SIGNED FILE TRANSFER

with ATECC508A

Team Grün:

Peter Lang, Harald Bogesch,

Felix Meißner, Christian Remmele

06.02.2019

Inhalt

- Aufgabenstellung
  - Projektszenario
  - Prinzipdarstellung
- Revolution Pi
- ATECC-Security-Chip
- Demonstrator Aufbau
  - Hardware
  - Software Module
    - ECDSA-Signatur
    - Graphischen Benutzeroberfläche
    - JPEG-Metadaten
    - TCP Verbindung
- Rückblick und mögliche Erweiterung
- Demonstration

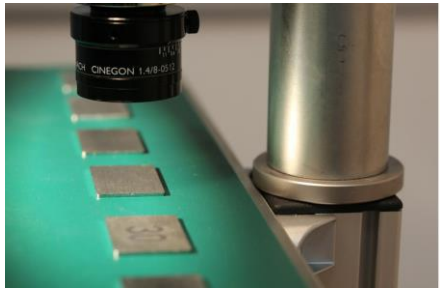
Aufgabenstellung

Aufgabenstellung

- Sichtung und Einarbeitung der Funktionsweiße des Revolution Pi
- Hardware Aufbau des Demonstrators
- Grundlegende Kommunikation zwischen Revolution Pi und ATECC508A-Security-Chip
- TCP-Verbindung zwischen zwei Revolution Pi
- Signieren und Verifizieren von Bildern mit ATECC508A
- Integritäts Geschütze Kommunikation zwischen zwei Revolution Pi
- Erstellung einer graphischen Benutzeroberfläche

