

Zu Beginn wollen wir zunächst den Unterschied zwischen den Begriffen **Authentisierung**, **Authentifizierung** sowie **Autorisierung** klären. Diese Begriffe werden beim Zugriff auf IT Applikationen verwendet, wobei sie oft synonym verwendet werden.

## Authentisierung

Bei der **Authentisierung** beweist eine Person, dass sie tatsächlich diejenige Person ist, die sie vorgibt zu sein. Eine Person legt also Nachweise vor, die ihre Identität bestätigen sollen. Je nach der eingesetzten Authentisierungsmethode, kann die Person ihre Identität unter anderem auf folgenden Wegen behaupten:

- Sie hat geheime Informationen, die nur ihr bekannt sind (z.B. Userid, Passwort)
- Sie besitzt einen Identifizierungsgegenstand, auch bekannt als **Token** (z.B. Personalausweis)
- Sie ist selbst das Identifizierungsobjekt (z.B. biometrische Merkmale wie Fingerabdruck).

Die Daten für die Authentisierung werden auch allgemein im Englischen als Credentials bezeichnet.

Kurz gesagt: Authentisierung bezeichnet das Nachweisen einer Identität.

In unserer Beispiellapplikation sind das: E-Mail-Adresse und Passwort.

## Authentifizierung

Die **Authentifizierung** stellt eine Prüfung der behaupteten Authentisierung dar. Bei der Authentifizierung ist nun der Prüfer an der Reihe. Er überprüft die Angaben auf ihre Echtheit. Zeitlich betrachtet findet eine Authentifizierung also nach einer Authentisierung statt.

In unserer Applikation werden die Informationen E-Mail-Adresse und Passwort an den Node.js Server übertragen und dort geprüft. Dazu werden die bereits gespeicherten Daten der registrierten User herangezogen und E-Mail-Adresse und Passwort verglichen.

## Autorisierung<sup>1</sup>

Die Autorisierung ist die Einräumung von speziellen Rechten (also etwa der Zugriff auf gespeicherte Informationen) nach erfolgreicher Überprüfung.

## Warum Cookies?

HTTP ist ein zustandsloses Protokoll. Das bedeutet, wenn du eine Seite in deinem Browser lädst und dann zu einer anderen Seite auf der gleichen Webseite navigierst, weder der Server noch der Browser irgendeine Möglichkeit haben zu wissen, dass es der gleiche Browser ist, der die gleiche Seite besucht. Daher muss jeder HTTP-Request alle Informationen beinhalten, die der Server benötigt, um die Anfrage zu erfüllen.

Moderne Webseiten wollen aber für den Kunden, die Kundin maßgeschneiderte Informationen bieten. Sie wollen mehr über die Nutzer wissen und in der Lage sein, diese Nutzer beim Surfen zu tracken. Beliebte Online-Shopping-Seiten wie **amazon.de** personalisieren ihre Seiten für dich auf verschiedene Arten:

- **Persönliche Willkommensbotschaft:** Willkommensnachrichten und Seiteninhalte werden speziell für den Nutzer generiert, um das Einkaufserlebnis persönlicher zu gestalten.
- **Gezielte Empfehlungen:** Indem Shops etwas über die Interessen des Kunden erfahren, können sie Produkte vorschlagen, von denen sie glauben, dass sie dem Kunden gefallen werden. Siehe auch spezielle Angebote wie Black Friday.

---

<sup>1</sup> Ja, Autorisierung hat kein h nach dem t. Im Englischen schreibt man jedoch *authorization*.

Empfehlung: [https://www.goetheschule-essen.de/PDF/MB\\_D\\_2\\_Fehler.pdf](https://www.goetheschule-essen.de/PDF/MB_D_2_Fehler.pdf)

- **Dauerhafte Informationen über den Nutzer:** Ehrlich: Wer will schon immer wieder umständliche Adress- und Kreditkartenformulare ausfüllen? Einige Websites speichern daher diese administrativen Details in einer Datenbank. Sobald sie dich identifiziert haben, können sie die Informationen aus der Datenbank verwenden, was das Einkaufserlebnis viel bequemer macht.
- **Session Tracking:** Viele Webseiten wollen dich tracken, während du mit der Seite interagierst (z. B. beim Füllen eines Online-Warenkorbs). Um dies zu tun, benötigen Webseiten eine Möglichkeit, HTTP-Transaktionen von verschiedenen Nutzern zu unterscheiden.
- **Marketing:** Manche Unternehmen wollen auch wissen, welche Webseiten du besuchst, um dich gezielt spammen zu können. Diese Art von Tracking wurde allerdings seit Anfang 2022 sehr erschwert. Siehe Third-Party Cookies weiter unten.

