CYBERNETICA

Euroopa Liit
Euroopa
Regionaalarengu Fond

Eesti
tuleviku heaks

CDOC 2.0 spetsifikatsioon

Tehniline dokument

Version 0.9

31.01.2023

ID D-19-12

Kuupäev	Versioon	Kirjeldus
28.09.2022	0.5	Lisatud RSA kapseldus. Uuendatud lisad.
14.11.2022	0.6	Uuendatud lisas OpenAPI kirjeldust.
25.11.2022	0.7	Põhitekst ja lisad andmestruktuuride kirjelduse ja nimetamise osas vastavusse viidud. Läbivalt kasutusele võetud `Capsule` nimetus `Details` asemel.
16.12.2022	0.8	Lisatud viited etalonrealisatsiooni lähtekoodile
31.01.2023	0.9	Täpsustatud tar formaadi kirjeldust. Lisatud LRO keelatud märkide nimekirja

Sisukord

1 Sissejuhatus

1.1 Eesmärk

Selle spetsifikatsiooni eesmärk on kirjeldada failide salastamiseks mõeldud CDOC 2.0 andmevorming.

1.2 Käsitlusala

Spetsifikatsioon kirjeldab:

- toetatud krüpteerimisskeemid
- abstraktse ja jadastatud andmevormingu
- krüptograafiliste operatsioonide detailid
- võtmeedastusserveri kasutamine
- realiseerimisjuhised

1.3 Määratlused ja lühendid

CDOC
: „Crypto Digidoc“, Eesti eID ökosüsteemis kasutatav vorming krüpteeritud failide edastamiseks.

CDOC 1.0
: mitteametlik termin kõigi sellele spetsifikatsioonile eelnevate (XML-ENC põhiste) CDOC vormingute tähistamiseks.

CDOC 2.0
: selles spetsifikatsioonis kirjeldatud CDOC vormingu versioon.

CEK
: *Content Encryption Key*, sisukrüpteerimisvõti. Võti, mida kasutatakse konteineri lasti vahetuks krüpteerimiseks.

ECC
: *Elliptic Curve Cryptography*, elliptkõverate krüptograafia.

ECDH
: *Elliptic-curve Diffie–Hellman*, võtmekehtestusprotokoll

FMK
: *File Master Key*, alusvõti, millest moodustatakse sisukrüpteerimisvõti CEK.

HKDF
: *HMAC-based Key Derivation Function*, funktsioon, mis piisava entroopiaga sisendil annab välja krüptograafilisteks operatsiooniseks sobivaid võtmeid.

KEK
: *Key Encryption Key*, võtmekrüpteerimisvõti, millega krüpteeritakse sisuktüpteerimisvõti CEK.

KEM
: *key encapsulation mechanism*, võtmekapseldusmehhanism

Konteiner
: üksik CDOC 2.0 vormingu alusel koostatud fail.

Last
: *Payload*, CDOC 2.0 vormingu abil edastatav failide kogum. See mõiste tähistab seda kogumit avateksti kujul.

OAEP
: *Optimal Asymmetric Encryption Padding*, täidistusskeem, kasutatav koos RSA krüpteerimise jaoks

Päis
: *Header*, CDOC 2.0 vormingu osa, mis kirjeldab konteineri vastuvõtjaid ja rakendatavaid krüptograafilisi meetmeid.

RSA
: *Rivest–Shamir–Adleman*, avaliku võtmega krüptosüsteem

Vastuvõtja
: osapool, kellele on CDOC 2.0 konteiner suunatud ning kelle kontrolli all on võtmematerjalid, mille abil saab konteineri sisu dekrüpteerida.

Vastuvõtja tunnus
: krüptograafiline avalik võti, isikukood, sertifikaadi omaniku eraldusnimi või muu andmeelement, mille abil on võimalik otsustada, kas mingi osapool on konkreetse konteineri vastuvõtjate hulgas.

Võtmekapsel
: CDOC 2.0 vormingu osa, mis sisaldab krüpteeritud lasti dekrüpteerimiseks vajaliku võtit, selle võtme osa või selle võtme vastuvõtja poolt arvutamiseks vajalike andmeid. Saatja võib võtmekapsli lisada ümbrikule või edastada võtmeedastusserveri kaudu.

Ümbrik
: CDOC 2.0 vormingu kõige välimine kiht, mis kirjeldab päise ja lasti kodeerimist ühtseks konteineriks.

1.4 Viited

[1] *Encrypted DigiDoc Format Specification*. AS Sertifitseerimiskeskus, https://www.id.ee/wp-content/uploads/2020/06/sk-cdoc-1.0-20120625_en.pdf. Juuni 2012.

[2] *Required modifications to CDOC for elliptic curve support*. Cybernetica AS, Report number A-101-7, https://www.ria.ee/media/1974/download. September 2017.

[3] Kristjan Krips, Mart Oruaas ja Jan Willemson. *CDOC 2.0 analüüs*. Tehniline raport. Cybernetica, 2020.

[4] Matús Nemec *et al.* "The Return of Coppersmith's Attack: Practical Factorization of Widely Used RSA Moduli". Teoses: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*. Toim. Bhavani Thuraisingham *et al.* ACM, 2017, lk. 1631–1648. DOI: 10.1145/3133956.3133969. URL: https://doi.org/10.1145/3133956.3133969.

[5] The IEEE and The Open Group. *Portable archive interchange*. https://pubs.opengroup.org/onlinepubs/9699919799/utilities/pax.html. 2018.

[6] L. Peter Deutsch ja Jean-loup Gailly. *ZLIB Compressed Data Format Specification version 3.3*. RFC 1950. Mai 1996. URL: https://rfc-editor.org/rfc/rfc1950.txt.

[7] MITRE. *CAPEC-126: Path Traversal*. https://capec.mitre.org/data/definitions/126.html. 2021.

[8] SEI CERT. *SEI CERT Oracle Coding Standard for Java*. https://wiki.sei.cmu.edu/confluence/display/java/IDS04-J.+Safely+extract+files+from+ZipInputStream. 2018.

[9] *Control character*. https://en.wikipedia.org/wiki/Control_character. 2022.

[10] The Linux Foundation. *OpenAPI Specification v3.1.0*. https://spec.openapis.org/oas/latest.html. 2022.

[11] *Algorithms, Key Size and Protocol Report*. ECRYPT CSA D5.4. Veebruar 2018. URL: https://www.ecrypt.eu.org/csa/documents/D5.4-FinalAlgKeySizeProt.pdf.

[12] Elaine Barker. *Recommendation for Key Management: Part 1 – General*. 2020. DOI: 10.6028/NIST.SP.800-57pt1r5.

[13] *Cryptographic Mechanisms: Recommendations and Key Lengths*. BSI - Technical Guideline TR-02102-1. Jaanuar 2022. URL: https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/TechGuidelines/TG02102/BSI-TR-02102-1.pdf.

[14] *Krüptoalgoritmid ning nende tugi teekides ja infosüsteemides*. Cybernetica AS, Report number T-184-7, https://www.ria.ee/media/1473/download. Märts 2021.

[15] Hugo Krawczyk. *Cryptographic Extraction and Key Derivation: The HKDF Scheme*. Cryptology ePrint Archive, Report 2010/264. https://ia.cr/2010/264. 2010.

[16] Gordon Procter. *A Security Analysis of the Composition of ChaCha20 and Poly1305*. Cryptology ePrint Archive, Report 2014/613. https://ia.cr/2014/613. 2014.

[17] Dr. Hugo Krawczyk ja Pasi Eronen. *HMAC-based Extract-and-Expand Key Derivation Function (HKDF)*. RFC 5869. Mai 2010. URL: https://rfc-editor.org/rfc/rfc5869.txt.

[18] Dr. Hugo Krawczyk, Mihir Bellare ja Ran Canetti. *HMAC: Keyed-Hashing for Message Authentication*. RFC 2104. Veebruar 1997. URL: https://rfc-editor.org/rfc/rfc2104.txt.

[19] Eric Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.3*. RFC 8446. August 2018. URL: https://rfc-editor.org/rfc/rfc8446.txt.

[20] *Digital Signature Standard (DSS)*. National Institute of Standards and Technology, https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf. 2013.

[21] K. Moriarty *et al. PKCS #1: RSA Cryptography Specifications Version 2.2*. RFC 8017. RFC. November 2016.

[22] Yoav Nir ja Adam Langley. *ChaCha20 and Poly1305 for IETF Protocols*. RFC 8439. Juuni 2018. DOI: 10.17487/RFC8439. URL: https://www.rfc-editor.org/info/rfc8439.

1.5 Ülevaade

[This section should describe what the rest of the document contains and explain how the document is organized.]

2 Ülesandepüstitus

CDOC 2.0 lahendab CDOC 1.0 spetsifikatsioonide [1, 2] ning realisatsioonidega seotud probleeme, lähtudes varasemas analüüsis [3] välja pakutud lahendusvariantidest.

CDOC 2.0 lahendab järgmised CDOC 1.0 probleemid:

- CDOC 1.0 ei paku mingit tulevikuturvalisust. Ründaja, kes on salvestanud CDOC konteineri, saab selle tulevikus avada, kui kompromiteeruvad konteineri loomiseks kasutatud võtmed või krüptoalgoritmid. ROCA juhtum [4] näitas, et tegu ei ole pelgalt teoreetilise probleemiga.
- CDOC 1.0 vorming ja selle töötluseks loodud tarkvara ei erista dokumendi krüpteerimist transpordiks osapoolte vahel ning dokumendi krüpteerimist säilitamiseks ühe osapoole poolt.
- CDOC 1.0 vorming ei võimalda saata krüpteeritud dokumenti vastuvõtjale, kes kasutab vaid mobiilset eID vahendit (Mobiil-ID või Smart-ID).

2.1 CDOC 2.0 krüpteerimisskeemid

Spetsifikatsioon kirjeldab järgmised krüpteerimisskeemid:

SC.01
: Krüpteerimine vastuvõtja ECC ID-kaardi avalikule võtmele, kusjuures kogu selle käigus tekkiv ECC KEM (*Key Encapsulation Mechanism*) materjal on CDOC konteineris kaasas. Ei paku täiendavat turvalisust, võrreldes CDOC1.0 vorminguga. Kasutatav nii transpordi- kui ka säilituskrüptograafia jaoks.

SC.02
: Krüpteerimine vastuvõtja ECC ID-kaardi avalikule võtmele, kusjuures kogu selle käigus tekkiv ECC KEM materjal edastatakse läbi võtmeedastusserveri. Pakub täiendavat turvalisust (osaline tulevikuturvalisus) eeldusel, et CDOC konteineri edastamiseks kasutatav sidekanal ning võtmeedastusserveri sisu ega sidekanalid ei ole mõlemad ründaja poolt loetavad. Kasutatav transpordikrüptograafia jaoks.

SC.03
: Krüpteerimine vastuvõtja RSA krüptopulga avalikule võtmele, kusjuures kogu selle käigus tekkiv RSA-ga krüpteeritud võtmematerjal on CDOC konteineris kaasas. Ei paku täiendavat turvalisust, võrreldes CDOC1.0 vorminguga. Kasutatav nii transpordi- kui ka säilituskrüptograafia jaoks.

SC.04
: Krüpteerimine vastuvõtja RSA krüptopulga avalikule võtmele, kusjuures kogu selle käigus tekkiv RSA-ga krüpteeritud võtmematerjal edastatakse läbi võtmeedastusserveri. Pakub täiendavat turvalisust (osaline tulevikuturvalisus) eeldusel, et CDOC konteineri edastamiseks kasutatav sidekanal ning võtmeedastusserveri sisu ega sidekanalid ei ole mõlemad ründaja poolt loetavad. Kasutatav transpordikrüptograafia jaoks.

SC.05
: Krüpteerimine sümmeetrilisele võtmele. Kasutatav nii transpordi- kui ka säilituskrüptograafia jaoks. Transpordikrüptograafia jaoks kasutamine eeldab võtme eelnevat edastamist pealtkuulamiskindlate süsteemiväliste kanalite kaudu.

2.2 Eeldused ja nõuded sidekanalitele

Sarnaselt eelmisele spetsifikatsioonile CDOC 1.0 pakub CDOC 2.0 kaitset andmete edastamiseks avalike ning potentsiaalselt ründajale loetavate sidekanalite (näiteks elektronpost, USB-mäluseade, failivahetusteenused) kaudu.

