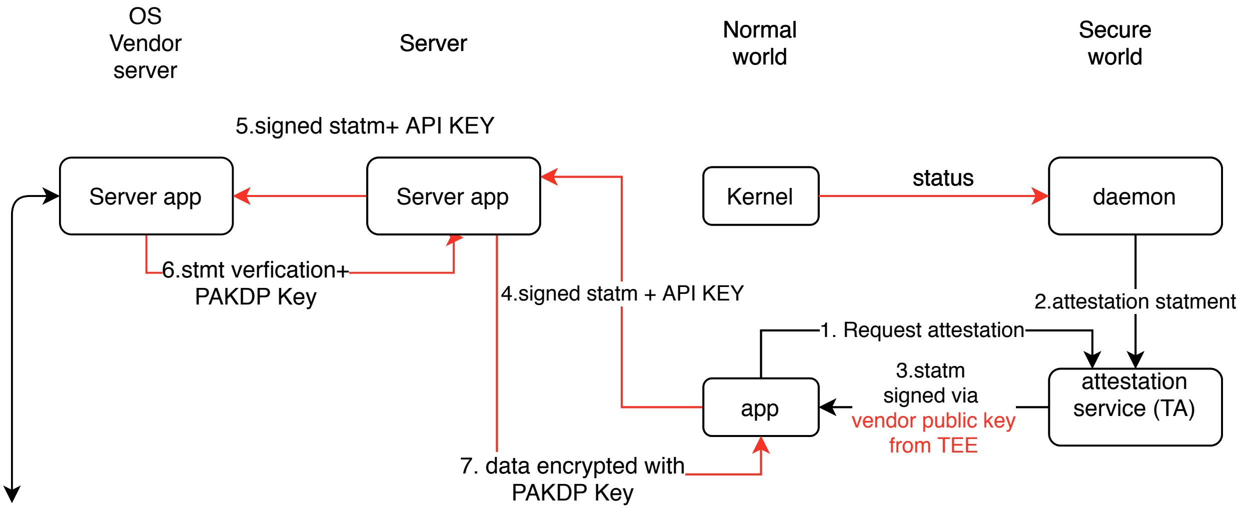


Verify the signature of stmt
Verify the content IMEI number belong to the vendor and obtain Device Public key from vendor database
Generate PAKDP enecrytpion Key as follow :

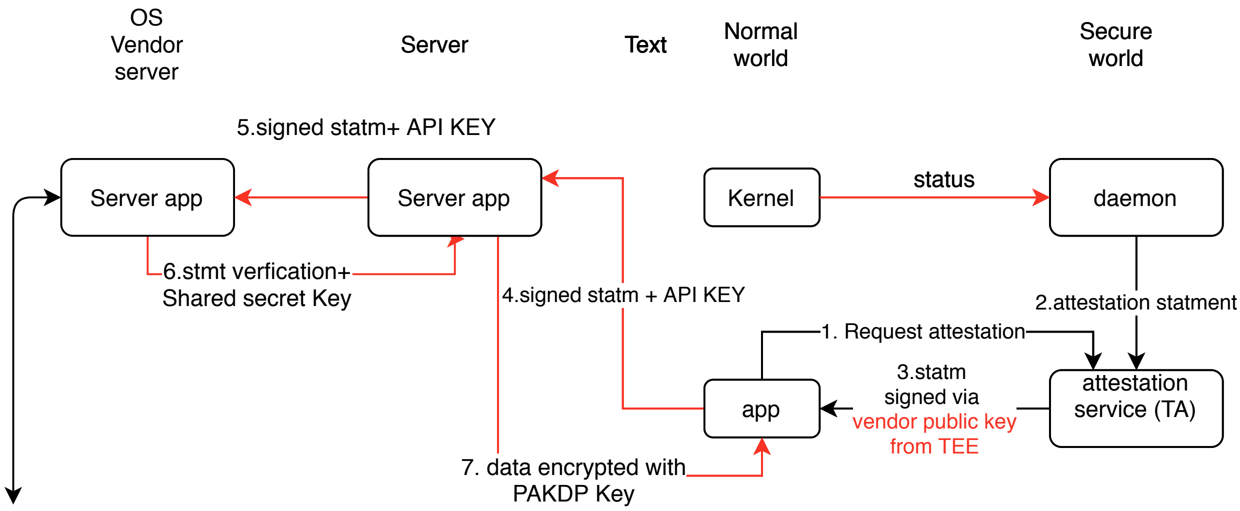
PAKDP Key = Per API KEY Device public key = KDF(API KEY, Device Public key)

Verify the signature of stmt
Text
Verify the content IMEI number belong to the vendor and obtain Device Public key from vendor database
Generate PAKDP enecrytpion Key as follow :

PAKDP Key = Per API KEY Device public key = KDF(API KEY, Device Public key)

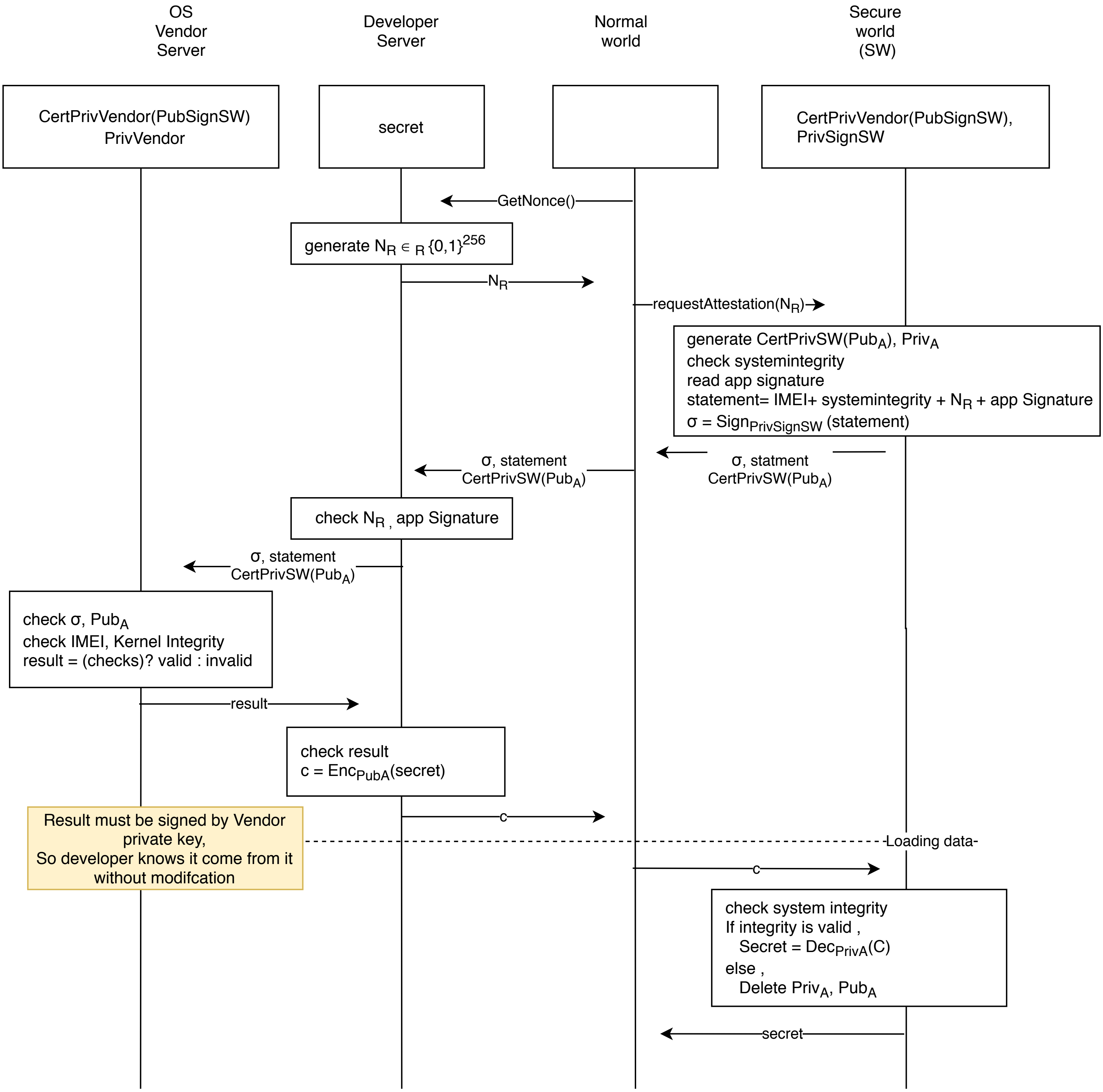
signed stmt = nonce
- generate never release private key and public key.
- enecrypt data fusing the key and deleted

Other option using a shared secret between the device and the server .no TLS
nonce ?
signature verification key?