Projektszenario

Produktionsanlage



Manipulation

Anlagen-
Überwachung

Bild + Messdaten

Bild + Messdaten

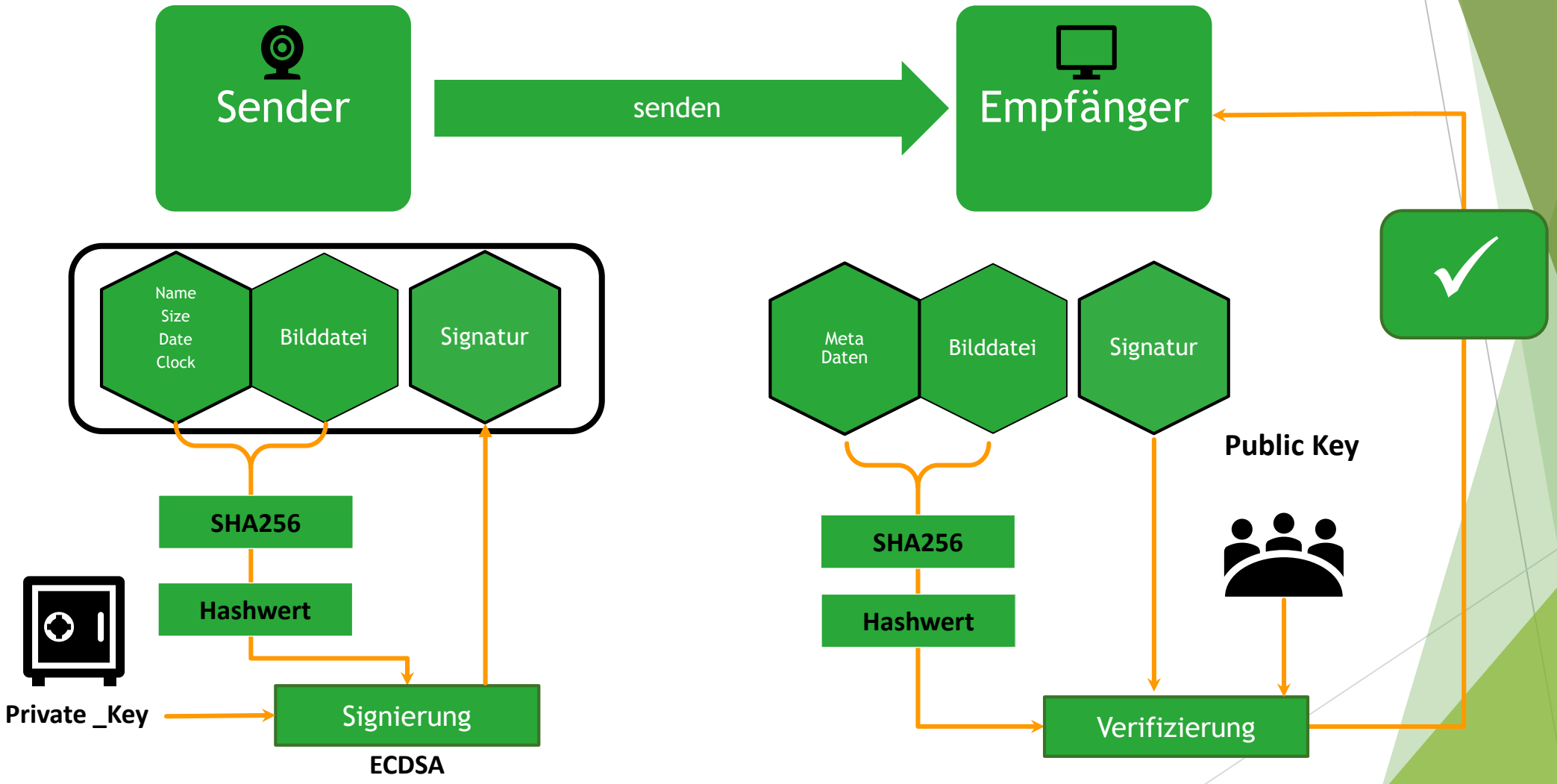
TCP

TCP

Signierung

Verifizierung

Prinzipdarstellung

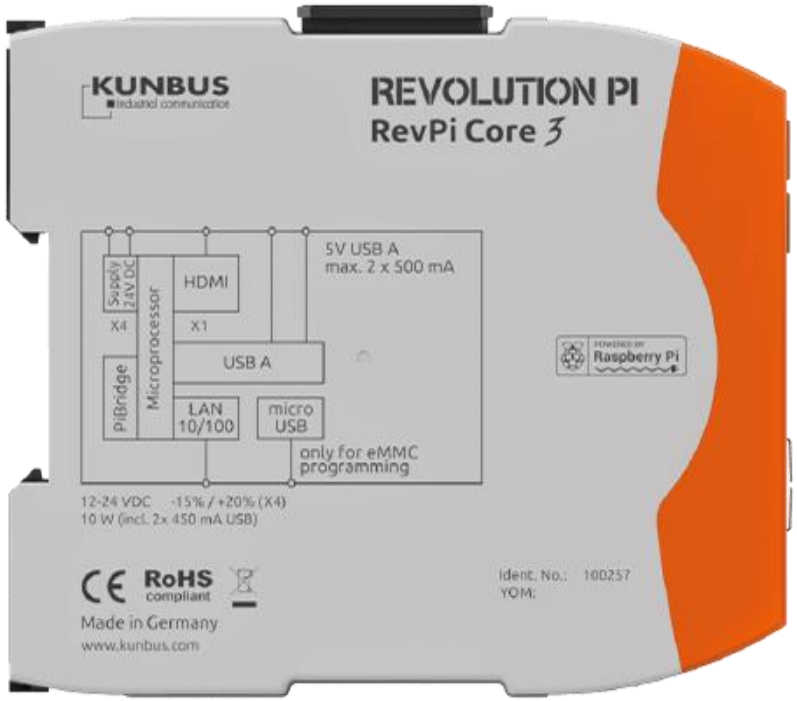


Schutzziele und Sicherheit

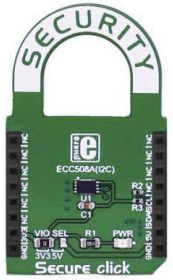
- Integrität
  - Daten können nicht verändert werden
- Authentizität
  - Objekt stammt von tatsächlich von einer spezifischen Identität
- Signierung mit ECDSA
  - Signierung mit geheimen Schlüssel
  - ca. 128 bit Sicherheit
  - Verifizierung mit öffentlichem Schüssel
- SHA-256 Hash-Funktion
  - ca. 128-bit Sicherheit
- **Durchgehende 128-bit Sicherheit**

