SSL/TLS: Ein Überblick

Wie funktioniert das *sichere* Internet?

Dirk Geschke

Linux User Group Erding

28. März 2012

Gliederung

Allgemeines

- SSL wurde ursprünglich von Netscape entwickelt um Webzugriffe zu verschlüsseln (1994)
- Microsoft hatte eine eigene Lösung: PCT, Private Communication Technology (Ende 1995)
- PCT hat sich nicht durchgesetzt, einige Aspkete davon wurden in SSL 3.0 aufgenommen
- Januar 1999: SSL wird von der IETF als Internetstandard in RFC 2246 veröffentlich: TLS – *Transport Layer Security*
- Heute kann fast jede TCP-Verbindung damit abgesichert werden.

Hashfunktionen

- Allgemein als Prüfsummen bekannt
- Diese stellen keine klassische Verschlüsselung dar, man kann den Ursprungswert nicht mehr ermitteln
- bilden große Datenmenge auf kleinere Zielmenge ab, oft konstante Größe, z.B. 128 Bits bei MD5
- Problem: Es gibt Kollisionen, unterschiedliche Ausgangswerte können gleiche Prüfsumme bilden.
- ganz großes Problem: Vorhersagbarkeit von Kollisionen, als Möglichkeit z.B. eine Datei mit gleicher Prüfsumme zu erstellen.
- klassische Vertreter: MD4, MD5, SHA, . . .

Symmetrische Verschlüsselung

- gleicher Schlüssel zum Ver- und Entschlüsseln von Nachrichten
- es gibt *stream cipher* (z.B. *xor*) und *block cipher*
- klassische Vertreter: DES, 3DES, AES, blowfish, twofish, ...
- größtes Problem: Schlüsseltausch! Beide Seiten müssen den gleichen Schlüssel besitzen
- Beispiel: xor-Verschlüsselung von LUG:
  ASCII: L=76, U=85, G=71

Beispiel: xor-Verschlüsselung

Dezimal	76	85	71
Binär	01001100	01010101	01000111
xor-Key	10100110	10111001	01011010
Ergebnis	11101010	11101100	00011101
Dezimal	234	236	29
Binär	11101010	11101100	00011101
xor-Key	10100110	10111001	01011010
Ergebnis	01001100	01010101	01000111

Problem: Key muss zufällig sein und mindestens so lang wie der zu verschlüsselnde Text!

Asymmetrische Verschlüsselung, public-key Verfahren

- unterschiedliche Schlüssel: ein Schlüssel zum Verschlüsseln und ein anderer zum Entschlüsseln
- Vorteil: Ein Schlüssel kann öffentlich bekannt sein (*public-key*)
- Nachteil: Im Vergleich zur symmetrischen Verschlüsselung sehr langsam
- klassiche Vertreter: DH, RSA, DSA, ...
- Beispiel: Diffie-Hellman

Beispiel: Diffie-Hellman–Schlüsseltausch

- Öffentlicher Schlüssel: p und g
- Alice hat den geheimen Schlüssel a, Bob hat b erstellt
- Alice berechnet: $A = g^a \mod p$
- Bob berechnet: $B = g^b \mod p$
- A und B werden öffentlich ausgetauscht: bekannt sind A,B,p,g
- Das *shared secret* ist nun $s = B^a \mod p$ und $s = A^b \mod p$
- Denn es gilt:
  $B^a \mod p = (g^b)^a \mod p = g^{ab} \mod p = (g^a)^b \mod p = A^b \mod p$
- Hybride Verfahren: Schlüsseltausch für symmetrische Verschlüsselung per Diffie-Hellman

RSA

- RSA ist eine Variante:

$$c = m^e \mod (n)$$

$$m = c^d \mod (n)$$

- Dabei ist m die Nachricht,
- c ist der *cipher*
- *public keys* sind n und e
- *private key* ist d

Praxis RSA

- Verschlüsselung
  1. Bob übermittelt *n* und *e* (öffentlich) an Alice.
  2. Alice nimmt *n* und *e* und berechnet aus *m* den cipher *c*.
  3. Der cipher *c* wird an Bob übermittelt
  4. Bob kann mittels dem geheimen *d* die Nachricht entschlüsseln.
- Digitale Signatur läuft umgekehrt
  1. Bob ermittelt die Prüfsumme und verschlüsselt diese mit dem *private key*
  2. Die Daten und die verschlüsselte Prüfsumme werden an Alice übermittelt
  3. Alice berechnet ebenfalls die Prüfsumme der Daten nach dem gleichen Verfahren
  4. Alice entschlüsselt mit dem *public key* die Prüfsumme von Bob und vergleicht diese mit der eigenen.