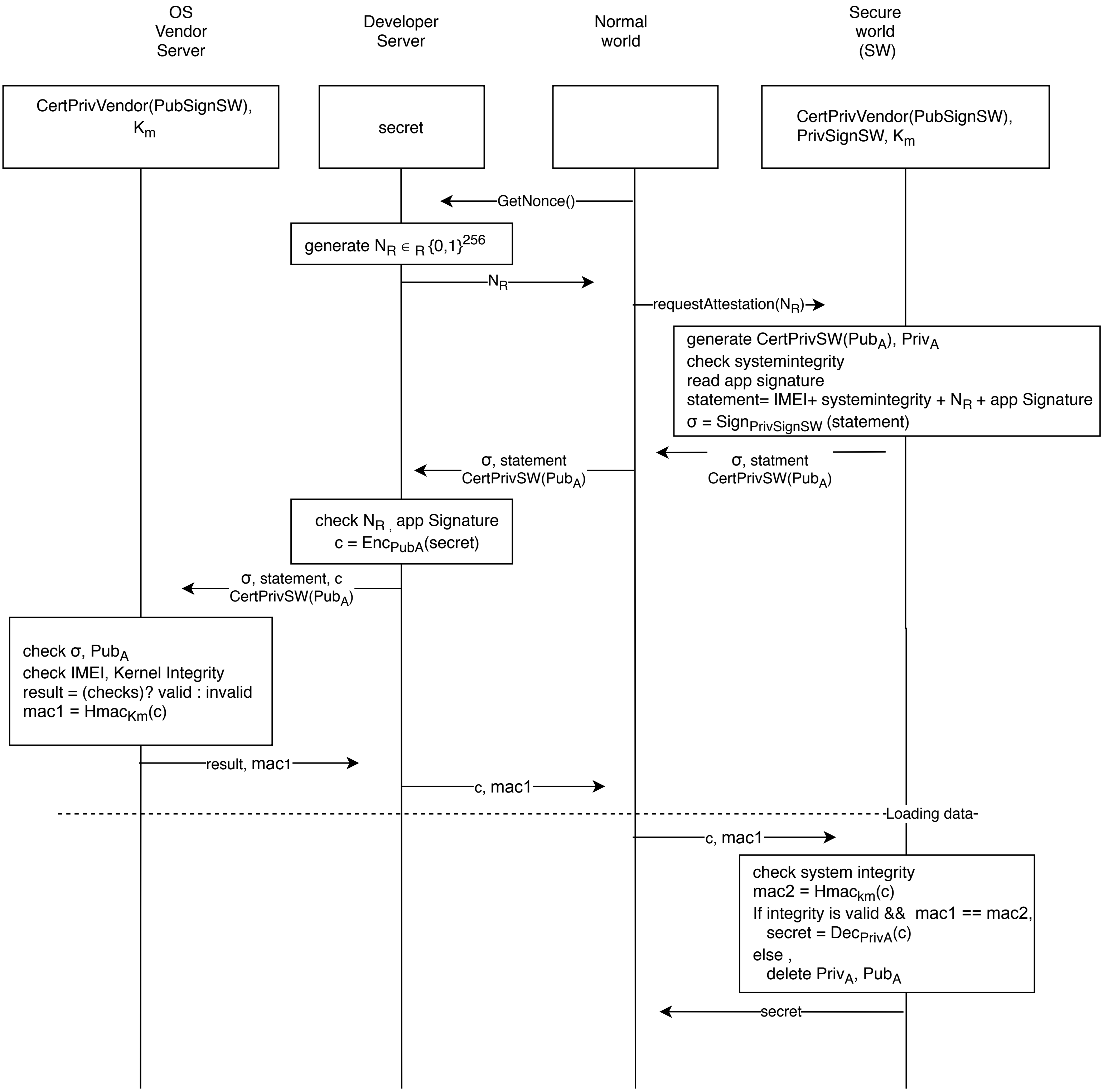


Shared secret key = Device derive a key using a shared key and API KEY

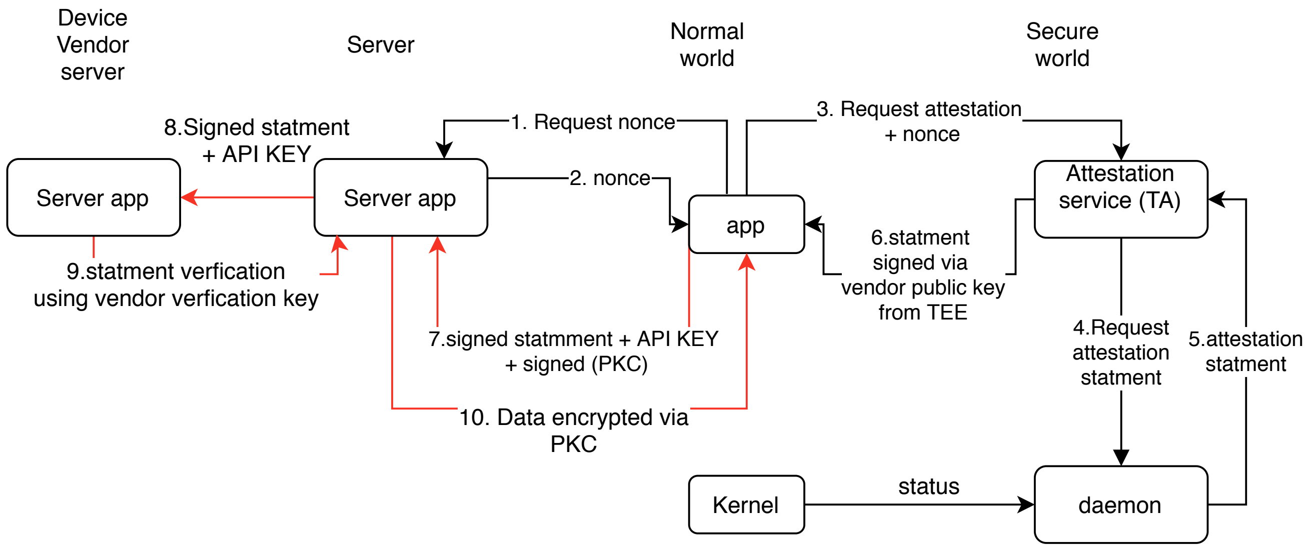
Basic


OS Vendor Server
Developer Server
Normal world
Secure world (SW)
CertPrivVendor(PubSignSW)
PrivVendor
secret
CertPrivVendor(PubSignSW),
PrivSignSW
GetNonce()
generate $N_R \in_R \{0,1\}^{256}$
N_R
requestAttestation(N_R)
generate CertPrivSW(Pub_A), $Priv_A$
check systemintegrity
read app signature
statement= IMEI+ systemintegrity + N_R + app Signature
σ = $Sign_{PrivSignSW}$ (statement)
σ, statment
CertPrivSW(Pub_A)
σ, statement
CertPrivSW(Pub_A)
check N_R , app Signature
σ, statement
CertPrivSW(Pub_A)
check σ, Pub_A
check IMEI, Kernel Integrity
result = (checks)? valid : invalid
result
check result
c = Enc_{PubA}(secret)
c
Result must be signed by Vendor private key,
So developer knows it come from it without modifcation
Loading data
c
check system integrity
If integrity is valid ,
Secret = Dec_{PrivA}(C)
else ,
Delete $Priv_A$, Pub_A
secret

