RIIGI INFOSÜSTEEMI AMET

Euroopa Liit
Euroopa
Regionaalarengu Fond

Eesti
tuleviku heaks

Krüptoalgoritmid ning nende tugi teekides ja infosüsteemides

UURING

2021

CYBERNETICA

Krüptoalgoritmid ning nende tugi teekides ja infosüsteemides

Tehniline dokument

Versioon 1.0

8. märts 2021. a.

71 lk

Dok T-184-7

Uuringu koostas Cybernetica AS Riigi Infosüsteemi Ameti tellimusel Euroopa Liidu struktuuritoetuse toetusskeemi „Infoühiskonna teadlikkuse tõstmine“ raames Euroopa Regionaalarengu Fondi rahastusel.

Projektijuhid:	Signe Meidla (Riigi Infosüsteemi Amet)
	Tõnis Reimo (Riigi Infosüsteemi Amet)
	Sandhra-Mirella Valdma (Cybernetica)
Autorid:	Mart Oruaas (Cybernetica)
	Aivo Kalu, PhD (Cybernetica)
	Jaak Pruulmann-Vengerfeldt, MSc (Cybernetica)
	Kristjan Krips, MSc (Cybernetica)
	Jan Willemson, PhD (Cybernetica)

Riigi Infosüsteemi Amet, Pärnu maantee 139a, 15169 Tallinn, Eesti
Email: ria@ria.ee, Web: `https://www.ria.ee`, Telefon: +372 663 0200

Cybernetica AS, Mäealuse 2/1, 12618 Tallinn, Eesti
E-mail: info@cyber.ee, Web: `https://www.cyber.ee`, Telefon: +372 639 7991

Sisukord

1 Sissejuhatus

See uuringuaruanne on järg kuuele varasemale uuringule, mis viidi läbi aastatel 2011–2018 [61, 62, 63, 22, 64, 84]. Uuringu eesmärk on anda ülevaade krüptograafiliste vahendite kasutamise hetkeseisust ja heast tavast. Alati on uuringu osaks olnud soovitused kasutatavate krüptoalgoritmide ja nende parameetrite (nt võtmepikkuste jm) kohta. Sel korral pöörame rohkem tähelepanu erinevatele levinud krüptograafilistele protokollidele ja nende kasutamise olulisematele asjaoludele. Samuti kirjeldame levinumaid krüptograafiateeke ja krüptograafiliste vahendite tuge levinud arendusplatvormidel.

Aruanne koosneb kuuest jaotisest. Jaotises 2 vaatleme krüptoprimitiive, nende tugevust ja soovitatavaid võtmepikkusi. Jaotises 3 vaatleme transpordiprotokollistikke. Jaotises 4 vaatleme levinumaid vorminguid ning kirjeldame, millist mõju vormingutele omavad muutused krüptoprimitiivides. Jaotises 5 kirjeldame levinumaid autentimis- ja delegeerimisprotokolle. Jaotises 6 kirjeldame vabavaraliste krüptograafia-teekide hetkeolukorda, sh viimasel ajal tekkinud populaarsemate arendus- ja käidukeskkondade vaatest.

Lähtematerjalidena on kasutatud eespool mainitud varasemaid Riigi Infosüsteemi Ameti tellitud uuringuid ning tuntud riiklike (NIST, BSI) ja rahvusvaheliste (SOG-IS) organisatsioonide samalaadseid uuringuid ja soovitusi. Lisaks toetub töö ETSI, NISTi ja IETFi standarditele ning eelretsenseeritud teadusartiklitele.

Aruande tarkvaratehnilistes osades on rakendatud Cybernetica AS pikaajalisi kogemusi krüptovormingute ja -teekide valikul ning nende kvaliteedi hindamisel.

Aruande koosseis on määratud riigihanke nr 213942 „Krüptoalgoritmid ning nende tugi teekides ja infosüsteemides“ tööde kirjeldusega.

Aruanne on suunatud lugemiseks kõigile IT-taristu ning tarkvaraarenduse valdkondade juhtidele ja spetsialistidele.

2 Krüptoprimitiivid

2.1 Võtmepikkuste soovitused

Soovituslikud võtmepikkused ning krüptoprimitiivide kasutatavuse ajahorisontide hinnangud tuginevad NISTi [5], SOG-IS [1] ja BSI [23] soovitustele.

Krüptograafia-alase tähtsaima uudisena võib osundada, et 2020. aasta mais andis NIST oma aruandest 800-57 välja 5nda versiooni [5]. Võrreldes 2017. aasta krüptoalgoritmide elutsükli uuringuga [64] ei ole hinnangud muutunud. Samuti on NIST asunud uuendama digisignatuuri standardit FIPS 186, tulles välja uue mustandiga FIPS 186-5 [29]. NISTi soovitused krüptograafiliste algoritmide perede turvatasemetele võtab kokku tabel 1.

Algoritmide turvatase määratakse bittides – mingi algoritmi b-bitine turvatase tähendab, et efektiivseim teadaolev rünne kasutab arvutusressurssi, mis kulub ligikaudu 2^b ploki krüpteerimiseks plokkšifriga. Turvataseme formaalsem definitsioon on antud NISTi aruandes 800-57 [5, jaotis 5.6.1.1].

Tabeli 1 veerg „DSA, DH“ viitab DSA signatuurialgoritmile ja Diffie-Hellmani võtmekehtestusprotokollile üle jäägiklassiringi. Nende konstruktsioonide puhul tagavad vajaliku turvataseme avaliku ja salajase võtme pikkused L ja N. Veerg „RSA“ annab soovituse RSA algoritmi mooduli pikkusele ja veerg „ECC“ elliptkõveratele tuginevate krüptoalgoritmide võtmepikkustele. Veerud „Plokkšifrid“ ja „SHA-2, SHA-3“ soovitavad vastavalt sümmeetriliste plokkšifri-algoritmide võtmepikkust ning räsifunktsioonide väljundi pikkust. Peamiseks soovitatavaks sümmeetriliseks krüptoalgoritmiks on jätkuvalt AES.

Tabel 1. Krüptograafiliste algoritmide soovitatavad võtmepikkused

Turvatase	DSA, DH	RSA	ECC	Plokkšifrid	SHA-2, SHA-3
128	$L = 3072$, $N = 256$	3072	$256 \ldots 383$	128	256
192	$L = 7680$, $N = 384$	7680	$384 \ldots 511$	192	384
256	$L = 15360$, $N = 512$	15360	$512+$	256	512

2.2 Krüptoprimitiivide rakendamisest

Standarditud krüptoprimitiive on võimalik rakendada laias laastus kahel viisil.

- Osana mingist standarditud protokollist, näiteks TLS.
- Osana rakendusespetsiifilisest liidestusest.

Kui esimese viisiga ei ole tavaliselt probleeme ning probleemide ilmnemisel käsitlevad neid nii standardiorganisatsioonid kui infoturbega professionaalselt tegelevad avalikud ja eraõiguslikud osapooled, siis teine viis on tihtipeale problemaatilisem.

Küsimus on selles, et krüptoprimitiivi turvatase ega korralikult realiseeritud algoritm koos võtmehaldusega ei ole piisav selleks, et saavutada turvaomadusi, mida süsteemi rakendaja soovib saavutada. Halvemal juhul võib ebakorrektselt rakendatud krüptograafiliste primitiivide kombinatsioon lausa nõrgendada mõne teise krüptograafiliste meetmete kaudu seotud süsteemi turvalisust (näiteks sama privaatvõtit kasutades [90]).

Krüptograafilistel protokollidel on väga palju muid nõrkusi peale krüptoprimitiivide otsese murtavuse [16, peatükk 1.4] ning tihtipeale ei ole need nõrkused väga ilmsed, kuivõrd ilmnevad sõnumivahetuse vaid väga spetsiifilistes kombinatsioonides.[1]

Krüptoprimitiivide rakendamisel väljaspool standardprotokolle või standardseid vorminguid soovitame alati küsida pädeva krüptograafi hinnangut plaanitavale tööle – krüptograafide rakendatavad uurimismeetodid võimaldavad selliseid nõrkusi tuvastada, vt näiteks ProVerifi rakendamist Mobiil-ID protokolli uurimiseks [66].

2.3 Krüptograafilised räsifunktsioonid

Räsifunktsioonid (*hash function, hash algorithm, digest function*) on funktsioonid, mis võtavad oma sisendiks vaid sõnumi ning väljastavad (tavaliselt fikseeritud pikkusega) sõnumilühendi ehk räsi.

On oluline tähele panna, et kuigi tehniliselt räsifunktsiooniks kvalifitseeruva funktsiooni koostamine ei ole ülearu keeruline, tohib krüptograafilistes rakendustes kasutada vaid aktsepteeritud ja läbiuuritud **krüptograafilisi räsifunktsioone** nagu näiteks SHA-2 või SHA-3 perekondade funktsioonid.

Krüptograafilise räsifunktsiooni kohustuslikeks omadusteks on kollisioonikindlus (*collision resistance*), mittepööratavus (*pre-image resistance*) ning lisaoriginaalikindlus (*second pre-image resistance*) – Rogaway ja Shrimpton seletavad need mõisted täpsemalt lahti krüptograafiliste räsifunktsioonide aluseid tutvustavas artiklis [89].

Tavaliselt on arvutuskiirus krüptograafilise räsifunktsiooni puhul omaette väärtus. Eraldi kategooria moodustavad spetsiaalselt lõppkasutaja-paroolide räsimiseks projekteeritud funktsioonid. Need funktsioonid esitavad suuri nõudmisi algoritmi käitamiseks kasutatavale riistvarale, et raskendada jõurünnaid. Sellised funktsioonid on näiteks bcrypt, *scrypt* ja Argon2.

2.3.1 SHA-2

2001. aastal avaldas USA *National Security Agency* (NSA) SHA-2 esimese spetsifikatsiooni. Praegu spetsifitseerib funktsiooni SHA-2 standard FIPS 180-4 [93].

SHA-2 puhul on standarditud funktsioonid SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224 ning SHA-512/256, kõigis neis nimedes näitab (viimane) kolmekohaline arv väljundi pikkust bittides. Vaid SHA-256 ja SHA-512 on iseseisvad algoritmid, ülejäänud variantide puhul on tegu pikema väljundiga funktsioonist lühema väljundiga funktsiooni saamisega

[1] https://buttondown.email/cryptography-dispatches/archive/cryptography-dispatches-the-most-backdoor-looking/

algväärtusvektori muutmise ning väljundi bittide äralõikamise teel. Praktilistes rakendustes ei oma see küll tähendust, kuniks kasutatakse sobiva väljundipikkuse ja turvatasemega räsifunktsiooni.

SHA-2 variantide turvatasemete hinnangud on toodud tabelis 1, kuid töö SHA-2 krüptoanalüüsi kallal kindlasti jätkub ning neid hinnanguid vaadatakse regulaarselt üle.

Oluline on märkida, et SHA-2 vastu ei ole täna teada klassikalistest otsingualgoritmidest efektiivsemaid kvantalgoritme (va üldotstarbeline Groveri algoritm, mida on varasemates uuringutes käsitletud [64]).

2.3.2 SHA-3

Räsialgoritm SHA-3 on NISTi poolt 2015. aastal standarditud kui täiendus algoritmidele SHA-1 ja SHA-2. Seejuures nägi NIST algoritmi SHA-3 standardides ette tänaseks edukalt realiseerunud ründeid SHA-1 vastu [95].

NISTi standard pakub nelja standardset fikseeritud väljundipikkusega räsifunktsiooni – SHA3-224, SHA3-256, SHA3-348 ja SHA3-512. Viimane arv funktsiooni nimes näitab väljundi pikkust bittides ning pikkused on valitud nii, et nad langeksid kokku SHA-2 perekonna funktsioonide väljundipikkustega.

SHA-3 perekonda kuuluvad ka muutuva (lõpmatu) väljundi pikkusega räsifunktsioonid SHAKE128 ja SHAKE256 – neid saab kasutada näiteks juhuarvugeneraatorina.