$\Longrightarrow$ gleiche Prüfsumme heißt unveränderte Daten von Bob!

Allgemeines

- SSL kann verwendet werden um TCP-Dienste abzusichern, z.B. HTTPS, POPS, IMAPS, FTPS, SMTPS, ...
- Teilweise Upgrade in der Verbindung möglich, z.B. SMTP mit STARTTLS
- Verwendung von SSL-Tunneln möglich: `stunnel`
- Weitere Akteure:
  1. *Alice* als z.B. Webnutzerin
  2. *Bob* als z.B. der Webserver
  3. *Charly* als Bösewicht
  4. *Vera* als Vertrauensperson, CA, z.B. *Verisign*

klassisches Vorgehen

1. Bob sendet den *public key* PB_B an Alice
2. Alice verschlüsselt damit die Nachricht (z.B. *shared key* für hybrides Verfahren)
3. Alice sendet diese verschlüsselte Nachricht an Bob zurück
4. Bob entschlüsselt diese mit seinem *private key* PR_B.
5. Eine symmetrisch verschlüsselte Verbindung könnten nun aufgebaut werden.

Frage: Wie wird er *public key* übermittelt?

Man-in-the-Middle

Charlie sitzt dabei *irgendwie* zwischen Alice und Bob

1. Bob sendet den *public key* PB_B an Charlie statt Alice
2. Charlie sendet seinen *public key* PB_C an Alice und behauptet Bob zu sein.
3. Alice verchlüsselt die Nachricht mit Charlys *public key* PB_C
4. Sie sendet das Ergebnis an Charly statt Bob!
5. Charly entschlüsselt die Nachricht mit seinem *private key* PR_C
6. Charly kann die Nachricht ändern oder nur mitlesen. Diese wird mit Bobs *public key* PB_B verschlüsselt.
7. Charly sendet die verschlüsselte Nachricht an Bob, der kann sie problemlos entschlüsseln

Woran merken Alice und Bob, dass hier etwas nicht stimmt?

Wie überträgt man Bobs *public key* sicher zu Alice?

Lösung: *trusted 3rd Party*, eine außenstehende Vertrauensperson: Vera

- Vera hat einen eigenen *public/private key*
- Vera hat ihren *public key* öffentlich verbreitet (z.B. im Browser integriert)
- Bob sendet nun seinen *public key* zu Vera (*Certificate Signing Request*, .crt)
- Vera verifziert, dass Bob auch wirklich Bob ist (Wie!?!)
- Vera signiert nun Bobs *public key* PB_B mit ihrem *private key* PR_V.
- Vera sendet diese Digitale Signatur als Zertifikat zurück an Bob

Wie sieht das nun in der Praxis aus?

- Alice fragt nach Bobs Zertifikat, dieses enthält nun Bobs *public key* und eine digitale Signatur von Vera
- Alice überprüft die digitale Signatur mit Veras *public key*
- Wenn die Signatur stimmt und Alice Vera vertraut(!!!), dann kann sie davon ausgehen, dass das Zertifikat von Bob stammt.
- Alice kann Bobs *public key* PB_B aus dem Zertifikat entnehmen und kann wie bisher mit Bob kommunizieren.
- Charly hat hier keine Chance!

WICHTIG: Sicherheit von Veras *private key*!

That's all!

Einige Punkte die noch zu beachten sind

- Was ist, wenn Charly auch ein Zertifikat hat?
- Im Zertifikat steht der *Common Name*: `CN`, gewöhnlich DNS-Name
- Extrem wichtig dabei: Man muss Vera vertrauen und sicher sein, dass man ihren *public key* besitzt
- Man kann damit den Server identifizieren (Zertifikat) und eine verschlüsselte Verbindung aufbauen.
- Damit wird nur der Server authentisiert, es gibt aber auch Client-Zertifikate: Damit kann ein Server den Client identifizieren. Bob kann so sicherstellen auch wirkich mit Alice zu reden!
- Henne-Ei-Problem: Erst wird die Verschlüsselung aufgebaut und dann erst der (virtuelle) Server festgelegt, d.h. pro IP-Adresse und Port nur ein https-Server.
- Lösung SNI: *Server Name Indication* nicht verbreitet!