Um nun zwischen den einzelnen HTTP Requests (GET, POST, etc.) den User eindeutig identifizieren zu können, benötigt der Server eine Session und einen Token. Sonst müsstest du dich für jedem Request authentisieren. Was mühsam wäre!

Daher findet gleich zu Beginn der Interaktion mit dem Server eine Authentisierung/Authentifizierung statt und der User bekommt einen Token. Er dient gleichsam als Ausweis. Er wird mit jedem Request an den Server mitgesendet. Damit ist klar, ob ein User authentifiziert ist oder nicht. Ist bei einem HTTP Request der Token nicht dabei, dann ist der User nicht authentifiziert und somit unbekannt.

Häufig enthält der Token nur eine eindeutige ID (etwa eine **Session-ID**).

Die beiden gängigen Arten von Token sind: **Cookies und JSON Web Token**. Über die JWTs gibt es ein eigenes Arbeitsblatt. Während Cookies automatisch vom Browser abgespeichert, respektive mitgesendet werden, musst du das Speichern/Senden eines JWTs selbst managen (kann aber auch ein Vorteil sein, wie du noch sehen wirst).

## Cookies

Die Idee eines Cookies ist einfach: Der Server sendet ein paar Daten an den Browser und dieser speichert sie für einen bestimmten Zeitraum. Danach werden diese Daten automatisch bei jedem Request an den Server mitgeschickt<sup>2</sup>. Es liegt am Server, was das für Daten sind. Limit: 4096 Bytes pro Cookie.

Cookies werden grob in zwei Arten eingeteilt: **Session Cookies** und **persistente (dauerhafte) Cookies**. Ein Session Cookie ist ein temporäres Cookie, das Einstellungen und Präferenzen speichert, während ein Nutzer auf einer Seite navigiert. Das Session Cookie wird gelöscht, wenn der Benutzer den Browser verlässt. Persistente Cookies können länger leben; sie werden auf der Festplatte gespeichert und überleben das Beenden des Browsers und den Neustart des Computers. Dauerhafte Cookies werden oft verwendet, um ein Konfigurationsprofil oder einen Login-Namen für eine Seite zu speichern, die ein Nutzer regelmäßig besucht.

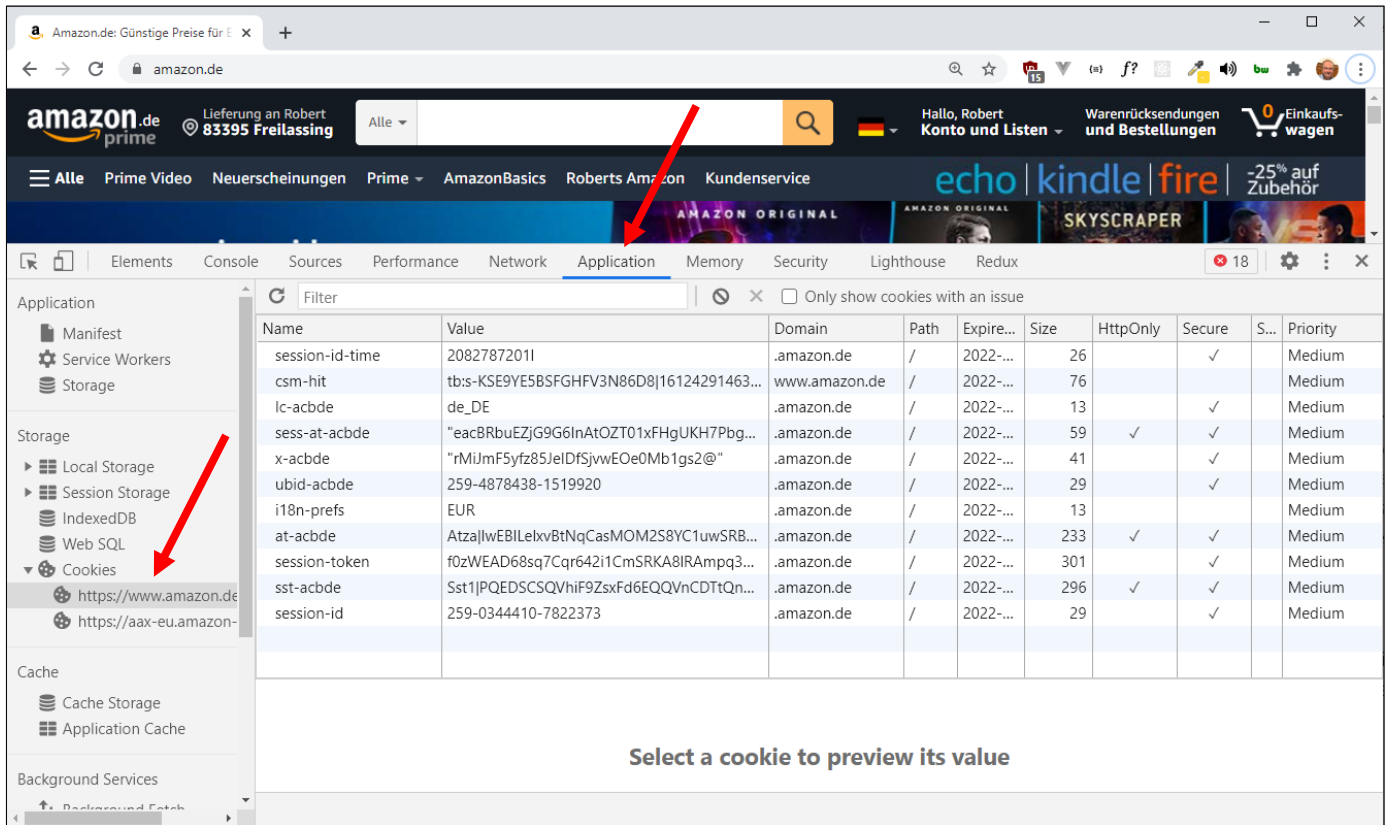
Der einzige Unterschied zwischen Session Cookies und dauerhaften Cookies ist, wann sie ablaufen. Ein Cookie ist automatisch ein Session-Cookie, wenn es keinen **Expires**- (veraltet) oder **Max-Age**-Parameter gibt, der eine längere Ablaufzeit angibt.