SHA-3 juurutamisel tasub kontrollida, kas algoritmi realisatsioon on saadaval kõigi seotud tarkvarasüsteemide ja -platvormide. Eriti oluline on see vanemate pärandsüsteemi puhul – näiteks SHA-3 perekond ei sisaldu Java 8 krüptoalgoritmide standardnimede loetelus, kuid Java rakenduste maailma valitsevad endiselt Java 7 ja 8 koos.

Pikemalt võib SHA-3 ja SHAKE kohta lugeda 2016. aasta krüptoalgoritmide elutsükli uuringust [22].

2.4 Parooliräsimise algoritmid

Turvaintsidentide statistikast[1] on selgelt näha, et viimasel viiel aastal on toimunud vähemalt paarsada infoleket erinevatest teenustest ja infosüsteemidest, millel on järgmine muster.

- Infosüsteemis kaitstakse lõppkasutajate paroole liiga nõrgalt (paroole hoitakse kas avatekstina või räsituna mõne lihtsa meetodiga – tavapärane on standardse räsifunktsiooni ühekordne rakendamine).
- Sissetungija varastab kasutajate andmebaasi, mis sisaldab kasutajatunnuseid (tüüpiliselt e-posti aadress) ning paroole või parooliräsisid ning nõrkade parooliräside korral arvutab neist jõuründega paroolide väärtused.[2]
- Murdunud paroolide abil saab teenuses kasutajakontosid üle võtta ning kuivõrd inimesed kasutavad sama parooli väga tihti paljudes teenustes, saab neid paroole pruukida kasutaja kontode ülevõtmiseks teisteski teenustes.

[1] https://haveibeenpwned.com/PwnedWebsites
[2] https://hashcat.net/hashcat/

Kuigi infosüsteemide turbe kõik meetmed on olulised, aitab konkreetselt sellise ründemustri vastu salvestatud parooli kaitsmine räsialgoritmiga, mille puhul on sisendiks olevat parooli jõuründega leidmine väga ressursimahukas. Järgnevalt vaatleme mõningaid selliseid algoritme. Osa neist algoritmidest on nimetatud kui võtmetuletusfunktsioonid, kuid sisuliselt on tegu sama probleemi lahendamisega – mingist jagatud saladusest raskestiennustatava baidijada tuletamisega.

Paroolide korralik räsimine võimaldab rakendada ja/või efektiivsemaks muuta ka muid nõudeid ja soovitusi, mis NIST juba 2016. aastal paroolide kohta andis [40, jaotis 5.1.1.2].

- Paroolide regulaarset vahetamist ei peaks nõudma.
- Paroolidele ei tohiks kehtestada maksimumpikkust, mis on vähem kui 64 tähemärki.
- Paroolidele ei tohiks kehtestada muid nõudeid peale miinimumpikkuse (näiteks nõue kasutada suur- ja väiketähti segamini ja muud sellised piirangud).
- Kasutaja valitud paroole peab võrdlema sõnastike või reeglistike vastu, et hoida ära levinud või kompromiteeritud paroolide kasutamist.

2.4.1 bcrypt

Algoritmi bcrypt avaldasid 1999. aastal Niels Provos ja David Mazières [77] ning see on sellest ajast saati olnud kõige populaarsem parooliräsimise algoritm. Algoritm baseerub plokkšifril Blowfish, mille võtmeloome rutiini keerukust kasutatakse parooliräsimise arvutusmahukuse tõstmiseks.

Algoritmi bcrypt positiivseks omaduseks on kahtlemata realisatsioonide suur valik erinevate programmeerimiskeelte ja -raamistike jaoks.

Negatiivse poole pealt võib ära märkida piirangut parooli pikkusele (72 baiti) ning suhtelist nõrkust FPGA riistvara kasutavate jõurünnete suhtes [98]. Samuti ei kuulu bcrypt ühegi riikliku agentuuri poolt soovitatud algoritmide nimekirja.

FPGA-põhiste rünnete vastase nõrkuse tõttu peab uutes süsteemides kasutama spetsiifilisi vastumeetmeid rakendavaid algoritme, nagu näiteks *scrypt*.

2.4.2 PBKDF2

PBKDF2 on IETF poolt standarditud [75] võtmetuletusfunktsioon, mis on ühtlasi ka üks väheseid NISTi poolt soovitatud võtmetuletus- ja parooliräsi algoritme [97] [40].

Kuigi PBKDF2 on defineeritud nii SHA-1 kui SHA-2 räsifunktsioonide baasil, ei soovita me SHA-1 varianti kasutada, kuivõrd SHA-1 ise on ebaturvaliseks tunnistatud algoritm [64]. Teadaolevalt ei mõjuta see otseselt PBKDF2-HMAC-SHA1 turvalisust, kuid segaduste vältimiseks on parem SHA-1 kasutamisest täielikult hoiduda.

NIST soovitab PBKDF2 kasutada vähemalt 10000 iteratsiooniga [40, jaotis 5.1.1.2].

2.4.3 scrypt

Algoritmi *scrypt* avaldas Colin Percival 2009. aastal [82] ning tänaseks on see standarditud kui IETF RFC 7914 [83].

Algoritmi *scrypt* põhiline erinevus selle eellastest on sõltuvus mälumahust, mitte kiiremast riistvarast. See teeb märksa kallimaks rööbitised jõuründed – ründamiseks pruugitava süsteemi

integraalskeemide maht peab olema suurem, mistõttu on ründe efektiivsust keerulisem parandada.

Interaktiivsete teenuste puhul soovitame kasutada *scrypt*i parameetreid $r = 8$, $p = 1$ ja $N = 2^{15} = 32768$. Sellised väärtused määravad üksiku parooliräsi arvutamiseks vajaliku mälumahu suuruseks 32MB ning 3GHz Intel Core i7 protsessoril võtab sellise räsi arvutamine aega 55 millisekundit. Valikumetoodika on pärit originaalartiklist [82, osa 8].

2.4.4 Argon2

Argon2 võitis 2015. aastal *Password Hashing Competition*[1] võistluse ning hetkel standarditakse seda IETF RFC protsessis [13].

Argon2 on tegelikult algoritmide perekond, millest igaühel on pisut erinevad omadused.

Argon2d – optimeeritud GPU-põhiste rünnete vastu, kuid võib sisaldada nõrkusi, mis lihtsustavad kõrvalkanalirünnete.

Argon2i – optimeeritud kõrvalkanalirünnete vastu.

Argon2id – eelmise kahe hübriid. Standardi mustand [13] soovitab kasutada Argon2id algoritmi kõigil juhtudel kui ülejäänud variantide eelistamiseks ei ole häid põhjuseid, samuti soovitab seda kasutada BSI [23].

Soovitatud parameetrid Argon2 perekonna algoritmidele on toodud standardimustandis [13, jaotis 4].

Argon2 on realiseeritud näiteks teegis *libsodium* (vt jaotis 6.1.7).

2.4.5 Balloon

NISTi digitaalidentiteedi halduse soovitused [40] loetlevad ainsa soovitusliku algoritmina PBKDF2 kõrval ka Balloon-i [15], kuid tolle puuduseks on kättesaadavate realisatsioonide napp kogus.

2.5 Sümmeetrilise võtmega primitiivid

Sümmeetrilise võtmega krüptoprimitiivid on sellised primitiivid, mille puhul kasutatakse kõigi operatsioonide teostamiseks sama võtmematerjali. Sümmeetrilised krüptoprimitiivid jagunevad kolme suurde kategooriasse: plokkšifrid, jadašifrid ning räsipõhised konstruktsioonid.

2.5.1 Plokkšifrid

Plokkšiffer on oma olemuselt üks-ühene teisendus n-bitiste plokkide vahel, mille määrab k-bitine võti. Plokipikkus n ja võtmepikkus k ei pea olema võrdsed.

Selle aruande kirjutamise ajal on ainus piisavalt turvaline standarditud plokkšifrialgoritm AES (*Advanced Encryption Standard*), mille turvatasemed vastavalt võtmepikkusele on toodud tabelis 1.

Plokkšifreid ei kasutata kunagi oma puhtal kujul, vaid mingis režiimis (AES-GCM, AES-CBC) või mingi teise konstruktsiooni (näiteks jadašifri või juhuarvude generaatori) ehituselemendina.

[1] https://www.password-hashing.net/

2.5.2 Jadašifrid

Jadašiffer võimaldab suvalise L-bitise avateksti krüpteerimist, genereerides k-bitisest võtmest L-bitise bitijada, mis kombineeritakse avatekstiga bitthaaval XOR operatsiooni abil (ehk $modulo\ 2$ liitmistehtega).

Jadašifrite saamiseks on mitmeid viise: jadašifrina kasutamiseks projekteeritud algoritmid, muutuva pikkusega räsifunktsioonid (SHAKE) või sobivas režiimis plokkšifrid (AES-CTR).

Jadašifrid pakuvad selle aruande kontekstis huvi ennekõike oma kasutuse tõttu TLSi šifrikomplektides, *ChaCha20* algoritmi näol. Üldisemalt on neil eelis plokkšifrite ees sellistes rakendustes, mis nõuavad ettemääramata suurusega andmesegmentide krüpteerimist ja edastamist piiratud oludes – näiteks raadiosides, kus plokkšifri kasutamine šifri ploki suurusest väiksemate segmentide edastamisel oleks vajaliku täidise tõttu raiskavam.

2.5.2.1 ChaCha20

ChaCha on Daniel J. Bernsteini 2008. aastal publitseeritud [11] jadašifrite perekond, number variandi *ChaCha20* nimes tähistab iteratsioonide arvu (vähema kui 20 iteratsiooniga ChaCha variandid ei ole piisava turvatasemega).

ChaCha20 on kasutusel TLS 1.2 ja TLS 1.3 jaoks soovitatud šifrikomplektides. *ChaCha20* kasutatakse nendes komplektides koos *Poly1305* autentimiskoodiga [78].

ChaCha20 hinnatakse ilma spetsiifilise AESi toeta riistvaral AESist kolm korda kiiremaks [78], seetõttu on tema kasutusvaldkond TLSis ennekõike väheste ressurssidega arvutisüsteemides (*IoT*, sardsüsteemid).

ChaCha20 võtmepikkus on 256 bitti ning selle turvatase on sama, kui Salsa20/20 jadašiffril, millel ChaCha põhineb – 256 bitti [12].

2.5.3 Räsipõhised sümmeetrilised konstruktsioonid

Räsipõhiste konstruktsioonide alla kuuluvad ennekõike sõnumiautentimiskoodid (*hash-based message authentication code*, HMAC) [59]. Standardiseeritud HMAC koodide turvatase sõltub ennekõike kasutatavast võtmepikkusest ning SOG-IS [1] soovitab kasutada HMAC võtmepikkust vähemalt 125 bitti. Tüüpiliselt on rakendustes HMAC võtme pikkus seotud kasutatava räsifunktsiooni väljundipikkusega.

2.6 Asümmeetrilised primitiivid

2.6.1 RSA

RSAst on pikemalt juttu 2016. aasta krüptoalgoritmide elutsükli uuringus [22, jaotis 3]. Turvatasemete hinnangud RSA algoritmile on sellest ajast muutunud ja vastavalt võtmepikkustele need on toodud tabelis 1.

Eestis vahepeal kasutusele võetud Smart-ID autentimis- ja allkirjastamisteenus kasutab oma krüptograafilise alusena ebatraditsioonilist nelja algarvulise teguriga RSA signatuuriskeemi, mille turvatase sama võtmepikkuse juures ei vasta tava-RSAle. Selle lahenduse kirjeldus ja turvataseme hinnangud koos metoodikaga on toodud temaatilises teadusartiklis [18].

Artikli järgi on $2N$-bitise avaliku mooduliga Smart-ID võtmepaari turvatase mitte rohkem kui 13 bitti madalam N-bitise avaliku mooduliga standardse RSA võtmepaari turvatasemest. Vastavalt tabelile 1 on selle aruande kirjutamise ajal Smart-ID poolt kasutatavate 6144-bitiste võtmete turvatase vähemalt 115 . Kuivõrd NIST [5, jaotis 5.6.3] annab 112-bitise turvatasemega võtmete kasutushorisondiks aasta 2030, siis kasutabki Smart-ID praegu 6144-bitised võtmeid ja seda loetakse neile võtmetele väljastatavate sertifikaatide kehtivusaja (3 aastat) piires turvaliseks.

