Webseiten mit HTTPS bereitstellen und mit HSTS sichern

Jörg Kastning

https://www.my-it-brain.de

10. März 2018

Inhalt

Inhalt

Inhalt

Inhalt

Inhalt

Ziele von HTTPS

- Authentizität
- Vertraulichkeit
- Integrität

Symmetrische Verschlüsselung

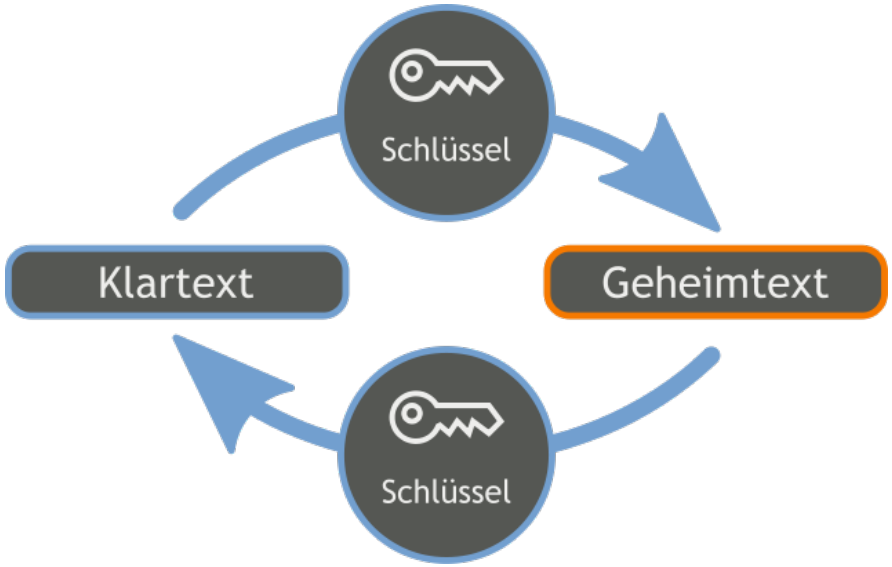


Abbildung: Zur Ver- und Entschlüsselung wird derselbe Schlüssel verwendet

Vor- und Nachteile Symmetrischer Verschlüsselung

Vorteile

Symmetrische Verschlüsselungsverfahren arbeiten in der Regel schneller als asymmetrische Verfahren.

Nachteile

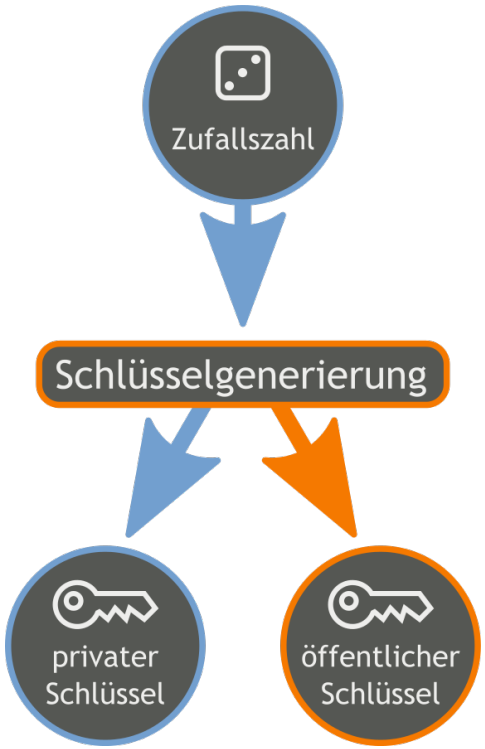
Der gemeinsame Schlüssel muss vor Kommunikationsbeginn allen Kommunikationsteilnehmern mitgeteilt werden. Der Schlüsselaustausch gestaltet sich oftmals sehr problematisch.

Asymmetrische Verschlüsselung

Schlüssel-Generierung

Es wird ein Schlüsselpaar aus großen Zufallszahlen erzeugt.

Während der private Schlüssel geheim zu halten ist, kann der öffentliche Schlüssel an Kommunikationspartner verteilt werden.