Revolution Pi

- Merkmale
  - Industrietauglich
  - Betriebstemperatur: -40°C…+55°C
  - Stromversorgung: min 10,7 V, max. 28,8 V
  - Leistungsaufnahme: max. 10 W
  - Schnittstellen:
    2 x USB 2.0, 1 x Micro-USB, 1 x Micro HDMI, 1 x RJ45
  - On bord Security Chip: ATECC508A

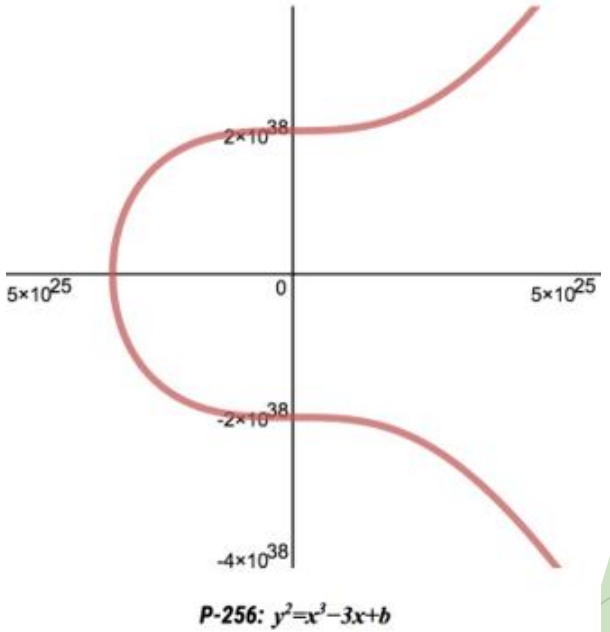


ATECC-Security-Chip



- Kryptographischer Co-Prozessor mit sicherer hardwarebasierter Schlüsselspeicherung
- Optimiert für Asymmetrische Krypto-Operationen:
  - ECDSA: Elliptic Curve Digital Signatur Algorithm
  - ECDH: Elliptic Curve Diffie-Hellman Algorithm
  - NIST Standard P256 Elliptic Curve Unterstützung (made by NSA)
- SHA-256-Hash-Algorithmus mit HMAC-Option
- Speicher für bis zu 16 Keys (je 256-bit)
- I^2C-Bus Kommunikation

- Physikalische Sicherheitsfunktionen zum Schutz vor unbefugtem Zugriff:
  - Active Shield Schaltung
  - Interne Speicher-Verschlüsselung
  - Spannungs-Manipulationserkennung



Byte - Position	Byte value	Bit in Byte	Bit - Nr.	Bit Name	Bit value	Byte Value	Bit value
0 - 12							
13							
14							
15							
16							
17	00						0
18	55						
19							
20 - 51							
52 - 59							
60 - 67							
68 - 83							
84	00						
85	00						
86	55						
87	55						
88 - 89							
90 - 91							
92 - 95							

