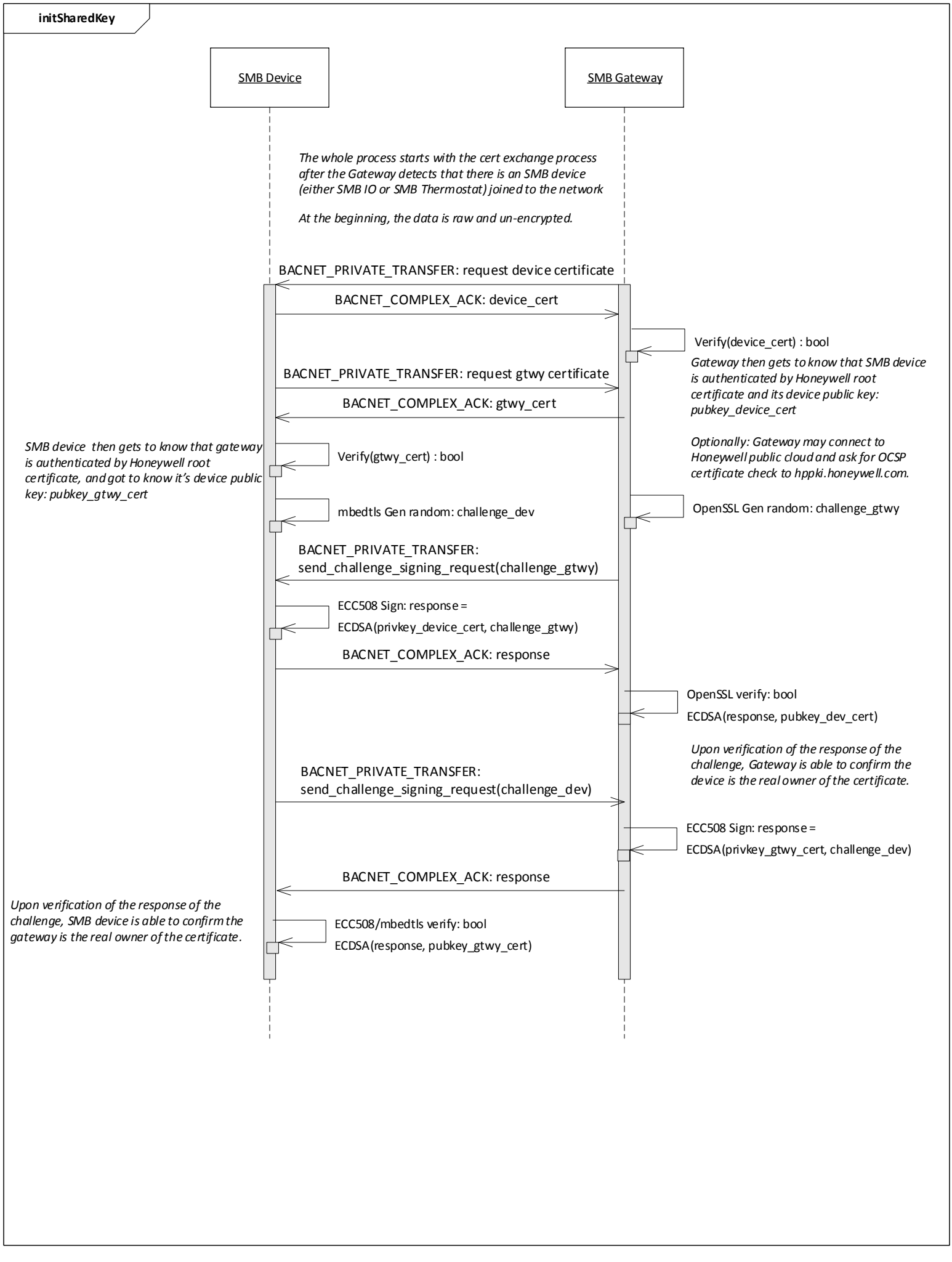
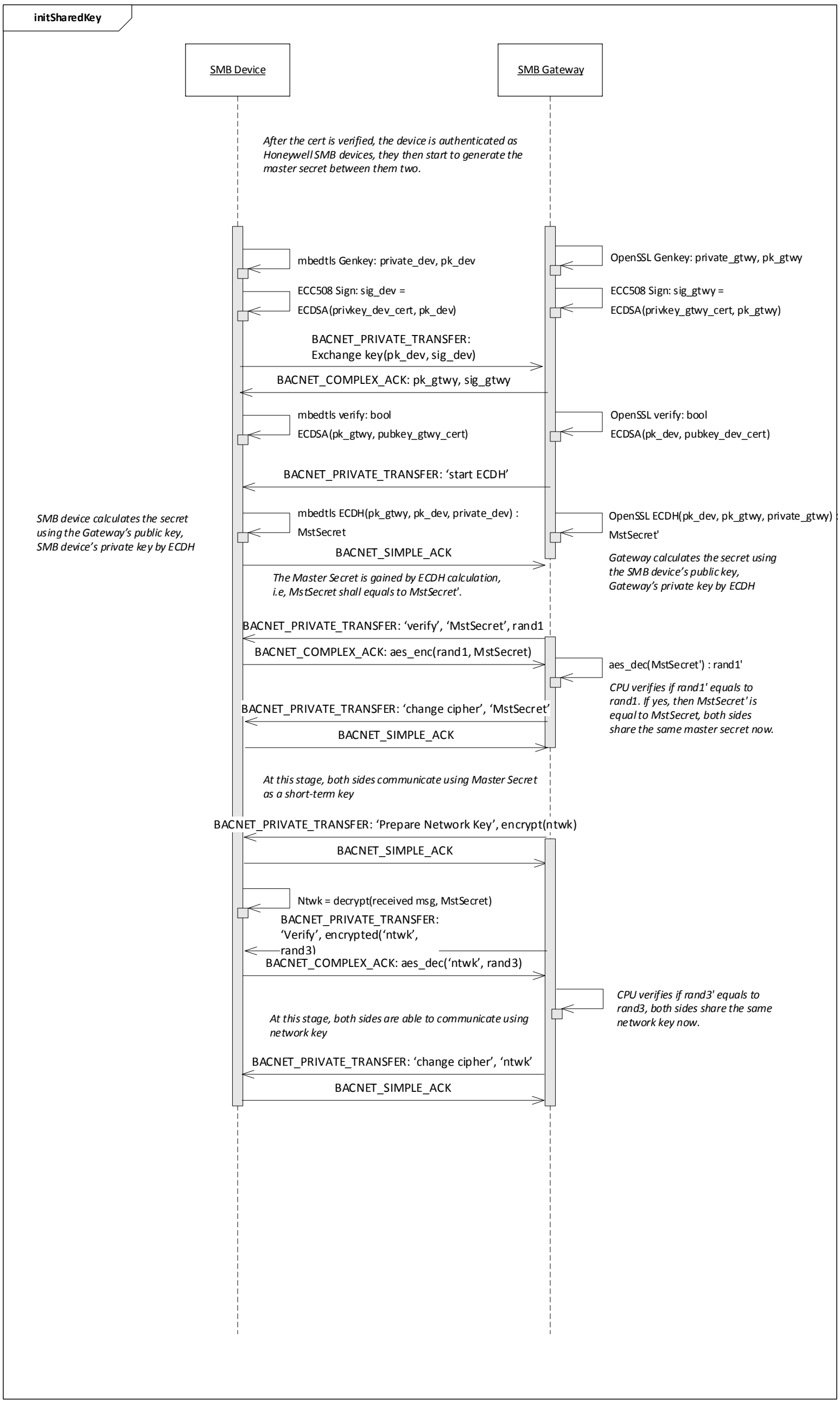
Revision History