CDOC 2.0 ei tee mingeid eeldusi konteineri edastamiseks kasutatava sidekanali kohta ning tagab selles olevate andmete salastatuse konteineri edastamise ajal. Üldjuhul ei taga CDOC 2.0 edastatud sõnumi tulevikuturvalisust – st konteineri kinni püüdnud ja salvestanud ründaja võib saada selle tulevikus dekrüpteerida.

CDOC 2.0 kirjeldab ka osalist tulevikuturvalisust pakkuva krüpteerimisskeemid (SC.02 ja SC.04), mille korral osa võtmematerjalist edastatakse saatjalt vastuvõtjale võtmeedastusserveri(te) kaudu. Võtmeedastusserveritele ning nende sidekanalitele esitatavad nõuded on kirjeldatud jaotises 4 iga skeemi kohta eraldi.

2.3 Käsitlusalasse mittekuuluv funktsionaalsus

CDOC 2.0 ei paku lahendusi järgmistele vajadustele:

- Saatja autentimine. CDOC 2.0 ei autendi saatjat. Saatja autentimiseks võib krüpteeritava informatsiooni enne krüpteerimist signeerida.

3 Krüpteerimisskeemid

See jaotis esitab kõik toetatavad krüpteerimisskeemid abstraktsel kujul, kirjeldades osapoolte vahelise sõnumivahetuse ning vahetatavate sõnumite sisu. Jaotis on kasulik ülevaate saamiseks eri skeemide toimepõhimõtetest.

Kõigis stsenaariumites tahab saatja (Alice, A) saata sõnumit M krüpteeritult vastuvõtjale (Bob, B). Ta võib seda teha kas otse või kasutades võtmeedastusserveri S (või serverite $S_1, S_2, \ldots, S_n$) abi. Ka vastuvõtjaid võib üldiselt olla mitu, sel juhul tähistame neid $B_1, B_2, \ldots, B_\ell$.

Alice kasutab sümmeetrilist krüptosüsteemi Sym, milles on järgnevad komponendid.

1. Võtmegenereerimisalgoritm $GenKey_{Sym}$ – sellega genereeritakse salajane võti. Vt jaotist 6.3.2.1.
2. Krüpteerimisalgoritm Enc_{Sym} – see on funktsioon, mis nõuab argumendiks võtit ja sisendit (mida soovime krüpteerida) ning väljastab krüptogrammi.
3. Dekrüpteerimisalgoritm Dec_{Sym} – see on funktsioon, mis nõuab argumendiks võtit ja krüptogrammi. Kui argumendiks antakse võti, millega krüptogramm on krüpteeritud, siis väljundiks on algne sisend. Vastasel juhul on väljundiks juhuslik võimalik sisend.

Võtmete genereerimisel kasutatakse HKDF (*extract, then expand*) konstruktsiooni. HKDF-i *extract* faasis genereeritakse funktsiooniga $GenKeyExtract_{Sym}$ alusvõti (*File Master Key, FMK*), millest edasi tuletatakse *expand* faasis funktsiooniga $GenKeyExpand_{Sym}$ sisukrüpteerimisvõti (*Content Encryption Key, CEK)*. Viimast kasutatakse sümmeetrilisel krüpteerimisel salajase võtmena. Sümmeetrilisel krüpteerimisel ja dekrüpteerimisel kasutatakse algoritmi ChaCha20-Poly1305. Detailsemad selgitused leiab sektsioonidest 6.2 ja 6.6.

Alice krüpteerib alusvõtme iga vastuvõtja jaoks eraldi, kasutades vastuvõtja-kohast sümmeetrilist võtmekrüpteerimisvõtit (*KEK*).

Alljärgnevalt kirjeldatud stsenaariumid erinevadki üksteisest selles, kuidas luuakse *KEK* ning kuidas edastatakse krüpteeritud alusvõtit *FMK* sisaldavad võtmekapslid vastuvõtjatele.

3.1 Otsesuhtlusega ECDH skeem

See skeem on kasutatav krüpteeritud sõnumite saatmiseks vastuvõtjatele, kes valdavad ECC privaatvõtit.

Alice kaitseb sõnumisaladust võtmekapseldusmehhanismiga (*Key Encapsulation Mechanism, KEM*), mis koosneb algoritmidest $Encaps_{KEM}$ ja $Decaps_{KEM}$.

Enne kapseldamist saab Alice teada vastuvõtja avaliku võtme kui elliptkõvera punkti. Edasi jooksutab Alice vastuvõtjaga ECDH võtmekehtestusprotokolli.

Kapseldusfunktsioon $Encaps_{KEM}$ võtab sisendiks vastuvõtja avaliku võtme ning väljastab võtme ja kapsli. Võtmeks on võtmekrüpteerimisvõti KEK, mille tuletamine on kirjeldatud jaotises 6.3.1 ja kapsliks $caps$ ECDH realisatsioonis Alice'i efemeerne avalik võti.

Lahtikapseldusfunktsioon $Decaps_{KEM}$ võtab sisendiks võtmekapsli ja vastuvõtja salajase võtme, kontrollib, et saadetud elliptkõvera punkt on korrektne, viib läbi ECDH võtmekehtestusprotokolli teise osapoole tegevuse ning tuletab KEKi. Täpsem selgitus on jaotises 6.3.1, alajaotises "KEKi arvutamine dekrüpteerimise käigus".

Kasutatav otsesuhtlusega skeem:

1. $A : fmk \leftarrow GenKeyExtract_{Sym}(Nonss)$
2. $A : cek \leftarrow GenKeyExpand_{Sym}(fmk)$
3. $A : c \leftarrow Enc_{Sym}(cek, M)$
4. A hangib vastuvõtjate $B_1, B_2, \ldots, B_\ell$ avalikud võtmed $PK_1, PK_2, \ldots, PK_\ell$; vastuvõtjatel on vastavad salajased võtmed $SK_1, SK_2, \ldots, SK_\ell$
5. $A : (kek_i, caps_i) \leftarrow Encaps_{KEM}(PK_i)\ (i = 1, 2, \ldots, \ell)$
6. $A : ck_i \leftarrow XOR(kek_i, fmk)\ (i = 1, 2, \ldots, \ell)$
7. $A \rightarrow B_i : c, ck_i, caps_i\ (i = 1, 2, \ldots, \ell)$ [1]
8. $B_i : kek_i \leftarrow Decaps_{KEM}(caps_i, SK_i)$
9. $B_i : fmk \leftarrow XOR(kek_i, ck_i)$ [2]
10. $B_i : cek \leftarrow GenKeyExpand_{Sym}(fmk)$
11. $B_i : M \leftarrow Dec_{Sym}(cek, c)$

3.2 Võtmeedastusserveriga ECDH skeem

Ka selles skeemis kaitseb Alice sõnumisaladust ECDH võtmekapseldusmehhanismiga, kuid ainsa erinevusena edastatakse siin kapsel võtmeedastusserveri kaudu, pakkudes sellega lisaturvalisust, eeldusel, et võtmeedastusserver käitub reeglipäraselt. Täiendava turvalisuse tagab autentimisprotokolli $Auth$ kasutamine, mille abil saab server vastuvõtja autentida.

Kasutatav võtmeedastusserveriga skeem:

1. $A : fmk \leftarrow GenKeyExtract_{Sym}(Nonss)$
2. $A : cek \leftarrow GenKeyExpand_{Sym}(fmk)$
3. $A : c \leftarrow Enc_{Sym}(cek, M)$
4. A hangib vastuvõtjate $B_1, B_2, \ldots, B_\ell$ avalikud võtmed $PK_1, PK_2, \ldots, PK_\ell$; vastuvõtjatel on vastavad salajased võtmed $SK_1, SK_2, \ldots, SK_\ell$
5. $A : (kek_i, caps_i) \leftarrow Encaps_{KEM}(PK_i)\ (i = 1, 2, \ldots, \ell)$
6. $A : ck_i \leftarrow XOR(kek_i, fmk)\ (i = 1, 2, \ldots, \ell)$
7. $A \rightarrow B_i : c, ck_i\ (i = 1, 2, \ldots, \ell)$
8. $A \rightarrow S : caps_i\ (i = 1, 2, \ldots, \ell)$
9. $B_i \rightarrow S : Auth$ [3]
10. $S \rightarrow B_i : caps_i$ [4]
11. $B_i : kek_i \leftarrow Decaps_{KEM}(caps_i, SK_i)$
12. $B_i : fmk \leftarrow XOR(kek_i, ck_i)$ [5]
13. $B_i : cek \leftarrow GenKeyExpand_{Sym}(fmk)$
14. $B_i : M \leftarrow Dec_{Sym}(cek, c)$

[1] Info kätte saamisel kontrollib vastuvõtja, et saadetud elliptkõvera punkt on korrektne. Kui mitte, väljastatakse veateade “saatja genereeritud efemeerne avalik võti ei ole korrektne” ja protokoll seiskub.

[2] Peale fmk dekrüpteerimist kontrollib vastuvõtja, et päisest saadud sõnumiautentimiskood on korrektne (täpsem info jaotises 6.5). Kui mitte, väljastatakse veateade “sõnumi autentimine ebaõnnestus” ja protokoll seiskub.

[3] Server autendib vastuvõtja. Server tagastab vastuvõtjale vaid talle saadetud võtmekapsli.

[4] Vt allmärkust 1.

[5] Vt allmärkust 2.

3.3 Otsesuhtlusega RSA-OAEP skeem

See skeem on kasutatav krüpteeritud sõnumite saatmiseks vastuvõtjatele, kes valdavad RSA privaatvõtit.

Alice soovib sõnumisaladust kaitsta RSA-OAEP skeemiga, mis kasutab krüpteerimisalgoritmi Enc_{RSA} ja dekrüpteerimisalgoritmi Dec_{RSA}.

Võtmeedastusvõtme salastamiseks krüpteerib saatja selle vastuvõtja avaliku võtmega. Võtmekapsliks on saadud krüptogramm, mille vastuvõtja dekrüpteerib oma privaatvõtmega.

1. $A : fmk \leftarrow GenKeyExtract_{Sym}(Nonss)$
2. $A : cek \leftarrow GenKeyExpand_{Sym}(fmk)$
3. $A : c \leftarrow Enc_{Sym}(cek, M)$
4. $A : kek_i \leftarrow GenKey_{Sym}\ (i = 1, 2, \ldots, \ell)$
5. $A : ck_i \leftarrow XOR(kek_i, fmk)\ (i = 1, 2, \ldots, \ell)$
6. A hangib vastuvõtjate $B_1, B_2, \ldots, B_\ell$ avalikud võtmed $PK_1, PK_2, \ldots, PK_\ell$; vastuvõtjatel on vastavad salajased võtmed $SK_1, SK_2, \ldots, SK_\ell$
7. $A : caps_i \leftarrow Enc_{RSA}(PK_i, kek_i)\ (i = 1, 2, \ldots, \ell)$
8. $A \rightarrow B_i : c, ck_i, caps_i\ (i = 1, 2, \ldots, \ell)$
9. $B_i : kek_i \leftarrow Dec_{RSA}(SK_i, caps_i)$
10. $B_i : fmk \leftarrow XOR(kek_i, ck_i)$
11. $B_i : cek \leftarrow GenKeyExpand_{Sym}(fmk)$
12. $B_i : M \leftarrow Dec_{Sym}(cek, c)$

3.4 Võtmeedastusserveriga RSA-OAEP skeem

Sarnane eelmise skeemiga, kuid saatja edastab võtmekapsli vastuvõtjale võtmeedastusserveri kaudu.