Am Beispiel von Amazon siehst du, wie viele Cookies allein von dieser Webseite verwendet werden! Du siehst auch, dass die Cookies domainspezifisch sind. Das bedeutet, dass wenn Cookies vom Server **www.rb.com** gekommen sind, in der Regel nur von dieser Domain gelesen werden können<sup>3</sup>.

---

<sup>2</sup> Es ist in Wirklichkeit komplexer, wie wir noch sehen werden.

<sup>3</sup> Ausnahme: Es wird im Set-Cookie Header eine zusätzliche Domain spezifiziert!



The screenshot shows the Amazon.de homepage in a web browser. The developer tools are open, and the 'Application' tab is selected. On the left sidebar, 'Cookies' is highlighted under the 'Storage' section. A red arrow points to the 'Cookies' entry. The main pane displays a table of cookies for the domain .amazon.de.

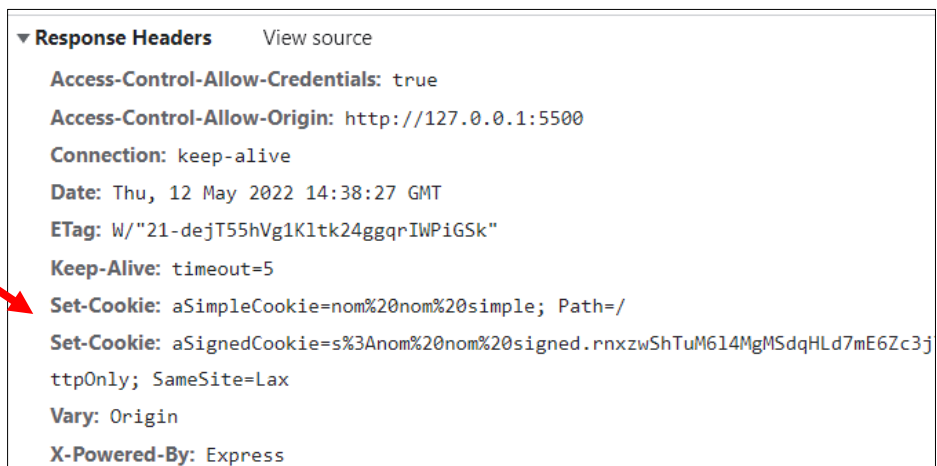
Name	Value	Domain	Path	Expire...	Size	HttpOnly	Secure	S...	Priority
session-id-time	20827872011	.amazon.de	/	2022-...	26		✓		Medium
csm-hit	tb:s-KSE9YE5BSFGHFV3N86D8 16124291463...	www.amazon.de	/	2022-...	76				Medium
lc-acbde	de_DE	.amazon.de	/	2022-...	13		✓		Medium
sess-at-acbde	"eacBRbuEZjG9G6InAtOZT01xFHgUKH7Pbg...	.amazon.de	/	2022-...	59	✓			Medium
x-acbde	"rMjMf5yFz85JelDfSjvwEOe0Mb1gs2@"	.amazon.de	/	2022-...	41		✓		Medium
ubid-acbde	259-4878438-1519920	.amazon.de	/	2022-...	29		✓		Medium
i18n-prefs	EUR	.amazon.de	/	2022-...	13				Medium
at-acbde	AtzajlwEBILelxvBtNqCasMOM2S8YC1uwSRB...	.amazon.de	/	2022-...	233	✓	✓		Medium
session-token	f0zWEAD68sq7Cqr642i1CmSRKA8IRampq3...	.amazon.de	/	2022-...	301				Medium
sst-acbde	Sst1JPQEDSCSQVhiF9ZsxFd6EQQVnCDTtQn...	.amazon.de	/	2022-...	296	✓	✓		Medium
session-id	259-0344410-7822373	.amazon.de	/	2022-...	29		✓		Medium

