constrained environments and integrates seamlessly into web communication without introducing latency. Moreover, it is cross-platform and works on any operating system that supports USB. Although the prototype is implemented with an external dongle, PUSH can be adapted to use the device's internal SRAM, eliminating the need for additional hardware.

7 Conclusion and Future Work

In this paper, we propose a novel SRAM PUF-based hardware-binding protocol, PUSH, which associates a web session with the user's device (dongle) and enables continuous verification of session legitimacy to mitigate session hijacking and related attacks. The results of a comprehensive evaluation of PUSH, including performance and security analysis, and comparison with existing approaches, have shown that PUSH is a secure, lightweight, and flexible protocol. It addresses the limitations of existing solutions and demonstrates great potential to enhance the security of web communication.

The following aspects are not covered in this paper but represent interesting directions for future research: evaluating the impact of different implementation choices on the security of PUSH; exploring the possibility of using alternative types of PUFs; and implementation of PUSH without a dongle, relying solely on the internal SRAM module of the user's device.

Acknowledgments. This work was funded by the Bavarian State Ministry of Science and Arts (BayStMWK), under the project "Secure Encapsulation" of the Bavarian Research Association "FORDaySec" ("Security in Everyday Use of Digital Technologies") and by the Interreg VI-A Programme Germany/Bavaria–Austria 2021–2027 – Programm INTERREG VI-A Bayern–Österreich 2021–2027, as part of Project BA0100016: "CySeReS-KMU: Cyber Security and Resilience in Supply Chains with focus on SMEs", which is co-funded by the European Union.

References

1. Ahmed, A.A., Ahmed, W.A.: An Effective Multifactor Authentication Mechanism Based on Combiners of Hash Function over Internet of Things. Sensors **19**(17) (2019). https://doi.org/10.3390/s19173663
2. Al-Sadi, M., Di Pietro, R., Lombardi, F., Signorini, M.: LENTO: Unpredictable Latency-based continuous authEntication for Network inTensive IoT envirOnments. Future Generation Computer Systems **139**, 151–166 (2023). https://doi.org/10.1016/j.future.2022.09.023
3. Aljrees, T., Kumar, A., Singh, K.U., Singh, T.: Enhancing IoT Security through a Green and Sustainable Federated Learning Platform: Leveraging Efficient Encryption and the Quondam Signature Algorithm. Sensors **23**(19) (2023). https://doi.org/10.3390/s23198090
4. Alshaeri, A., Younis, M.: Distributed Hardware-Assisted Authentication and Key Agreement Protocol for Internet of Things. In: 2024 IEEE 21st Consumer Communications & Networking Conference (CCNC). pp. 152–158 (2024). https://doi.org/10.1109/CCNC51664.2024.10454706
5. Arduino: Nano 33 BLE Documentation (2025), https://docs.arduino.cc/hardware/nano-33-ble/
6. Badshah, A., Abbas, G., Waqas, M., Tu, S., Abbas, Z.H., Muhammad, F., Chen, S.: USAF-IoD: Ultralightweight and Secure Authenticated Key Agreement Framework for Internet of Drones Environment. IEEE Transactions on Vehicular Technology **73**(8), 10963–10977 (2024). https://doi.org/10.1109/TVT.2024.3375758
7. Barker, E.B., Kelsey, J.M., McKay, K.A., Roginsky, A.L., Turan, M.S.: Recommendation for Random Bit Generator (RBG) Constructions. Tech. Rep. NIST SP 800-90C (Fourth Public Draft), National Institute of Standards and Technology (NIST), Gaithersburg, MD (2024). https://doi.org/10.6028/NIST.SP.800-90C.4pd
8. Barker, E.B., Roginsky, A.L.: Transitioning the Use of Cryptographic Algorithms and Key Lengths. Tech. Rep. NIST SP 800-131A Rev. 3, National Institute of Standards and Technology (NIST), Gaithersburg, MD (2024). https://doi.org/10.6028/NIST.SP.800-131Ar3.ipd
9. Bharti Kumar, A., Chaudhary, M.: Prevention of Session Hijacking and Ipspoofing with Sensor Nodes and Cryptographic Approach. International Journal of Computer Applications **76**(9), 22–28 (Aug 2013). https://doi.org/10.5120/13275-0821
10. Bugliesi, M., Calzavara, S., Focardi, R., Khan, W.: CookiExt: Patching the browser against session hijacking attacks. Journal of Computer Security **23**(4), 509–537 (Sep 2015). https://doi.org/10.3233/jcs-150529
11. Cherupally, S.K., Yin, S., Kadetotad, D., Bae, C., Kim, S.J., Seo, J.s.: A Smart Hardware Security Engine Combining Entropy Sources of ECG, HRV, and SRAM PUF for Authentication and Secret Key Generation. IEEE Journal of Solid-State Circuits **55**(10), 2680–2690 (2020). https://doi.org/10.1109/JSSC.2020.3010705
12. Clark, L.T., Medapuram, S.B., Kadiyala, D.K.: SRAM Circuits for True Random Number Generation Using Intrinsic Bit Instability. IEEE Transactions on Very Large Scale Integration (VLSI) Systems **26**(10), 2027–2037 (2018). https://doi.org/10.1109/TVLSI.2018.2840049
13. Dacosta, I., Chakradeo, S., Ahamad, M., Traynor, P.: One-time cookies: Preventing session hijacking attacks with stateless authentication tokens. ACM Trans. Internet Technol. **12**(1) (Jul 2012). https://doi.org/10.1145/2220352.2220353
14. Dolev, D., Yao, A.: On the security of public key protocols. IEEE Transactions on Information Theory **29**(2), 198–208 (1983). https://doi.org/10.1109/TIT.1983.1056650