Nutzung der Schlüssel zur Ver- und Entschlüsselung

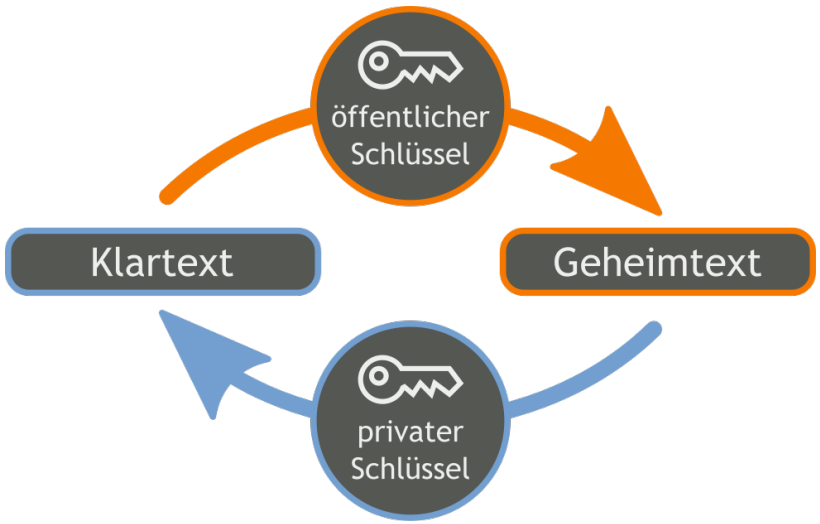


Abbildung: Zur Ver- und Entschlüsselung werden unterschiedliche Schlüssel verwendet

Nutzung der Schlüssel zur Signatur

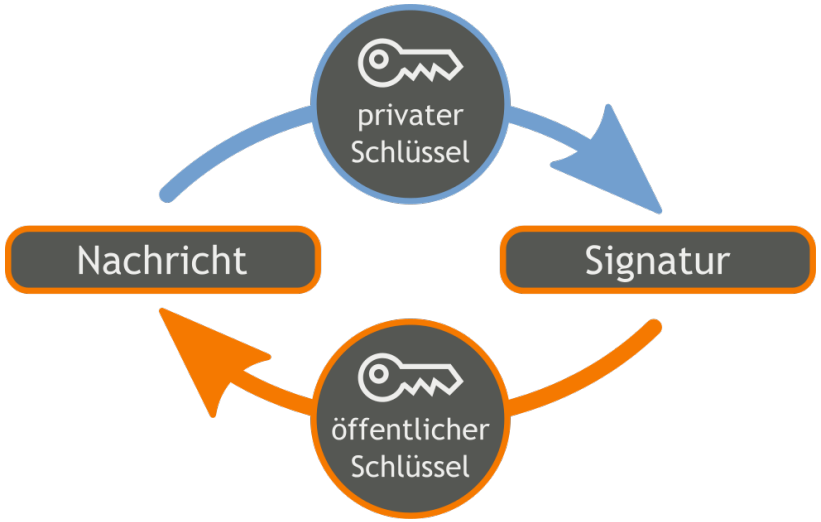


Abbildung: Das Schlüsselpaar ermöglicht Signatur und Signaturprüfung von Nachrichten

Vor- und Nachteile Asymmetrischer Verschlüsselung

Vorteile

Das Problem des Schlüsselaustauschs existiert bei diesen Verfahren nicht. Der öffentliche Schlüssel kann gefahrlos weltweit bekannt gemacht werden.

Nachteile

Asymmetrische Verschlüsselungsverfahren sind wesentlich langsamer als symmetrische Verfahren.

Kombination beider Verfahren bei HTTPS

- Nutzung eines asymmetrischen Verfahrens zum Austausch eines gemeinsamen Sitzungsschlüssels
- Für die weitere Kommunikation wird der gemeinsame Schlüssel verwendet

Der Prozess im Überblick

1. Erzeugung eines privaten Schlüssels
2. Generierung einer Zertifikatsanfrage (CSR)
3. Übermittlung des CSR an eine Zertifizierungsstelle (CA)
4. Ausstellung eines signierten Zertifikats durch die CA
5. Implementierung des privaten Schlüssels und des Zertifikats auf dem Webserver

Erzeugung eines privaten Schlüssels

```
/ tmp$ openssl genrsa -aes256 -out test.key 2048
Generating RSA private key, 2048 bit long modulus
.............................+++
.......................................+++
e is 65537 (0 x10001 )
Enter passphrase for test.key:
Verifying - Enter passphrase for test.key:
```

Achtung

Der private Schlüssel ist stets auf einem vertrauenswürdigen Host zu erzeugen. Er sollte niemals im Web auf den Seiten einer CA generiert werden.

Generierung einer Zertifikatsanfrage (CSR)

```
openssl req -batch -sha256 -new -key test.key -out test.csr \
-subj "/C=DE/L=Musterstadt/O=Musterfirma/OU=Musterabteilung/ \
CN=foo.example.com/emailAddress=foo@example.com"
```

Achtung

Auch dieser Schritt sollte auf einem vertrauenswürdigen Host ausgeführt werden. Der private Schlüssel darf den Host nicht verlassen.