HMAC

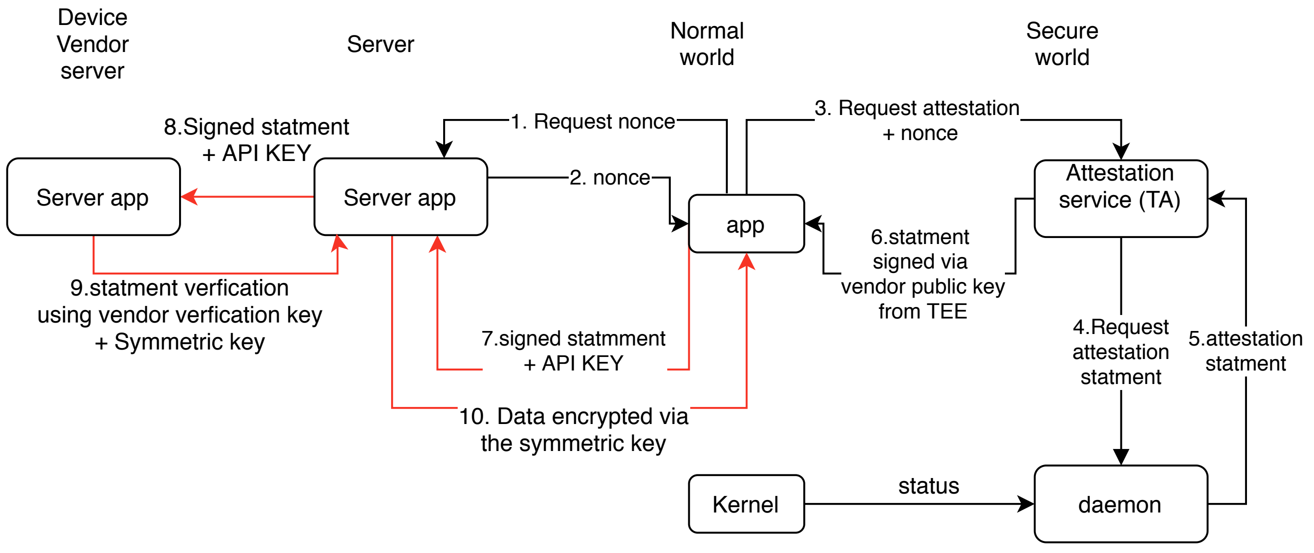

OS Vendor Server
Developer Server
Normal world
Secure world (SW)
CertPrivVendor(PubSignSW), K_m
secret
CertPrivVendor(PubSignSW), PrivSignSW, K_m
GetNonce()
generate $N_R \in_R \{0,1\}^{256}$
N_R
requestAttestation(N_R)
generate CertPrivSW(Pub_A), $Priv_A$
check systemintegrity
read app signature
statement= IMEI+ systemintegrity + N_R + app Signature
σ = $Sign_{PrivSignSW}$ (statement)
σ, statment
CertPrivSW(Pub_A)
σ, statement
CertPrivSW(Pub_A)
check $N_{R\,,}$ app Signature
c = Enc_{PubA}(secret)
σ, statement, c
CertPrivSW(Pub_A)
check σ, Pub_A
check IMEI, Kernel Integrity
result = (checks)? valid : invalid
mac1 = $Hmac_{Km}$(c)
result, mac1
c, mac1
Loading data-
c, mac1
check system integrity
mac2 = $Hmac_{km}$(c)
If integrity is valid && mac1 == mac2,
secret = Dec_{PrivA}(c)
else ,
delete $Priv_A$, Pub_A
secret

Public key cerftciate (PKC) approach


Device Vendor server
Server
Normal world
Secure world
Server app
Server app
app
Attestation service (TA)
Kernel
daemon
1. Request nonce
2. nonce
3. Request attestation + nonce
4.Request attestation statment
5.attestation statment
6.statment signed via vendor public key from TEE
7.signed statmment + API KEY + signed (PKC)
8.Signed statment + API KEY
9.statment verfication using vendor verfication key
10. Data encrypted via PKC
status

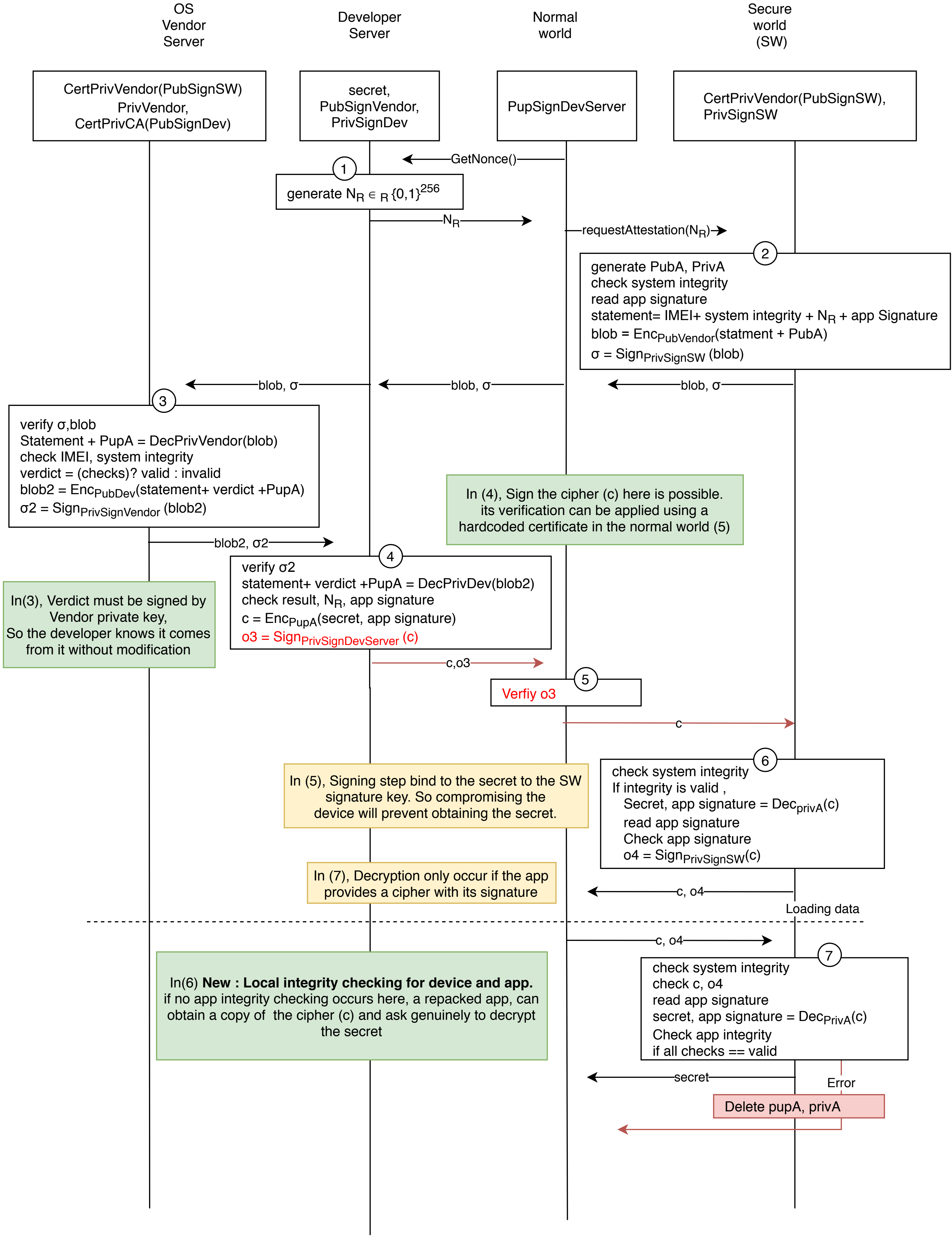
Daemon: measures kernel status periodically
Attestation statement: contains a nonce, IMEI number, kernel status, **public key certificate (PKC)**. The statement signed by vendor public key
Public-key certificate (PKC): a key generated in the secure world. it's a private part never leave the secure world.
API KEY: per developer key to use the attestation service.