15. D'silva, K., Vanajakshi, J., Manjunath, K.N., Prabhu, S.: An effective method for preventing SQL injection attack and session hijacking. In: 2017 2nd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT). pp. 697–701 (2017). https://doi.org/10.1109/RTEICT.2017.8256687
16. Guajardo, J., Kumar, S.S., Schrijen, G.J., Tuyls, P.: FPGA Intrinsic PUFs and Their Use for IP Protection. In: Paillier, P., Verbauwhede, I. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2007, pp. 63–80. Springer (2007). https://doi.org/10.1007/978-3-540-74735-2_5
17. Hemavathy, S., Bhaaskaran, V.S.K.: Arbiter PUF—A Review of Design, Composition, and Security Aspects. IEEE Access **11**, 33979–34004 (2023). https://doi.org/10.1109/ACCESS.2023.3264016
18. Hwang, W.S., Shon, J.G., Park, J.S.: Web Session Hijacking Defense Technique using User Information. Human-centric Computing and Information Sciences **12**, 16 (2022)
19. Kapko, M.: Slack resets passwords en masse after invite link vulnerability. Cybersecurity Dive (2022), https://www.cybersecuritydive.com/news/slack-password-vulnerability/629026/, accessed: 2025-04-01
20. Katzenbeisser, S., Schaller, A.: Physical Unclonable Functions: Sicherheitseigenschaften und Anwendungen. Datenschutz und Datensicherheit - DuD **36**(12), 881–885 (Nov 2012). https://doi.org/10.1007/s11623-012-0295-z
21. Kumar, A., Saha, R., Conti, M., Kumar, G., Buchanan, W.J., Kim, T.H.: A comprehensive survey of authentication methods in Internet-of-Things and its conjunctions. Journal of Network and Computer Applications **204**, 103414 (Aug 2022). https://doi.org/10.1016/j.jnca.2022.103414
22. Kumar Baitha, A., Vinod, S.: Session Hijacking and Prevention Technique. International Journal of Engineering & Technology **7**(2.6), 193–198 (Mar 2018). https://doi.org/10.14419/ijet.v7i2.6.10566
23. Linus Tech Tips: My Channel Was Deleted Last Night (2023), https://www.youtube.com/watch?v=yGXaAWbzl5A, accessed: 2025-04-01
24. Lo, N.W., Yohan, A.: BLE-Based Authentication Protocol for Micropayment Using Wearable Device. Wirel. Pers. Commun. **112**(4), 2351–2372 (Jun 2020). https://doi.org/10.1007/s11277-020-07153-0
25. Lounis, K., Zulkernine, M.: T2T-MAP: A PUF-Based Thing-to-Thing Mutual Authentication Protocol for IoT. IEEE Access **9**, 137384–137405 (2021). https://doi.org/10.1109/ACCESS.2021.3117444
26. Maes, R.: Physically unclonable functions: Properties, pp. 49–80. Springer Berlin Heidelberg, Berlin, Heidelberg (2013). https://doi.org/10.1007/978-3-642-41395-7_3
27. Mall, P., Amin, R., Das, A.K., Leung, M.T., Choo, K.K.R.: PUF-Based Authentication and Key Agreement Protocols for IoT, WSNs, and Smart Grids: A Comprehensive Survey. IEEE Internet of Things Journal **9**(11), 8205–8228 (2022). https://doi.org/10.1109/JIOT.2022.3142084
28. Mangard, S., Oswald, E., Popp, T.: Power analysis attacks: Revealing the secrets of smart cards. Springer New York, NY (2007). https://doi.org/10.1007/978-0-387-38162-6
29. Mexis, N.: A Comprehensive Comparison of Fuzzy Extractor Schemes Employing Different Error Correction Codes. M.Sc. Thesis, University of Passau, Passau, Germany (Oct 2023). https://doi.org/10.15475/ccfesedecc.2023
30. Microchip Technology Inc.: 23LC512 - 512 Kbit SPI Serial SRAM (2025), https://www.microchip.com/en-us/product/23lc512#Documentation