Below the table, it says: **Select a cookie to preview its value**

## Verwendung von Cookies

Wenn ein Client einen HTTP-Request an einen Server schickt, kann der Server der Antwort ein Cookie mitgeben, das am Client automatisch gespeichert wird. Wie funktioniert das? Der Trick ist ein zusätzlicher HTTP Header: **Set-Cookie**<sup>4</sup>. Wenn der Browser diesen Header sieht, speichert er das Cookie und sendet es mit jedem weiteren Request an den Server (bis es abläuft).

Antwort vom Server  
mit zwei Cookies



The screenshot shows the 'Response Headers' section of a browser's developer tools. It lists several headers, with two 'Set-Cookie' headers highlighted by a red arrow pointing from the text 'Antwort vom Server mit zwei Cookies'.

```

▼ Response Headers    View source
Access-Control-Allow-Credentials: true
Access-Control-Allow-Origin: http://127.0.0.1:5500
Connection: keep-alive
Date: Thu, 12 May 2022 14:38:27 GMT
ETag: W/"21-dejT55hVg1K1tk24ggqrIWPiGsk"
Keep-Alive: timeout=5
Set-Cookie: aSimpleCookie=nom%20nom%20simple; Path=/
Set-Cookie: aSignedCookie=s%3Aanom%20nom%20signed.rnxzwShTuM614MgMSdqHLd7mE6Zc3j
             ttpOnly; SameSite=Lax
Vary: Origin
X-Powered-By: Express
  
```

<sup>4</sup> Siehe: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Set-Cookie>

Ein nachfolgender  
Request an den Server.

▼ **Request Headers** View source

```

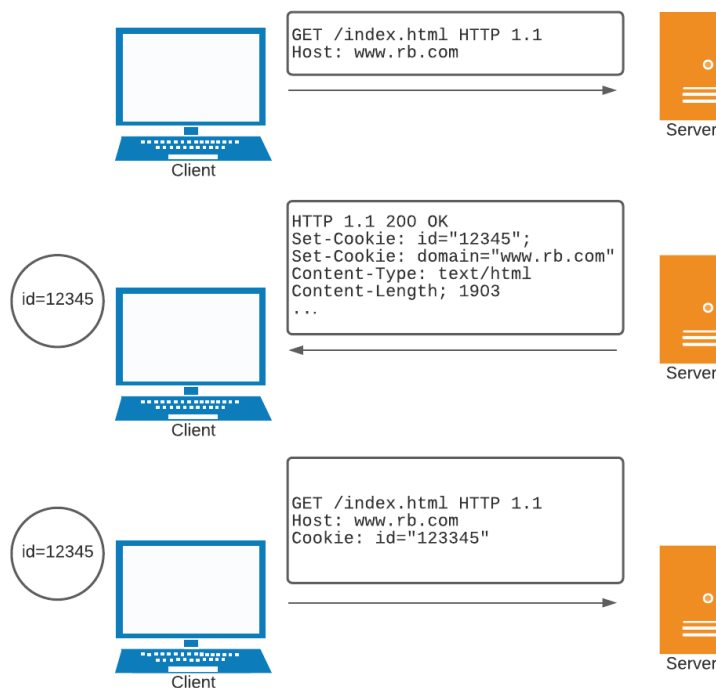
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9,de-DE;q=0.8,de;q=0.7
Connection: keep-alive
Cookie: aSimpleCookie=nom%20nom%20simple; aSignedCookie=s%3A%20nom%20signed.
Host: 127.0.0.1:3000
sec-ch-ua: " Not A;Brand";v="99", "Chromium";v="101", "Google Chrome";v="101"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
  
```

Verschiedene Browser speichern Cookies auf unterschiedliche Weise in unterschiedlichen Plätzen auf deinem Rechner.

Beispiel Chrome unter Windows für User *Phrozen*: C:\Users\Phrozen\AppData\Local\Google\Chrome\User Data\Default

Der Inhalt kann nur über den Browser gelesen/manipuliert werden.

Hier erneut der gesamte Ablauf:



## Sicherheitsrisiken

Eigentlich ist die Bereitstellung einer standardisierten, überprüften Methode zur Authentifizierung und Zugriff auf persönliche Informationen in entfernten Datenbanken mittels Verwendung anonymer Cookies als Schlüssel optimal, da die Häufigkeit der Kommunikation sensibler Daten vom Client zum Server reduziert wird.