2.6.2 ElGamal krüptoskeem

ElGamal on asümmeetrilise võtmega krüpteerimisskeem, mis põhineb Diffie-Hellmani võtmekehtestusel [37].

Eestis on ElGamali krüptosüsteemi kasutatud interneti-hääletamise süsteemis IVXV [49].

2.6.3 Elliptkõveratel põhinevad süsteemid

Elliptkõverate krüptogaafiast on pikemalt juttu varasemates uuringutes aastatest 2015 ja 2016 [63] [22], seetõttu vaatame siinkohal ainult neid aspekte, mis on nendest aruannetest saadik muutunud või mida tasub üle rõhutada.

Ühe tähelepanuväärsema uuendusena võib ära märkida Daniel J. Bernsteini Curve25519 standardimist IETF poolt RFC 7748 [65] raames ning NISTi käimasolevat standardimist *Special Publication* 800-186 mustandi näol. Curve25519 on oluline, sest selle tööks vajalikud arvutused on kiired ja selle kasutus pole piiratud patentidega. Samuti on sel kõveral elliptkõverate krüptograafia seisukohast olulisi kõrvalomadusi (näiteks keeruturvalisus – *twist security*), mis muudavad algoritmide realiseerimise lihtsamaks.[1]

Teise uuendusena võib märkida EdDSA signatuuriskeemi, eriti selle konkreetset realisatsiooni, mis on tuntud kui Ed25519 – vt jaotis 2.6.3.1.

Tabelis 2 on loetletud elliptkõverad, mida SOG-IS, NIST ja IETF soovitavad.

Tabel 2. Standarditud elliptkõverad

Kõvera nimi	Standard	Kuju	Märkused
Brainpool P256r1 Brainpool P348r1 Brainpool P512r1	RFC 5639 [73]	Weierstraß	
NIST P-256 NIST P-384 NIST P-521	NIST FIPS186-4 [28]	Weierstraß	P-256 ja P-384 on kasutusel Eesti Mobiil-ID ja ID-kaardi võtmepaaride jaoks.
Curve25519 Edwards25519	RFC 7748 [65]	Montgomery Edwards	Omavahel algebraliselt seotud.

[1] https://safecurves.cr.yp.to/

2.6.3.1 Ed25519

Rääkides Dan Bernsteini elliptkõverast Curve25519 on vaja silmas pidada järgmist terminoloogiat[1]:

Curve25519 – Elliptkõver ise.

X25519 – Kõveral 25519 baseeruv Diffie-Hellmani võtmekehtestusprotokoll, mis kasutab kõvera esitust Montgomery koordinaatides.

Ed25519 – Kõveral 25519 baseeruv signatuuriskeem, mis kasutab kõvera esitust Edwardsi koordinaatides.

Ed25519 on EdDSA digisignatuuriskeemi konkreetne isend – on oluline tähele panna, et kui Weierstraßi kõveratel baseeruvad digisignatuurid vastavad ECDSA skeemile, siis Ed25519 aluseks olev EdDSA on Schnorri signatuuriskeemi edasiarendus Edwardsi elliptkõverate jaoks.

X25519 ja Ed25519 realisatsioonid on võrreldes eelmise aruandega [64] rohkem levinud,[2] muuhulgas on NIST alustanud Ed25519 standardimist [29]. Seega tasub Ed25519 endiselt vaadata kui perspektiivset signatuuriskeemi.

2.7 Postkvant-krüptograafia standardimustandi hetkeseis

Praktiliselt kõik hetkel standarditud ja laialt kasutatavad asümmeetrilised primitiivid (nt RSA, Diffie-Hellmani võtmekehtestus, ElGamali krüptosüsteem, elliptkõveratel põhinevad krüptosüsteemid) muutuvad piisavalt võimsa kvantarvuti valmimisel haavatavateks. Seda asjaolu silmas pidades algatas NIST 2016. aastal postkvant-algoritmide standardimiseks konkursilaadse protsessi.[3]

Konkursi esimesse vooru esitati üle 60 krüpteerimis- ja signatuuriskeemi. Järgnevate aastate jooksul on osa neist ilmnenud nõrkuste tõttu kõrvale jäetud ning osa omavahel sarnaseid kandidaate on ühendatud. 2020. aasta suvel kuulutas NIST välja teise vooru tulemused. Nende alusel läheb esimeses järjekorras standardimisele neli krüpteerimis-võtmekehtestusskeemi ning kolm signatuuriskeemi. Peale nende on välja kuulutatud 5+3 nn alternatiivset skeemi, mida NIST peab samuti piisavalt tugevateks, et kaaluda nende standardimist tulevikus.

Konkursi kolmanda vooru põhifinalistid on kirjeldatud tabelis 3, kus lühend PKE-KEM tähendab *public key encryption key encapsulation mechanism*. Skeemide aluseks olevate ülesannete tüüpidest annavad ülevaate aruanded [22, 84].

Millised neist skeemidest rakendustes ja teekides populaarseteks osutuvad, näitab lähitulevik. Hetkel võib postkvant-algoritmide tuge krüptograafilistes teekides pigem eksperimentaalseks pidada.

Open Quantum Safe projekt[4] on teostanud rea NISTi konkursi kandidaate (sh kõik tabelis 3 toodud) nii, et neid saab kasutada näiteks TLS või SSH protokolli krüptoprimitiividena. Ka Bouncy

[1] https://mailarchive.ietf.org/arch/msg/cfrg/-9LEdnzVrE5RORux3Oo_oDDRksU/

[2] https://ianix.com/pub/curve25519-deployment.html

[3] https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization

[4] https://openquantumsafe.org/

Tabel 3. NISTi postkvant-konkursi põhifinalistid

Nimi	Liik	Tüüp
Classic McEliece	PKE-KEM	Koodipõhine
CRYSTALS-KYBER	PKE-KEM	Võrepõhine
NTRU	PKE-KEM	Võrepõhine
SABER	PKE-KEM	Võrepõhine
CRYSTALS-DILITHIUM	Signatuur	Võrepõhine
FALCON	Signatuur	Võrepõhine
Rainbow	Signatuur	Mitmemuutuja polünoomide põhine

Castle teegis on olemas eksperimentaalne tugi tabeli 3 algoritmidele Classic McEliece, NTRU ja Rainbow.[1]

Kuna NISTi konkursi tulemuste väljakuulutamiseni on veel vähemalt aasta aega (NIST pole konkreetset kuupäeva välja kuulutanud), võib nendes algoritmides lähitulevikus muutusi esineda. See tähendab, et stabiilsust vajavates süsteemides neid veel kasutada ei saa, kuid pilootprojekte saab põhimõtteliselt juba läbi viia.

[1] `https://javadoc.io/doc/org.bouncycastle/bcprov-jdk15on/latest/index.html`

3 Protokollid ja protokollistikud

3.1 TLS

TLS (*Transport Layer Security*) on valdava osa Interneti-liikluse turvamiseks kasutatav protokollistik. Eelmiste uuringutega võrreldes on kõige suurem sisuline muutus TLS versiooni 1.3 standardimine augustis 2018 [85].

Võrreldes varasemate uuringutega [22] on arenenud ka viisid RSA-põhise võtmekehtestuse ründamiseks [90]. RSA kasutamine võtmekehtestuseks on nüüd „tingimusteta mittesoovitatav". Autentimine RSAga (kui näiteks serveri sertifitseeritud võtmepaar on RSA võtmepaar) on turvaline, kui autentimise võtmepaari ei kasutata samal ajal muuks otstarbeks.

3.1.1 Soovitused

Kasutada võib TLS versiooni 1.3 ja mööndustega ka versiooni 1.2. TLS 1.2 võib kasutada vaid juhul kui kasutatavad šifrikomplektid ja laiendused on teadaolevalt turvalised.

TLS 1.2 šifrikomplektid määravad algoritmid võtmekehtestuse, (autentimisel) signeerimise ja sümmeetrilise krüpteerimise jaoks ning fikseerivad kasutatava räsifunktsiooni.

RSA võtmekehtestuse kasutamine ei ole enam lubatud, sest seni pole õnnestunud seda turvaliseks muuta. Oluline on ka, et kuna rünnakud RSA võtmekehtestuse vastu võimaldavad võltsida RSA allkirja, siis saab ebaturvalist võtmekehtestust pakkuva serveri abil rünnata ka servereid, mis kasutavad sedasama RSA võtmepaari ainult autentimiseks (nt TLS 1.3 servereid). Seepärast on võtmekehtestuseks lubatud kasutada vaid efemeersete[1] Diffie-Hellmani võtmepaaridega võtmekehtestust (ECDHE ja DHE).

Kui koostalitlusvõime tagamiseks on RSA võtmekehtestuse kasutamine kasvõi ajutiselt vältimatu, peab hoolitsema, et kasutataks unikaalseid, vaid selle serveriga seotud võtmeid. Võimalusel tuleb sellise serveri kasutamist piirata ka kliendi IP-aadressi järgi või vähendada võimalikku ründepinda mingi muu meetodiga. Üldisemalt ei ole soovitatav sama võtmepaari kasutamine mitmes iseseisvas TLS-serveris. Eriti ettevaatlik peab olema võtmetega, millele on olemas *wildcard*-sertifikaadid (nimedega kujul `*.example.com`) või mitut nime sisaldavaid sertifikaadid ning mida sellisena on mugav kasutada mitmes, sageli erineva TLS-protokolli teostuse või konfiguratsiooniga serveris.

RSA kasutamine suhtluspoolte autentimiseks (st võtmetüübina serveri või kliendi sertifikaadis) on lubatud.

Efemeersete võtmetega Diffie-Hellmani võtmekehtestuseks astendamisega jäägiklassirühmas (DHE) jaoks peab kasutama piisavalt suure järguga rühma. DHE-võtmekehtestuse korral sõltub

[1] Ühekordsete, lühikese elueaga. Täpsemat käsitlust vt [16, jaotis 1.5.7].

võtmekehtestusprotokolli abil kokkulepitud peasaladuse turvalisus otseselt ja ainult kasutatud rühma turvaomadustest ja mitte näiteks serveri autentimissertifikaadis kajastatud võtmepaari omadustest.[1] TLS 1.2 ja vanemad lubavad kasutada isegenereeritud rühmi, mille turvaomadusi on raske hinnata. TLS 1.3 lubab kasutada vaid valitud hulka, teadaolevalt turvalisi rühmi ning RFC 7919 [38] lisab nimega rühmade toe ka varasematele TLSi versioonidele. Kasutada tohib vaid rühmi, mis on piisava turvatasemega vastavalt tabelile 1. Kui DHE kasutamine pole vajalik tagasiühilduvuse jaoks, soovitame seda mitte kasutada, sest elliptkõveratepõhine ECDHE on parema jõudlusega ja seda toetab suurem osa tänapäevaseid TLS-teostusi (sh värskemad brauserid).

Sümmeetrilise krüpteerimise jaoks on lubatud kasutada vaid autenditud töörežiime. Plokkšifritest võib kasutada vaid AESi, jadašifritest toob TLS 1.3 sisse ChaCha20. Lubatud šifrid/töörežiimid on AES-GCM, AES-CCM ja CHACHA20-POLY1305. AES-CCM kasutamine on lubatud, aga vaid siis, kui teised kaks pole kasutatavad.

Šifrikomplektis määratud räsifunktsiooni kasutatakse sõnumiautentimiskoodi (HMAC) koosseisus. Sümmeetriliste šifrite autenditud töörežiimides (AEAD – *Authenticated Encryption with Additional Data*) kasutatakse sõnumiautentimiskoodi vaid pseudojuhusliku jada generaatori (PRF – *Pseudorandom Function*) sisendi komponendina. Kui šifrikomplekt räsifunktsiooni eraldi ei maini, kasutatakse TLS 1.2 ja uuemate juures vaikimisi SHA-256. See on ka soovitus – kasutada SHA-256 või tugevamat.