Beispiel einer PKI

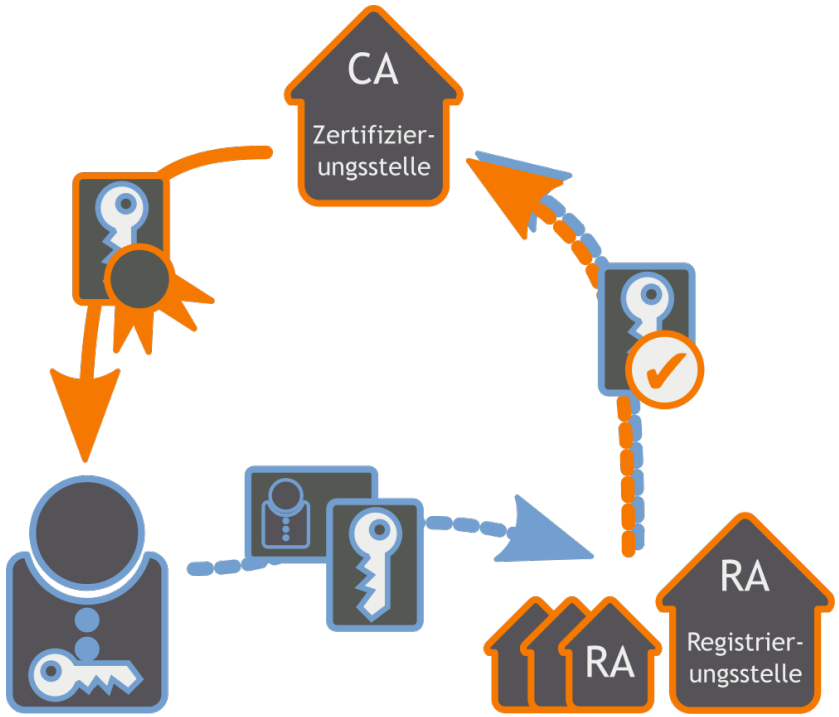


Abbildung: Verarbeitung eines CSR durch RA und CA

Beispiele für Zertifizierungsstellen

- DFN-PKI
- Let's Encrypt
- WoSign
- CAcert
- Symantec
- Hosting-Provider wie z. B. Strato, Hosteurope, 1&1, usw.
- Selbstsignierte Zertifikate (durch eigene CA)

Implementierung auf einem Webserver

- Die Dateien gehören **nicht** ins DocumentRoot

Implementierung auf einem Webserver

- Die Dateien gehören **nicht** ins DocumentRoot
- Zugriffsrechte auf den privaten Schlüssel soweit wie möglich beschränken

Implementierung auf einem Webserver

- Die Dateien gehören **nicht** ins DocumentRoot
- Zugriffsrechte auf den privaten Schlüssel soweit wie möglich beschränken
- Zertifikatskette mitausliefern

Die Zertifikatskette

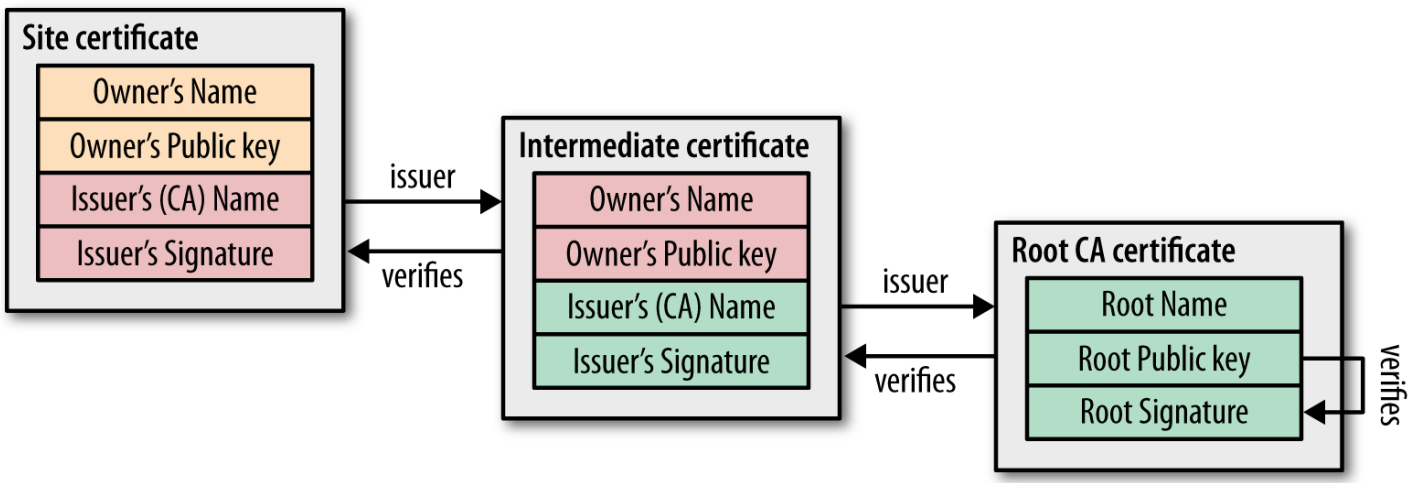


Abbildung: Zertifikatskette vom Leaf Certificate bis zum Root Certificate im Trust Store des Webbrowsers