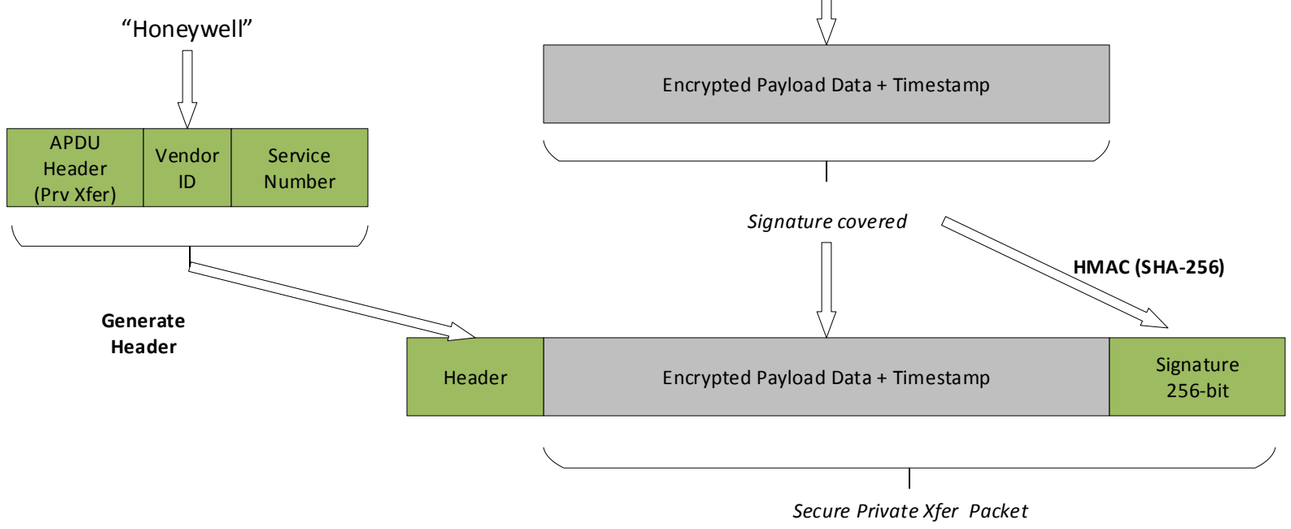
V0.1 – Li Lei, 2018-11-05, initial draft.
V1.0 – Li Lei, 2019-11-06, Update to do the challenge of the devices and suitable for SMB system.