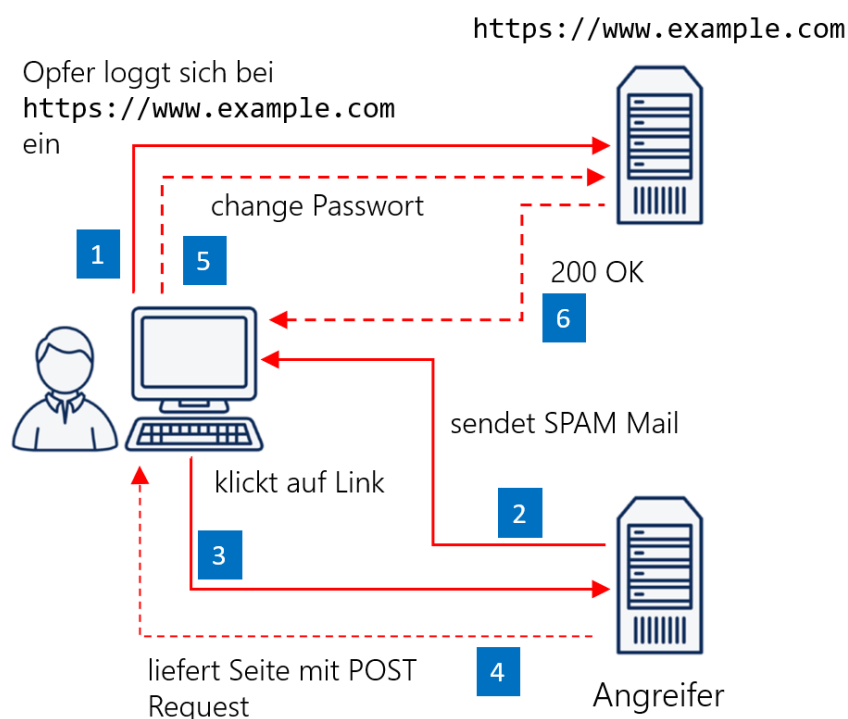
Jedoch hat das automatische Mitsenden von Cookies Nachteile. So kann dieses Verhalten für eine Art von Angriff verwendet werden, der als **Session Riding** oder **CSRF** (Cross-Site Request Forgery) bekannt ist.

Angenommen, die Website <https://www.example.com> erlaubt es authentifizierten Nutzern, ihre Passwörter zu ändern, indem sie via POST Request ein neues Passwort an <https://www.example.com/changepassword> senden. Dazu müssen sie nicht einmal das alte Passwort und den Benutzernamen angeben. Die Seite verwendet das mitgesendete Cookie, um zu überprüfen, ob der Benutzer eingeloggt und der Request gültig ist.

Nehmen wir weiter an, du hast dort einen Account und bist noch immer auf deren Website angemeldet [1] (das Cookie ist drei Monate gültig, du hast vergessen/warst zu faul, dich auszuloggen).

Nun bekommst du über SPAM, Social Media, dubiose Webseite \*hüstel\*, etc. einen Link präsentiert [2], den du anklickst [3]. Dieser führt zu einer Webseite, die, sobald sie in in deinen Browser geladen ist [4], automatisch einen POST Request mit einem neuen, vom Angreifer festgelegten Passwort, an <https://www.example.com/changepassword> auslöst [5].

Da dein Browser das Authentifizierungs-Cookie an den Request anhängt (es ist ja die gleiche Domain wie die, von der das Cookie stammt), wird dein Passwort geändert [6].



## First-Party Cookies und Third-Party Cookies

First-Party-Cookies sind Cookies, die von der besuchten Domain erstellt wurden. Third-Party-Cookies sind Cookies, die von einer anderen als der besuchten Domäne erstellt wurden. Betrachten wir ein Beispiel. Du besuchst [example.com](https://www.example.com) und nutzt ein eingebettetes HTML-Element (z.B. ein `iframe`) von [my.plugin.com](https://www.my.plugin.com).

Alle Cookies von [example.com](https://www.example.com), [static.example.com](https://static.example.com), etc. sind First-Party-Cookies, aber alle Cookies von [my.plugin.com](https://www.my.plugin.com), [plugin.com](https://www.plugin.com) sind Third-Party-Cookies.

Seit Anfang des Jahres 2022 hat Google die Benutzung von Third-Party Cookies eingeschränkt. Diese können nur unter gewissen Umständen (siehe späteres Arbeitsblatt) mit dem HTTP-Request mitgesendet werden.