Konfiguration einer TLS/SSL-Webseite

Ein Beispiel für Apache 2.4

```
LoadModule ssl_module modules/mod_ssl.so

Listen 443
<VirtualHost *:443>
ServerName www.example.com
SSLEngine on
SSLCertificateFile "/path/to/www.example.com.cert"
SSLCertificateChainFile "/usr/local/apache2/conf/ssl.crt/ca.crt"
SSLCertificateKeyFile "/path/to/www.example.com.key"
</VirtualHost>
```

Quelle: `https://httpd.apache.org/docs/2.4/`

Konfiguration einer TLS/SSL-Webseite

Ein Beispiel für NGINX

```
server {
  listen 443 ssl;
  server_name www.example.com
  ssl_certificate /path/to/www.example.com.cert
  ssl_certificate_key /path/to/www.example.com.key
  ...
  }
```

Quelle: `http://nginx.org/en/docs/http/configuring_https_servers.html`

Gefahr durch Man-In-The-Middle-Angriffe

Es ist Vorsicht geboten

Obwohl TLS/SSL-Verschlüsselung implementiert wurde, sind Man-In-The-Middle-Angriffe weiterhin möglich.

Gefahr durch Man-In-The-Middle-Angriffe

Es ist Vorsicht geboten

Obwohl TLS/SSL-Verschlüsselung implementiert wurde, sind Man-In-The-Middle-Angriffe weiterhin möglich.
Denn der Webbrowser weiß nicht, ob TLS/SSL auf einer Webseite implementiert sind oder nicht.

Schwachstelle im Design

- Es gibt hunderte von „vertrauenswürdigen“ CAs, denen Browser und E-Mail-Clients vertrauen
- Jede dieser CAs kann Zertifikate für beliebige Domains ausstellen und die Browser und E-Mail-Clients vertrauen ihnen
- Man-In-The-Middle-Angriffe sind so auch auf HTTPS-Verbindungen möglich

HTTP Strict Transport Security

Welchen Nutzen bietet HSTS?

- Dem Webbrowser wird mitgeteilt, dass eine Seite über HTTPS erreichbar ist
- Die URL wird transparent zu einer `https`-URL umgeschrieben
- Der Webbrowser wird angewiesen, diese Seite ausschließlich über HTTPS aufzurufen

HTTP Strict Transport Security

Wie implementiert man HSTS?

```
<VirtualHost www.example.com:443>
Header always set Strict-Transport-Security "max-age=63072000; \
includeSubdomains;"
</VirtualHost>
```

```
server {
  listen 443 ssl;
  add_header Strict-Transport-Security "max-age=63072000; \
  includeSubdomains; ";
}
```

HTTP Strict Transport Security

Vor- und Nachteile von HSTS

Vorteile

- Leicht zu konfigurieren
- Verhindert MITM-Angriffe auf HTTP-Verbindungen

Nachteile

- Problematisch bei Mixed-Content
- MITM-Angriff mit gefälschtem TLS/SSL-Zertifikat weiterhin möglich

Fazit

Mit HSTS steht ein wirksames Mittel zur Verfügung, um die Angriffsfläche auf HTTPS-Verbindungen zu verkleinern.

Fazit

Mit HSTS steht ein wirksames Mittel zur Verfügung, um die Angriffsfläche auf HTTPS-Verbindungen zu verkleinern.

Darüber hinaus stehen weitere Mittel wie z. B. *Certificate Transparency* (RFC 6962), *Certification Authority Authorization* (RFC 6844) und HTTP Public Key Pinning (RFC 7469) zur Verkleinerung der Angriffsfläche zur Verfügung.

Fazit

Mit HSTS steht ein wirksames Mittel zur Verfügung, um die Angriffsfläche auf HTTPS-Verbindungen zu verkleinern.

Darüber hinaus stehen weitere Mittel wie z. B. *Certificate Transparency* (RFC 6962), *Certification Authority Authorization* (RFC 6844) und HTTP Public Key Pinning (RFC 7469) zur Verkleinerung der Angriffsfläche zur Verfügung.

Weiterführende Informationen findet ihr im TLS-Kochbuch unter: `https://www.my-it-brain.de/wordpress/mein-tls-kochbuch/`