TLS 1.2 kasutamisel on oluline keelata ka andmete pakkimine (*compression*) TLSi kihis ning kätluse kordamine juba loodud seansis (*renegotiation*), sest mõlemad muudavad TLSi rünnatavaks. Võimalusel tuleb keelata ka andmete pakkimine TLS-seansi sees kasutatavas rakendusprotokollis (nt HTTP), sest selline pakkimine võimaldab andmete lekkimist sarnaselt pakkimisele TLS-kihis [39]. Kui pakkimise keelamine pole võimalik, peab rakendama muid leevendavaid meetmeid nagu päringuvõltsimise (*Cross-Site Request Forgery*) vastane kaitse ja väljundi pikkuse juhuslik muutmine, et peita pakkimise tekitatud pikkuseerinevusi, mida seda tüüpi ründed kasutavad.

TLSis kasutatavaid šifrikomplekte ja nende kahebaidiseid identifikaatoreid registreerib IANA.[2] Samas registris on toodud ka väli iga konkreetse šifrikomplekti soovitatavuse kohta.

TLS 1.2 jaoks on praegu lubatud vaid šifrikomplektid:

- TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
- TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
- TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256
- TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
- TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
- TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256
- TLS_DHE_RSA_WITH_AES_128_CCM (see ja kõik järgmised üksnes turvalise DHE rühmaga! Vaata ka ülalolevat lõiku DHE võtmekehtestuse kohta – see on lubatud kuid võimalusel peaks seda vältima)

[1] https://weakdh.org/

[2] https://www.iana.org/assignments/tls-parameters/tls-parameters.xhtml#tls-parameters-4

- TLS_DHE_RSA_WITH_AES_128_GCM_SHA256
- TLS_DHE_RSA_WITH_AES_256_CCM
- TLS_DHE_RSA_WITH_AES_256_GCM_SHA384
- TLS_DHE_RSA_WITH_CHACHA20_POLY1305_SHA256

TLS 1.3 šifrikomplektid määravad vaid sümmeetrilise šifri ja räsifunktsiooni ning seetõttu neid TLSi eelmiste versioonidega kasutada ei saa.

Lubatud TLS 1.3 šifrikomplektid on:

- TLS_AES_128_CCM_SHA256 (vaid juhul, kui AES-GCM või ChaCha20-Poly1305 pole kasutatavad)
- TLS_AES_128_GCM_SHA256
- TLS_AES_256_GCM_SHA384
- TLS_CHACHA20_POLY1305_SHA256

3.1.2 Muud TLS rakendamist kirjeldavad dokumendid

TLS turvalise konfigureerimise soovitusi annavad ka muud organisatsioonid. Seetõttu on kasulik neid soovitusi regulaarselt kontrollida. Viidatud veebilehti on lihtsam uuendada kui käesolevat teksti ning teiste dokumentide uuendamistsükkel võib olla käesoleva teksti omast erinev. Järgnevalt toome mõned viited materjalidele, mida jälgida.

IETF haldab heade tavade kirjeldust, selle värskeim versioon on praegu RFC 7525 [96], kuid ettevalmistamisel on ka uuem, TLS 1.3 olemasoluga arvestav versioon.[1]

OWASPil on spikker (*cheat sheet*) TLSi turvaliseks konfigureerimiseks,[2] muuhulgas leiab sealt konfiguratsiooni testimiseks mõeldud tööriistade nimekirja.

Brauseritootja Mozilla haldab soovitusi TLS serverite turvaliseks konfigureerimiseks.[3] Mozilla soovitused on jagatud kolme gruppi vajaliku tagasiühilduvuse taseme järgi: moodsad brauserid/kliendid (TLS 1.3 toega, tagasiühilduvust pole vaja), keskmine/mõõdukas tase (soovituslik konfiguratsioon üldkasutatavale serverile) ja vana (erandlik tagasiühilduvus; kasutamiseks vaid erandkorras).

Ka NSA on värskelt avaldanud soovitused [33] vanematest TLSi versioonidest ja nõrkadest algoritmidest loobumise kohta. Üheks oluliseks erinevuseks selle aruande ja NSA soovituste vahel on, et meie soovitame RSA võtmekehtestusest loobuda.

Veebibrauserites kasutatava TLS ja PKI osas lepivad brauseritootjad ja CAd kokku omavahelises foorumis.[4] Selle organisatsiooni kaudu kehtestatakse alusnõuded (*baseline requirements*),[5] mis on kohustuslikud CAdele, kes soovivad oma juursertifikaatide levitamist brauserites.

[1] https://datatracker.ietf.org/doc/html/draft-ietf-uta-rfc7525bis-00

[2] https://cheatsheetseries.owasp.org/cheatsheets/Transport_Layer_Protection_Cheat_Sheet.html

[3] https://wiki.mozilla.org/Security/Server_Side_TLS

[4] https://cabforum.org

[5] https://cabforum.org/baseline-requirements-documents/

3.1.3 TLS 1.3

Eelmise aruande ilmumise ajal ei olnud TLS 1.3 veel lõplikult avaldatud. Nüüd on TLS 1.3 avaldatud, selle spetsifitseerib RFC 8446 [85]. Järgnev on ülevaade TLS 1.3st.

TLS 1.3 on protokolli TLS uus, seni viimane versioon. Hoolimata minimaalsest versiooninumbrimuutusest on tegu senisest üsnagi erineva protokolliga. Protokollikirjelduse autor on TLS 1.3 projekteerimise eesmärkidena nimetanud[1]:

- Koristamine: ebaturvaliste ja kasutusel mitteolevate omaduste, algoritmide jms eemaldamine;
- Parem konfidentsiaalsus: kätluse (*handshake*) osiste senisest ulatuslikum krüpteerimine;
- Parem latents: lühemate sõnumivahetustega kätlus, 2 sõnumit (1-RTT) tavalisel juhul ja 1 sõnum (0-RTT) korduval ühendumisel;
- Järjepidevus: seniste oluliste kasutusviiside jätkuv võimaldamine;
- Turvalisus: tehtavad valikud peavad olema põhjendatud turvaanalüüsiga.

Kasutajavaatest on olulisemad muudatused:

- Teadaolevalt ebaturvalised algoritmid on keelatud. RSA võtmekehtestus, SHA-1 ja MD5 räsifunktsioonid, RC4, CBC-režiimis plokkšifrid jne on ilmutatult keelatud. Sümmeetrilist krüpteerimisalgoritmi on lubatud kasutada vaid AEAD-režiimis. RSAga signeerimiseks kasutatakse turvalisemat signatuuriskeemi RSASSA-PSS (alates PKCS#1 v2.1 defineeritud, turvatõestustega signatuuriskeem [55, jaotis 8.1]). DSA on keelatud. Kasutaja valitud rühmad DHE võtmekehtestusel on keelatud.
- Tavakasutuses on lubatud vaid efemeersete võtmepaaridega võtmekehtestusprotokollid, mis pakuvad tulevikuturvalisust (*Perfect Forward Secrecy*). See tähendab, et sümmeetrilisteks krüpteerimiseks kasutatavad seansivõtmed on iga kord uued ning poolte pikaajalise võtmematerjali (sertifikaadiga seotud salajase võtme) lekkimine ei võimalda dekrüpteerida enne leket loodud seansse.
- Lisati vastussõnumita (0-RTT) kätlusrežiim, mis võimaldab kliendil hakata kaitstud andmeid saatma enne kui server oma esimese paketi saadab või see pakett kliendini jõuab. Selliselt saadetud nn „varajased andmed“ (*early data*) on kaitstud nõrgemini kui tavalise kätluse järel vahetavad andmed ja seetõttu tuleb 0-RTT režiimi lubamisel olla ettevaatlik.[2]
- Kõik kätlussõnumid, mis järgnevad serveripoolse võtmekehtestusosaku saatmisele, on krüpteeritud, sh serveri vastus laienduste kohta ja sertifikaadid. Kui varem (kuni TLS versioonini 1.2 kaasa arvatud) oli pealtkuulajal võimalik näha mõlema poole esitatud sertifikaate ja autentimissignatuure, siis versioonis 1.3 enam mitte. Protokolli versioonis 1.3 on nähtavad kliendi saadetud laiendused (sõnum *Client Hello* koos laiendustega), nt serveri nimi (SNI, *Server Name Indication*). Käib töö ka kliendi esimese sõnumi krüpteerimise võimaldamiseks, kuid 2021. aasta alguses pole see veel standardina valmis.[3]

[1] https://cs155.stanford.edu/lectures/19-rescorla.pdf

[2] https://blog.trailofbits.com/2019/03/25/what-application-developers-need-to-know-about-tls-early-data-0rtt/

[3] https://tlswg.org/draft-ietf-tls-esni/draft-ietf-tls-esni.html

- Elliptkõveratel põhinevate algoritmide kasutamine autentimiseks ja võtmekehtestuseks on nüüd põhistandardi osa. Elliptkõveratega seotud andmevahetust lihtsustati, iga toetatud kõvera jaoks on määratud oma konkreetne punktiesituse vorming[1] varasema vormingute igakordse kokkuleppimise asemel.
- Kasutatava TLSi versiooni kokkuleppimine muutus, nüüd kasutatakse selleks laiendusi kätlussõnumites. TLS 1.3 kirjeldamise käigus selgus, et märkimisväärselt suur hulk Interneti-liiklust vahendavaid seadmeid (ruuterid, tulemüürid, liikluseanalüsaatorid; *middleboxes*) teevad neid läbivate pakettide kohta lubamatuid eeldusi. Seetõttu on TLS-pakettide päises oleva versiooninumbrivälja kasutamine versiooni 1.3 jaoks võimatu. Lahendus: TLS 1.3 liiklus maskeeritakse TLS 1.2 seansi jätkamiseks. Selliseks maskeerimiseks saadetakse täiendavaid sõnumeid, millel ei ole TLS 1.3 jaoks tähendust.
- Erinevad seansi jätkamise mehhanismid (seansi identifikaator, seansipilet) ja eeljaotatud võtmega šifrikomplektid asendati ühe eeljagatud võtmeid kasutava võtmekehtestusmehhanismiga. Seansi jätkamise võimaldamiseks saab server seansi käigus väljastada kui tahes palju üksteisest sõltumatuid pileteid, millest tuletatakse eeljaotatud võtme identifikaatorid. Toetatud on nii skeemid, kus olekut hoiab server kui skeemid, kus olek säilib piletis.
- Eeljaotatud saladuste mehhanism võimaldab TLS 1.3 kasutada ka ilma avaliku võtme krüptograafiata. Sellisel juhul ei kasutata efemeerseid võtmeid ja seansid ei ole tulevikuturvalised. Samas on selline TLSi variant siiski turvaanalüüsitud [30] ja nt IoT-seadmete puhul eelistatud, võrreldes omaloodud protokollidega. Väljatöötamisel on ka vastav TLS 1.3 profiil.[2]
- Täienesid viisid, kuidas klient ja server saavad vastastikku kokku leppida, milliseid sertifikaate autentimiseks tunnistatakse. Näiteks saab kumbki pool deklareerida, milliseid signeerimisalgoritme, sertifitseerimisorganeid (*Certification Authority*, CA) ja sertifikaadilaiendusi ta teise poole sertifikaatides toetab või eeldab.
- Eemaldati TLS-tasemel pakkimise tugi. Selline pakkimine võimaldas teatud tingimustel lekitada kaitstud kanalis liikunud saladusi, nt HTTP seansivõtmeid jms. Oluline on tähele panna, et samasugust lekkimist võimaldab rakendusprotokolli tasemel pakkimine ning seda ka TLS 1.3 kasutamisel – võimalusel tuleb selline pakkimine keelata või rakendada täiendavaid leevendavaid meetmeid [39]
- Kätluse kordamine seansi käigus (*renegotiation*) eemaldati. Küll on võimalik kliendi autentimine pärast põhikätluse lõppu (*post-handshake authentication*), mis on see, milleks kätluse kordamist sageli varasemalt kasutati. Mis sellise hilisema autentimise tähendus on, sõltub rakendusprotokollist ning nt HTTP/2 jaoks on see hoopiski keelatud [10]. Samadel põhjustel on kätluse kordamine keelatud ka juhtudel, kui HTTP/2 kasutatakse koos mõne vanema TLSi versiooniga [9, jaotis 9.2.1].