Einige Punkte die noch zu beachten sind

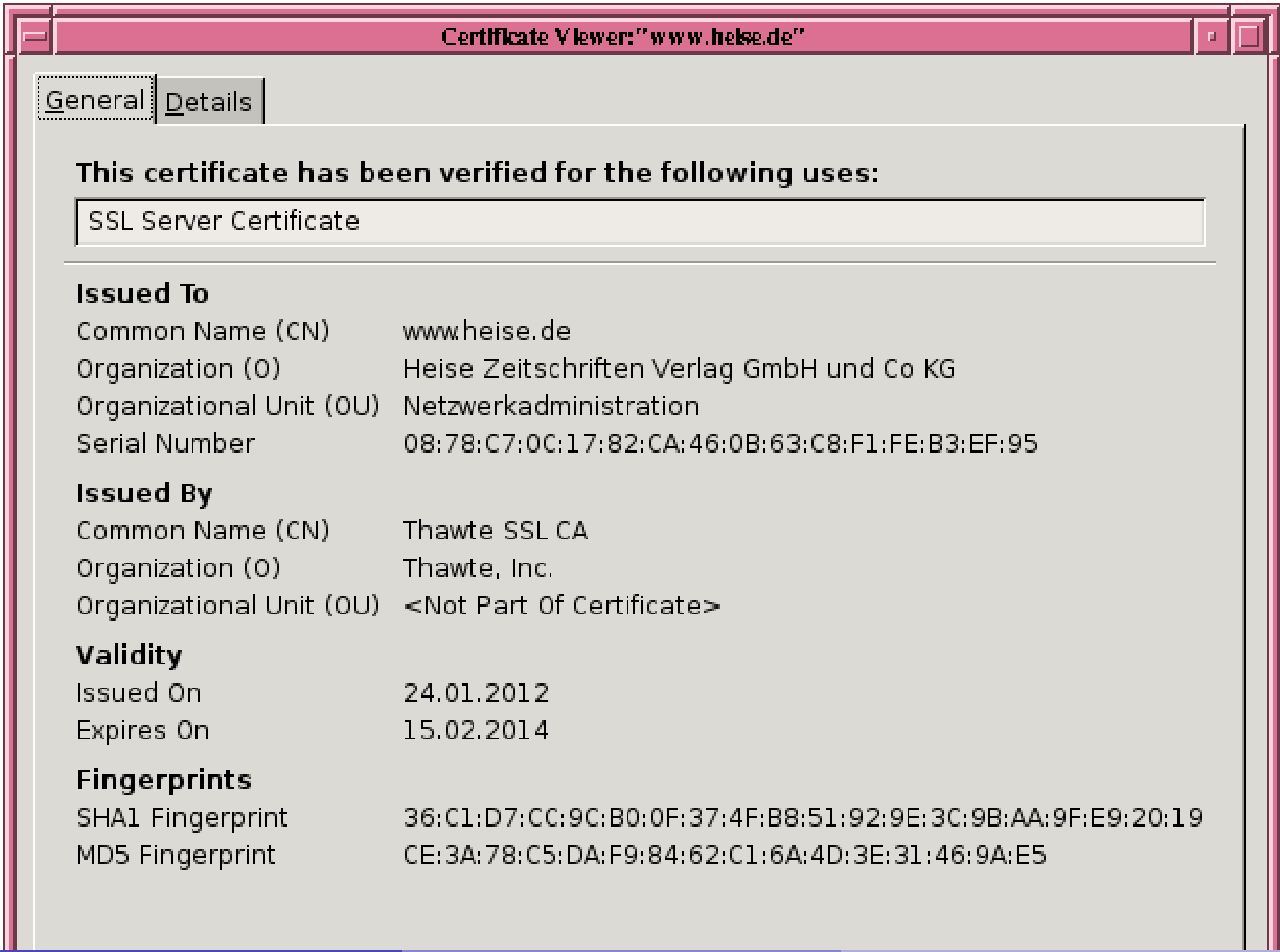
- Zertifikat enthält noch mehr, z.B. eine Lebensdauer
- Prüfung des Zertifikats übernimmt der Browser mit Hilfe der URL und den eingebauten Root-Zertifikaten
- Es gibt *Certificate Revocation Lists*, die aber kaum jemand nutzt
- Es gibt OSCP: *Online Certificate Status Protocol*, die URLs dazu sind im Zertifikat
- Problem: Was passiert wenn der OSCP-Server *down* oder *überbeschäftigt* ist und ein *Try later* sendet?
  - OSCP prüft nur auf Sperrstatus, nicht auf Gültigkeit!
  - Viele Clients sehen bei keiner negativen Antwort, z.b. *Try later* das Zertifikat als gültig an.
  - Chrome hat OSCP daher ausgebaut, sie setzen nun auf *CRL*-Dateien. Diese werden mit dem Browser und Updates ausgeliefert.