31. Microsoft: CNG DPAPI (Data Protection API). Microsoft Learn, https://learn.microsoft.com/de-de/windows/win32/seccng/cng-dpapi#:~:text=DPAPI%20ist%20Teil%20von%20CryptoAPI,zu%20verschl%C3%BCsseln%20und%20zu%20entschl%C3%BCsseln.
32. Microsoft: Pipes (Interprocess Communications) (2021), https://learn.microsoft.com/en-us/windows/win32/ipc/pipes
33. Monsen, K., Birgisson, A.: Fighting Cookie Theft Using Device-Bound Keys. https://blog.chromium.org/2024/04/fighting-cookie-theft-using-device.html (Apr 2024)
34. Nikiforakis, N., Meert, W., Younan, Y., Johns, M., Joosen, W.: SessionShield: Lightweight Protection against Session Hijacking. In: Erlingsson, Ú., Wieringa, R., Zannone, N. (eds.) Engineering Secure Software and Systems. pp. 87–100. Springer Berlin Heidelberg, Berlin, Heidelberg (2011). https://doi.org/10.1007/978-3-642-19125-1_7
35. Ogundele, I., O.Akinade, A., Alakiri, H.: Detection and Prevention of Session Hijacking in Web Application Management. IJARCCE **9**, 1–10 (Jun 2020). https://doi.org/10.17148/IJARCCE.2020.9601
36. Pothumarti, R., Jain, K., Krishnan, P.: A lightweight authentication scheme for 5G mobile communications: a dynamic key approach. Journal of Ambient Intelligence and Humanized Computing (Jan 2021). https://doi.org/10.1007/s12652-020-02857-4
37. Prapty, R.T., Azmin Md, S., Hossain, S., Narman, H.S.: Preventing Session Hijacking using Encrypted One-Time-Cookies. In: 2020 Wireless Telecommunications Symposium (WTS). pp. 1–6 (2020). https://doi.org/10.1109/WTS48268.2020.9198717
38. Siddiqui, Z., Gao, J., Khurram Khan, M.: An Improved Lightweight PUF–PKI Digital Certificate Authentication Scheme for the Internet of Things. IEEE Internet of Things Journal **9**(20), 19744–19756 (2022). https://doi.org/10.1109/JIOT.2022.3168726
39. Statista: Beyond Passwords: Biometrics, Multifactor, and Passwordless Authentication. Study (Jul 2023), https://www.statista.com/study/116099/beyond-passwords-biometrics-multifactor-and-passwordless-authentication/
40. Statista: Worldwide digital population (Oct 2024), https://www.statista.com/statistics/617136/digital-population-worldwide/
41. SurfShark: Number of user accounts exposed worldwide from 1st quarter 2020 to 3rd quarter 2024 (in millions). Chart (Oct 2024), https://www.statista.com/statistics/1307426/number-of-data-breaches-worldwide/
42. The Chromium Projects: SPDY: An Experimental Protocol for a Faster Web (2009), https://www.chromium.org/spdy/spdy-whitepaper/
43. Wang, R., Selimis, G., Maes, R., Goossens, S.: Long-term continuous assessment of SRAM PUF and source of random numbers. In: Proceedings of the 23rd Conference on Design, Automation and Test in Europe. p. 7–12. DATE '20, EDA Consortium, San Jose, CA, USA (2020). https://doi.org/10.23919/DATE48585.2020.9116353
44. Weatherley, R.: Crypto Library for Arduino. Rweather Arduino Libraries, https://rweather.github.io/arduinolibs/crypto.html
45. Zhang, Y., Ge, Y.: Evaluation of Microcontroller-Based SRAM PUF and the Authentication Scheme. In: Proceedings of the 4th International Conference on Computer, Internet of Things and Control Engineering. p. 79–86. CITCE '24, Association for Computing Machinery, New York, NY, USA (2025). https://doi.org/10.1145/3705677.3705691