1. $A : fmk \leftarrow GenKeyExtract_{Sym}(Nonss)$
2. $A : cek \leftarrow GenKeyExpand_{Sym}(fmk)$
3. $A : c \leftarrow Enc_{Sym}(cek, M)$
4. $A : kek_i \leftarrow GenKey_{Sym}\ (i = 1, 2, \ldots, \ell)$
5. $A : ck_i \leftarrow XOR(kek_i, fmk)\ (i = 1, 2, \ldots, \ell)$
6. $A \rightarrow B_i : c, ck_i\ (i = 1, 2, \ldots, \ell)$
7. A hangib vastuvõtjate $B_1, B_2, \ldots, B_\ell$ avalikud võtmed $PK_1, PK_2, \ldots, PK_\ell$; vastuvõtjatel on vastavad salajased võtmed $SK_1, SK_2, \ldots, SK_\ell$
8. $A : caps_i \leftarrow Enc_{RSA}(PK_i, kek_i)\ (i = 1, 2, \ldots, \ell)$
9. $A \rightarrow S : caps_i\ (i = 1, 2, \ldots, \ell)$
10. $B_i \rightarrow S : Auth$
11. $S \rightarrow B_i : caps_i$
12. $B_i : kek_i \leftarrow Dec_{RSA}(SK_i, caps_i)$
13. $B_i : fmk \leftarrow XOR(kek_i, ck_i)$

14. $B_i : cek \leftarrow GenKeyExpand_{Sym}(fmk)$
15. $B_i : M \leftarrow Dec_{Sym}(cek, c)$

3.5 Sümmeetrilise võtmega skeem

Alice kaitseb saladust võtmetuletusmehhanismiga, mis koosneb algoritmidest $Encaps_{HKDF}$ ja $Decaps_{HKDF}$. Enne kapseldamist teab Alice vastuvõtja sümmeetrilist salajast võtit ning selle nime (nime lepivad saatja ja vastuvõtja omavahel kokku, nimi aitab eri võtmeid eristada). Kapseldusfunktsioon $Encaps_{HKDF}$ võtab sisendiks vastuvõtja sümmeetrilise salajase võtme ning selle nime ning väljastab võtme ja kapsli. Võtmeks on võtmekrüpteerimisvõti KEK, mille tuletamine on kirjeldatud jaotises 6.3.4 ja kapsliks $caps$ dekrüpteerimisvõtme märgendit ning võtme tuletamisel kasutatud juhuarvu sisaldav andmestruktuur. Lahtikapseldusfunktsioon $Decaps_{HKDF}$ võtab sisendiks vastuvõtja sümmeetrilise salajase võtme, selle nime ja võtmekapsli. Õigete sisendite korral tuletab sealt KEKi. Täpsem selgitus on jaotises 6.3.4, alajaotises ”KEKi arvutamine dekrüpteerimise käigus”.

Kasutatav sümmeetrilise võtmega skeem:

1. $A : fmk \leftarrow GenKeyExtract_{Sym}(Nonss)$
2. $A : cek \leftarrow GenKeyExpand_{Sym}(fmk)$
3. $A : c \leftarrow Enc_{Sym}(cek, M)$
4. A valduses on sümmeetrilised võtmed $S_1, S_2, \ldots, S_\ell$ mille märgendid on $L_1, L_2, \ldots, L_\ell$
5. $A : (kek_i, caps_i) \leftarrow Encaps_{HKDF}(S_i, L_i)\ (i = 1, 2, \ldots, \ell)$
6. $A : ck_i \leftarrow XOR(kek_i, fmk)\ (i = 1, 2, \ldots, \ell)$
7. $A \rightarrow B_i : c, ck_i, caps_i\ (i = 1, 2, \ldots, \ell)$
8. $B_i : kek_i \leftarrow Decaps_{HKDF}(caps_i, S_i)$
9. $B_i : fmk \leftarrow XOR(kek_i, ck_i)$ [6]
10. $B_i : cek \leftarrow GenKeyExpand_{Sym}(fmk)$
11. $B_i : M \leftarrow Dec_{Sym}(cek, c)$

3.6 Turvaeeldused

Krüpteerimise kõige üldisem turvaeesmärk on, et

> *keegi peale määratud vastuvõtja (B) ei suuda sõnumit M dekrüpteerida.*

Selle eesmärgi saavutamiseks tuleb vaadeldavate skeemide korral lubada vastavaid eeldusi. Alati tuleb lubada, et

> *kasutatav sümmeetriline krüptosüsteem Sym ei murdu.*

Asümmeetrilist krüptograafiat kasutav skeem jaotisest 3.1 kasutab eeldust, et

> *kasutatav asümmeetriline krüptoalgoritm ei murdu.*

Asümmeetrilist krüptograafiat kasutav skeem jaotisest 3.2 kasutab eeldust, et

[6] Vt allmärkust 2.

kasutatav asümmeetriline krüptoalgoritm ei murdu.

või

võtmeedastusserver toimib reeglitepäraselt.

4 Konteinervorming

CDOC 2.0 konteinervormingu kirjeldus on jagatud kaheks: abstraktseks ja konkreetseks osaks. Abstraktne osa kirjeldab andmeelemendid ja nendevahelised seosed, konkreetne osa kirjeldab, kuidas neid andmeelemente jadastada.

4.1 Abstraktne vorming

See jaotis kirjeldab CDOC 2.0 vormingut abstraktsest vaatest, andes andmekoosseisud ja andmemudelid, kuid mitte jadastatud vormingu kirjeldust.

4.1.1 Aluspõhimõtted

Siin toodud põhimõtted annavad spetsifikatsiooni kasutajale pidepunkte detailsema spetsifikatsiooni mõtestamiseks.

- Abstraktne vorming koosneb päisest ja krüpteeritud lastist.
- Abstraktne vorming sisaldab üht krüpteeritud lasti, mille sisse on krüpteeritud üks kuni mitu faili. Info nende failide nimede kohta, samuti see, millised on mitme faili puhul nende suurused ja järjestused, on samuti krüpteeritud.
- Last on krüpteeritud ühe sümmeetrilise võtmega (*Content Encryption Key*; CEK), kasutates AEAD (*Authenticated Encryption with Additional Data*) krüpteerimisviisi.
- CEK saadakse võtmepärimise teel CDOC faili alusvõtmest (*File Master Key*; FMK). Vt jaotis 6.2.
- FMK võib olla paralleelselt krüpteeritud ühe kuni mitme võtmekrüpteerimisvõtmega (*Key Encryption Key*; KEK), üks iga vastuvõtja kohta. KEKi genereerimise kohta vt jaotis 6.3.
- Päis kirjeldab seda, kuidas on FMK kaitstud (kuidas saavad vastuvõtjad FMK dekrüpteerimiseks vajaliku KEK-i omandada).
- Päise terviklus tagatakse sõnumiautentimiskoodiga, mille arvutamiseks kasutatakse FMK-st päritud sõnumiautentimisvõtit (*Header HMAC Key*; HHK). Vt. jaotis 6.5.
- Vormingu universaalsuse tagamiseks ei ole selle kirjeldamise juures kasutatud otseselt Eesti eID infrastruktuuri elemente. Näiteks kirjeldatakse vastuvõtjat tema avaliku võtme, mitte sertifikaadi abil.
- Dekrüpteerimine vastab alati ühele mustrile: 1) vastuvõtja omandab KEK-i, 2) vastuvõtja dekrüpteerib FMK, 3) vastuvõtja pärib HHK ning valideerib päise, 4) vastuvõtja pärib CEK-i, 5) vastuvõtja dekrüpteerib lasti.

4.1.2 Päise struktuur

Päise struktuuri kirjeldamiseks kasutatakse pseudokoodi, mis ei vasta ühelegi programmeerimisega skeemikeelele, kuid mis võiks olla intuitiivselt mõistetav.

Päis koosneb ühest kuni mitmest vastuvõtjat kirjeldavast struktuurist. Iga vastuvõtja struktuur sisaldab täielikku infot selle kohta, kuidas konkreetne vastuvõtja saab (isiku tuvastamisel, isiklikule krüptomaterjalile ligipääsemisel jne) FMK-le ligipääsu.

Päisele arvutatakse sõnumiautentimiskood, kasutades FMK-st tuletatud võtit. See on vajalik, et vältida päise manipuleerimist selle edastajate poolt, näiteks osade vastuvõtjate varjamiseks. Sõnumiautentmiskood arvutatakse konkreetsel viisil jadastatud (vt jaotis 4.2) päisele.

```
    Header = {
        Recipients              = :Recipient[](1..k)
        PayloadEncryptionMethod = :enum(CHACHA20-POLY1305)
    }
```

Päisele arvutatakse sõnumiautentimiskood (detaile vaata jaotisest 6.5):

```
    Checksum = {
        value = HMAC(HHK, Serialize(Header))
    }
```

Vastuvõtja kirjeldatakse struktuuriga `Recipient`. Struktuuri ülesehitus annab lugejale võimaluse kiirelt ja üheselt otsustada, kas tal on olemas võimalus konkreetse `Recipient`'i eksemplari alusel lasti dekrüpteerida.

```
    Recipient = {
        Capsule = Union(:ECCPublicKeyCapsule | :KeyServerCapsule |
                :SymmetricKeyCapsule | :RSAPublicKeyCapsule )
        KeyLabel = :string
        EncryptedFMK = :byte[]
        FMKEncryptionMethod = :enum(XOR)
    }
```

Struktuur `Recipient` koosneb võtmekapslist, vastuvõtja võtme nimest, krüpteeritud FMK-st ning FMK krüpteerimise meetodi tunnusest.

- `Capsule` – krüpteerimisskeemi-spetsiifilised andmed, millede abil vastuvõtja saab dekrüpteerida FMK.
- `KeyLabel` – FMK dekrüpteerimiseks vajaliku privaat- või salajase võtme inimloetav nimi. Selle olemasolu on vajalik mõistliku kasutajaliidese ehitamiseks. Saatja täidab selle välja võtme või sellega seotud sertifikaadi põhjal. Spetsifikatsioon ei täpsusta, kuidas seda tehakse, kuna see ei ole krüptograafilise töötluse jaoks oluline. Vorminguks on UTF-8 string.
- `EncryptedFMK` – krüpteeritud FMK.
- `FMKEncryptioMethod` – FMK krüpteerimise meetodid tüüp.

Struktuuri `Capsule` eduka töötlemise tulemuseks on krüptograafiline võti, millega dekrüpteerida FMK, kasutades meetodit, mille määrab väli `FMKEncryptionMethod`. Krüptograafiliste operatsioonide detaile vaata jaotisest 6.4.

Erinevate krüpteerimisskeemide (jaotis 3) toetamiseks on spetsifitseeritud järgmised võtmekapslite tüübid.

- `ECCPublicKeyCapsule` – Vastuvõtja tunnuseks on ECC avalik võti `RecipientPublicKey` (näiteks ID-kaardi esimese võtmepaari avalik võti). KEK päritakse ECDH abil. Kasutatakse krüpteerimisskeemi SC.01 korral. Vt jaotis 3.1.
- `RSAPublicKeyCapsule` – Vastuvõtja tunnuseks on RSA avalik võti `RecipientPublicKey`. KEK saadakse võtmekapsli dekrüpteerimisel RSA privaatvõtme abil. Kasutatakse krüpteerimisskeemi SC.03 korral. Vt jaotis 3.3.

- `KeyServerCapsule` – Vastuvõtja tunnuseks on ECC või RSA avalik võti `RecipientPublicKey`, millega vastuvõtja end võtmeedastusserverile autendib. Võtmeedastusserver väljastab objekti `ECCPublicKeyCapsule` või `RSAPublicKeyCapsule`, mida kasutatakse vastavalt selle kirjeldusele. Kasutatakse krüpteerimisskeemide SC.02 ja SC.04 korral. Vt jaotis 3.2 ja jaotis 3.4.
- `SymmetricKeyCapsule` – Vastuvõtja tunnuseks on võtme nimi `KeyLabel`. KEK päritakse HKDF abil sümmeetrilisest võtmest, mille kasutaja ette annab. Kasutatakse krüpteerimisskeemi SC.05 korral. Vt jaotis 3.5.

Spetsifikatsiooni tulevased versioonid võivad seda loetelu täiendada.

4.1.3 Võtmekapslite tüübid

ECC avaliku võtme kapsel. Vastuvõtja tunnuseks on ECC avalik võti `RecipientPublicKey`.

```
    ECCPublicKeyCapsule = {
        Curve              = :enum(secp384r1)
        RecipientPublicKey = :byte[]
        SenderPublicKey    = :byte[]
    }
```

- `Curve` – kasutatava elliptkõvera tunnus.
- `RecipientPublicKey` – vastuvõtja ECC avalik võti, mille alusel vastuvõtja leiab talle mõeldud vastuvõtja kirje.
- `SenderPublicKey` – saatja avalik võti, mida vastuvõtja kasutab ECDH abil KEK tuletamiseks.