Einige Punkte die noch zu beachten sind

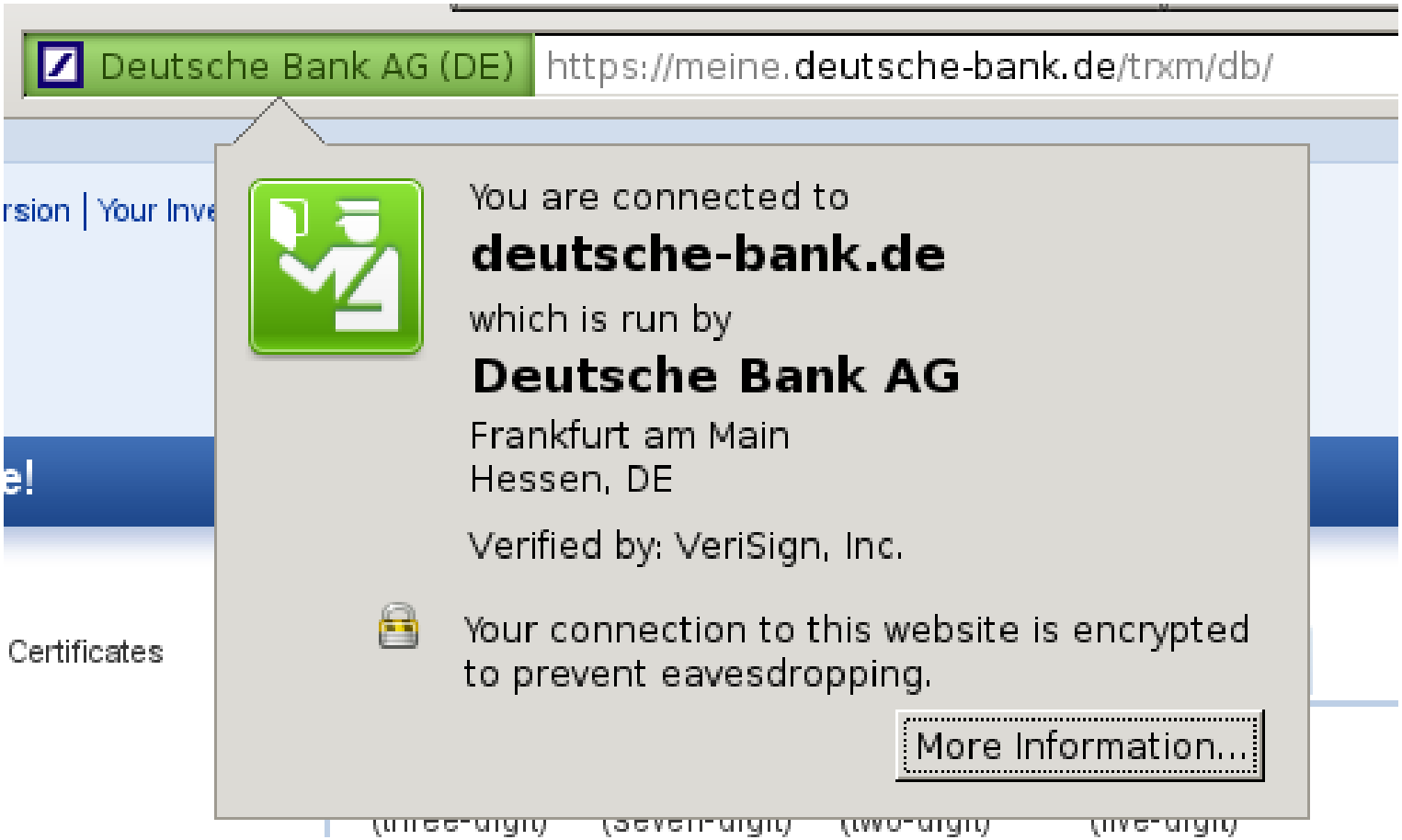
- Man kann auch *self-signed* Zertifikate verwenden, sogenanntes *snake-oil*
- Man kann die eigene CA im Browser einbinden!
- Es ist möglich, dass man unbedacht falsche Root-CAs einspielt!
- Aufpassen bei Typos!:
  - `www.goog1e.de`
  - `www.amazone.com`
  - `www.he1se.de`
  - `www.gooogle.com`
  - `wwww.lug-erdinger.de`
- Es gibt auch Intermediate-Zertifikate: Zertifikatsketten