[1] Elliptkõveratel põhinevates süsteemides on võtmed ja krüptogrammid punktid (täisarvudena esitatud koordinaadid) kasutataval kõveral. Nende esitamiseks baidijadadena on kasutusel olnud mitmeid erinevaid vorminguid, kusjuures erinevatele kõveratele sobivad erinevad esitused.

[2] `https://tools.ietf.org/html/draft-ietf-uta-tls13-iot-profile-00`

3.1.4 TLS-CCA

TLS *Client Certificate Authentication* (TLS-CCA) on standardne viis kliendi autentimiseks TLS-seansis. Juhul, kui autentitakse nii server kui klient, nagu tavaliselt levinud praktika on, nimetatakse tekkivat komplekti ka kui *Mutual TLS Authentication* (mTLS).

Server võib paluda kliendilt autentimist, näidates millist kliendisertifikaati ta ootab (kriteeriumiteks võivad olla toetatud CAde eraldusnimed, toetatud algoritmid, TLS 1.3 puhul sertifikaadi laiendused ja tingimused nende väärtustele).

Klient esitab sertifikaadi ning signeerib sama, sertifikaadis näidatud võtmepaariga senised kätlussõnumid (ja sertifikaadipäringu kontekstiinfo, TLS 1.3 korral). Klient võib sertifikaadi esitamisest ka loobuda, nt kui sobivatele tingimustele vastav sertifikaat puudub. Sellisel juhul on serveri otsustada, kas seansiga saab jätkata või mitte.

TLS versioonides kuni TLS 1.2 liigub kliendi sertifikaat esmase kätluse ajal kaitsmata kujul ning on sellisena pealkuulajale nähtav. TLS 1.2 korduskätlusel ja TLS 1.3 kätluses on sertifikaat kaitstud. TLS 1.3 korral toimub sertifikaatide edastamine krüpteeritult ja kliendisertifikaat edastatakse vaid eelnevalt edukalt autenditud serverile.

Eestis levinud TLS-CCA kasutus on ID-kaardiga autentimine, aga TLS-CCAd kasutatakse ka süsteemidevahelises kommunikatsioonis, nt X-tee turvaserverite vahel.

TLS-CCA on ainuke standardne viis kliendi võtmepaari krüptograafiliseks sidumiseks TLS-seansiga.[1]

Tänu kahepoolsele ja kätlusega seotud autentimisele on TLS-CCA ainus praktiliselt kasutatav viis, mille abil vältida infovahetuse krüptograafilise kaitse märkamatut nõrgenemist või kadu keskkondades, kus kasutatakse TLS liikluse inspekteerimise vahendeid (tulemüürid, antiviirused) või rakendatakse liikluse vahendamist rakenduskihis (*phishing* ründed, PSD2 *embedded* ja *decoupled* liidestused[2]). Mööndustega aitab sama probleemi vastu ka kinnistamine (vt jaotis 3.1.5.5), kuid ainult esimesena mainitud vahendite puhul.

3.1.4.1 TLS-CCA ja Eesti ID-kaardiga autentimine

Kuivõrd ID-kaardi kasutuse algusaegadel oli ID-kaart ja temaga tehniliselt samaväärsed digiallkirjavahendid ainsad eksisteerivad digiallkirjavahendid, läks käibele lihtsustatud oskusteave ID-kaardi jaoks TLS-CCA konfigureerimise kohta, mis tegeleb ainult ühe aspektiga kogu vajalikust konfiguratsioonist: usaldatud sertifitseerimisahelate määramisega.

Tänaseks on toimunud kaks olulist muutust:

- ID-kaartide sertifikaate väljastatakse mitmest sõltumatust sertifitseerimishierarhiast.
- ID-kaartidega samast sertifitseerimishierarhiast väljastatakse sertifikaate ka teistsugustele QSCD (*Qualified Signature Creation Device*) vahenditele, näiteks Mobiil-ID.

See tähendab, et laialt kasutusel olev konfiguratsioon,[3] mille sisuks on õigesti paigaldatud

[1] Seda võimaldaks ka *Token Binding* aga selle standardimine on peatunud ja pole tõenäoline, et brauserid seda nähtavas tulevikus toetama hakkavad. Standardimist vedasid seni just brauseritootjad.

[2] https://www.europeanpaymentscouncil.eu/sites/default/files/kb/file/2018-05/APIEG30-18Authenticationguidance(SCA).pdf

[3] https://wiki.itcollege.ee/index.php/ID_kaardiga_autentimine_Apache2_veebiserveriga

sertifikaadihierarhiad, ei anna tegelikult soovitud turvaomadust – kindlust, et krüptograafilisel tasemel oleks TLS-CCA autentimine võimalik vaid ID-kaardiga.

Selleks, et saavutada korrektset konfiguratsiooni, on vaja pidada silmas järgmist.

- Sertifikaadi kuulumine kindlasse sertifikaadihierarhiasse näitab seda, et sertifikaadi sees olev informatsioon on usaldusväärne, seejuures tuleb aga silmas pidada seda, et vaadata tuleb hierarhiat tervikuna, mitte ainult juursertifikaati. Sama juursertifikaadi all võib olla hierarhia kõrvalharusid, mida ei tohi etteantud otstarbeks usaldada.
- Informatsioon sertifikaadi esitaja ning lubatud kasutusotstarbe kohta on saadaval üksnes sertifikaadis endas.

Sertifikaadihierarhiate konfigureerimise osas soovitame pöörduda vastavate tehniliste juhendmaterjalide poole[1,2]

Selleks, et tuvastada, kas konkreetne sertifikaat kuulub ID-kaardi autentimissertifikaatide hulka, tuleb kontrollida sertifikaadi laiendust `X509v3 Certificate Policies`, mis sisaldab sertifikaadi väljastamise aluseks olevate poliitikate identifikaatoreid (OID-sid)[3,4] Selle aruande kirjutamise ajal kehtivad OID-d on loetletud tabelis 4 – ainult sellised sertifikaadid, mis on väljastatud tabelis loetletud CAde poolt ning milles sisaldub vastav poliitika OID, on ID-kaardi (ja sarnaste kiipkaartide) autentimissertifikaadid.

Tabel 4. ID-kaardi autentimisvõtme sertifitseerimispoliitikad

Poliitika nimi	Väljastav CA	OID
CP for older ID-card	ESTEID-SK 2015	1.3.6.1.4.1.10015.1.1
CP for older Digi-ID and older Digi-ID of e-residents	ESTEID-SK 2015	1.3.6.1.4.1.10015.1.2
CP for newer ID-card of Estonian citizen	ESTEID2018	1.3.6.1.4.1.51361.1.1.1
CP for newer ID-card of EU citizen	ESTEID2018	1.3.6.1.4.1.51361.1.1.2
CP for newer Digi-ID	ESTEID2018	1.3.6.1.4.1.51361.1.1.3
CP for digital identity card of e-resident	ESTEID2018	1.3.6.1.4.1.51361.1.1.4
CP for residence card of long-term resident	ESTEID2018	1.3.6.1.4.1.51361.1.1.5
CP for residence card of temporary resident	ESTEID2018	1.3.6.1.4.1.51361.1.1.6
CP for residence card of family members of citizen of EU	ESTEID2018	1.3.6.1.4.1.51361.1.1.7
CP for diplomatic identity card	ESTEID2018	1.3.6.1.4.1.51455.1.1.1

3.1.5 TLSi turvamise muud aspektid

Lisaks TLSi protokollile ja kasutatavatele krüptoalgoritmidele vajavad kaitsmist ka muud ühenduse turvalisust mõjutavad aspektid.

[1] https://github.com/SK-EID/smart-id-documentation/wiki/Secure-Implementation-Guide

[2] https://www.id.ee/artikkel/id-kaardiga-isikutuvastus-veebilehel-serveriseadistused-probleemilahendused-jms-2/

[3] https://www.skidsolutions.eu/en/repository/CP/

[4] https://www.skidsolutions.eu/upload/files/SK-CPR-ESTEID2018-EN-v1_2_20200630.pdf

Paljude rakendusprotokollide juures on TLS kasutusel valikuna – alustatakse mittekaitstud ühendusega ja kooskõlastuse (*negotiation*) käigus võidakse kokku leppida TLSi kasutamine.[1] Kuna teadmise TLS-ühenduse võimalikkusest saab klient üle turvamata ühenduse, on võimalik madaldusrünne (*downgrade attack*), kus vahemees eemaldab kliendile saadetavatest pakettidest teadmise TLSi kasutatavuse kohta ja sunnib sedasi ka edasises ühenduses TLSi mitte kasutama (sunnib ühenduse turvataseme madalamaks, kui see oleks ilma rünnakuta). Sarnane teguviis oleks HTTP-kaudu serveeritud lehelt ümbersuunamise eemaldamine HTTPSi kasutavale lehele. Keerulisemal juhul suhtleb vahemees päris serveriga üle turvatud kanali, kuid kliendiga suhtlemiseks kasutab jätkuvalt turvamata ühendust. Ainus soovitud turvagarantiisid andev lahendus on, et klient teab algusest peale, kui peab kasutama turvatud ühendust ning omab toimivat viisi serveri autentimiseks. Sellise info edastamiseks on välja pakutud DNSi-põhiseid lahendusi (näiteks DANE ja MTA-STS[2]), aga ka veidi nõrgemate turvagarantiidega TOFU-stiilis (*Trust On First Use*) lahendusi (mööndustega näiteks HSTS [44]).

Oluline osa TLS kätlusest on poolte autentimine. Tavaliselt autenditakse TLS tasemel vähemalt server, teenustevaheliste ühenduste korral sageli mõlemad pooled. TLS protokollina pakub autentimiseks kas eeljaotatud saladuse kasutamist või viisi tõestamiseks, et tuvastatav pool saab kasutada deklareeritud avalikule võtmele vastavat salajast võtit. Eeljaotatud saladuse kasutamine on mõistlik vaid väga spetsiifilistes olukordades – nt seadmetes, mille jõudlus ei võimalda avaliku võtme krüptograafiat kasutada – ning on sellisena erand.

Avaliku võtme krüptograafia kasutamisel on autentimiseks vaja TLSi-välist viisi avaliku võtme sidumiseks vastava võtmepaari kasutaja identiteediga. Tavaliselt kasutatakse avaliku võtme kuuluvuse näitamiseks X.509 sertifikaatidel põhinevat avaliku võtme taristut (PKIX, *Public Key Infrastructure (X.509)* [14]). PKIX kõige levinuma kasutusjuhu – HTTPS serverite sertifikaatide – puhul esitab HTTPS-server sertifikaadi, mis sisaldab kokkulepitud kujul serveri DNS-nime (mille järgi klient tavaliselt pöördumist alustab) ning mille kohta server suudab näidata kliendi poolt usaldatud CAga lõppeva sertifitseerimisahela.

PKIX asemel või selle kõrval võivad pooled kasutada sertifikaatide või avalike võtmete kinnistamist (*pinning*). Sellisel juhul on poolel mingi informatsioon teise poole sertifikaadi või selles sisalduva avaliku võtme kohta. Kinnistamist kasutatakse enamasti rakendustevahelises sides, kus osapooli on vähe ning need on eelnevalt teada. Avalikes protokollides (nt HTTPS) on probleemiks kinnistamisinfo usaldusväärne levitamine.

Võimalik probleem PKIX kasutamisel on ka võltssertifikaadid. Sertifikaatide võltsimist võimaldavad väga mitmesugused nõrkused alustades krüptograafilistest probleemidest mõne sertifikaadi või kasutatava võtmepaariga ahelas ja lõpetades organisatsiooniliste probleemidega CAdes. TLSi kontekstis on võltssertifikaadi tagajärjel võimalik vahemeherünne (MITM, *Man in the Middle*) mõne osapoole vastu. Ajalooliselt on sertifikaatide võltsimiseni viinud mõne CA võtme murdmine, pahatahtlik või lohakas (sertifikaaditaotlust mitte piisavalt kontrollinud) CA. Võltssertifikaatide probleemi leevendamiseks kasutatakse kinnistamist ja sertifikaatide läbipaistvuslogi (*Certificate Transparency*).