RSA avaliku võtme kapsel. Vastuvõtja tunnuseks on ECC avalik võti `RecipientPublicKey`.

```
    RSAPublicKeyCapsule = {
        RecipientPublicKey = :byte[]
        EncryptedKEK       = :byte[]
    }
```

- `RecipientPublicKey` – vastuvõtja RSA avalik võti, mille alusel vastuvõtja leiab talle mõeldud vastuvõtja kirje.
- `EncryptedKEK` – vastuvõtja avaliku võtmega krüpteeritud võtmeedastusvõti.

Võtmeserveri kapsel. Vastuvõtja tunnuseks on tema ECC või RSA avalik võti `RecipientPublicKey`.

```
    KeyServerCapsule = {
        RecipientKey = Union(:EccKeyDetails | :RsaKeyDetails)
        KeyServerID         = :string
        TransactionID       = :string
    }

    RsaKeyDetails = {
        RecipientPublicKey  = :byte[]
    }

    EccKeyDetails = {
         Curve              = :enum(secp384r1)
         RecipientPublicKey = :byte[]
    }
```

- `RecipientKey` – vastuvõtja võtme, millega vastuvõtja end võtmeedastusserverile autendib, andmed.
- `KeyServerID` – võtmeedastusserveri identifikaator. Vastuvõtja peab selle alusel suutma leida võtmeedastusserveri võrguaadressi ning sellega ühenduma.
- `TransactionID` – saatja poolt võtmevahetusserverile vastuvõtjale edastamiseks saadetud võtmekapsli identifikaator.

Sümmeetrilise võtme kapsel. Vastuvõtja tunnuseks on kasutaja valduses oleva sümmeetrilise võtme nimi `KeyLabel`.

```
1    SymmetricKeyCapsule = {
2        Salt   = :byte[]
3    }
```

- `Salt` – saatja poolt genereeritud juhuarv, mida kasutatakse HKDF-Extract funktsiooni sisendina.

4.1.4 Vormingu laiendamine

Vormingu laiendamiseks ning üldisemalt edasiühilduvuse tagamiseks on ette nähtud üks viis.

Selleks on päise struktuuris `Recipient` olev ühendtüüpi (*union type*) väli `Capsule` – ühendi iga tüüp kirjeldab mingit liiki vastuvõtjat oma krüptograafiliste primitiivide ja võtmehalduse vahenditega. Vorming nii oma abstraktsel kui konkreetsel juhul võimaldab neid tüüpe vastavalt vajadustele lisada.

4.2 Jadastatud vorming

Selles spetsifikatsioonis on kirjeldatud abstraktse vormingu realiseerimise viis vormingu FlatBuffers[1] baasil.

4.2.1 Vormingu üldine kirjeldus

Vorming koosneb ümbrikust, mis on sisuliselt jadastatud ja üksteise järele liidetud päis, sõnumiautentimiskood ja last.

Sõnumiautentimiskood ja last on on lihtviisil jadastatud.

Päis, tulenevalt oma laiendusvajadustest ning vajadusest päisega töötlusloogika vaatest samaväärseid sõnumeid läbi võtmeedastusserveri transportida, on kirjeldatud vormingu FlatBuffers baasil.

Lisaks päise laiendusmehhanismile defineerib konkreetse vormingu ümbrik veel ühe laiendus-punkti.

Selleks on ümbriku viiendas positsioonis olev versioonitunnus, mis on selle spetsifikatsiooni kontekstis fikseeritud väärtusele „2“ (baidi väärtus). Uute versioonide kirjeldamisel tuleb seda tunnust muuta.

[1] https://google.github.io/flatbuffers/

4.2.2 Ümbrik

Ümbrik koosneb järgmistest andmelementidest, mis on esitatud üksteise järel baitidena. Ümbriku alguse ja lõpu markeerimine ei ole selle spetsifikatsiooni käsitlusalas, kuivõrd kõige peamine ja loomulik kasutusjuht on see, mille puhul on ühes CDOC-failis täpselt üks ümbrik.

- 4 baiti: string „CDOC“ – vormingu marker (i.k. *prelude*), UTF-8 kodeeringus.
- 1 bait: versiooni tunnus, selle spetsifikatsiooni kohaselt väärtusega 2.
- 4 baiti: järgneva päise pikkus, *big endian* esituses. Päise pikkus on 32-bitiline märgiga täisarv, ehk päise maksimumsuurus saaks olla 2 GB. Realisatsiooni lihtsuse huvides on päise suurus piiratud 1MB (2^{20}) baidiga.
- Eelnevaga määratud kogus baite: jadastatud FlatBuffers päis.
- 32 baiti: päise sõnumiautentimiskood (vt jaotis 6.5).
- Ülejäänud baidid kuni lõpuni: päise poolt määratud skeemi ja võtmega krüpteeritud last.

Tabel 1 kujutab ümbriku struktuuri.

Tabel 1. Ümbriku struktuur

Väli	”CDOC”	Versioon	Päise pikkus	Päis	HMAC	Last
Pikkus	4	1	4	Päise pikkus	32	Ümbriku lõpuni
Algus	1	5	6	10	10 + Päise pikkus	10 + Päise pikkus + 32

4.2.3 Päis ja selle sõnumiautentimiskood

FlatBuffers vormingu skeemi tehniline kirjeldus (skeem) on toodud etalonrealisatsiooni lähtekoodihoidlas, kataloogis `cdoc20-schema/`.

Skeem on kirjeldatud kahes failis, mis on toodud selle spetsifikatsiooni lisades.

src/main/fbs/header.fbs
: FlatBuffers päise kirjeldus.

src/main/fbs/recipients.fbs
: Vastuvõtja tüüpide kirjeldused, ühiskasutatav teistes failides toodud skeemide poolt.

FlatBuffers’i reeglite alusel jadastatud päis kirjutatakse ümbrikku ning vastavalt ümbriku kirjeldusele kirjutatakse selle ette 4-baidine pikkuseväli.

Päise sõnumiautentimiskood arvutatakse vastavalt jaotisele 6.5 ning kirjutatakse baitide kaupa vahetult päise järele. sõnumiautentimiskoodi algoritm ja seega ka pikkus on selle spetsifikatsiooniga määratud.

4.2.4 Last

Last kirjutatakse CDOC-vormingu alusel koostatud konteinerisse vahetult sõnumiautentimiskoodi järele, kõige viimasena. Vorming eeldab, et lasti lõpu tunnus on kuidagi vorminguväliselt määratud, näiteks faili lõpuga.

On oluline tähele panna, et lasti lõpu tunnus on ainult soovitusliku iseloomuga – lasti tegeliku tervikluse määrab see, kas last on võimalik tervikuna dekrüpteerida või mitte.

Lasti avateksti moodustamine on kirjeldatud jaotises 4.3.

Lasti krüpteerimine on kirjeldatud jaotises 6.6.

4.2.5 Vormingu koostamise juhendid

See jaotis viitab etalonrealisatsiooni lähtekoodile, kasutades selleks Java paketinimesid ja muid identifikaatoreid. Sellised viited on antud `püsisammuga kirjas`.

CDOC 2.0 konteineri koostamisel tuleb teha järgmised sammud.

- Koguda kõigi vastuvõtjate loetelu.
- Genereerida FMK, HHK ja CEK.
- Koostada päis, koos kõigi kaasnevate krüptograafiliste operatsioonidega.
- Arvutada päisele sõnumiautentimiskood.
- Valmistada ette lasti avatekst.
- Krüpteerida last.
- Koostada ümbriku jadastatud kuju.
- Turvaliselt kustutada töö käigus kasutatud FMK, HHK ja CEKi väärtused.

Lasti avateksti ettevalmistamine on kirjeldatud jaotises 4.3. `container.Tar.archiveFiles()`

Järgmisena tuleb ette valmistada krüptograafiline materjal, mille abil päist ja lasti kaitstakse. Vastavate võtmete (FMK, HHK ja CEK) genereerimine ja tuletamine on kirjeldatud jaotises 6.2. `container.Envelope.prepare()` ja `container.Envelope()`

Seejärel tuleb koguda kokku ja jadastada kõigi soovitud vastuvõtjate loetelu, kuivõrd konteineri tervikluse tagamiseks kasutatavad krüptograafilised meetodid opereerivad tervikliku jadastatud päisega.

Iga vastuvõtja kohta tuleb läbi viia vajalikud krüptograafilised protseduurid, mis on kirjeldatud jaotistes 6.3 ja 6.4

Seejärel tuleb arvutada päise sõnumiautentimiskood, vastavalt jaotisele 6.5.

Lasti krüpteerimine on kirjeldatud jaotises 6.6. `crypto.ChaChaCipher.encryptPayload()` ja `crypto.ChaChaCipher.initChaChaOutputStream()`

Ümbriku täpne jadastatud vorming on kirjas jaotises 4.2.2.

Krüpteerimisprotsessi lõppedes tuleb kasutatud krüptomaterjal (sümmeetrilised võtmed, efemeersed privaatvõtmed) turvaliselt kustutada. Turvaline kustutamine sõltub olulisel määral kasutatavas käidukeskkonnast, mõnel pool (näiteks JVM-is) ei pruugi see ka võimalik olla. Arendaja peab hindama, millised võimalusi pakub selleks tema poolt kasutatav programmeerimiskeel ja käidukeskkond.

4.2.6 Vormingu parsimise juhendid

See jaotis viitab etalonrealisatsiooni lähtekoodile, kasutades selleks Java paketinimesid ja muid identifikaatoreid. Sellised viited on antud `püsisammuga kirjas`.

Konteineri parsimise etalonrealisatsioon on funktsioon `container.Envelope.decrypt()`. See on primaarne sisendpunkt dekrüpteerimise loogikasse ning tema eesmärk on võtta sisendina antud krüpteeritud konteiner ning kirjutada selles sisalduvad failid etteantud kausta.

Mainitud funktsioon teeb järgmist ja kõik alternatiivsed realisatsioonid peavad tegema sedasama, rakendades kõiki asjakohaseid turvakontrolle.

- Parsib ümbriku ja dekooderib ümbrikust päise
- Dekrüpteerib/tuletab KEKi.
- Dekrüpteerib/tuletab konteinerispetsiifilised võtmed FMK, HHK ja CEK.
- Kontrollib päise sõnumiautentimiskoodi.
- Dekrüpteerib arhiivi.
- Eraldab krüpteeritud failiarhiivist failid ja kirjutab need etteantud kausta. Voogtöötluse režiimis on dekrüpteerimine ja failide eraldamine üks operatsioon.

Ümbriku parsimise ja päise dekodeerimise etalonrealisatsiooniks on funktsioon `container.Envelope.readFBSHeader()`.

Päis tuleb Flatbuffers teegi abil parsida, kasutades selleks `fbs.header.Header` juurtüüpi (etalonrealisatsioonis on selleks FlatBuffers-skeemist genereeritud funktsioon `fbs.header.Header.getRootAsHeader()`).

Päise parsimine täielikul kujul on toodud etalonrealisatsiooni funktsioonis `container.Envelope.deserializeFBSHeader()`.

Päisest tuleb leida selline vastuvõtja (`Recipient`), mis vastab konteinerit töötlevale osapoolele ning pärida või dekrüpteerida KEK, FMK ja HHK. Vastuvõtja identifitseerimise viisi on kirjeldatud iga krüpteerimisskeemi juures jaotises 6.3. Kui töötlevale osapoolele vastavat vastuvõtjat ei leitud, ei ole võimalik konteinerit dekrüpteerida. Sel juhul peab algoritm väljastama vea, et konteiner ei ole mõeldud töötlejale avamiseks ning lõpetama töö.

KEKi arvutamine on kirjeldatud jaotises 6.3. Kui KEK arvutamise käigus tekib viga (näiteks ei ole punkt elliptkõveral), siis peab algoritm väljastama vea ning lõpetama töö. KEK arvutamise funktsioonid on klassis `crypto.KekTools`

FMK dekrüpteerimine on kirjeldatud jaotises 6.4. `crypto.Crypto.xor()`

HHK pärimise protseduur on kirjeldatud jaotises 6.2. `crypto.Crypto.deriveHeaderHmacKey()`

HHK ning päise originaalse jadastatud kuju alusel tuleb kontrollida ära päise sõnumiautentimiskood. `container.Envelope.checkHmac()`

Alles sõnumiautentimiskoodi õnnestunud kontrolli järel võib asuda lasti dekrüpteerima. Kui päise sõnumiautentimiskoodi kontroll ebaõnnestus, siis peab algoritm väljastama vea ning lõpetama töö.