Shared secret approach


Device Vendor server
Server
Normal world
Secure world
Server app
Server app
app
Attestation service (TA)
Kernel
daemon
1. Request nonce
2. nonce
3. Request attestation + nonce
4.Request attestation statment
5.attestation statment
6.statment signed via vendor public key from TEE
7.signed statmment + API KEY
8.Signed statment + API KEY
9.statment verfication using vendor verfication key + Symmetric key
10. Data encrypted via the symmetric key
status

Shared secret: secret data can be set once before releasing the device (e.g device unique key). It never leaves the secure world.
Symmetric key: KDF(nonce, shared secret)
Daemon: measures kernel status periodically
Attestation statement: contains a nonce, IMEI number, kernel status, **public key certificate (PKC)**. The statement signed by vendor public key
Public-key certificate (PKC): a key generated in the secure world. it's a private part never leave the secure world.
API KEY: per developer key to use the attestation service.

Code Runner (Optimal)



CertPrivVendor(PubSignSW)
PrivVendor,
CertPrivCA(PubSignDev)

secret,
PubSignVendor,
PrivSignDev

PupSignDevServer

CertPrivVendor(PubSignSW),
PrivSignSW

generate $N_R \in_R \{0,1\}^{256}$

generate PubA, PrivA
check system integrity
read app signature
statement= IMEI+ system integrity + N_R + app Signature
blob = $Enc_{PubVendor}$(statment + PubA)
σ = $Sign_{PrivSignSW}$ (blob)

verify σ,blob
Statement + PupA = DecPrivVendor(blob)
check IMEI, system integrity
verdict = (checks)? valid : invalid
blob2 = Enc_{PubDev}(statement+ verdict +PupA)
σ2 = $Sign_{PrivSignVendor}$ (blob2)

In (4), Sign the cipher (c) here is possible. its verification can be applied using a hardcoded certificate in the normal world (5)

verify σ2
statement+ verdict +PupA = DecPrivDev(blob2)
check result, N_R, app signature
c = Enc_{PupA}(secret, app signature)
o3 = $Sign_{PrivSignDevServer}$ (c)

In(3), Verdict must be signed by Vendor private key, So the developer knows it comes from it without modification

Verfiy o3

In (5), Signing step bind to the secret to the SW signature key. So compromising the device will prevent obtaining the secret.

check system integrity
If integrity is valid ,
Secret, app signature = Dec_{privA}(c)
read app signature
Check app signature
o4 = $Sign_{PrivSignSW}$(c)

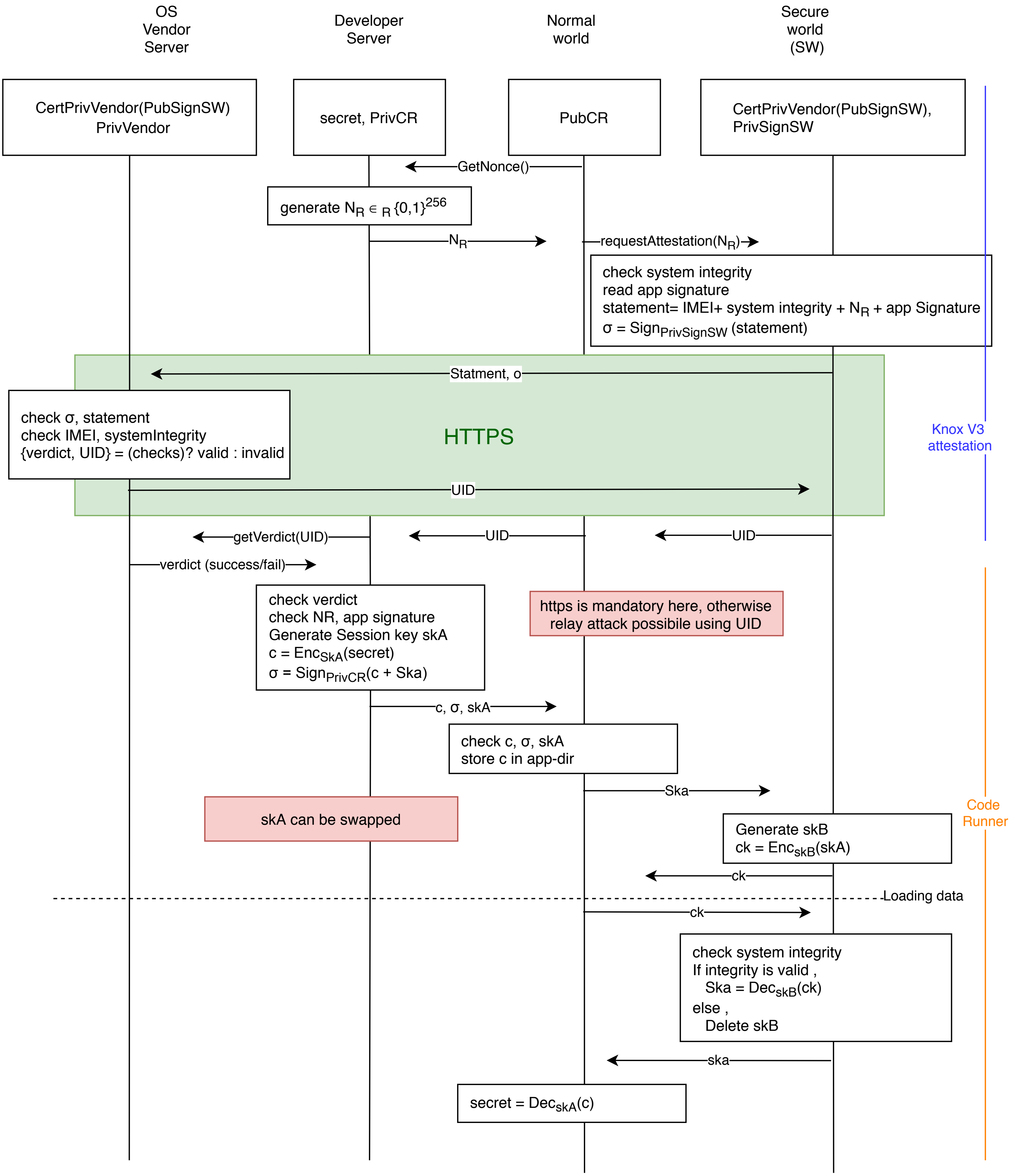
In (7), Decryption only occur if the app provides a cipher with its signature