Chip Konfiguration

Chip Konfiguration

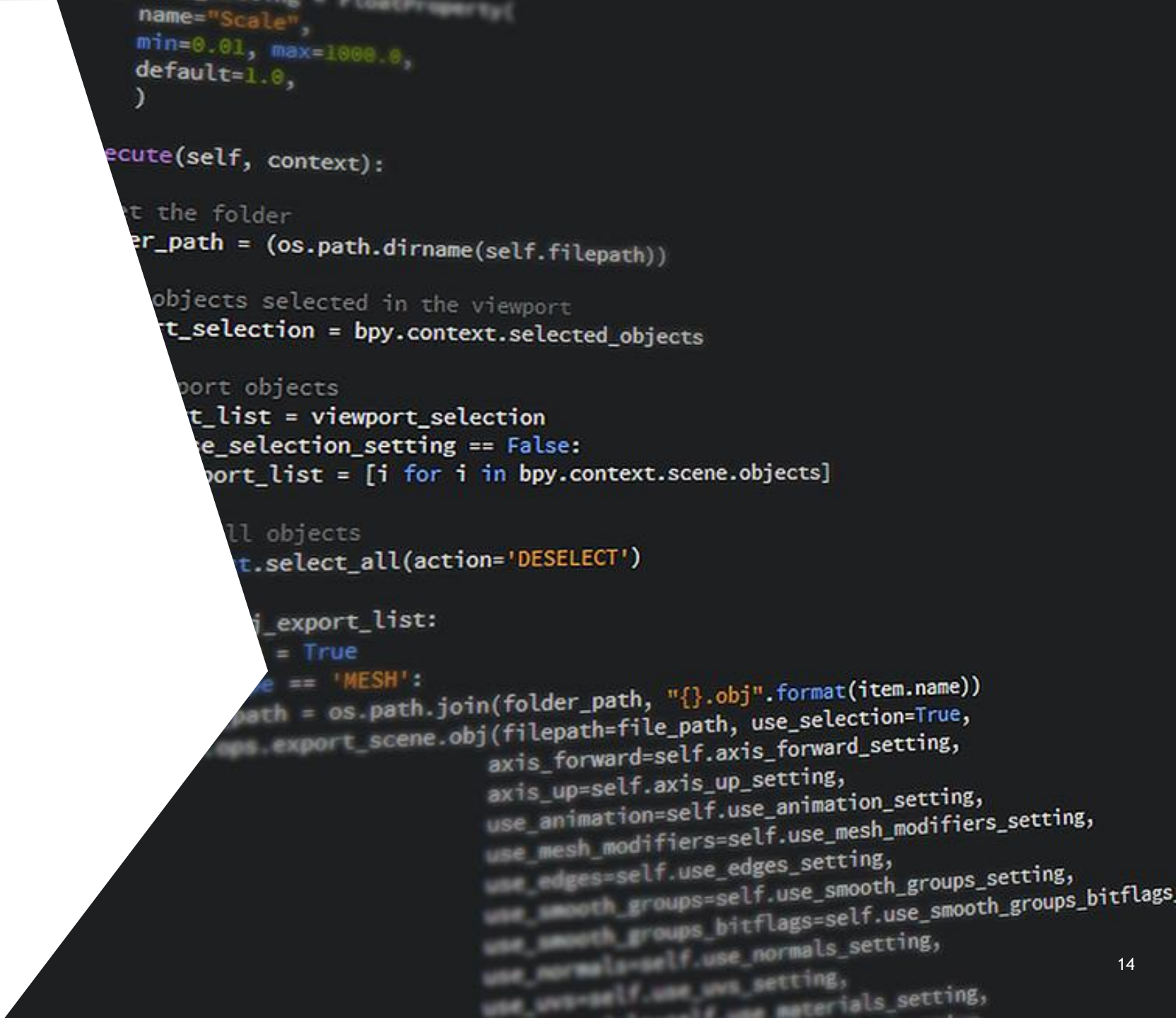
Name		Byte - Position	Byte value	Bit in Byte	Bit - Nr.	Bit Name	Bit value	Ausführung aktueller Konfiguration
SlotConfig								
Slot 0	Slot 0	20	8F	10001111	0	ReadKey	1	externe Signatu Externe Signaturen von wilkürlichen Nachrichten aktiviert
					1		1	Interne Signatur von Nachrichten mithilfe von "GenDig" oder "GenKey" aktiviert
					2		1	ECDH operationen für diesen key erlaubt
					3		1	
					4	NoMac	0	Der Schlüssel in diesem Slot kann von allen Befehlen benutzt werden
					5	LimitedUse	0	keine einschränkung des Verwendungszwecks
					6	EncryptRead	0	Lesebefehle sind erlaubt
					7	IsSecret	1	Der Inhalt dieses Slots ist geheim da in "*WriteConfig*" ein anderer Wert als Always steht
		21	20	00100000	8	WriteKey	0	Es können 4 oder 32 Byte große Daten in den Slot geschrienen werden ist auf Alwys gesetzt (siehe Tabellenbedeutung Table 2-8)
					9		0	
					10		0	
					11		0	
					12	WriteConfig	0	in dieser Bit Config kann mit GenKey, Zufallsschlüssel in diesen Slot geschrieben werden. (siehe Tabellenbedeutung Table 2-9)
					13		1	
					14		0	
					15		0	
		20 - 21						