Dekrüpteerimiseks tuleb pärida CEK, vastav protseduur on kirjeldatud jaotises 6.2. `crypto.Crypto.deriveContentEncryptionKey()`

Lasti dekrüpteerimine koosneb kolmest erinevast etapist: dekrüpteerimisest, krüptogrammi autentimisest ning dekrüpteeritud failiarhiivi lahtipakkimisest.

Dekrüpteerimine ja krüptogrammi autentimine on kirjeldatud jaotises 6.6.

Failiarhiivi lahtipakkimine on kirjeldatud jaotises 4.3.2

4.3 Krüpteerimata last

See jaotis kirjeldab üksikasjalikumalt krüpteerimata lasti vormingut ja töötlemist.

Vormingu põhiomadused:

- Edastatavad failid arhiveeritakse, kasutades POSIX tar vormingut [5].
- Arhiveeritud failid pakitakse meetodiga ZLIB, standarditud kui IETF RFC 1950 [6].

Konteineri lasti avatekst moodustatakse järgmisel moel: edastatavatest failidest (või ka ühest failist) moodustatakse POSIX tar arhiiv, mis pakitakse tervikuna ZLIB vormingusse.

> Realisatsioonimärkus: DD4 klient kasutab zlib teegi poole pöördumiseks Qt vastavaid mähkurfunktsioone. Kuna neid ei saa voolrežiimis (streaming mode) kasutada, siis kaasneb spetsifikatsiooniga soovitus asendada Qt mähkurite kasutamine zlib'i voolrežiimis väljakutsetega. See muutub eriti oluliseks säilituskrüpto juures, kus mahud võivad olla suured ning andmete ühekorraga krüpteerimine mälupuhvrites ei ole võimalik.

4.3.1 Nõuded POSIX tar arhiivi koostamisele

Kuna vorming `tar` on iseenest pika ajaloo ja mitmete variatsioonidega, on siinkohal kirjeldatud, millistele nõuetele peab vastama CDOC 2.0 tarbeks koostatud arhiiv. Nende nõuete eesmärgiks on vähendada ühilduvusprobleeme erinevate klientrakenduste ja/või käidusüsteemide vahel ning võimaldada faile arhiivist võimalikult turvaliselt failisüsteemi kirjutada.

- Kasutatakse standarditud POSIX tar dialekti [5]. See vorming on tuntud ka kui „POSIX 1003.1-2001“ või „PAX“.
- Kõik failinimed on esitatud UTF-8 kodeeringus.
- Üle 100B failinimed on toetatud PAX päise laiendusega [5]
- Üle 8GiB suurused failid on toetatud PAX päise laiendusega [5]
- Arhiivi kirjutatakse failinimed ilma kataloogiteedeta (*basename*).
- Arhiivi kirjutatud loabitte ja muid turvaargumente ignoreeritakse (neid võib kirjutada, aga mitte lugeda).
- Arhiivi kirjutatakse ainult tavafaile (tüüp 0).
- Faile käsitletakse binaarfailidena.

4.3.2 Nõuded lasti lahtipakkimisele

Lasti vorming on valitud selliselt, et seda on võimalik lahti pakkida voolrežiimis. See tähendab, et kogu krüpteeritud lasti ei pea töötlemiseks mällu laadima. Lasti on võimalik dekrüpteerida, lahti pakkida ning avakujul faile kettale kirjutada järjest.

Vooltöötluse korral kasutatakse dekrüpteeritud andmeid enne krüpteeringu kontrollsumma kontrollimist. Lahtipakkimisel peab arvestama sellega, et last võib olla riknenud ning ei vasta spetsifikatsioonis toodud reeglitele või on lausa pahatahtlikult ründaja poolt kokku pandud. Kuna CDOC 2.0 konteineri saatja ei ole autenditud, siis peab igal juhul arvestama võimalusega, et lasti võib olla koostanud ründaja – ka siis, kui krüpteeringu kontrollsumma klapib.

Voolrežiimis töötlemisel ei tohi avateksti töötlemisel tekkinud vigu (pakkimise või arhiveerimise vead) enne käsitleda, kui kogu last on läbitud ning krüptogramm autenditud. Kui krüptogrammi

autentimine ebaõnnestus, siis tuleb raporteerida see viga. Alles siis, kui krüptogrammi autentimine õnnestus, võib raporteerida avateksti töötlemisel toimunud vea. Vea korral tuleb kõik loodud failid kustutada.

Järgnevalt on kirjeldatud kaht rünnet, mille vastu spetsifikatsioonile vastav tarkvara peab rakendama kaitsemeetmeid.

> Võimalike rünnete nimekiri ei ole lõplik – nii näiteks võib faili sisuks olla viirus või pahavara ning enne selle kasutamist peab seda kontrollima antiviirusega – kuid see rünne ei ole CDOC 2.0 spetsiifiline vaid kehtib mistahes ebausaldusväärsest allikast saadud faili kasutamisel ja seetõttu seda siin pikemalt ei kirjeldata.

Esimene rünne: ründaja saab luua pakitud lasti, mille lahtipakkimisel moodustub hiiglaslik fail. Kui vastuvõtja töötleb lasti mälus, võib rakendus krahhida. Kui vastuvõtja kirjutab lasti kettale, võib ketas täis saada. Lahtipakkimisel on mõistlik kehtestada maksimaalne lubatud suurus lahtipakitud failidele ja jooksvalt kontrollida vaba mälu või vaba kettaruumi suurust lahtipakkimisel. Kui lahtipakitavad failid on suuremad kui lubatud või on vaba mälu või vaba kettaruum on kahanenud allapoole lubatud piiri, tuleb lahtipakkimine katkestada, eemaldada seni kettale kirjutatud failid ning raporteerida viga.

Teine rünne: ründaja saab manipuleerida tar failis olevate failide atribuutidega – nimede, loabittide, turvaargumentide ja tüüpidega. Sellise tar faili lahtipakkimisel ilma lisakontrolle rakendamata võib ründaja saada üle kirjutada olemasolevaid süsteemifaile, lisada uusi faile, tekitada faile, mis ei ole tavakasutajale nähtavad, kuid võivad olla vajalikud mõne ründe läbiviimiseks jne.

Kuna CDOC 2.0 konteineri eesmärk ei ole olla universaalne arhiivivorming, vaid lihtsalt pakkuda võimalust mitme faili samaaegseks krüpteerimiseks säilitades kasutajate mugavuse huvides failide algsed nimed, siis on tar faili lahtipakkimisele kehtestatud hulk reegleid, mille järgmine tagab kaitse eelpoolmainitud manipulatsioonide vastu:

- Failide loomisel tuleb ignoreerida arhiivis olevaid loabitte, faili omaniku ja grupi tunnuseid ning muid turvaargumente – kõik failid tuleb luua mittekäivitatavatena, rakenduse käivitanud kasutaja omadena ning talle loetavate-kirjutatavatena.
- Luua tohib vaid tavafaile (tüüp 0). Kui arhiivis on mõnd muud tüüpi fail, tuleb katkestada lahtipakkimine, eemaldada seni kettale kirjutatud failid ning anda viga. Korrektne CDOC 2.0 klientprogramm ei tohi luua faile, milles on muud tüüpi faile.
- Enne faili kettale kirjutamist kontrollida failinime ohutust. Lubamatuid sümboleid sisaldava failinime leidmisel katkestada lahtipakkimine, eemaldada seni kettale kirjutatud failid ning anda viga.

Failinime ohutuse kontrollimisel on järgmised eesmärgid:

- Vältida kataloogihüppe (*Path traversal*) [7] rünnet ning failide loomist väljapoole kasutaja poolt määratud kataloogi.
- Vältida kasutajale kättesaamatute või raskelt kättesaadavate, erimärke sisaldavate nimedega failide loomist.

Eri operatsioonisüsteemide nõuded failinimedele on erinevad. Mõistlik on kasutada mõnd läbiproovitud lahendust ebaturvalisest allikast saadud failinimede kontrollimiseks. Mõistlik on kasutada mitut kontrollimehhanismi.

Pathvalidate [2] on väga põhjalik Pythoni teek failinimede kontrollimiseks – muudes keeltes programmeerides tuleb teha sarnased kontrollid.

SEI CERT kataloog [8] kirjeldab täiendavat viisi kataloogihüpete vastu kaitseks.

Nimekiri etalonrealisatsioonis `container.FileNameValidator` failinimedele kehtivatest piirangutest:

- ei tohi alata tühiku ega sidekriipsuga;
- ei tohi lõppeda tühiku ega punktiga;
- ei tohi olla ükski järgnevatest: CON, PRN, AUX, NUL, COM[1-9], LPT[1-9];
- ei tohi sisaldada järgnevaid sümboleid: <, >, :, \, /, |, ?, *;
- ei tohi sisaldada kontrollsümboleid [9].
- ei tohi sisaldada Unicode märki Right-To-Left Override (U+202E)

[2] https://github.com/thombashi/pathvalidate

5 Võtmeedastusserver

See jaotis defineerib võtmeedastusserveri (siin jaotises ka: serveri), tema välised liidesed ja kasutamise reeglid.

5.1 Sissejuhatus

Võtmeedastusserver on alamsüsteem, mille ülesanne on edastatada CDOC-konteineri dekrüpteerimiseks vajalik võtmekapsel saatjalt vastuvõtjale, järgides reegleid, mis on konkreetse krüpteerimisskeemi jaoks kirjeldatud jaotises 3.

Võtmeedastusserveri poolt pakutav sidekanal on turvalisem kui avalikud sidekanalid, mille kaudu edastatakse CDOC 2.0 konteinereid. Reeglipäraselt toimiv võtmeedastusserver tagab krüpteeritult edastatud andmete tulevikuturvalisuse, kuna avalikku sidekanalit jälgiv ründaja ei saa enda valdusesse avaliku võtmega krüptoalgoritmide abil salastatud sümmeetrilisi krüpteerimisvõtmeid – need liiguvad turvatult läbi võtmeedastusserveri. Seega ei saa ründaja tulevikus, peale avaliku võtmega krüptoalgoritmide murdumist või privaatvõtmete kompromiteeriumist, krüpteerimisvõtmeid murda. Võtmeedastusserver ei pea vahendama mahukaid krüpteeritud dokumente – seega on tema käitamise kulud madalad.

Korraga võib kasutusel olla mitmeid võtmeedastusservereid, seejuures võivad neid käitada erinevad organisatsioonid. Juurutuse käigus kehtestatud turvanõuded võivad kohustada iga üksiku võtmeedastusserveri käitamist üksteisest sõltumatute organisatsioonide poolt.

5.2 Serveri tööpõhimõte

Kõige lihtsamal juhul töötab server järgmiselt.

1. Saatja loob krüpteerimise käigus võtmekapsli, mis on mõeldud konkreetsele vastuvõtjale.
2. Saatja valib välja serveri, ühendub sellega ning saadab sellele võtmekapsli koos vastuvõtja identifikaatoriga.
3. Server genereerib transaktsiooni identifikaatori ning salvestab selle koos võtmekapsli ja vastuvõtja identifikaatoriga.
4. Saatja lisab konteinerile valitud serveri identifikaatori, transaktsiooni identifikaatori ning vastuvõtja identifikaatori.
5. Saatja saadab konteineri vastuvõtjale.
6. Vastuvõtja leiab konteinerist talle mõeldud võtmekapsli kohta käiva info.
7. Vastuvõtja ühendub saatja poole valitud serveriga ning autendib end sellele.
8. Vastuvõtja saadab serverile konteinerist saadud transaktsiooni identifikaatori.
9. Server otsib transaktsiooni identifikaatori ning autentimise käigus tuvastatud vastuvõtja identifikaatori alusel võtmekapsli.
10. Server tagastab võtmekapsli vastuvõtjale.
11. Vastuvõtja kasutab võtmekapslis olevat infot konteineri dekrüpteerimiseks.

5.3 Serveri olek

Serveri oleku moodustavad serverile edastamiseks antud võtmekapslid koos nende juurde kuuluva infoga.