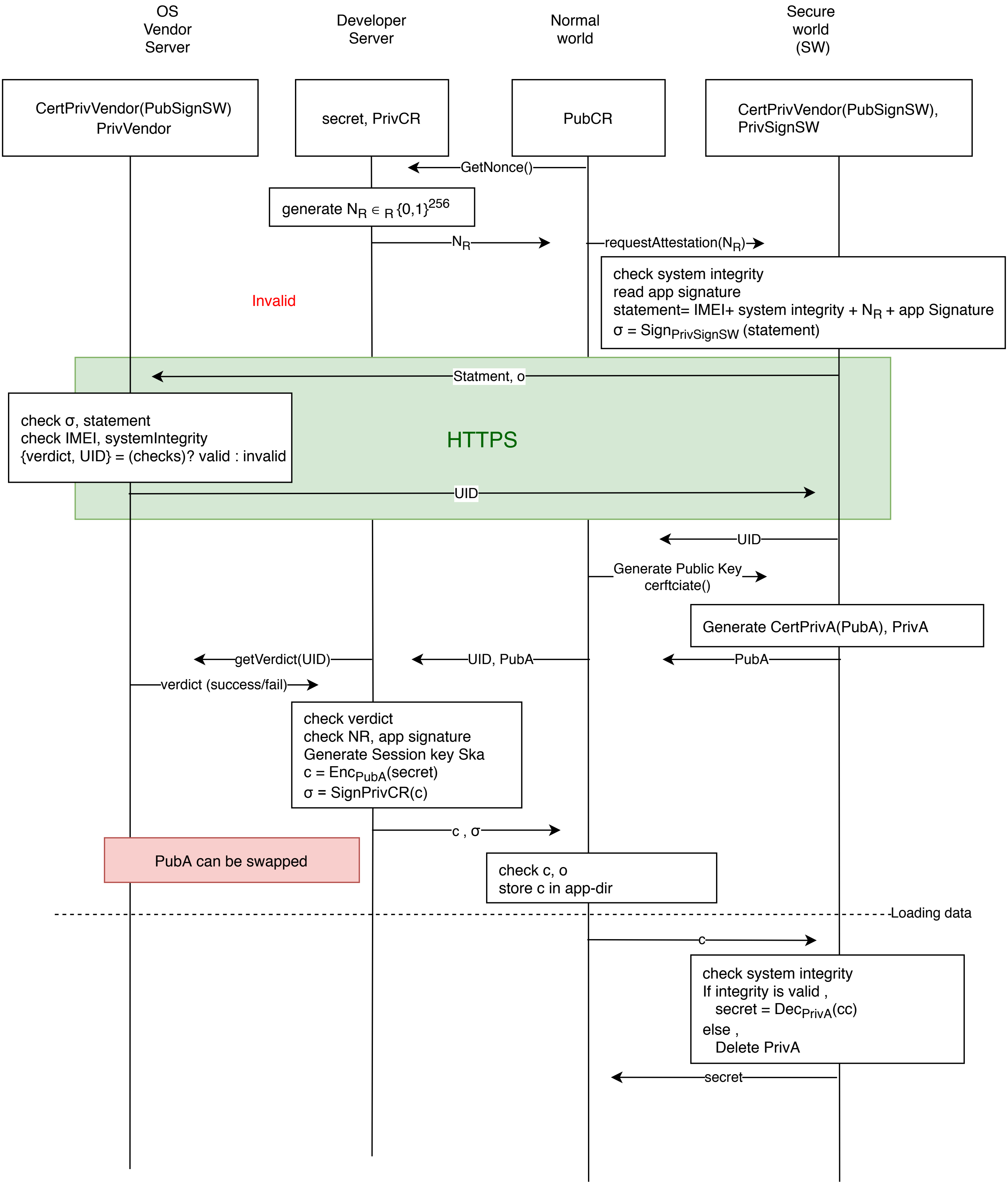
In(6) **New : Local integrity checking for device and app.** if no app integrity checking occurs here, a repacked app, can obtain a copy of the cipher (c) and ask genuinely to decrypt the secret

check system integrity
check c, o4
read app signature
secret, app signature = Dec_{PrivA}(c)
Check app integrity
if all checks == valid

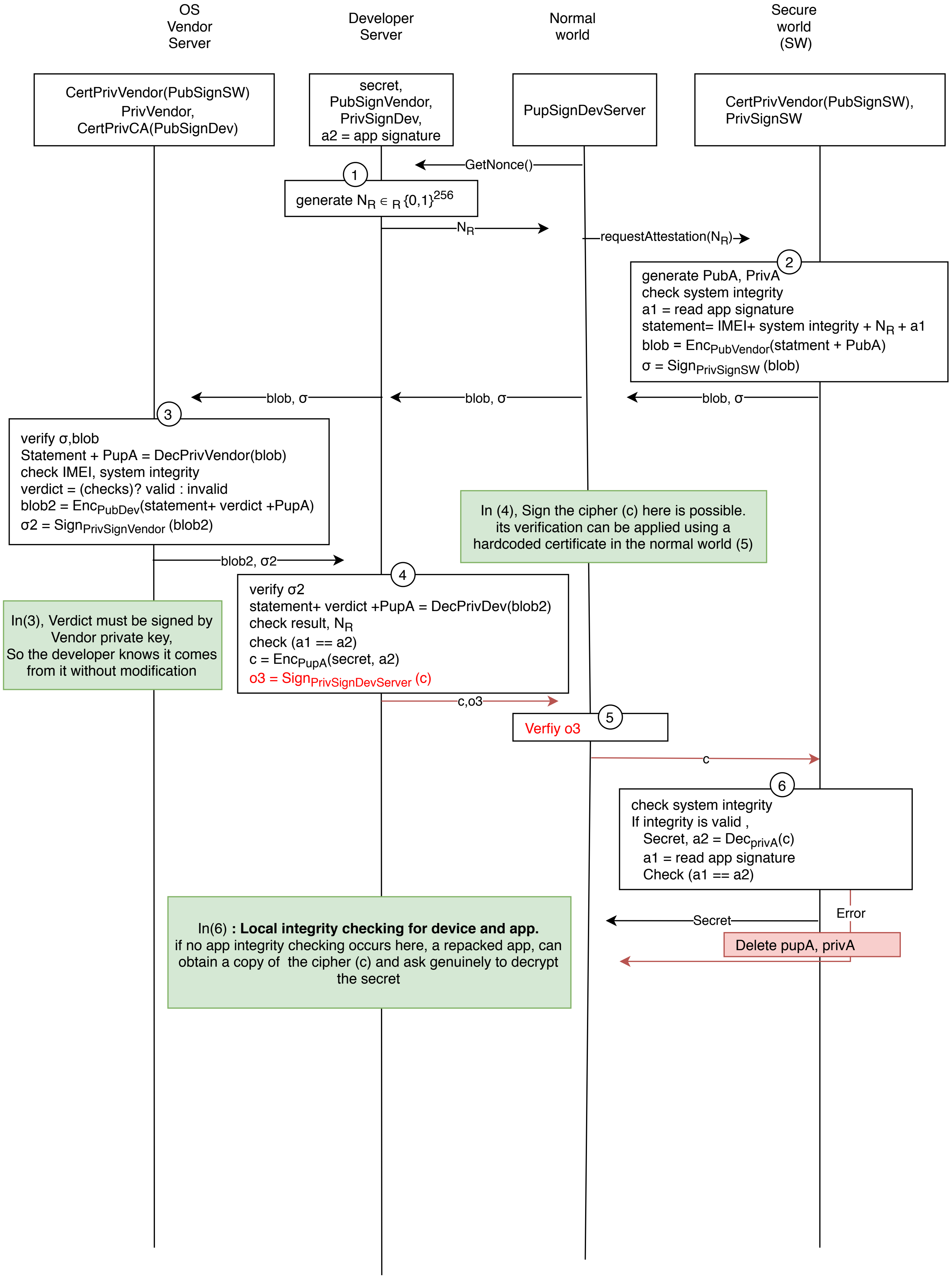
Delete pupA, privA

Code Runner using Knox v3




OS Vendor Server
Developer Server
Normal world
Secure world (SW)
CertPrivVendor(PubSignSW)
PrivVendor
secret, PrivCR
PubCR
CertPrivVendor(PubSignSW),
PrivSignSW
GetNonce()
generate $N_R \in_R \{0,1\}^{256}$
N_R
requestAttestation(N_R)
check system integrity
read app signature
statement= IMEI+ system integrity + N_R + app Signature
σ = $\text{Sign}_{\text{PrivSignSW}}$ (statement)
Invalid
Statment, σ
HTTPS
check σ, statement
check IMEI, systemIntegrity
{verdict, UID} = (checks)? valid : invalid
UID
UID
Generate Public Key cerftciate()
Generate CertPrivA(PubA), PrivA
getVerdict(UID)
UID, PubA
PubA
verdict (success/fail)
check verdict
check NR, app signature
Generate Session key Ska
c = Enc_{PubA}(secret)
σ = SignPrivCR(c)
c , σ
PubA can be swapped
check c, σ
store c in app-dir
Loading data
c
check system integrity
If integrity is valid ,
secret = $\text{Dec}_{\text{PrivA}}$(cc)
else ,
Delete PrivA
secret