Chip Konfiguration

```
config = bytearray.fromhex(

    'C0 00 55 00 8F 20 C4 44 87 20 87 20 8F 0F C4 36'
    '9F 0F 82 20 0F 0F C4 44 0F 0F 0F 0F 0F 0F 0F 0F'
    '0F 0F 0F 0F FF FF FF FF 00 00 00 00 FF FF FF FF'
    '00 00 00 00 FF FF FF FF FF FF FF FF FF FF FF FF'
    'FF FF FF FF 00 00 55 55 FF FF 00 00 00 00 00 00'
    '33 00 1C 00 13 00 13 00 7C 00 1C 00 3C 00 33 00'
    '3C 00 3C 00 3C 00 30 00 3C 00 3C 00 3C 00 30 00')


atcab_write_bytes_zone(0, 0, 16, config, len(config))
```

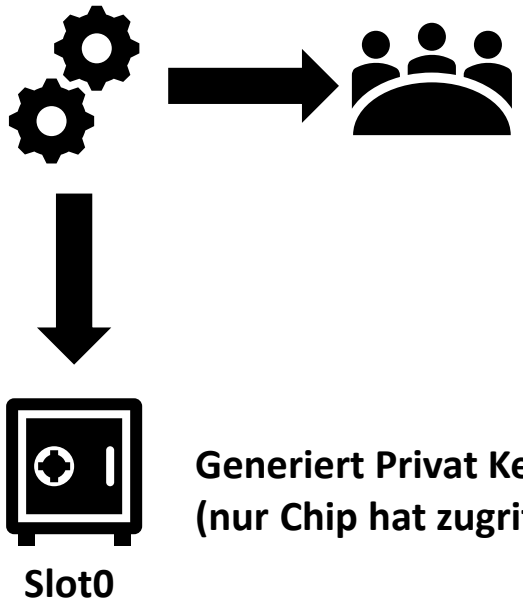
Demonstrator Aufbau

Software Module



ECDSA-Schlüssel

Atcab_genkey(slot)



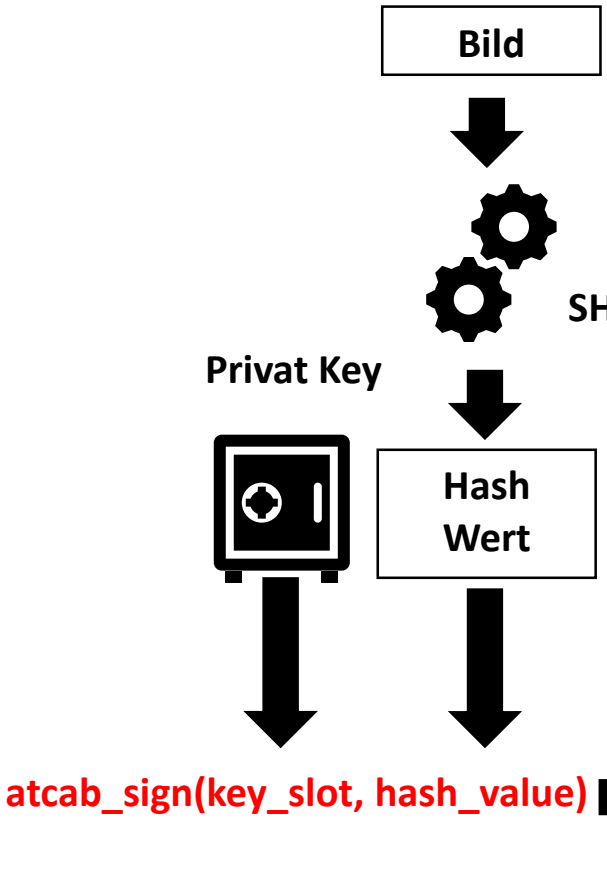
Gibt Public Key aus

Generiert Privat Key
(nur Chip hat zugriff)

Slot0

```
self.key_slot = 0
self.pub_key = bytearray(64)
assert ATCA_SUCCESS == atcab_genkey(self.key_slot, self.pub_key)
```