- Võtmekapsel, baidimassiivina. Võtmeserveri jaoks tähenduseta.
- Transaktsiooni ID – serveri enda poolt krüptograafiliselt tugeva juhuarvude generaatori abil genereeritud UUID.
- Vastuvõtja identifikaator – mingi selline identifikaator, mille saab server vastuvõtjat autentides autentimisprotsessi väljundina.
- Kehtivusaeg. Võtmeedastusserver hoiab võtmekapslit ainult ettenähtud ajavahemiku jooksul, mis määratakse saatja ning võtmeedastusserveri poolt rakendatavate võtmehalduspoliitikate koostöös. Pärast selle aja möödumist kapsel kustutatakse.

5.4 Serveri liidesed

Server pakub kaht liidest: üht võtmekapsli üleandmiseks serverile ning teist võtmekapsli üleandmiseks vastuvõtjale.

Liidesed on formaalselt kirjeldatud OpenAPI [10] keeles (vt lisa C).

5.4.1 Saatja liides

Võtmeedastusserveri saatja liidest kasutab saatja, et edastada võtmekapsel serverile ning saada vastu transaktsiooni identifikaatori, mille ta lisab konteineri päisesse.

Saatja saadab serverile võtmekapsli ning vastuvõtja identifikaatori ja saab vastu serveri poolt genereeritud transaktsiooni identifikaatori.

See liides ei ole autenditud – seda võivad kasutada kõik saatjad.

5.4.2 Vastuvõtja liides

Võtmeedastusserveri vastuvõtja liidest kasutab vastuvõtja, et saada serveri käest võtmekapsel.

Vastuvõtja autendib end serverile ning saadab transaktsiooni identifikaatori. Server otsib selle alusel välja võtmekapsli. Kapsel peab olema saadetud samale vastuvõtjale kes end serverile autentis – server võrdleb autentimise tulemusel saadud vastuvõtja identifikaatorit võtmekapsli saatja poolt määratud vastuvõtja identifikaatoriga. Server tagastab vastuvõtjale võtmekapsli.

5.4.3 Liideste turvalisus

Liideste turvamiseks kasutatakse TLS 1.3 protokolli. Server omab avalikust, tunnustatud CA-st võetud sertifikaati. Kliendid kontrollivad igal ühendumisel selle sertifikaadi kehtivust CA-st OCSP protokolli abil.

Protokolli turvalisuse tagamiseks on oluline tagada, et võtmekapsel jõuaks ainult võtmeedastusserverini. Selleks kasutatakse serveri TLSi võtmete kinnistamist (*pinning*). See tagab, et tänapäeval levinud [1] TLSi inspekteerimine ei riku võtmematerjali konfidentsiaalsust.

[1] https://zakird.com/papers/https_interception.pdf

5.5 Vastuvõtja autentimine

Iga võtmeedastusserverit kasutav võtmekapsli tüüp kirjeldab vastuvõtja identifitseerimise ning autentimise viisi.

Spetsifikatsiooni selles versioonis on defineeritud üks võtmekapsli tüüp, mis kasutab võtmeedastusserverit: `KeyServerCapsule`.

Spetsifikatsiooni tulevased versioonid võivad seda loetelu täiendada. Samaaegselt võib olla kasutusel mitmeid erinevaid autentimisskeeme.

5.5.1 `KeyServerCapsule` autentimisskeem

Selles skeemis identifitseeritakse vastuvõtjat tema avaliku võtmega, mida ta kasutab konteineri dekrüpteerimiseks. Avalik võti on määratud struktuuri `KeyServerCapsule` väljaga `RecipientKey`

Server kasutab vastuvõtja autentimiseks TLS kliendiautentimist (mTLS – *Mutual TLS*). Server on konfigureeritud kontrollima kliendi sertifikaadi kehtivust (näiteks OCSP protokolli abil). Kui vastuvõtja kaotab kontrolli oma dekrüpteerimisvõtme üle ja tühistab oma sertifikaadi, siis ei väljasta võtmeedastusserver võtmekapslit kaardi uuele omanikule (ründajale) ja ründaja ei saa konteinerit dekrüpteerida.

Pärast edukat autentimist loeb server kliendi poolt kasutatud sertifikaadist kliendi avaliku võtme ning võrdleb seda transaktsiooni identifikaatori poolt viidatud võtmekapsliga seotud avaliku võtmega. Kui need langevad kokku, siis tagastab server võtmekapsli. Vastasel juhul tagastab server vea.

5.6 Serverite identifitseerimine ja usaldamine

CDOC 2.0 poolt pakutavad täiendavad turvaomadused kehtivad ainult sellisel juhul, kui võtmekapsel edastatakse konkreetse krüpteerimisstsenaariumi poolt ette nähtud omadustega serverite kaudu (vt jaotis 3).

Et vastuvõtja ja saatja saaks olla kindlad selles, milliste serveritega nad suhtlevad, tuleb kas DigiDoc tarkvara paigalduspaketis või muul viisil levitada igale CDOC 2.0 vormingut kasutavale kliendile nimekiri usaldusväärsetest võtmeedastusserveritest. Nimekirja kasutatakse ka TLSi võtmete kinnistamiseks.

Selles nimekirjas on järgmised elemendid.

- Serveri identifikaator.
- Serveri poolt toetatav võtmekapsli tüüp.
- Saaja liidese URL.
- Vastuvõtja liidese URL.
- Serverit haldava organisatsiooni tunnus.
- Serveri avalikud võtmed, mis võimaldavad kliendil serveri identiteeti krüptograafiliselt kontrollida. Avalikud võtmed on sertifikaatide kujul.

Saatja ei edasta kunagi vastuvõtjale serveri tehnilist pöörduspunkti, vaid ainult sellest nimekirjast pärineva identifikaatori. See on vajalik, et vältida ründeid, mille puhul meelitatakse vastuvõtja ebausaldusväärse serveriga suhtlema.

Juhul kui kunagi seatakse üles mitu sõltumatut võtmeedastusserverite infrastruktuuri, mis ei

koordineeri omavahel serveritele identifikaatorite omistamist ning sama identifikaator võetakse mõlemas süsteemis kasutusele eri serverite tähistamiseks, siis võib tekkida olukord, kus ühte infrastruktuuri kasutav klient loob konteineri, mida proovib avada teist infastruktuur kasutav klient.

Vastuvõtja võtab sel juhul ühendust vale võtmeedastusserveriga, autendib end sellele ning saadab konteinerist saadud transaktsiooni identifikaator. Server ei leia sellele transaktsiooni identifikaatorile vastavat võtmekapslit ning tagastab vea – dekrüpteerimine ebaõnnestub.

Vastuvõtja poolt kasutatav võtmeedastusserver saab teada transaktsiooni identifikaatori, kuid kuna tal puudub võimalus õigele võtmeedastusserverile vastuvõtja nimel autentida, siis ei saa ta ka sealt võtmekapslit alla laadida.

Serveri poolt toetatav võtmekapsli tüüp võimaldab saatjal valida korrektse võtmekapsli tüübi ning võimaldab vastuvõtjal end serverile õige protokolliga autentida. Kuna serverid on kergekaalulised, siis juhul kui üks organisatsioon soovib toetada mitut erinevat vastuvõtja tüüpi, peab ta käitama mitut võtmeedastusserverit. See võimaldab iga võtmeedastusserveri muuta lihtsamaks ja seega turvalisemaks. See on eriti oluline vastuvõtja liidese juures, kus kasutusel võivad olla väga erinevate omadustega autentimisprotokollid, mida on raske turvaliselt ühendada.

Serverit haldava organisatsiooni tunnus ei pea olema ilmutatult seotud organisatsiooni nimega, kuid peab võimaldama tuvastada, millised serverid on sama organisatsiooni kontrolli all. Seda informatsiooni on vaja tulevaste ühissalastustusel baseeruvate krüptoskeemide toetamiseks.

Igale serverile võib olla määratud rohkem kui üks avalik võti – see on vajalik sertifikaatide ja võtmete sujuva vahetamise korraldamiseks. Klient peab serveriga ühenduse võtmisel alati kontrollima, et server kasutab üht loetletud sertifikaatidest. See aitab välistada vahendusründeid.

6 Krüptograafilised detailid

See jaotis kirjeldab lähemalt kõiki CDOC 2.0 vormingu juures tehtavaid krüptograafilisi arvutusi. Suurem osa nendest arvutustest on laiema tehnoloogilise infrastruktuuri suhtes neutraalsed, kuid osadel juhtudel on olemas konkreetsed seosed Eestis kasutatavate eID vahenditega (ennekõike ID-kaardiga) – need seosed on vastavates kohtades ära märgitud.

6.1 Turvatase ja kasutatavad krüptograafilised algoritmid

Hiljutiste uuringute [11] ja [12] tulemusena hinnatakse, et 128-bitiline turvalisus pakub kaitset ka pärast 2031. aastat. Seda pakkumist kinnitab ka värske 2022. aasta uuring [13]. Sellest lähtuvalt kasutame järgnevaid algoritme:

- HKDF-SHA-256,
- HMAC-SHA-256,
- ChaCha20-Poly1305.

Räsifunktsioonide pere SHA2 on standardiseeritud juba pikalt. Hiljutiste uuringute põhjal (vt [14]) on räsifunktsioonide pere SHA2 jätkuvalt turvaline ning SHA-256 pakub 128-bitilist turvalisust.

Kasutatavale võtmepärimisfunktsioonile HKDF (täpsem esitus sektisoonis 6.2) on tehtud põhjalik turvalisuse analüüs [15]. HKDF kasutamist turvalise räsifunktsiooniga soovitab ka [11].

HMACi turvalisuseks piisab 128-bitisest võtmest [14].

ChaCha20-Poly1305 on turvaline (IND-CPA ja INT-CTXT) autentiv krüpteerimisalgoritm (vt nt [16]). ChaCha20 kui jadašiffer pakub ka ise 256-bitist turvalisust [14].

FMK krüpteerimiseks kasutakse XOR operatsiooniga krüpteerimist (*one-time pad*), mis pakub 256-bitist turvalisust.

Sümmeetriliste võtmetete kasutatavad võtmepikkused.

- FMK - 256 bitti
- HHK - 256 bitti
- CEK - sõltub kasutatavast lasti krüpteerimise algoritmist, 256 bitti ChaCha20-Poly1305 korral.
- KEK - sõltub kasutatavast FMK krüpteerimise algoritmist, 256 bitti XOR korral.

6.2 Võtmete pärimine

Võtmete pärimiseks kasutatakse CDOC 2.0 vormingu puhul võtmepärimisfunktsiooni HKDF, mis on spetsifitseeritud kui IETF RFC 5869 [17].

RFC 5869 defineerib kaks võtmepärimisfunktsiooni: `HKDF-Extract` ja `HKDF-Expand`, mille kasutamine on kirjeldatud allpool. Edaspidi kasutatakse nende funktsioonide märkimiseks lühemaid variante $Extract$ ja $Expand$.

Võtmepärimise käigus kasutatakse igal pool räsifunktsiooni SHA-256 ning seetõttu on nende funktsioonide väljundiks 256-bitine väärtus.

Kõik tekstilised konstandid tuleb enne funktsioonidele sisendiks andmist kodeerida UTF-8 kodeeringu alusel (krüptograafiliste funktsioonide sisendiks on siinkohal baidimassiivid ja täisarvud), spetsifikatsioonis toodud jutumärgid ei ole stringide osad.

Üldkasutatavad sümmeetrilised võtmed päritakse järgnevalt.

File Master Key, FMK

$FMK \leftarrow Extract(”CDOC20salt”, random)$

Kus $CDOC20salt$ on konstantne string ja $random$ on vähemalt 256-bitine väärtus, mis on genereeritud krüptograafiliselt tugeva juhuarvude generaatori (CSPRNG) abil.

Content Encryption Key, CEK

$CEK \leftarrow Expand(FMK, ”CDOC20cek”, L_{octets})$

L_{octets} määrab funktsiooni $Expand$ väljundi pikkuse baitides ning see peab olema sama kui kasutatava sümmeetrilise krüpteerimisalgorimi võtmepikkus (vt jaotis 6.6).

Header HMAC Key, HHK

$HHK \leftarrow Expand(FMK, ”CDOC20hmac”, 32_{octets})$

On oluline tähele panna, et sedasi päritud HHK pikkus on 256 bitti ehk sama palju kui kasutatava SHA-256 räsialgoritmi väljund. See tuleneb HMACi standardis antud soovitustest [18].