## Cookie Secret

Um Cookies sicherer zu machen, ist ein geheimes Passwort (Cookie Secret) notwendig. Das Cookie Secret ist ein String, der dem Server bekannt ist und verwendet wird, um die Cookies zu signieren, bevor sie an den Client gesendet werden. Dadurch können Cookies nicht manipuliert werden! Sie sind aber immer noch lesbar. Daher, um ganz sicherzugehen, können Cookies auch verschlüsselt werden. Das Cookie Secret ist kein Passwort, das man sich merken muss.

Robert Baumgartner

SEW 4. Jahrgang

OK, genug Theorie. Los geht's!

## Cookies mit Express(o)

**Aufgabe 1.** Wir starten mit unserem selbst gebauten Basis-Server. Entpacke daher die Datei **START server.zip**.

Speichere dein Cookie Geheimnis gemeinsam mit anderen Konstanten im **.env** File ab!

```
PORT=3000
```

```
COOKIE_SECRET=I love SEW
```

**Aufgabe 2.** Installiere mit **npm** das Package **cookie-parser** und binde das Package in **app.js** ein<sup>5</sup>.

```
const cookieParser = require('cookie-parser');
```

Verwende die Funktion als Middleware (achte dabei auf die Reihenfolge der **app.use** Anweisungen):

```
app.use(cookieParser(process.env.COOKIE_SECRET));
```

Sobald du dies getan hast, kannst du bequem Cookies verwenden.

Beim Erzeugen des Cookies kannst du ein Option-Objekt angeben. Fehlt es, gelten die Defaults!

**Aufgabe 3.** Lege nun die Route **GET /setcookies** an und teste im Browser

```
router.get('/getcookies', (req, res) => {  
  res.cookie('aSimpleCookie', 'nom nom simple'); // einfaches Cookie  
  res.cookie('aSignedCookie', 'nom nom signed', {  
    maxAge: 1000 * 60 * 15, // hält 15 Minuten  
    httpOnly: true, // Zugriff nur vom Webserver, Browser kann es nicht lesen!  
    signed: true, // signiertes Cookie  
  });  
  
  res.send('Hello! You just received cookies!');  
});
```

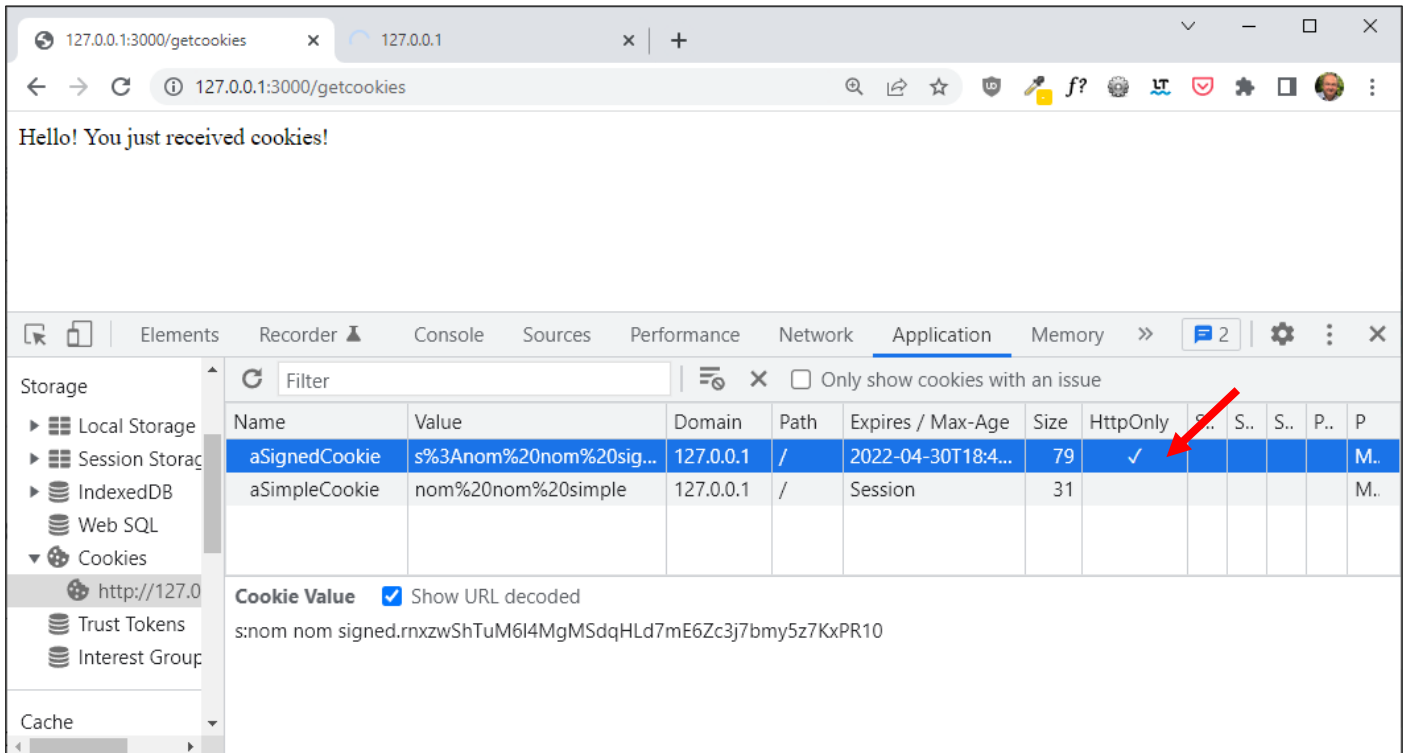
**Anmerkung:** Das 2002 erstmals eingeführte Property **httpOnly** bedeutet, dass du als Anwender das Cookie zwar mit den DevTools des im Browsers lesen kannst, aber der Browser kann das Cookie nicht lesen. Stattdessen kommt ein leerer String zurück. Damit soll eine Art der XSS (Cross Side Script) Attacke verhindert werden.