Code Runner (Optimal)


OS Vendor Server
Developer Server
Normal world
Secure world (SW)
CertPrivVendor(PubSignSW) PrivVendor, CertPrivCA(PubSignDev)
secret, PubSignVendor, PrivSignDev, a2 = app signature
PupSignDevServer
CertPrivVendor(PubSignSW), PrivSignSW
GetNonce()
1
generate $N_R \in_R \{0,1\}^{256}$
N_R
requestAttestation(N_R)
2
blob, σ
blob, σ
blob, σ
3
blob2, σ2
4
c,o3
5
Verfiy o3
c
6
Secret
Error
Delete pupA, privA

generate PubA, PrivA
check system integrity
a1 = read app signature
statement= IMEI+ system integrity + N_R + a1
blob = $Enc_{PubVendor}$(statment + PubA)
σ = $Sign_{PrivSignSW}$ (blob)

verify σ,blob
Statement + PupA = DecPrivVendor(blob)
check IMEI, system integrity
verdict = (checks)? valid : invalid
blob2 = Enc_{PubDev}(statement+ verdict +PupA)
σ2 = $Sign_{PrivSignVendor}$ (blob2)

In (4), Sign the cipher (c) here is possible. its verification can be applied using a hardcoded certificate in the normal world (5)

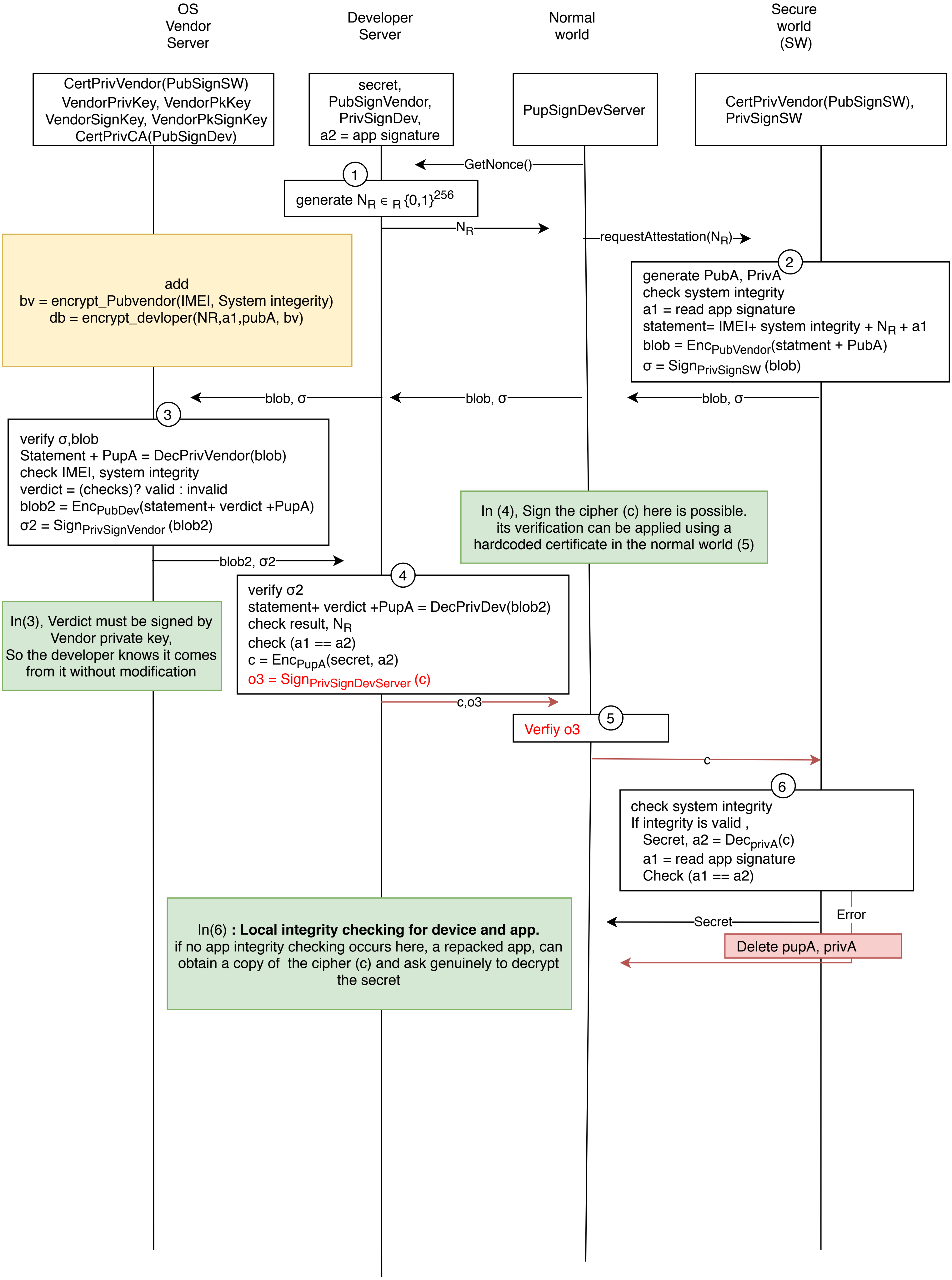
In(3), Verdict must be signed by Vendor private key, So the developer knows it comes from it without modification

verify σ2
statement+ verdict +PupA = DecPrivDev(blob2)
check result, N_R
check (a1 == a2)
c = Enc_{PupA}(secret, a2)
o3 = $Sign_{PrivSignDevServer}$ (c)