- Auf URL achten! Tippfehler passieren schnell, Sonderzeichen aus anderen Sprachen.
- Keine anderen Root-CAs unbedacht einspielen (Bildschirm sperren!)
- Warnungen des Browsers beachten! Nicht blind ignorieren!
  - Passt der CN-Name zum DNS-Name?
  - Ist das Zertifikat noch gültig?
  - Wer hat es signiert?
- Produktupdates können falsche Root-CAs mitbringen! Quelle beachten!

Sicherer Webserver

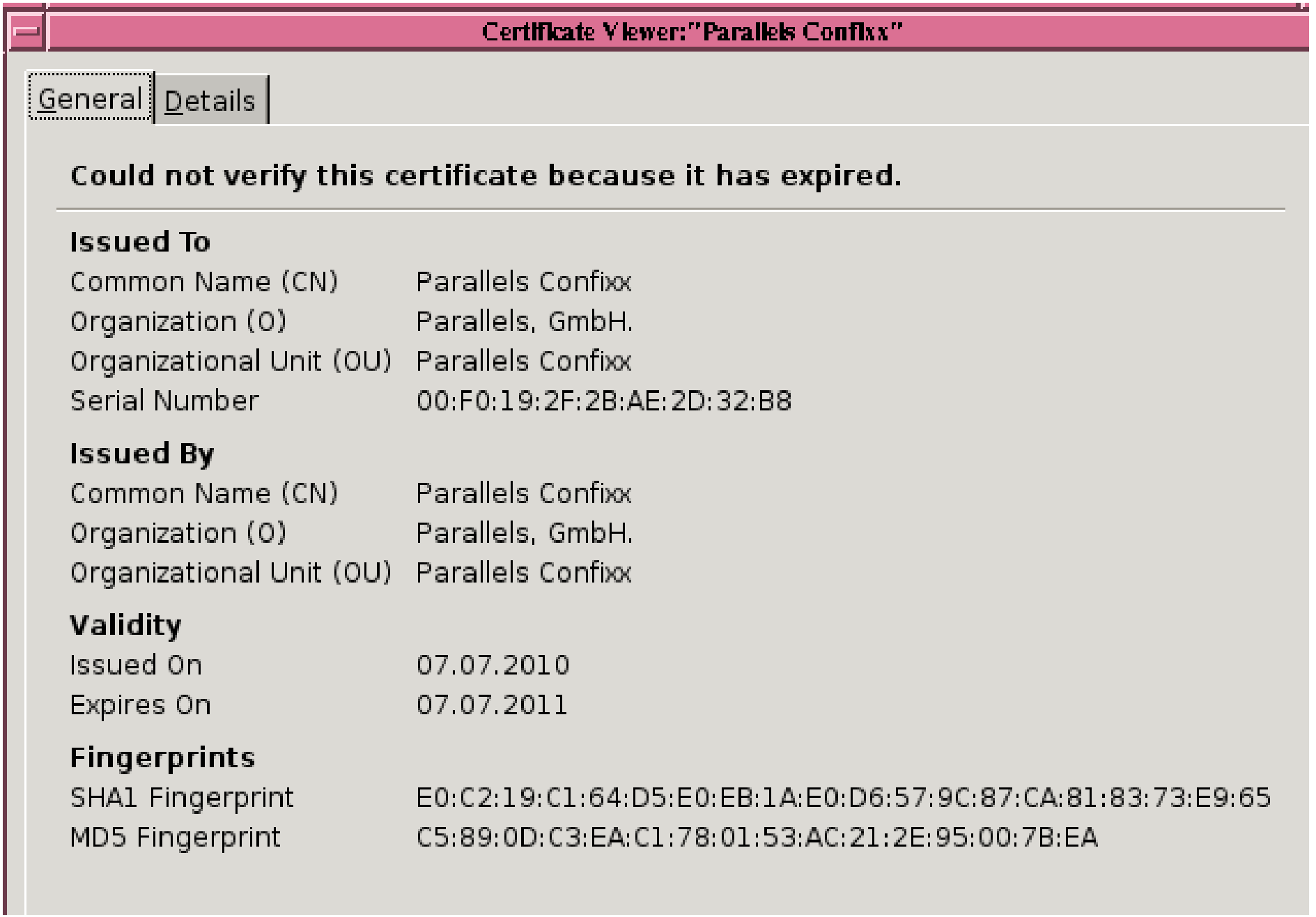

Certificate Viewer:"www.heise.de"
General
Details
This certificate has been verified for the following uses:
SSL Server Certificate
Issued To
Common Name (CN) www.heise.de
Organization (O) Heise Zeitschriften Verlag GmbH und Co KG
Organizational Unit (OU) Netzwerkadministration
Serial Number 08:78:C7:0C:17:82:CA:46:0B:63:C8:F1:FE:B3:EF:95
Issued By
Common Name (CN) Thawte SSL CA
Organization (O) Thawte, Inc.
Organizational Unit (OU) <Not Part Of Certificate>
Validity
Issued On 24.01.2012
Expires On 15.02.2014
Fingerprints
SHA1 Fingerprint 36:C1:D7:CC:9C:B0:0F:37:4F:B8:51:92:9E:3C:9B:AA:9F:E9:20:19
MD5 Fingerprint CE:3A:78:C5:DA:F9:84:62:C1:6A:4D:3E:31:46:9A:E5

Extasicherer Webserver: EV-Zertifikat




Certificate Viewer:"meine.deutsche-bank.de"
General
Details
This certificate has been verified for the following uses:
SSL Server Certificate
Issued To
Common Name (CN) meine.deutsche-bank.de
Organization (O) Deutsche Bank AG
Organizational Unit (OU) <Not Part Of Certificate>
Serial Number 14:A3:4C:D2:AA:2A:6F:53:AB:72:41:46:48:6C:3F:6D
Issued By
Common Name (CN) VeriSign Class 3 Extended Validation SSL SGC CA
Organization (O) VeriSign, Inc.
Organizational Unit (OU) VeriSign Trust Network
Validity
Issued On 28.09.2011
Expires On 18.10.2012
Fingerprints
SHA1 Fingerprint 6C:F2:11:14:DC:4F:77:95:1A:67:63:E1:64:2B:AE:2F:DF:33:EB:BC
MD5 Fingerprint AE:4E:CE:CD:E2:B4:F5:54:E8:5D:CA:22:CC:9E:EB:F0

Untrusted Webserver


Certificate Viewer:"Parallels Conflxx"
General
Details
Could not verify this certificate because it has expired.
Issued To
Common Name (CN) Parallels Confixx
Organization (O) Parallels, GmbH.
Organizational Unit (OU) Parallels Confixx
Serial Number 00:F0:19:2F:2B:AE:2D:32:B8
Issued By
Common Name (CN) Parallels Confixx
Organization (O) Parallels, GmbH.
Organizational Unit (OU) Parallels Confixx
Validity
Issued On 07.07.2010
Expires On 07.07.2011
Fingerprints
SHA1 Fingerprint E0:C2:19:C1:64:D5:E0:EB:1A:E0:D6:57:9C:87:CA:81:83:73:E9:65
MD5 Fingerprint C5:89:0D:C3:EA:C1:78:01:53:AC:21:2E:95:00:7B:EA

Das war es schon!