ECDSA-Signatur

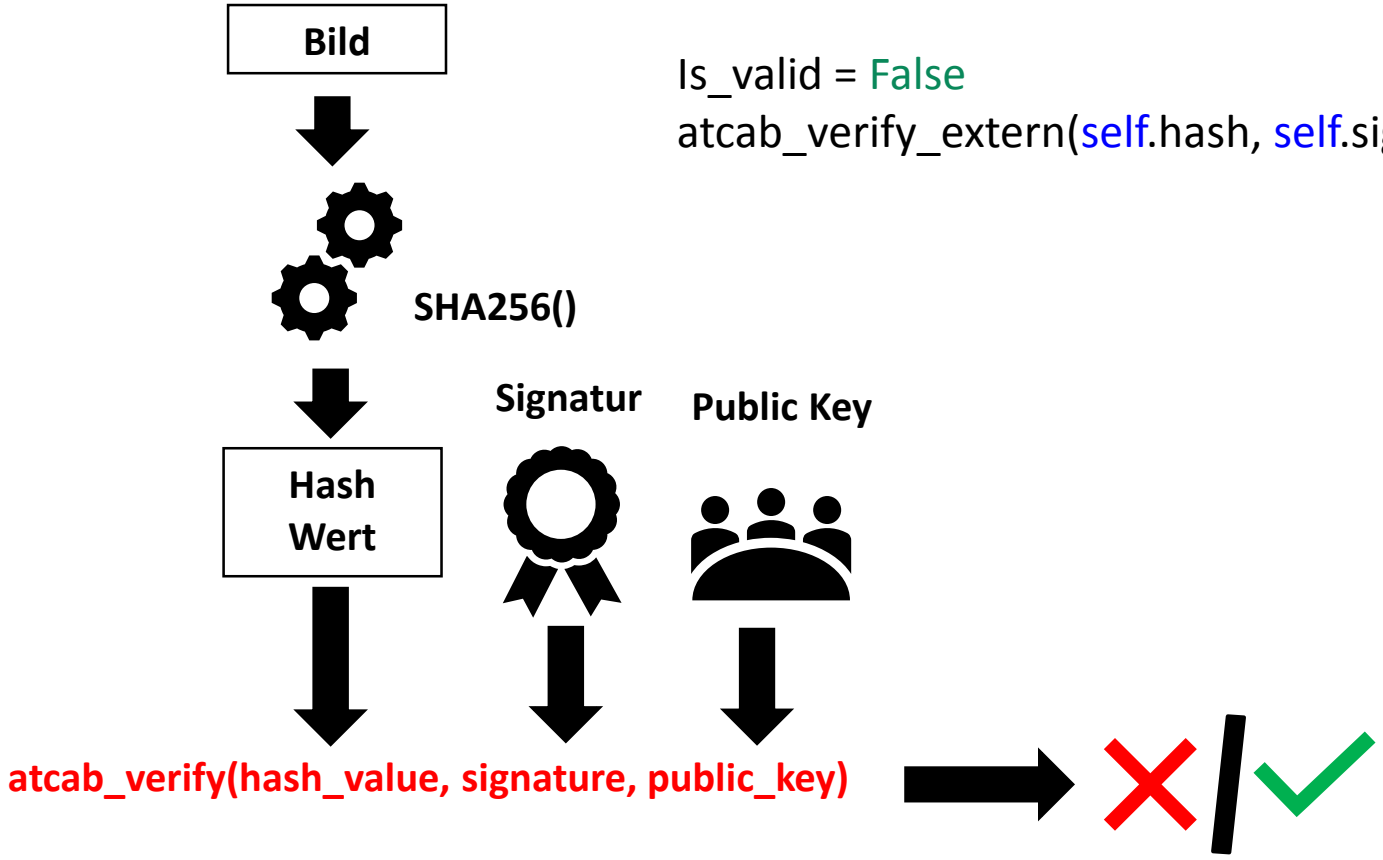



```
digest = hashes.Hash(hashes.SHA256())
digest.update(picture_bytes)
self.hash = digest.finalize()
```

```
self.signature = bytearray(64)
assert atcab_sign(self.key_slot, self.hash, self.signature ) == ATCA_SUCCESS
```

Signatur

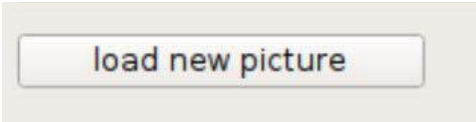
ECDSA-Verifikation




```
Is_valid = False
atcab_verify_extern(self.hash, self.signature, self.pub_key, is_valid)
```