TLSiga seotud turvameetmena saab vaadelda ka CAde ja brauseritootjate foorumi[3] alusnõuetes kehtestatud (serveri)sertifikaatide maksimaalset lubatud kehtivusaega. Piiratud kehtivusaeg vähendab valesti väljastatud sertifikaadi ajalist mõju (kui selline sertifikaat muul viisil avastamata

[1] https://en.wikipedia.org/wiki/Opportunistic_TLS
[2] https://tools.ietf.org/html/rfc8461
[3] https://cabforum.org/

jääb) ning võimaldab kiiremini kehtestada uusi nõudeid sertifikaatidele. Praegu (alates 1. septembrist 2020) kehtib nõue, et avalikud CAd ei tohi väljastada sertifikaate kehtivusega rohkem kui 398 päeva. Mozilla soovitab oluliselt lühemaid (kuni 90 päeva) kehtivusaegu, väites, et see soodustab sertifikaatide uuendamise automatiseerimist ja vähendab praegust, sageli veaohtlikku käsitsi uuendamist. Sertifikaadi harva käsitsi uuendamine ununeb kergesti ning selle käigus tehakse sageli vigu usaldusahela konfigureerimisel jm, see kõik põhjustab käideldavusprobleeme.

Järgnevalt kirjeldame lähemalt valikut mehhanisme, millega TLS ühendusi täiendavalt turvatakse.

3.1.5.1 HSTS – *HTTP Strict Transport Security*

HSTS [44] on mehhanism, mille kaudu veebiserveri omanik saab deklareerida, et tema serverit peaks kasutama vaid turvalist ühendust kasutades. See on mõeldud kaitsena madaldusrünnete vastu, aga aitab ka serveripidaja näpuvigadest põhjustatud mitteturvalise kanali kasutamise vastu.

Turvalise ühenduse kasutamise poliitika võib kehtida kas ainult täpselt sellele serverile/domeeninimele, mille poole kasutaja pöördus või ka kõigile alamdomeenidele. Koos HSTS-poliitika deklareerimisega määrab server selle kehtimise lõpu aja. Kui kasutaja brauser on HSTS-deklaratsioonist teadlik, siis asendab see kõik asjassepuutuva(te) domeeni(de) suunas tehtud mitteturvalised (`http`) päringud automaatselt turvalistega (`https`). Kui turvalise ühenduse tegemine ei õnnestu, siis annab brauser kasutajale vea ning spetsifikatsiooni kohaselt ei tohi brauser võimaldada HSTS poliitika eiramist.

Levinumad veebibrauserid sisaldavad kõik eelkonfigureeritud poliitikaid.

HSTS poliitika kasutuselevõtmisel tuleb arvestada asjaoluga, et see rakendub ka samas domeenis või selle alamdomeenis olevatele test- ja arendusserveritele. HSTS poliitika rakendamisel on mõistlik rakendada see kogu domeenile (nt mitte ainult `www.example.com`), et välistada kasutaja suunamine aadressidele, mis on küll samas domeenis (nt `vvv.example.com`) kuid HSTS-poliitikaga katmata ning kuhu seetõttu võidakse saata seansivõtmeid vms tundlikku infot sisaldavaid brauserikooke (kui nende skoobiks on määratud terve domeen). Alamdomeenidele rakenduva poliitika korral on oluline jälgida, et poliitika deklareeriks juba domeeni tipmine nimi (nt `example.com`), vajadusel peab sellisele nimele vastava veebiserveri tekitama ning muudest serveritest (nt `www.example.com`) laetud sisust tipmisele nimele viitama, et vastav poliitika kindlasti laetaks.

HSTS ei asenda mehhanisme nagu CSP[1] ning brauserikookide turvalist transporti nõudvat atribuuti *secure*.

Olukorras, kus HTTPS on juba kasutusel ja kogu domeeni ja kõigi alamdomeenide veebiliikluse sundimine HTTPSile on võimalik, on HSTSi kasutuselevõtmine ja HTTP kasutamisest loobumine soovitatav. HTTP kasutamisest loobumine tähendab siinkohal ka seda, et veebiserveri ressursid ei ole üldse üle HTTP kättesaadavad ning päringutele vastatakse ümbersuunamisega HTTPS peale. HSTS rakendamisel on kasulik jälgida OWASPi soovitusi.[2] Kui HSTS on korrektselt

[1] *Content Security Policy*, `https://www.w3.org/TR/CSP/`, sellega saab keelata osa sisu mitteturvalise laadimise

[2] `https://cheatsheetseries.owasp.org/cheatsheets/HTTP_Strict_Transport_Security_Cheat_Sheet.html`

konfigureeritud ja toimib, siis tuleks turvalisuse tagamiseks esitada oma domeen lisamiseks brauseri eelkonfigureeritud nimekirjadesse.

3.1.5.2 MTA-STS – *SMTP MTA Strict Transport Security*

MTA-STS [71] on DNSi- ja HTTPSi-põhine mehhanism, mis võimaldab DNS-domeeni omanikul deklareerida poliitikaid selle kohta, kas ja millised selle domeeni meilivahetusserveritest (MX, *mail exchanger*) on võimelised kasutama TLSi ning mida tuleb teha juhul, kui TLS-ühenduse tekitamine mingil põhjusel ei õnnestu. DNSis hoitakse väidet selle kohta, et domeeni kohta on olemas MTA-STS poliitika ning kehtiva poliitika versiooninumbrit. Tegelik poliitika tuleb alla laadida HTTPSi kaudu aadressilt, mille saab MTA-STSi rakendava domeeni nimest tuletada. Saadud poliitikat võib puhverdada, kuni DNSis olev versiooninumber pole muutunud.

MTA-STS on meede madaldusrünnete vastu, kuid selle kasutamisel peab arvestama, et MTA-STS rakendamine ei ole saatvatele serveritele kohustuslik ning suurem osa neist seda ei teegi. Ka vastuvõttev server ei tea, millised saatjad MTA-STSi toetavad ning nii on muu kanali kaudu edastatud infota keeruline keelduda mitteturvalistest ühendustest. Sellisena pakub MTA-STS märkimisväärselt madalamaid turvagarantiisid kui HSTS ja üldjuhul peab arvestama, et avalik e-postiliiklus ilma täiendavate meetmete rakendamiseta on kolmandatele osapooltele kergesti pealtkuulatav, muudetav ja takistatav.

Sarnast probleemi lahendab DANE SMTP profiil [31]. Praktikute hinnangul on MTA-STS loodud suurte meiliteenuste poolt ning sobibki sellisena neile, samas kui DANE SMTP profiil on populaarsem väiksemate ISPde ja meiliteenuse pakkujate juures.[1] DANE SMTP profiili kasutamisest on juttu ka allpool, DANE kohta käiva jaotise lõpus.

Nii DANE SMTP profiili kui MTA-STSi toetavad saatjad saavad TLS-ühenduse tekitamisel tekkinud vigadest teatada, kasutades standardiettepanekus *SMTP TLS Reporting* [72] kirjeldatud meetodeid.

3.1.5.3 CAA – *Certificate Authority Authorization*

CAA [42] on DNSi-põhine mehhanism, mis võimaldab DNS-domeeni omanikul deklareerida poliitikaid selle kohta, millistel CAdel on lubatud anda välja domeeni kohta käivaid sertifikaate või koondsertifikaate (*wildcard certificate*, `*.example.com`). Sellisena on see osa kaitsest võltssertifikaatide vastu.

CAA-poliitikate tarbijad on on CAd ja see kaitseviis pole mõeldud (publitseeritud) sertifikaatide lõppkasutajale. Sel põhjusel ei põhjusta CAA kasutamine ka lisatööd iga konkreetse TLS-ühenduse autentimisel. Kõik suuremate brauserite toetatud CAd peavad kohustuslikus korras toetama CAA kirjeid. Toetatud CAde deklareerimine CAA kirjetes on hea tava.

3.1.5.4 DANE – *DNS-based Authentication of Named Entities*

DANE [45] on DNSSECil põhinev mehhanism, mille abil saab kirjeldada nõudeid TLS-serveri avalikule võtmele või avaliku võtme autentsust tõestavale sertifikaadiahelale. DANE eesmärk on vähendada TLS-ühenduste turvalisuse sõltuvust usaldatavatest kolmandatest osalistest. DANE rakendamisel kasutatakse kas täiendavat kinnistamist või väliste CAde asendamist DNSSEC poolt pakutava usaldusmudeliga.

[1] `https://github.com/internetstandards/toolbox-wiki/blob/master/DANE-for-SMTP-how-to.md`

DANE defineerib DNS kirjetüübi TLSA. TLSA kirje seob konkreetse DNS-nime, protokolli (TCP, UDP või SCTP) ja pordiga (number) ühe või mitu objekti koos iga objekti kohta käiva metainfoga. Metainfo kirjeldab objekti tüübi (sertifikaat või avalik võti), esituse (sidumisviisi) ja kasutusviisi (st kuidas seda serveri autentimisel kasutada).

Seotav objekt on standardne sertifikaat või avalik võti.[1] Sidumise viis on kas täielik esitus või räsi. Toetatud räsialgoritmid on SHA-256 ja SHA-512. Ühe aadressi, protokolli ja pordi kombinatsiooni kohta võib leiduda mitu TLSA kirjet. Piisab, kui neist ühe määratud tingimused on täidetud. Kui DANE spetsifikatsiooni toetav klient leiab DNSSEC-mõttes turvalised TLSA-kirjed, siis on talle näidatud aadressile ja pordile TLS-ühendust algatades kohustuslik kontrollida, et serveri avalik võti ja/või lisatud sertifikaadiahel vastavad mõnes TLSA-kirjes toodud tingimusele.

DANE kirjeldab neli võimalikku kasutusviisi (nimed täiendavast RFCst 7218 [41]):

- PKIX-TA: „*CA constraint*“. Serveri sertifikaati kontrollitakse tavalisel, PKIX defineeritud [14] viisil, kasutades kliendile varem teadaolevaid ankurvõtmeid (*trust anchor*). Kirjes seotud objekt peab vastama[2] mõnele sertifikaadile leitud sertifitseerimisahelas. Sisuliselt on tegemist CA sertifikaadi või avaliku võtme kinnistamisega.
- PKIX-EE: „*Service certificate constraint*“. Serveri sertifikaati kontrollitakse tavalisel, PKIX defineeritud viisil, kasutades kliendile varem teadaolevaid ankurvõtmeid. Kirjes seotud objekt peab vastama ahela viimasele, serveri enda avalikku võtit sertifitseerivale sertifikaadile. Sisuliselt on tegemist serveri sertifikaadi või avaliku võtme kinnistamisega.
- DANE-TA: „*Trust anchor assertion*“. Serveri sertifikaati kontrollitakse tavalisel, PKIX defineeritud viisil, kuid ankurvõti peab vastama kirjes seotud objektile. Sel viisil saab serveri (domeeni) omanik deklareerida uue CA. Kui TLSA-kirjes on sertifikaadi või avaliku võtme räsi, peab server sertifikaadiahelas esitama ka juursertifikaadi, kuigi see tavaliselt nõutud pole.
- DANE-EE: „*Domain issued certificate*“. Serveri sertifikaat peab otse vastama kirjes seotud objektile. Erinevalt kasutusviisist PKIX-EE, ei kontrollita selle kasutusviisi korral sertifikaadiahelat. See kasutusviis võimaldab DNS tsooni (domeeni) halduril sertifitseerida avaliku võtme TLS-kasutuseks. TLS 1.3 võimaldab poolte autentimisel kasutada ka otse SPKI-struktuuri, ilma sertifikaadita – kasutusviis DANE-EE on üks võimalik viis sellise avaliku võtme sertifitseerimiseks. Kui server esitab avaliku võtme sertifikaadis, siis võib klient mõnel juhul ignoreerida sertifikaadis olevaid nimepiiranguid ja kehtivusaegu (DANE-teadliku SMTP kasutamisel isegi peab neid ignoreerima [31]).

Oportunistlikku TLSi toetavad protokollid (SMTP, IMAP, FTP jne) kasutavad TLSi samal pordil kui turvamata ühendust. See tähendab, et TLS kasutamise alustamine lepitakse kokku turvamata kanalis ning seega on võimalik madaldusrünne – ründaja saab jätta mulje, et server ei toeta TLSi. DANE algse RFC järgi ei too TLSA kirjete olemasolu üldjuhul kliendile kohustust kasutada otse turvatud ühendust või loobuda ühendusest kui ei õnnestunud TLSile üleminekus kokku leppida. Sellise nõude SMTP ja SRV-kirjete kaudu kirjeldatud teenuste jaoks defineerivad RFC 7672 [31] ja 7673 [35] – nii saab DANEi kasutada HSTS analoogina ka teiste teenuste jaoks.