Key Encryption Key, KEK

KEKi pärimine sõltub otseselt kasutatavast vastuvõtja tüübist ning on kirjeldatud vastavate vorminguelementide juures:

- `ECCPublicKeyCapsule` – p. 6.3.1.
- `RSAPublicKeyCapsule` – p. 6.3.2.
- `KeyServerCapsule` – p. 6.3.3.
- `SymmetricKeyCapsule` – p. 6.3.4.

6.3 Päiseelementide kirjeldus ja KEK arvutamine

Päise abstraktne struktuur on kirjeldatud jaotises 4. Väljad on abstraktses struktuuris kirjeldatud primitiivsete andmetüüpidega. See jaotis kirjeldab, kuidas välju arvutada ning väärtuseid primitiivse andmetüübi kujule viia.

Key Encryption Key (KEK) arvutamine sõltub sellest, millist tüüpi on vastuvõtja. Allpool defineeritakse KEKi arvutamine iga vastuvõtja tüübi kohta eraldi.

6.3.1 ECCPublicKeyCapsule

`ECCPublicKeyCapsule` (vt tabel 2) viitab vastuvõtjale, keda identifitseeritakse tema ECC avaliku võtmega. Vorming CDOC 2.0 toetab ükskõik millise `secp384r1` elliptkõveral genereeritud avaliku võtme kasutamist adressaadina. Näiteks võib selleks võtmeks olla Eesti ID-kaardi autentimisvõtmepaari avalik võti.

Kasutatav TLS 1.3 kodeering elliptkõvera punktide jaoks on secp384r1 kõvera puhul identne CDOC 1.0 poolt kasutatava kodeeringuga.

Struktuur `ECCPublicKeyCapsule` vastab võtmekapslile $caps_i$ jaotistes 3.1 ja 3.2 toodud proto-

Tabel 2. ECCPublicKeyCapsule elemendid

Väli	Sisu	Kodeering
`Curve`	Kasutatav elliptkõver, praegu vaid secp384r1.	Vastavalt kasutatavale vormingule, vt skeemi kirjeldust.
`RecipientPublicKey`	Vastuvõtja avalik võti. Näiteks ID-kaardi 1. võtmepaari avalik võti	Avalik võti kodeeritakse vastavalt TLS 1.3 reeglitele [19, p. 4.2.8.2].
`SenderPublicKey`	Saatja efemeerse (lühikese elueaga või lausa ühekordse) võtmepaari avalik võti.	Avalik võti kodeeritakse vastavalt TLS 1.3 reeglitele [19, p. 4.2.8.2].

kollide mõttes.

Saatja arvutab KEKi enda genereeritud efemeerse võtmepaari salajase võtme ning vastuvõtja avaliku võtme abil, kasutades ellipkõrveratel defineeritud Diffie-Hellmani võtmekehtestust (ECDH) ning rakendab saadud tulemusele määratud võtmepärimisfunktsiooni. Vastuvõtja teeb samalaadse arvutuse, kasutades selleks saatja efemeerset avalikku võtit ning ID-kaardi autentimisvõtmepaari. Arvutuste detailid on toodud allpool.

Saatja efemeerne võtmepaar on anonüümne, st ta ei ole kuidagi seotud saatja avaliku identiteediga.

Mitme vastuvõtja puhul võib saatja ühe CDOC-konteineri piires kasutada sama efemeerset võtmepaari.

Saatja efemeerne võtmepaar koosneb avalikust ja salajasest võtmest:

$$(pk_{eph}, sk_{eph}).$$

Vastuvõtjal on tema ID-kaardi autentimisvõtmepaari avalik võti pk_{rec} (eeldame, et vastuvõtja privaatvõti ei ole otse ligipääsetav).

See võti on ka saatjale kättesaadav. Vastuvõtja avaliku võtme levitamise mehhanismid ei ole selle spetsifikatsiooni käsitlusalas.

6.3.1.1 KEKi arvutamine krüpteerimise käigus

Jagatud ECDH saladus arvutatakse saatja poolt järgmiselt:

$$S_{ecdh} \leftarrow (sk_{eph} \cdot pk_{rec})_x \ ,$$

st jagatud saladuseks on selliselt arvutatud elliptkõvera punkti x-koordinaat. Jagatud saladus on kodeeritud *big-endian* baidimassiiviks, mille pikkus täisbaitides vastab kasutatava elliptkõvera mooduli pikkusele täisbaitides. Näiteks secp384r1 kõvera puhul on see 48 baiti.

KEK arvutatakse jagatud saladusest järgmiselt:

$$KEK_{pm} \leftarrow Extract(\text{"}CDOC20kekpremaster\text{"}, S_{ecdh})$$
$$KEK \leftarrow Expand(KEK_{pm}, \text{"}CDOC20kek\text{"} \parallel algId \parallel pk_{rec} \parallel pk_{eph}, L_{octets})$$

Siin $algId$ on FMK krüpteerimiseks kasutatava krüptoalgoritmi tunnus defineeritud stringina, vastavalt `Recipient.FMKEncryptionMethod` väljale (jaotis 6.4).

L_{octets} määrab funktsiooni $Expand$ väljundi pikkuse baitides ning selle määrab FMK krüpteerimiseks kasutatav sümmeetriline krüpteerimisalgoritm.

6.3.1.2 KEKi arvutamine dekrüpteerimise käigus

Eesti ID-kaart toetab autentimisvõtmepaari kasutamist ECDH ühe osapoolena.

Sobiv viis selle funktsionaalsuse kasutamiseks on läbi PKCS#11 funktsiooni `C_DERIVEKEY`, kasutades mehhanismi `CKM_ECDH1_DERIVE`.

Võtme tuletamiseks saab kasutada ka Windowsi krüptofunktsioone `NCryptSecretAgreement` ning `NCryptDeriveKey`. `NCryptSecretAgreement` arvutab S_{ecdh} ning `NCryptDeriveKey` arvutab KEK_{pm}. Seejuures tuleb `NCryptDeriveKey` parameetrid seada järgnevalt:

- `hSharedSecret` - `NCryptSecretAgreement` poolt tagastatud pide.
- `pwszKDF` - konstant `BCRYPT_KDF_HMAC`.
- `pParameterList` - algoritmi parameetrid:
  - `KDF_HASH_ALGORITHM` - konstant `BCRYPT_SHA256_ALGORITHM`.
  - `KDF_HMAC_KEY` - konstant `CDOC20kekpremaster`.
  - `KDF_SECRET_PREPEND` - see parameeter tuleb ära jätta.
  - `KDF_SECRET_APPEND` - see parameeter tuleb ära jätta.

Teadaolevalt valideerivad kõik täna kasutusel olevad ID-kaardid ECDH võtmekehtestuse sisendiks olevat elliptkõvera punkti, kuid selle kontrolli võib CDOC-vormingut toetavas tarkvaras täiendavalt realiseerida, et parandada veahalduse kasutajasõbralikkust.

Selleks, et kontrollida, kas punkt $Q = (x, y)$ asub kõveral, tuleb kontrollida, kas x ja y on vahemikust $[0..p-1]$, rahuldavad kõvera võrrandit ja kas punkt asub õiges alamrühmas. Näiteks kõvera P-384 võrrand on kujul

$$y^2 \equiv x^3 - 3x + b \bmod p\ .$$

Lisaks tuleb kontrollida, et $nQ = 0$ ja $Q \neq 0$. Konstandid b, p ja n on kirjeldatud standardis [20].

`C_DERIVEKEY` funktsiooni sisendiks tuleb anda saatja efemeerne avalik võti pk_{eph} ning viide vastavale ID-kaardi võtmepaarile (pk_{rec}, sk_{rec}). Vastuvõtja arvutab

$$S'_{ecdh} \leftarrow (sk_{rec} \cdot pk_{eph})_x\ .$$

Tänu elliptkõvera algebralistele omadustele kehtib võrdus $S_{ecdh} = S'_{ecdh}$.

Sellest jagatud saladusest tuleb arvutada KEK täpselt samamoodi nagu krüpteerimise juures kirjeldatud.

6.3.2 RSAPublicKeyCapsule

`RSAPublicKeyCapsule` (vt tabel 3) viitab vastuvõtjale, keda identifitseeritakse tema RSA avaliku võtmega.

Struktuur `RSAPublicKeyCapsule` vastab võtmekapslile $caps_i$ jaotistes 3.1 ja 3.2 toodud protokollide mõttes.

Saatja genereerib juhusliku KEKi ja krüpteerib selle vastuvõtja RSA avaliku võtmega kasutades OAEP täidistust. Vastuvõtja dekrüpteerib krüpteeritud KEKi oma RSA privaatvõtmega. Arvutuste detailid on toodud allpool.

Vastuvõtja avalik võti on ka saatjale kättesaadav. Vastuvõtja avaliku võtme levitamise mehhanismid ei ole selle spetsifikatsiooni käsitlusalas.

Tabel 3. RSAPublicKeyCapsule elemendid

Väli	Sisu	Kodeering
RecipientPublicKey	RSA avalik võti	Väärtuseks on ASN.1 struktuuri `RSAPublicKey` DER kodeering vt [21, p. A.1.1].
EncryptedKEK	Vastuvõtja avaliku võtmega krüpteeritud võtmeedastusvõti.	XXX

6.3.2.1 KEKi arvutamine krüpteerimise käigus

FMK krüpteerimiseks kasutatav sümmeetriline krüpteerimisalgoritm määrab KEK pikkuse L_{octets}. Saatja genereerib L_{octets} pikkuse juhuarvu KEK.

Saatja kasutab RSA-OAEP (vt [21, p. 7.1]) krüpteerimisfunktsiooni KEK krüpteerimiseks. RSA-OAEP vajab kolme parameetrit (vt [21, p. A.2.1]):

- `hashAlgorithm` – algoritmi poolt kasutatav räsifunktsioon. Kasutame räsifunktsiooni SHA-256 (`id-sha256`)
- `maskGenAlgorithm` – algoritmi poolt kasutatav maskigenereerimise funktsioon. Kasutame funktsiooni MGF1 (`id-mgf1`), mis on parametriseeritud SHA-256 räsifunktsiooniga.
- `pSourceAlgorithm` – märgendi allika funktsioon. Kasutame tühja märgendit (`pSpecifiedEmpty`).

6.3.2.2 KEKi arvutamine dekrüpteerimise käigus

Vastuvõtja dekrüpteerib krüpteeritud KEK-i oma privaatvõtmega kasutades samu parameetreid mida kasutati krüpteerimisel.

6.3.3 KeyServerCapsule

Struktuuri `KeyServerCapsule` (vt tabel 4) poolt kirjeldatud võtmeserver väljastab järgmisi struktuure, mille töötlemine on kirjeldatud vastavates jaotistes:

- `ECCPublicKeyCapsule`, jaotis 6.3.1.
- `RSAPublicKeyCapsule`, jaotis 6.3.2.

`KeyServerCapsule` kasutamise detailid on kirjeldatud jaotises 5.

6.3.4 SymmetricKeyCapsule

`SymmetricKeyCapsule` (vt tabel 5) viitab vastuvõtjale, keda identifitseeritakse võtme nimega.

Selle skeemi kasutamisel on saatja ja vastuvõtja kas üks ja sama isik (säilituskrüptograafia kasutusjuhtum) või on eelnevalt omavahel süsteemiväliselt vahetanud sümmeetrilise salajase võtme (transpordikrüptograafia kasutusjuhtum).

Igal juhul on nii saatja kui vastuvõtja valduses üks ja sama salajane võti, mida identifitseeritakse võtme nimega. Spetsifikatsioon ei sea piiranguid sellele, kuidas nime valida.

Tabel 4. KeyServerCapsule elemendid

Väli	Sisu	Kodeering
RecipientKey	Vastuvõtja võtme, millega vastuvõtja end võtmeedastusserverile autendib, andmed.	
KeyServerID	Võtmeedastusserveri identifikaator.	UTF-8 string, mille määrab tarkvara usaldusankrute konfiguratsioon 5.6.
TransactionID	Transaktsiooni identifikaator.	UTF-8 string, mille määrab võtmeedastusserver.