Überprüfe das Ergebnis!

Siehe nächste Seite!

---

<sup>5</sup> Der **cookie-parser** ist eine Middleware, die unter anderem den HTTP Header parst und automatisch ein Property **cookies** in dem **req** Objekt erzeugt. Sie wird auch benötigt, um signierte Cookies zu verwenden.



Storage

- Local Storage
- Session Storage
- IndexedDB
- Web SQL
- Cookies
  - http://127.0.0.1:3000/
- Trust Tokens
- Interest Group
- Cache

Name	Value	Domain	Path	Expires / Max-Age	Size	HttpOnly	S..	S..	S..	P..	P
aSignedCookie	s%3Anom%20nom%20sig...	127.0.0.1	/	2022-04-30T18:4...	79	✓					M..
aSimpleCookie	nom%20nom%20simple	127.0.0.1	/	Session	31						M..

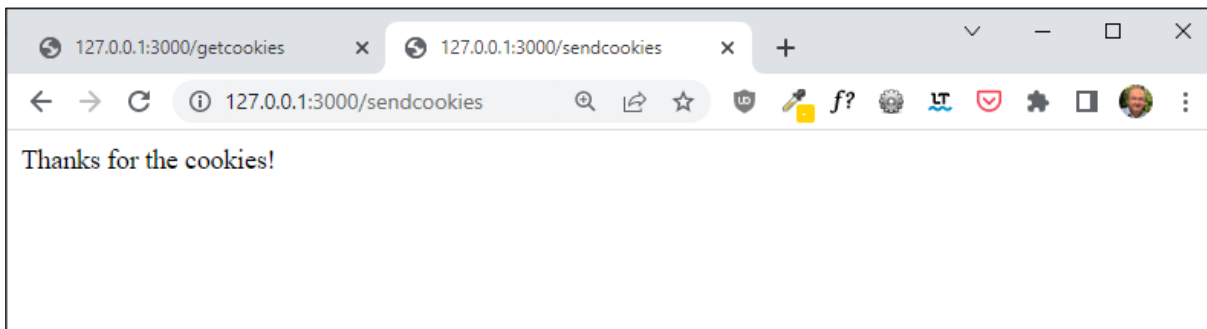
**Cookie Value** ☒ Show URL decoded

s:nom nom signed.rnxzwShTuM6l4MgMSdqHLd7mE6Zc3j7bmy5z7KxPR10

Du siehst, man kann den Inhalt der Cookies lesen, auch wenn sie signiert und httpOnly sind.

Nun werden die Cookies bei jedem Request mitgesendet. Sie stehen als Property in `req.cookies` und `req.signedCookies`.

**Aufgabe 4.** Erstelle eine Route `GET /sendcookies`, welche die Cookies auf der Console am Server ausgibt.



Thanks for the cookies!

```
nom nom simple
nom nom signed
GET /sendcookies 200 8.161 ms - 23
```