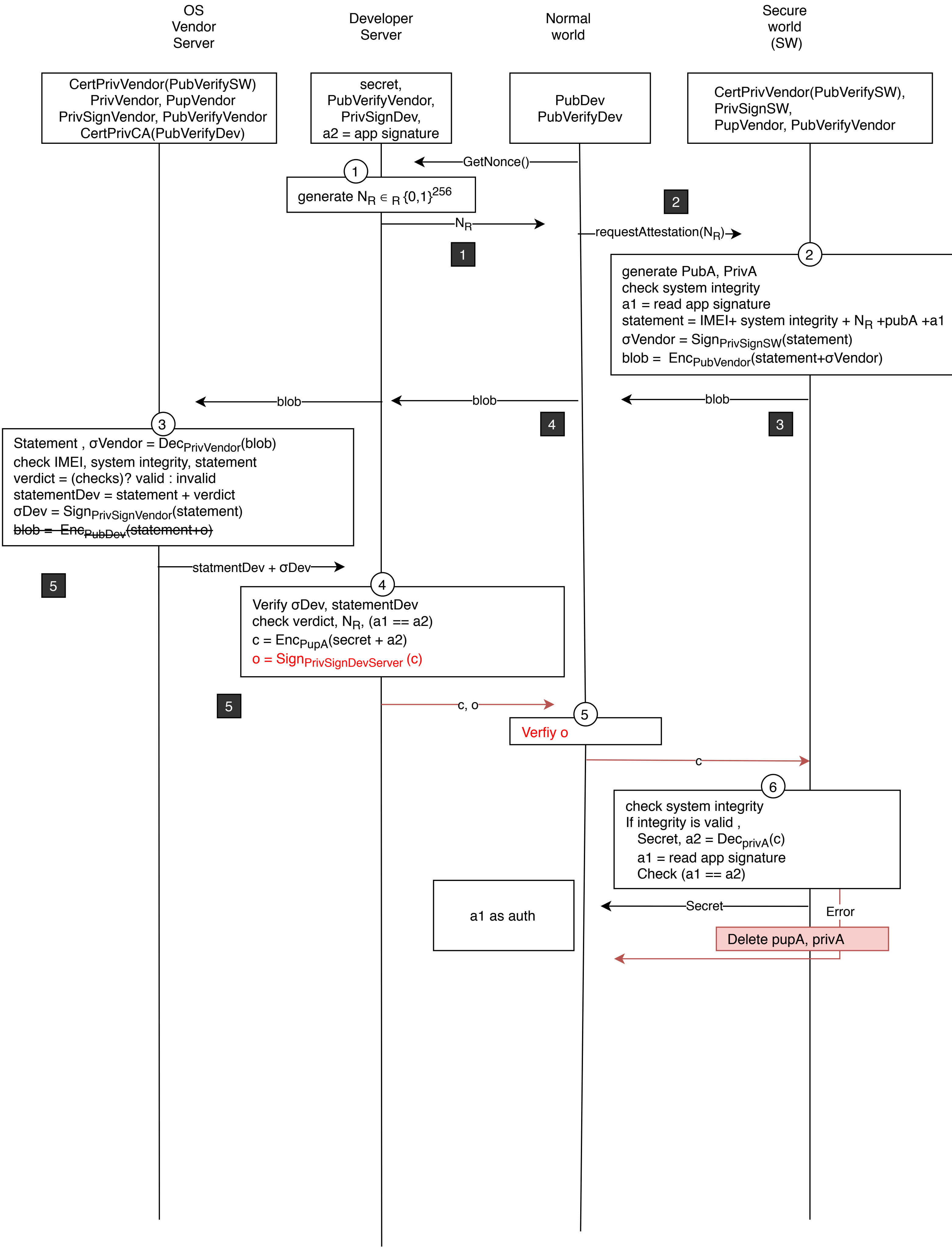
check system integrity
If integrity is valid ,
  Secret, a2 = Dec_{privA}(c)
  a1 = read app signature
  Check (a1 == a2)

In(6) **: Local integrity checking for device and app.** if no app integrity checking occurs here, a repacked app, can obtain a copy of the cipher (c) and ask genuinely to decrypt the secret

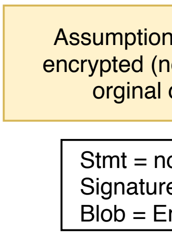
Code Runner (Optimal)


OS Vendor Server
Developer Server
Normal world
Secure world (SW)
CertPrivVendor(PubSignSW)
VendorPrivKey, VendorPkKey
VendorSignKey, VendorPkSignKey
CertPrivCA(PubSignDev)
secret,
PubSignVendor,
PrivSignDev,
a2 = app signature
PupSignDevServer
CertPrivVendor(PubSignSW),
PrivSignSW
GetNonce()
1
generate $N_R \in_R \{0,1\}^{256}$
N_R
requestAttestation(N_R)
add
bv = encrypt_Pubvendor(IMEI, System integerity)
db = encrypt_devloper(NR,a1,pubA, bv)
2
generate PubA, PrivA
check system integrity
a1 = read app signature
statement= IMEI+ system integrity + N_R + a1
blob = $Enc_{PubVendor}$(statment + PubA)
σ = $Sign_{PrivSignSW}$ (blob)
blob, σ
blob, σ
blob, σ
3
verify σ,blob
Statement + PupA = DecPrivVendor(blob)
check IMEI, system integrity
verdict = (checks)? valid : invalid
blob2 = Enc_{PubDev}(statement+ verdict +PupA)
σ2 = $Sign_{PrivSignVendor}$ (blob2)
In (4), Sign the cipher (c) here is possible.
its verification can be applied using a
hardcoded certificate in the normal world (5)
blob2, σ2
4
verify σ2
statement+ verdict +PupA = DecPrivDev(blob2)
check result, N_R
check (a1 == a2)
c = Enc_{PupA}(secret, a2)
o3 = $Sign_{PrivSignDevServer}$ (c)
In(3), Verdict must be signed by
Vendor private key,
So the developer knows it comes
from it without modification
c,o3
5
Verfiy o3
c
6
check system integrity
If integrity is valid ,
Secret, a2 = Dec_{privA}(c)
a1 = read app signature
Check (a1 == a2)
Error
Secret
Delete pupA, privA
In(6) : **Local integrity checking for device and app.**
if no app integrity checking occurs here, a repacked app, can
obtain a copy of the cipher (c) and ask genuinely to decrypt
the secret

Code Runner (Optimal)


OS Vendor Server
Developer Server
Normal world
Secure world (SW)
CertPrivVendor(PubVerifySW)
PrivVendor, PupVendor
PrivSignVendor, PubVerifyVendor
CertPrivCA(PubVerifyDev)
secret,
PubVerifyVendor,
PrivSignDev,
a2 = app signature
PubDev
PubVerifyDev
CertPrivVendor(PubVerifySW),
PrivSignSW,
PupVendor, PubVerifyVendor
GetNonce()
1
generate $N_R \in_R \{0,1\}^{256}$
2
N_R
requestAttestation(N_R)
1
2
generate PubA, PrivA
check system integrity
a1 = read app signature
statement = IMEI+ system integrity + N_R +pubA +a1
σVendor = $Sign_{PrivSignSW}$(statement)
blob = $Enc_{PubVendor}$(statement+σVendor)
blob
blob
blob
4
3
3
Statement , σVendor = $Dec_{PrivVendor}$(blob)
check IMEI, system integrity, statement
verdict = (checks)? valid : invalid
statementDev = statement + verdict
σDev = $Sign_{PrivSignVendor}$(statement)
~~blob = Enc_{PubDev}(statement+o)~~
statmentDev + σDev
5
4
Verify σDev, statementDev
check verdict, N_R, (a1 == a2)
c = Enc_{PupA}(secret + a2)
o = $Sign_{PrivSignDevServer}$ (c)
5
c, o
5
Verfiy o
c
6
check system integrity
If integrity is valid ,
Secret, a2 = Dec_{privA}(c)
a1 = read app signature
Check (a1 == a2)
a1 as auth
Secret
Error
Delete pupA, privA