Tabel 5. SymmetricKeyCapsule elemendid

Väli	Sisu	Kodeering
Salt	Saatja poolt genereeritud juhuarv, mida kasutatakse HKDF-Extract funktsiooni sisendina KEKi tuletamisel.	Baidijada

6.3.4.1 KEKi arvutamine krüpteerimise käigus

Saatjal on sümmeetriline võti sym nimega $label$. Saatja genereerib juhuarvu $salt$. Selle arvu eesmärk on kindlustada uus KEK isegi juhul, kui võtit sym taaskasutatakse. Kuna võtme taaskasutamisele pole võimalik mõistlikku ülempiiri anda, võtame väärtuse $salt$ pikkuseks tugeva varuga 256 bitti.

KEK arvutatakse sümmeetrilisest võtmest ja genereeritud juhuarvust järgmiselt:

$$KEK_{pm} \leftarrow Extract(salt, sym)$$
$$KEK \leftarrow Expand(KEK_{pm}, "CDOC20kek" \parallel algId \parallel label, L_{octets})$$

Siin $algId$ on FMK krüpteerimiseks kasutatava krüptoalgoritmi tunnus stringina, vastavalt `Recipient.FMKEncryptionMethod` väljale (jaotis 6.4).

L_{octets} määrab funktsiooni $Expand$ väljundi pikkuse baitides ning selle määrab FMK krüpteerimiseks kasutatav sümmeetriline krüpteerimisalgoritm.

6.3.4.2 KEKi arvutamine dekrüpteerimise käigus

Vastuvõtjal on sümmeetriline võti sym märgendiga $label$. `SymmetricKeyCapsule` väljast `Salt` saab vastuvõtja saatja poolt genereeritud juhuarvu $salt$.

Neist andmetest tuleb arvutada KEK täpselt samamoodi, nagu krüpteerimise juures kirjeldatud.

6.4 FMK krüpteerimine ja dekrüpteerimine

FMK krüpteerimiseks kasutatakse XOR-operatsiooni.

FMK eeldab, et KEK on sama pikk kui FMK ja seda on võimalik saavutada võtmepärimismeetodi HKDF kasutamisega.

FMK krüpteerimiseks rakendatakse XOR operatsiooni bitikaupa FMK ja KEKi vastavatele bittidele.

Dekrüpteerimiseks rakendatakse XOR operatsiooni bitikaupa krüptogrammi ja KEKi vastavatele bittidele.

6.5 Päise sõnumiautentimiskood

Päise sõnumiautentimiskood arvutatakse algoritmi HMAC [18] abil, kasutades räsifunktsiooni SHA-256.

Võtmena kasutatakse HHK-d (vt jaotis 6.2).

HHK pikkus peab olema vähemalt sama kui kasutatava räsifunktsiooni väljundi pikkus ehk 256 bitti.

$$HMACValue_{header} \leftarrow HMAC_{SHA-256}(HHK, header),$$

kus $header$ on jadastatud päis sellisel kujul, nagu ta ümbriku osaks kirjutatakse.

Sõnumiautentimiskoodi kontrollimisel tuleb arvutada parsitud päisest HHK ning seejärel arvutada konteinerist loetud jadastatud päisele sõnumiautentimiskood ja võrrelda seda konteinerist loetud sõnumiautentimiskoodiga – need kaks väärtust peavad olema võrdsed.

6.6 Lasti moodustamine ja krüpteerimine

Tähelepanu! See jaotis kirjeldab lasti kui baidijada krüpteerimist. Krüpteeritavatest failidest ühtse baidimassiivi moodustamine on kirjeldatud jaotises 4.3.

Lasti krüpteerimiseks kasutatakse ChaCha20-Poly1305 AEAD krüpteerimisskeemi [22] järgmiste parameetritega.

- Võtme pikkus: 256 bitti.
- Nonsi pikkus: 96 bitti.
- Autentimismärgise pikkus: 128 bitti.

Võtmena kasutatakse CEK-i.

Nonss ($nonce$) genereeritakse alati värske, kasutades krüptograafiliselt tugevat juhuarvude generaatorit (CSPRNG).

Lisainfona ($additionalData$) kasutatakse ettemääratud stringi UTF-8 kodeeringus, jadastatud päist ja päise sõnumiautentimiskoodi.

$$additionalData \leftarrow "CDOC20payload" \parallel header \parallel headerHMAC$$

Avateksti ($payLoad$) krüpteerimiseks rakendatakse krüpteerimisfunktsiooni, mille väljundi pikkus on avateksti pikkus pluss autentimismärgise pikkus.

$$encryptedPayload \leftarrow encrypt_{cc20p1305}(CEK, nonce, payLoad, additionalData)$$

Krüpteeritud lasti dekrüpteerimiseks rakendatakse dekrüpteerimisfunktsiooni.

$$payload \leftarrow decrypt_{cc20p1305}(CEK, nonce, encryptedPayLoad, additionalData)$$

> Lasti voolrežiimis töötlemisel käsitletakse avateksti enne, kui dekrüpteerimisfunktsioon on kontrollinud autentimismärgise ehtsust. Nõuded avateksti töötlemisele ning vigade käsitlemisele on toodud jaotises 4.3.2. Oluline on, et kõik failid autentimismärgise ehtsuse kontrolli ebaõnnestumisel kustutataks.

Krüpteeritud last jadastatakse koos nonsiga, kuivõrd nonss on vaja vastuvõtjale edastatada.

$$serializedPayload \leftarrow nonce \parallel encryptedPayLoad$$

Vormingu ümbriku mõttes on vastab lastile krüpteeritud ja jadastatud last ($serializedPayload$). Nonssi jms detailid ei ole ümbriku kirjelduses ilmutatud, kuivõrd sõltuvad kasutatavast krüpteerimismeetodist.

7 Realiseerimisjuhised

7.1 Etalonrealisatioon

CDOC 2.0 etalonrealisatsioon (*reference implementation*) on programmeerimiskeeles Java realiseeritud käsurearakendus, mis on kättesaadav `https://stash.ria.ee/projects/CDOC2/repos/cdoc20_java/`.

Etalonrealisatsioon on CDOC 2.0 spetsifikatsiooni autorite poolt kirjutatud ning selle lähtekoodi on spetsifikatsiooni teistel realiseerijatel tungivalt soovitav lugeda.

Ühtlasi saab etalonrealisatsiooni kasutada ühilduvustestide läbiviimiseks.

7.2 Testivektorid

Testivektorid on andmekomplektid, mis on võimaldavad CDOC 2.0 realisatsioone testida.

Testivektorid on esitatud etalonrealisatsiooni koosseisus, kataloogis `test/testvectors/`.

Lisa A `header.fbs`

```
include "recipients.fbs";

namespace Header;

// Union for communicating the recipient type
union Capsule {
    recipients.ECCPublicKeyCapsule,
    recipients.RSAPublicKeyCapsule,
    recipients.KeyServerCapsule,
    recipients.SymmetricKeyCapsule
}

// FMK encryption method enum.
enum FMKEncryptionMethod:byte {
    UNKNOWN,
    XOR
}

// Payload encryption method enum.
enum PayloadEncryptionMethod:byte {
    UNKNOWN,
    CHACHA20POLY1305
}

// Intermediate record, some languages act very poorly when it comes
// to an array of unions.
// Thus it is better to have an an array of tables that
// contains the union as a field.
table RecipientRecord {
    capsule:                Capsule;
    key_label:              string (required);
    encrypted_fmk:          [ubyte] (required);
    fmk_encryption_method:  FMKEncryptionMethod = UNKNOWN;
}

// Header structure.
table Header {
    recipients:                [RecipientRecord];

    payload_encryption_method: PayloadEncryptionMethod = UNKNOWN;
}

root_type Header;
```

Lisa B `recipients.fbs`

```
namespace Recipients;

//for future proofing and data type
union KeyDetailsUnion {
    EccKeyDetails, RsaKeyDetails
}

// Elliptic curve type enum for ECCPublicKey recipient
enum EllipticCurve:byte {
    UNKNOWN,
    secp384r1
}


table RsaKeyDetails {
    //RSA pub key in DER
    recipient_public_key:   [ubyte] (required);
}

table EccKeyDetails {
     // Elliptic curve type enum
     curve:                 EllipticCurve = UNKNOWN;

     //EC pub key in TLS format
     //for secp384r1 curve: 0x04 + X 48 coord bytes + Y coord 48 bytes)
     recipient_public_key:  [ubyte] (required);
}

// ECC public key recipient
table ECCPublicKeyCapsule {
    curve:                 EllipticCurve = UNKNOWN;
    recipient_public_key:  [ubyte] (required);
    sender_public_key:     [ubyte] (required);
}

table RSAPublicKeyCapsule {
    recipient_public_key:  [ubyte] (required);
    encrypted_kek:         [ubyte] (required);
}


table KeyServerCapsule {
    recipient_key_details: KeyDetailsUnion;
    keyserver_id:          string (required);
    transaction_id:        string (required);
}


// symmetric long term crypto
table SymmetricKeyCapsule {
    salt:                [ubyte] (required);
}
```

Lisa C cdoc20-key-capsules.yaml

```yaml
# Key Capsules API, version 2.0 of cdoc20services API
openapi: 3.0.3
info:
  contact:
    url: http://cyber.ee
  title: cdoc20-key-capsules
  version: '2.0'
  description: API for exchanging CDOC2.0 ephemeral key material in key capsules
servers:
  - url: 'https://cdoc2-keyserver-01.test.riaint.ee:8443'
    description: RIA test TLS
  - url: 'https://cdoc2-keyserver-01.test.riaint.ee:8444'
    description: RIA test mutualTLS

paths:
  '/key-capsules/{transactionId}':
    get:
      summary: Get key capsule for transactionId
      description: Get key capsule for transactionId
      tags:
        - cdoc20-key-capsules
      parameters:
        - name: transactionId
          in: path
          schema:
            type: string
            minLength: 18
            maxLength: 34
          required: true
          description: transaction id from recipients.KeyServerCapsule.transaction_id (fbs
              )
      responses:
        '200':
          description: OK
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/Capsule'
        '400':
          description: 'Bad request. Client error.'
        '401':
          description: 'Unauthorized. Client certificate was not presented with the
              request.'
        '404':
          description: 'Not Found. 404 is also returned, when recipient id in record does
              not match with public key in client certificate.'
      operationId: getCapsuleByTransactionId
      security:
        - mutualTLS: []
  '/key-capsules':
    post:
      summary: Add Key Capsule
      description: Save Capsule and generate transaction id using secure random. Generated
           transactionId is returned in Location header
      operationId: createCapsule
      responses:
```

```
        '201':
          description: Created
          headers:
            Location:
              schema:
                type: string
                example: /key-capsules/KC0123456789ABCDEF
              description: 'URI of created resource. TransactionId can be extracted from
                 URI as it follows pattern /key-capsules/{transactionId}'
        '400':
          description: 'Bad request. Client error.'
      requestBody:
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Capsule'
      security: []
      tags:
        - cdoc20-key-capsules
components:
  schemas:
    Capsule:
      title: Capsule
      type: object
      properties:
        recipient_id:
          type: string
          format: byte
          minLength: 97 # EC public key
          maxLength: 2100 # 16 K RSA public key = 2086 bytes
          description: 'Binary format is defined by capsule_type'
        ephemeral_key_material:
          type: string
          format: byte
          maxLength: 2100
          description: 'Binary format is defined by capsule_type'
        capsule_type:
          type: string
          enum:
            - ecc_secp384r1
            - rsa
          description: |
            Depending on capsule type, Capsule fields have the following contents:
             - ecc_secp384r1:
                  * recipient_id is EC pub key with secp384r1 curve in TLS format (0x04 +
                     X coord 48 bytes + Y coord 48 bytes) (https://www.rfc-editor.org/rfc
                     /rfc8446#section-4.2.8.2)
                  * ephemeral_key_material contains sender public EC key (generated) in
                     TLS format.
             - rsa:
                  * recipient_id is DER encoded RSA recipient public key - RsaPublicKey
                     encoding [https://www.rfc-editor.org/rfc/rfc8017#page-54](RFC8017
                     RSA Public Key Syntax A.1.1)
                  * ephemeral_key_material contains KEK encrypted with recipient public
                     RSA key
      required:
        - recipient_id
        - ephemeral_key_material
        - capsule_type
  securitySchemes:
```

```
    mutualTLS:
      # since mutualTLS is not supported by OAS 3.0.x, then define it as http basic auth.
          MutualTLS must be implemented
      # manually anyway
      #type: mutualTLS
      type: http
      scheme: basic
tags:
  - name: cdoc20-key-capsules
```