Graphischen Benutzeroberfläche


```
# load necessary GUI libraries

from PyQt5 import QtWidgets, QTCore, QtGui
from send_design import Ui_MainWindow  # importing the QT File

 #button init

 self.ui.pB_load_new_pic.clicked.connect(self.click_new_pic)

# button functions

def click_new_pic(self):
      #load new pic from pool
      if self.pic_count >= 19:
         self.pic_count = 0
      else:
         self.pic_count += 1

      self.current_pic = self.pic_path + self.pic_list[self.pic_count]
      display = QPixmap(self.current_pic)
      self.ui.display.setPixmap(display)
      self.ui.out_file_name.setText(self.current_pic)
      self.ui.out_hash_value.setText("no hash value avilable")
      self.ui.out_signature.setText("no signature avilable")
      self.ui.out_signature.setStyleSheet("background-color: rgb(233, 233, 233);")
```



JPEG-Metadaten

```
5590    00015d40: D9 FF FE 00 73 65 6E 73 6F 72 5F 32 2A 36 33 34    Y.~.sensor_2*634
5591    00015d50: 39 38 33 3B 63 6F 75 6E 74 65 72 2A 31 37 3B 64    983;counter*17;d
5592    00015d60: 65 76 69 63 65 2A 30 31 32 33 46 44 30 43 41 37    evice*0123FD0CA7
5593    00015d70: 44 35 45 38 46 31 45 45 3B 64 61 74 65 5F 74 69    D5E8F1EE;date_ti
5594    00015d80: 6D 65 2A 30 39 2E 30 31 2E 32 30 31 39 5F 31 35    me*09.01.2019_15
5595    00015d90: 3A 33 33 3A 35 34 3B 73 65 6E 73 6F 72 5F 33 2A    :33:54;sensor_3*
5596    00015da0: 33 33 31 31 31 39 3B 73 65 6E 73 6F 72 5F 34 2A    331119;sensor_4*
5597    00015db0: 36 39 39 37 32 37 3B 73 65 6E 73 6F 72 5F 31 2A    699727;sensor_1*
5598    00015dc0: 31 34 37 37 36 32 00                               147762.
```