Kasutusviisid DANE-TA ja DANE-EE võimaldavad saada hakkama ilma väliste CAdeta, CA rollis on DNSi tsooni (domeeni) haldaja, kes saab tekitada TLSA kirjeid. Võrreldes praeguste brauseri-CAdega on sellise mikro-CA võimalus võltsida sertifikaate (kui see osapool peaks pahatahtlik

[1] DER-kodeeritud X.509 sertifikaat või SPKI (*Subject Public Key Information*) struktuur

[2] Spetsifikatsioonis kirjeldatud tähenduses

olema või vastav võti lekkima või murduma) piiratud vaadeldava tsooniga. Brauserite toetatud juur-CAd võivad praegu väljastada sertifikaate suvalistele nimedele või ka tekitada uusi CA-sertifikaate.

DANE toimimise eelduseks on toimiv DNSSEC. DNSSECi kasutatakse järjest rohkem, kuid olukorrast, kus kõik või vähemalt enamus kasutajaid omaks võimekust DNSSEC-kirjeid valideerida, ollakse veel kaugel.[1] Sellel on mitmesuguseid põhjuseid, alustades probleemidest võrguseadmetega. Sarnaselt TLS 1.3 juurutamisel leituga – paljud olemasolevad võrguseadmed teevad kasutuselolevate protokollide osas eeldusi ning ei luba läbi või moonutavad pakette, mis neile eeldustele ei vasta. DNSSECi (ja DANE) uued, seni mitte kasutatud kirjetüübid ning signatuuride arvel senisest palju suuremad sõnumid on küll standardikohased, kuid ei tööta sellegi poolest märkimisväärselt suure osa internetikasutajate jaoks korrektselt.

Teine suur DANE kasutuselevõttu aeglustav probleem on TLSi suurima kasutuse, HTTPS-ühenduste põhikliendiks olevate brauserite loojate (majanduslikud) huvid. Brauseritootjad on ärisuhetes CAdega, mille juursertifikaate nad toetavad ning on aastate jooksul loonud mitmesuguseid lahendusi, leevendamaks CAde-põhise PKI probleeme. Nii aitab DNSikirjete võltsimise (probleem, mida DNSSEC lahendab) vastu osaliselt *DNS over TLS* ning sertifikaatide võltsimise vastu aitavad *Certificate Transparency* ja *CAA*-kirjed DNSis. See tähendab, et brauseritootjatel pigem pole huvi toetada lahendusi, mis soodustavad DANE levikut.

DANE-põhised lahendused teevad keerukamaks või välistavad ka legitiimse TLS-liikluse inspekteerimise antiviiruste või tulemüüride poolt.

DANE on võrdlemisi hästi toetatud SMTP jaoks.[2] DANE võimaldab saavutada meiliserverite vaheliste TLS-ühenduste vastastikkuse autentimise[3] ning seetõttu on DANE kasutamine meililiikluse turvamiseks soovitatav.

DANE kasutamine täiendava või peamise autentimisinfo kirjeldamiseks on mõistlik, kuid sõltub kaitstavast teenusest ja selle klientidest (nende võimekusest DANE TLSA-kirjeid kätte saada ja valideerida).

3.1.5.5 Sertifikaatide ja avalike võtmete kinnistamine

Kinnistamisest üldiselt ja selle mõnedest esinemisvormidest oli juttu jaotistes 3.1.5 ja 3.1.5.4 – DANE TLSA kirjed on sisuliselt üks viis sertifikaatide või avalike võtmete kinnistamiseks ning selle info levitamiseks. Kinnistamine vähendab CAde rolli (ainsa) usaldusallikana ning seeläbi maandab võltssertifikaatidest tulenevaid turvariske.

Sisuliselt on ka (ainult) oma CA kasutamine või autentimine eelnevalt vahetatud sertifikaatide või avalike võtmete alusel kinnistamise vorm, nagu on seda ka süsteemi või brauseri sisseehitatud CAde sertifikaadid. Tavaliselt mõeldakse kinnistamise all siiski lõppolemi sertifikaadi või mõne CA sertifikaadi või võtme kontrollimist lisaks tavalisele, süsteemseid usaldusankruid kasutavale PKIX kontrollile.

Kinnistatakse tavaliselt kas terve X.509 sertifikaat või SPKI (*subject public key info*, subjekti avaliku võtme info [14, jaotis 4.1.2.7]) struktuur selle sees. SPKI struktuur sisaldab kasutatava

[1] https://blog.apnic.net/2020/07/10/why-has-dnssec-increased-in-some-economies-and-not-others/

[2] https://github.com/baknu/DANE-for-SMTP/wiki/4.-Adoption-statistics

[3] https://www.ria.ee/sites/default/files/content-editors/kuberturve/turvaline_meilivahetus_avalikus_sektoris_2019.pdf

algoritmi identifikaatorit ning võtmematerjali – avalikku võtit ja asjasse puutuvaid parameetreid vastavalt viidatud algoritmile.

Kinnistust kasutatakse sageli rakendustevahelises suhtluses. Kui suhtlevad osapooled on eelnevalt teada ning neid ei ole palju, võib kasutatavate avalike võtmete eelnev vahetamine olla lihtsam, võrreldes kolmanda osapoole (CA) kasutamisega.

Järgnevad lõigud kirjeldavad lähemalt aspekte, mis on vaja kinnistamise juures läbi mõtelda.

Sertifikaat või avalik võti? Üldjuhul on mõistlik kinnistada avalik võti, kui vastav salajane võti on kinnistaja kontrolli all, ning kinnistada sertifikaat, kui sertifikaat sisaldab kellegi teise kontrolli all oleva võtmepaari avalikku võtit.

Sertifitseerimisahelas olevat, CA kontrolli all olevat võtit võib CA meelega või kogemata kasutada teistsuguste omadustega vahesertifikaadi tekitamiseks ning vaid võtme kinnistamine aitab vältida usaldatavate osapoolte hulga ootamatut laienemist (vt ka jaotis 3.1.4.1).

Sertifikaadi kujul esitatud kinnistus on sageli kasutatav ka otse usaldusankruna, erinevalt SPKI-struktuurist. Sertifikaadi sisse saab lisada ka reegleid avaliku võtme või CA sertifikaadi abil välja antud sertifikaatide kasutusviiside jms kohta, mida PKIX valideerimisalgoritme kasutavad rakendused kasutada oskavad.

Teisalt, avaliku võtme kinnistamine võimaldab kinnistada sellise võtme, mille jaoks sertifikaati veel välja antud polegi. Näiteks võib paljude hajutatud klientidega veebiteenusel olla otstarbekas lisaks aktiivses kasutuses olevale võtmele või sertifikaadile kinnistada „varuvõti“, millele hangitakse sertifikaat siis, kui aktiivne võti mingil põhjusel kasutuskõlbmatuks muutub – salajane võti lekib või murtakse, sertifikaat aegub vms. Erakorralises olukorras võib uue kinnistuse levitamine osutuda võimatuks, seetõttu on hea, kui seda saab levitada eelnevalt. Samamoodi – kui aegunud või mingi muu probleemiga sertifikaat asendada, siis on sageli võimalik kasutada sama võtmepaari ning kui kinnistus käib võtmepaari kohta, siis kehtib see edasi.

Terve struktuur või räsi? Kinnistuse esitamise viis sõltub peamiselt kinnistust kasutava süsteemi võimekusest (milliseid esitusi on see võimeline kinnistusena kasutama) ning sellest, kuidas kinnistust kommunikeerima peab.

Süsteemid, mis kasutavad kinnistavat sertifikaati (või ka avalikku võtit) usaldusankruna, vajavad üldjuhul tervet struktuuri. Olukordades, kus on vajalik kinnistusinfo sage edastamine (näiteks DANE TLSA kirjetes) või algse võtmematerjali varjamine („varuvõtmed“ rakendustes, mida levitatakse olulise viivitusega, näiteks mobiilirakendused), on eelistatum räsi.

CA või lõpp-olem? Üldjuhul kehtib väide, et mida lähemal paikneb kinnistus lõppolemile, seda vähem on võtmeid ja neid kontrollivaid osapooli, mida usaldama peab. Siiski võib olla otstarbekas kinnistada CA, kui lõppolemi avalik võti muutub sageli või lõppolemeid on palju. CA sertifikaadi (või võtme) kinnistamisel peab kindlasti muul viisil (PKIX kontrollid, nimekontroll) kontrollima, kas lõppolemi sertifikaat on sobiv ja kehtib.

Üks või mitu kinnistust? Peamised kaalutlused on kinnistusi kasutava süsteemi võimekus (mõnel juhul saabki defineerida vaid ühe sertifikaadi või võtme) ning kinnistuse edastamise meetod. Kui kinnistust saab värskendada vaid sama kanalit pidi, mille turvalisust kinnistus

parandab, peab kasutama mitut kinnistust. Ainult ühe kinnistuse kasutamisel on väga suur oht sellise kanali rikkumiseks ja seisundi lukustamiseks.

Kinnistuse uuendamise meetod Juhuks kui kasutusel olevate ja kinnistatud võtmete või sertifikaatidega midagi juhtub, peab leiduma meetod, kuidas kinnistust uuendada. Lihtsamal juhul võib uuendamine käia käsitsi: administraator lisab sihtsüsteemi konfiguratsiooni info uue kinnistuse kohta. Automaatse uuendamise korral peab arvestama uuenduse levimise ajaga ning hoolitsema uuenduse autentsuse eest. Kui uuendamiseks kasutatakse sama kanalit, mida kinnistus kaitseb, tuleb kogu protseduur väga hoolikalt läbi mõelda, et vältida ebaõnnestunud uuendamisest tulenevaid käideldavusprobleeme.

HPKP – *HTTP Public Key Pinning* HPKP [34] on standard, mis kirjeldab HTTP päise, mille abil HTTP server saab kinnistada mõne sertifikaadi oma sertifikaadi sertifitseerimisahelast. Kinnistamiseks kasutatakse avaliku võtme räsi, kinnistamine on täienduseks tavalisele PKIX valideerimisele.

Praktilises kasutuses selgus, et selline kinnistamine on kergesti rünnatav ning suuremad brauserid on HPKP toe praeguseks eemaldanud. Selle asemel soovitatakse kasutada *Certificate Transparency*'t ja sellega seotud HTTP-päist `Expect-CT`. HPKP kasutamine ei ole soovitatav, vajadusel tuleb sertifikaadi kinnistamiseks kasutada muid meetodeid.

3.1.5.6 *Certificate Transparency*

Certificate Transparency [67] ehk CT on mehhanism CAde väljastatud ja kasutusel olevate sertifikaatide kohta avaliku ja muutmiskindla logi pidamiseks. CT tegeleb TLS-serverites kasutavate sertifikaatidega, mille on väljastanud avalikud, brauserite poolt vaikimisi usaldatavad CAd. RFC 6962 [67] kirjeldab kasutamise TLS-ühenduse ja serveri autentimise kontekstis, muudes kohtades – isikusertifikaadid, kliendiautentimine jm CTd ei kasutata.

CT eesmärk on vähendada väga suurt usaldusvajadust brauseritesse vaikimisi lisatud CAde vastu. CT vahenditega peetava logi abil saab tuvastada valesti väljastatud, nt sertifitseerimispoliitikale mittevastavad või domeeniomaniku teadmata väljastatud sertifikaadid ja seega ka halvasti käituvad CAd.

Logitakse väljastatud sertifikaadid ja/või veel väljastamata eelsertifikaadid[1] koos terve sertifitseerimisahelaga lõppsertifikaadist juursertifikaadini. Sertifikaate logisse lisamiseks võib esitada igaüks, tänapäeval on eelsertifikaatide ühte või mitmesse logisse esitamine paljude CAde sertifitseerimispoliitika osa.