initSharedKey
SMB Device
SMB Gateway
The whole process starts with the cert exchange process after the Gateway detects that there is an SMB device (either SMB IO or SMB Thermostat) joined to the network
At the beginning, the data is raw and un-encrypted.
BACNET_PRIVATE_TRANSFER: request device certificate
BACNET_COMPLEX_ACK: device_cert
Verify(device_cert) : bool
Gateway then gets to know that SMB device is authenticated by Honeywell root certificate and its device public key: pubkey_device_cert
BACNET_PRIVATE_TRANSFER: request gtwy certificate
BACNET_COMPLEX_ACK: gtwy_cert
SMB device then gets to know that gateway is authenticated by Honeywell root certificate, and got to know it's device public key: pubkey_gtwy_cert
Verify(gtwy_cert) : bool
Optionally: Gateway may connect to Honeywell public cloud and ask for OCSP certificate check to hppki.honeywell.com.
mbedtls Gen random: challenge_dev
OpenSSL Gen random: challenge_gtwy
BACNET_PRIVATE_TRANSFER: send_challenge_signing_request(challenge_gtwy)
ECC508 Sign: response = ECDSA(privkey_device_cert, challenge_gtwy)
BACNET_COMPLEX_ACK: response
OpenSSL verify: bool
ECDSA(response, pubkey_dev_cert)
Upon verification of the response of the challenge, Gateway is able to confirm the device is the real owner of the certificate.
BACNET_PRIVATE_TRANSFER: send_challenge_signing_request(challenge_dev)
ECC508 Sign: response = ECDSA(privkey_gtwy_cert, challenge_dev)
BACNET_COMPLEX_ACK: response
Upon verification of the response of the challenge, SMB device is able to confirm the gateway is the real owner of the certificate.
ECC508/mbedtls verify: bool
ECDSA(response, pubkey_gtwy_cert)


initSharedKey
SMB Device
SMB Gateway
After the cert is verified, the device is authenticated as Honeywell SMB devices, they then start to generate the master secret between them two.
mbedtls Genkey: private_dev, pk_dev
OpenSSL Genkey: private_gtwy, pk_gtwy
ECC508 Sign: sig_dev = ECDSA(privkey_dev_cert, pk_dev)
ECC508 Sign: sig_gtwy = ECDSA(privkey_gtwy_cert, pk_gtwy)
BACNET_PRIVATE_TRANSFER: Exchange key(pk_dev, sig_dev)
BACNET_COMPLEX_ACK: pk_gtwy, sig_gtwy
mbedtls verify: bool ECDSA(pk_gtwy, pubkey_gtwy_cert)
OpenSSL verify: bool ECDSA(pk_dev, pubkey_dev_cert)
BACNET_PRIVATE_TRANSFER: 'start ECDH'
SMB device calculates the secret using the Gateway's public key, SMB device's private key by ECDH
mbedtls ECDH(pk_gtwy, pk_dev, private_dev) : MstSecret
OpenSSL ECDH(pk_dev, pk_gtwy, private_gtwy) : MstSecret'
BACNET_SIMPLE_ACK
Gateway calculates the secret using the SMB device's public key, Gateway's private key by ECDH
The Master Secret is gained by ECDH calculation, i.e, MstSecret shall equals to MstSecret'.
BACNET_PRIVATE_TRANSFER: 'verify', 'MstSecret', rand1
BACNET_COMPLEX_ACK: aes_enc(rand1, MstSecret)
aes_dec(MstSecret') : rand1'
CPU verifies if rand1' equals to rand1. If yes, then MstSecret' is equal to MstSecret, both sides share the same master secret now.
BACNET_PRIVATE_TRANSFER: 'change cipher', 'MstSecret'
BACNET_SIMPLE_ACK
At this stage, both sides communicate using Master Secret as a short-term key
BACNET_PRIVATE_TRANSFER: 'Prepare Network Key', encrypt(ntwk)
BACNET_SIMPLE_ACK
Ntwk = decrypt(received msg, MstSecret)
BACNET_PRIVATE_TRANSFER: 'Verify', encrypted('ntwk', rand3)
BACNET_COMPLEX_ACK: aes_dec('ntwk', rand3)
CPU verifies if rand3' equals to rand3, both sides share the same network key now.
At this stage, both sides are able to communicate using network key
BACNET_PRIVATE_TRANSFER: 'change cipher', 'ntwk'
BACNET_SIMPLE_ACK

Service Number Nomenclature:
0x80: Exchange Device cert
0x81: Verify cert
.....

**0x90**: Private Transfer Extension: encrypted existing APDU commands







64-bit timestamp --> overflows in 292 million years
@ every increase per ms