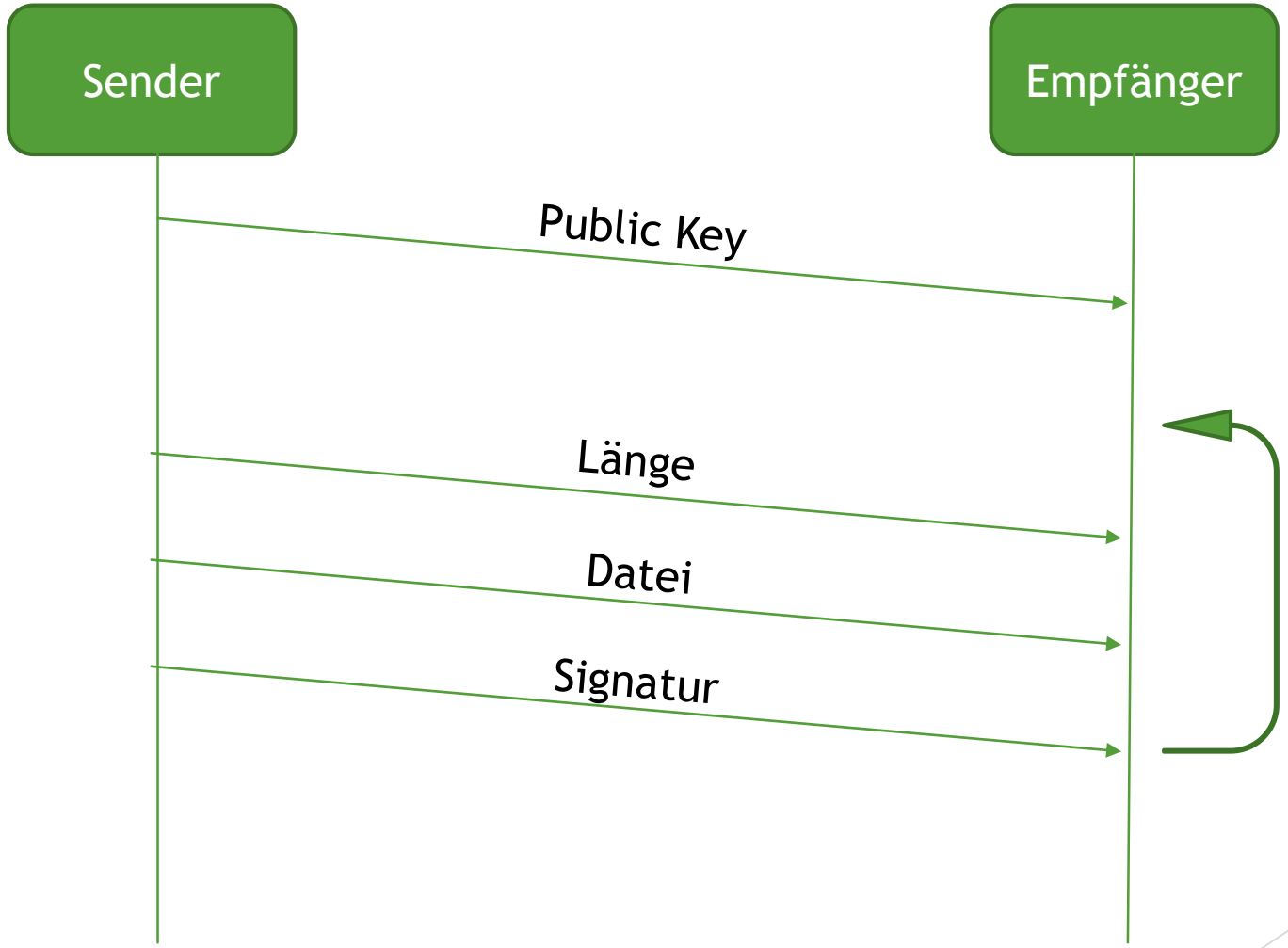
JPEG-Metadaten

```
self.metadict["device"] = self.serialnum

...

file = open(self.current_pic, 'rb')
data = file.read()
file.close()

metadata = str(self.metadict)

...

start = data.find(b'\xff\xfe')
if start == -1:
        datanew = data
else:
        datanew = data[:start]


metadata = b'\xff\xfe\x00' + metadata.encode() + b'\x00'


file2 = open(self.meta_pic, 'wb')
file2.write((datanew+metadata))
file2.close()
```

JPEG-Metadaten

```
file = open(self.current_pic, 'rb')
data = file.read()
file.close()


start = data.find(b'\xff\xfe')
start = start + 3
data = data[start:-1]
data = data.decode()


data = data.split(';')
for i in range(len(data)):
        data[i] = data[i].split('*')

keys = []
values = []

for i in range(len(data)):
        keys.append(data[i][0])
        values.append(data[i][1])

dictionary = dict(zip(keys, values))

self.metadict.clear()
self.metadict = dictionary
```

TCP-Verbindung



TCP-Verbindung

```
s.connect((self.dest_ip, int(self.dest_port_pic)))


f = open(self.meta_pic, 'rb')
message = f.read()
f.close()


length = len(message)
length = 'length ' + str(length)
length = length.encode()


signature = b'SIGNATURE:' + self.signature


s.sendall(length)
time.sleep(0.5)
s.sendall(message)
time.sleep(0.5)
s.sendall(signature)
```

Rückblick und Mögliche Erweiterungen

Rückblick

- ✓ RevPi eingerichtet
- ✓ Hardwareaufbau angefertigt
- ✓ Erfolgreich mit ATECC508A kommuniziert
- ✓ Python-Bibliothek cryptoauthlib eingebunden
- ✓ Zusatzinformationen in Bild gespeichert
- ✓ Daten Signiert und Verifiziert
- ✓ Zwei RevPi mit TCP verbunden
- ✓ Grafische Benutzeroberfläche erstellt

Erweiterung

- Locking Data-Zone
  - Integrität der Keys geschützt
- ECDH-Key-agreement für verschlüsselte Kommunikation
  - Wahrung der Vertraulichkeit bei der Kommunikation

Demonstration