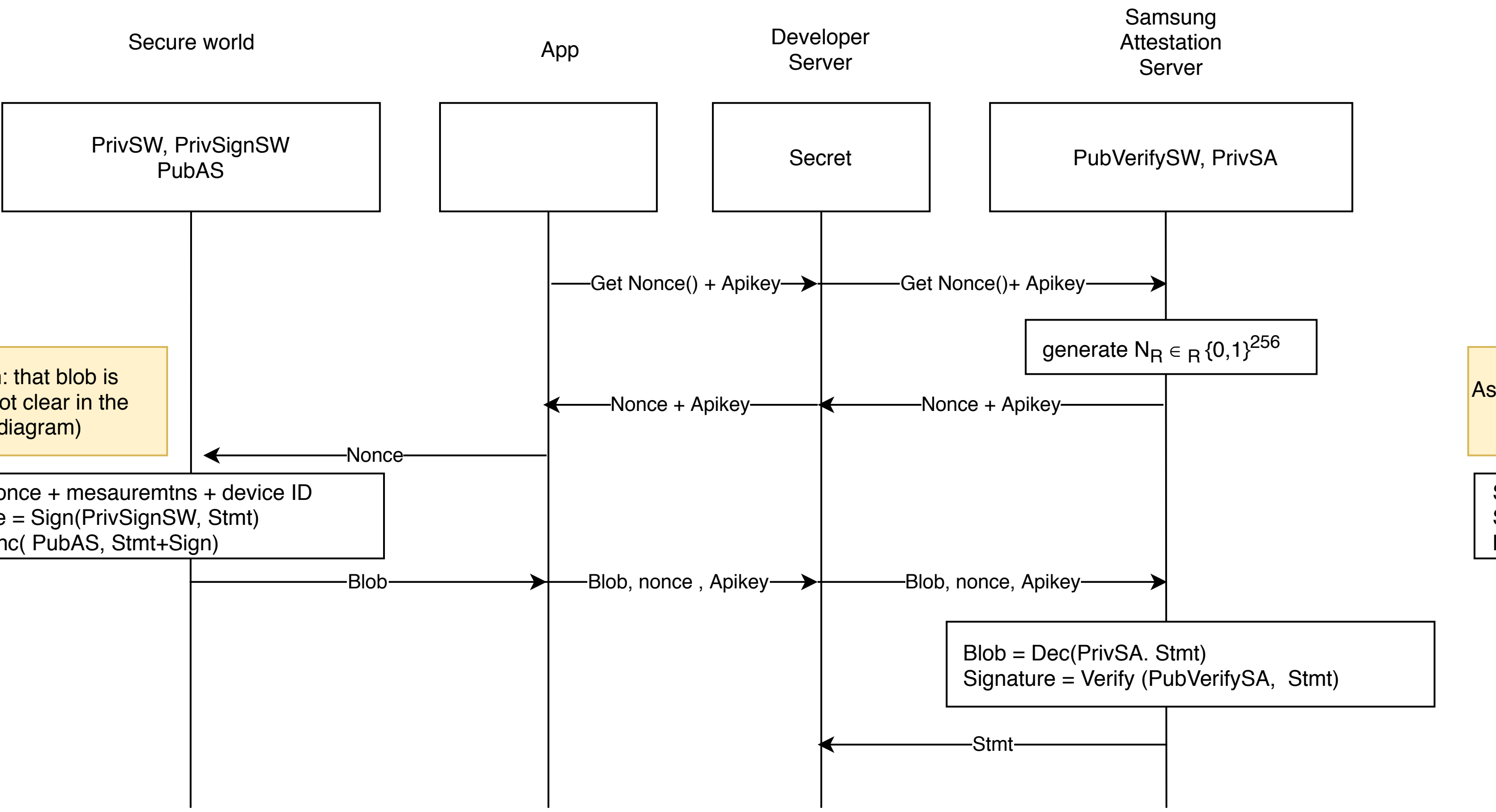
add
bv = encrypt_Pubvendor(IMEI, System integerity)
db = encrypt_devloper(NR,a1,pubA, bv)



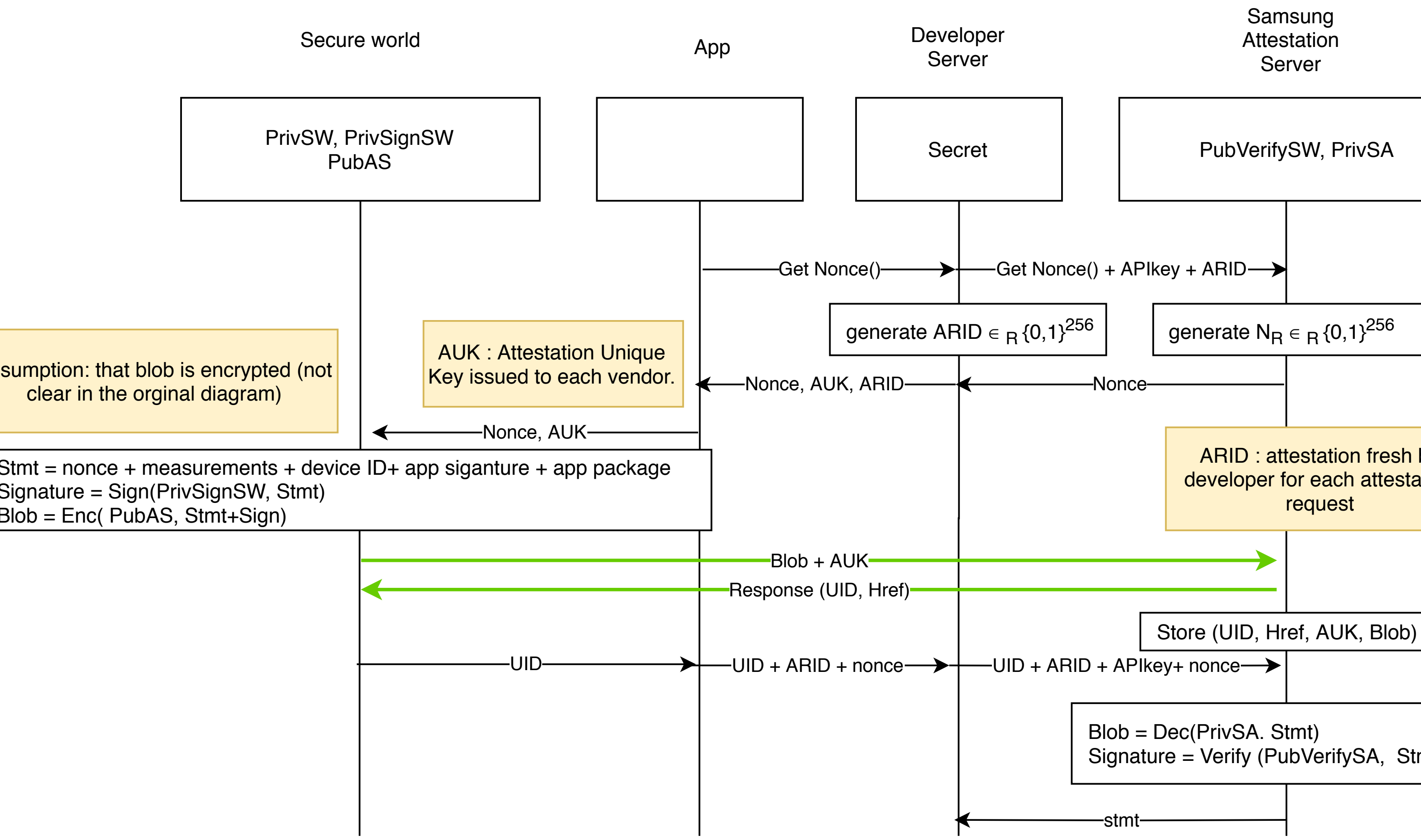
Assumption
encrypted (n
orginal

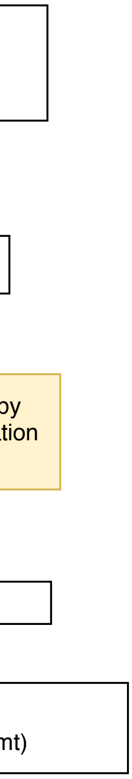
Stmt = n
Signature
Blob = E

Knox attestation v2



Knox attestation v3


Secure world
App
Developer Server
Samsung Attestation Server
PrivSW, PrivSignSW
PubAS
Secret
PubVerifySW, PrivSA
Get Nonce()
Get Nonce() + APIkey + ARID
generate ARID $\in_R \{0,1\}^{256}$
generate $N_R \in_R \{0,1\}^{256}$
sumption: that blob is encrypted (not clear in the orginal diagram)
AUK : Attestation Unique Key issued to each vendor.
Nonce, AUK, ARID
Nonce
Nonce, AUK
ARID : attestation fresh l developer for each attesta request
Stmt = nonce + measurements + device ID+ app siganture + app package
Signature = Sign(PrivSignSW, Stmt)
Blob = Enc(PubAS, Stmt+Sign)
Blob + AUK
Response (UID, Href)
Store (UID, Href, AUK, Blob)
UID
UID + ARID + nonce
UID + ARID + APIkey+ nonce
Blob = Dec(PrivSA. Stmt)
Signature = Verify (PubVerifySA, Str
stmt


py
tion
mt)