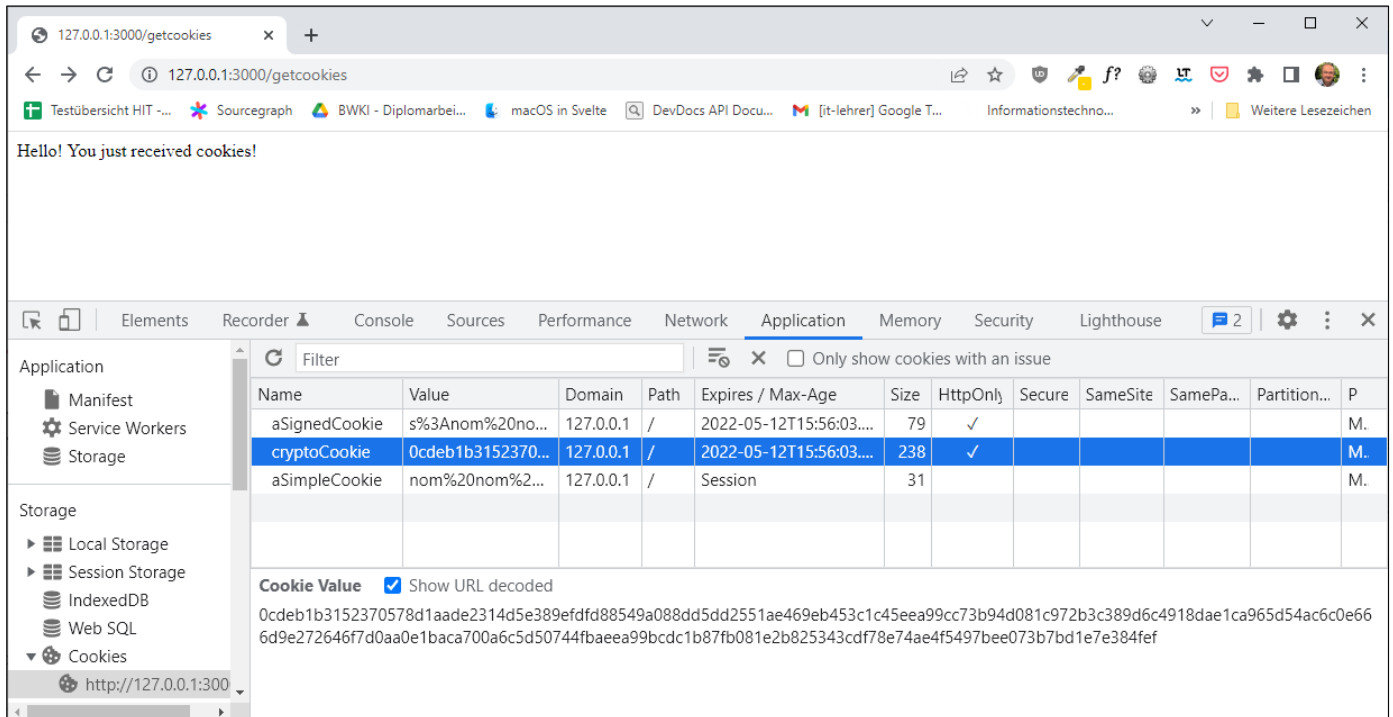
Wir wollen nun den Inhalt verschlüsseln, und zwar mit dem besten Algorithmus, der derzeit bekannt ist: AES 256 Bit.

**Aufgabe 5.** Installiere das Package **cryptr** und erstelle ein drittens Cookie, das verschlüsselt ist. Lies dazu:

<https://www.npmjs.com/package/cryptr>

Verwende für das Secret ebenfalls eine `.env` Konstante:

```
CRYPTR_SECRET=I love INSY
```



Hello! You just received cookies!

Application

- Manifest
- Service Workers
- Storage

Storage

- Local Storage
- Session Storage
- IndexedDB
- Web SQL
- Cookies

Filter

Name	Value	Domain	Path	Expires / Max-Age	Size	HttpOnly	Secure	SameSite	SamePa...	Partition...	P
aSignedCookie	s%3Anom%20no...	127.0.0.1	/	2022-05-12T15:56:03....	79	✓					M.
cryptoCookie	0cdeb1b3152370...	127.0.0.1	/	2022-05-12T15:56:03....	238	✓					M.
aSimpleCookie	nom%20nom%2...	127.0.0.1	/	Session	31						M.

Cookie Value ☒ Show URL decoded

0cdeb1b3152370578d1aade2314d5e389efdfd88549a088dd5dd2551ae469eb453c1c45eea99cc73b94d081c972b3c389d6c4918dae1ca965d54ac6c0e666d9e272646f7d0aa0e1baca700a6c5d50744fbabeea99bcd1b87fb081e2b825343cdf78e74ae4f5497bee073b7bd1e7e384fef

**Aufgabe 6.** Teste das Entschlüsseln!

```
GET /setcookie 304 70.630 ms - -
nom nom simple
nom nom signed
nom nom encrypted
GET /getcookie 304 70.950 ms - -
```

**Aufgabe 7:** Beantworte folgende Fragen:

- Wie kannst du erreichen, dass ein Cookie automatisch nach dem Schließen des Browsers gelöscht wird?
- Was ist CSRF?
- Was sind Thrid-Party Cookies?
- Wie kannst du Cookies signieren?
- Wieso ist das httpOnly Property wichtig?
- Welche Alternative gibt es zu Cookies?