Vastuseks logisse lisamise päringule tagastab logipidaja sertifikaadi ajatempli (SCT, *Signed Certificate Timestamp*). CT nõuab, et TLS server esitaks kätluse käigus oma lõppolemi sertifikaadi kohta SCT ühelt või mitmelt logilt ja TLS klient kontrolliks, et vähemalt üks esitatud SCTdest on kehtiv.

Server võib SCT esitada kas laiendusena X.509 sertifikaadis, selleks loodud uues TLS laienduses (`signed_certificate_timestamp`) või TLS-sertifikaadi olekupäringu laienduses („*OCSP stapling*"). Kliendilt oodatakse vähemalt ühega neist kolmest meetodist edastatud SCT aktsepteerimist.

[1] `https://www.thesslstore.com/blog/ssl-precertificates/`

CT defineerib jälgija (*monitor*) ja audiitori (*auditor*) rollid. Jälgija ülesanne on ühest küljest jälgida, et logi lõppu kord juba kirjutatud andmeid ei eemaldataks ning teisalt võrrelda lisandunud sertifikaate teadaolevate sertifitseerimispoliitikate ja sertifikaaditaotlejate kehtestatud reeglitega. Audiitor teeb pistelist logi kooskõlalisuse kontrolli ning kontrollib, et serverite esitatud SCTdele vastavad sertifikaadid tegelikult logisse jõuavad.

CT pakutav väärtus põhineb eeldustel, et a) kõik väljaantud ja kasutusel olevad sertifikaadid on avalikus logis kirjas ning b) jälgijad leiavad kahtlased sertifikaadid üles ning teavitavad neist CAd, domeeniomanikku jt. Algse lubaduse järgi võib igaüks olla jälgija ja seega otsustada ise, kas tema haldusala kohta väljastatud sertifikaadid on õiguspärased. Paraku on teada, et uute CT kirjete maht on märkimisväärselt suur – ööpäevas rohkem kui 40GB uusi andmeid [68]. Sama allikas väidab, et ükski viiest sõltumatust jälgijast, kes võimaldavad domeeniomanikel enda domeeniga seotud logitud sertifikaate otsida, ei anna täielikke vastuseid.

2021. aasta alguse seisuga on oma poliitika CT-kontrolli osas avaldanud Google Chrome[1] ja Apple.[2] Google nõuab, et CT-ühilduvad oleks kõik EV (*Extended Validation*) sertifikaadid, mis on väljastatud pärast 1. jaanuari 2015 ja kõik teised sertifikaadid, mis on väljastatud pärast 1. maid 2018. Apple nõuab CT-ühildumist sertifikaatidelt, mis on väljastatud pärast 15. oktoobrit 2018.

Tähele peab panema ka, et CT ei tegele mitteavalike CAdega. See tähendab, et kasutaja arvutis usaldusväärseks konfigureeritud „kohalik“ CA võib välja anda sertifikaate suvaliste veebiaadresside kohta. Organisatsioonide arvutivõrkudes võib see olla soovitud käitumine (vahemeheründele ülesehitatud sisukontrolli tarkvara jaoks), aga teisalt avab sellise CA privaatvõtme leke võimalusi rünneteks organisatsiooni sisevõrgu kasutajate vastu.

3.2 SSH

Secure Shell (SSH) on krüptoprotokoll, mis on projekteeritud võrguteenuste turvamiseks. SSH esimene rakendus oli kaugterminal, turvamata protokollide nagu *telnet* ja *rexec* asendusena. Praktikas võib SSH-d kasutada igasuguste TCP-põhiste võrguprotokollide turvamiseks. SSH turvalise kasutamise kohta on oma soovitused andnud ka BSI [25].

Tarkvarapaketi SSH esmaversiooni, mis toetas vaid protokolli SSH-1, avaldas Tatu Ylönen 1995. aastal. Protokolli versioon SSH-1 sisaldab mitmeid vigu[3] mille tulemusena ei soovitata seda enam kasutada.

IETF standardis protokolli SSH-2 2006. aastal [70]. Protokoll on krüptograafiliselt parema lahendusega kui SSH-1 ning sisaldab tuge suurema hulga kasutusjuhtude jaoks (näiteks rakenduse-ühenduste multipleksimine üle ühe TCP/SSH ühenduse[4]).

Protokolli SSH-2 krüptograafilisi omadusi määratakse kolme komponendiga: šiffer (*Encryption Algorithm*), autentsustõendi algoritm (*MAC Algorithm*) ja võtmekehtestuse algoritm (*Key Exchange Method*). Normeeritud väärtused nendele parameetritele on kirjas IANA vastavas registris,[5] kuid realisatsioonid ei pea piirduma loetletud algoritmidega, vaid võivad lisada omi.

Seetõttu on raske anda üksikasjalikku juhist SSH šifrite konfigureerimiseks (erinevalt näiteks

[1] https://chromium.github.io/ct-policy/ct_policy.html
[2] https://support.apple.com/en-gb/HT205280
[3] https://www.cvedetails.com/cve/CVE-2001-1473/
[4] https://en.wikibooks.org/wiki/OpenSSH/Cookbook/Multiplexing
[5] https://www.iana.org/assignments/ssh-parameters/ssh-parameters.xhtml

TLSist). On võimalik anda mõned järgimiseks sobivad suunised, kuid pärandsüsteemide ühilduvusnõuded võivad seada omad nõuded.

Vastavad seadistused tuleks sisse viia nii SSH serveri kui kliendi konfiguratsiooni, et vältida madaldusrünnet.

Andmevahetuse kaitsmisest sõltumatult võidakse krüptograafilisi meetodeid kasutada ka kasutaja autentimiseks. Siinkohal on oluline tähele panna, et kord genereeritud võtmepaare kasutavad kasutajad autentimisvahendina aastaid ja seega peab võtmete genereerimisel valima piisavalt pika eeldatava elueaga võtmetüübi ja piisavalt pikaks ajaks turvalisust pakkuva võtmepikkuse. Samuti peab regulaarselt kontrollima süsteemide kaugpääsu võimaldavate võtmete turvataset ja kui see pole enam piisav, siis nõrgaks osutunud võtmed asendama või eemaldama. Lisaks võtme tüübile on oluline ka kasutatav autentimisalgoritm. Näiteks on kõik esimeses selleteemalises RFCs [70] kirjeldatud algoritmid praeguseks kasutuskõlbmatud, sest kasutavad SHA-1 räsi.

Milliseid algoritme konkreetne SSH realisatsioon toetab, tuleb selgitada vastava tarkvara juhendmaterjalidest.

- Šifrite hulgast tuleks kasutada ainult ChaCha20-Poly1305 ning AES-GCM variante, nende puudumisel AES-CTR.
- Autentimiskoodide hulgast tuleks kasutada ainult HMAC algoritme SHA-2 perekonna räsidega.
- Võtmekehtestuse algoritmide hulgast tuleks valida sellised, mis kasutavad DH või ECDH võtmekehtestust, SHA-2 või SHA-3 perekonna räsialgoritme ning selliseid Diffie-Hellmani rühmi või elliptkõveraid, mis vastavalt tabelile 1 pakuvad vähemalt 128-bitist turvataset (näiteks DH rühmad *group1* ja *group14* ei kvalifitseeru, sest nende järgu bitipikkus ei ole piisav).
- Serveri ja kasutaja (kui kasutatakse kasutaja autentimist võtmepaariga) võtmetena tuleb kasutada piisava turvatasemega (v.t tabel 1) elliptkõveratel baseeruvaid võtmeid ja neile vastavaid autentimisalgoritme.

Internetis leidub hulgaliselt materjale[1] SSH turvalise konfigureerimise kohta, kuid nendega tutvudes tuleb alati veenduda, et nood pole aegunud ning kehtiks lugejat huvitava SSH realisatsiooni ja versiooni kohta.

3.3 IPsec

IPsec (RFC 4301 [94] jt) on IETFi defineeritud turvaarhitektuur ja protokollistik, mis lisab krüptograafilise kaitse OSI-mudeli võrgukihis (vastandina nt TLSile, mis teeb seda transpordikihis). IPSeci abil turvatakse näiteks asutustevahelist sidet (kahe paljudest arvutitest koosneva IP-võrgu ühendamine üle üldkasutatava sidevõrgu) aga ka kaugpääsu, mille korral ühenduvad üksikud arvutid turvaliselt kaitstud võrku või serverisse.

IPsec turvaparameetrite konfigureerimise kohta on soovitused andnud ka BSI [24] ja NIST [6].

IPsec kirjeldab:

- turvaühendi (*security association*) mõiste,

[1] Näiteks: `https://linux-audit.com/audit-and-harden-your-ssh-configuration/`

- protokollid turvaühendi kokkuleppimiseks ja haldamiseks ning
- protokollid turvaliseks andmevahetuseks kokkulepitud turvaühendit kasutades.

IKEv2 (*Internet Key Exchange version 2*) [56] on protokoll poolte vastastikuseks autentimiseks ning turvaühendite tekitamiseks ja haldamiseks. Praegune versioon 2 on edasiarendus varasematest IKE v1 ja ISAKMP (*Internet Security Association and Key Management Protocoll*) protokollidest. IKE v2 pakub võrreldes IKE v1ga hulka mugavusteenuseid (laiendatava autentimise tugi, IP-rändluse tugi, seansi elutukse jms) kuid ka IKEv1-l ei ole teadaolevaid nõrkusi ning vajadusel võib ka seda kasutada turvaühendi kokkuleppimiseks. Eelistatud on siiski IKEv2 kasutamine.

IP-pakettide krüptograafilise kaitsega tegelevad protokollid AH (*Authentication Header*) [57] ja ESP (*Encapsulating Security Payload*) [58]. AH tagab tervikluse paketi lastile (*payload*) ja valitud päiseväljadele saatja identiteedi kontrolli. ESP tagab lisaks AH pakutavale ka paketi sisu konfidentsiaalsuse ja piiratud kaitse liiklusvoo analüüsi vastu. IPsec arhitektuur võimaldab AH ja ESP kasutamist ükshaaval või kombinatsioonis, konkreetne kasutus sõltub vajadusest.

Mõlemad protokollid toetavad transportrežiimi ja tunnelirežiimi.

Transportrežiimis lisatakse algsele paketile, algse päise järele, lisapäis, mis seob paketiga turvaühendi ja sisaldab (ESP korral valikuliselt) tervikluse kontrolliks vajalikke andmeid. Marsruutimiseks kasutatakse algse paketi päist ning selles olnud aadresse. Transportrežiimis ei saa tagada IP-paketi muutuvate väljade terviklust ning ESP korral on paketi päis pealtkuulajale nähtav (võimalik on liiklusvoo analüüs).

Tunnelirežiimis on kogu algne pakett uue, turvalüüside vahel saadetava paketi lastiks (*payload*) ning seda ei saa paketi edastamisel muuta. Terviklust kontrollitakse kogu (algse) paketi ulatuses. ESP korral krüpteeritakse kogu pakett – pealtkuulaja ei näe algse paketi lasti ega ka päist.

Soovitame kasutada ESP protokolli tunnelirežiimis.

AH ja ESP poolt kasutavad krüptoalgoritmid, parameetrid ja võtmed määratakse turvaühendi poolt ning need lepitakse kokku IKEt kasutades.Võtmete käsitsi konfigureerimine ilma IKEt kasutamata on tehniliselt võimalik, kuid seda ei peaks lubama, kuivõrd seda on keerulisem hallata ning siis jäävad pooled ilma võtmevärskendamise võimalusest.

IKE protokolli esimese sammuna lepivad IKE pooled kokku IKE turvaühendis ja teise sammuna autendivad üksteist. Esimeses sammus tuletatakse DH-kehtestatud saladusest ja vahetatud nonssidest põhisaladus, millest tuletatakse edaspidi kõik kasutatavad võtmed. IKE turvaühendis kokkulepitud algoritme kasutatakse IKE liikluse (sh ka autentimissõnumite) kaitsmiseks ja vastastikuseks autentimiseks.

Autentimiseks tuletatakse esimestest vahetatud pakettidest ja kokkulepitud saladustest autentimissõnumid, mille kumbki pool peab signeerima. Jagatud saladuse (parooliga) autentimisel kasutatakse sõnumiautentimiskoodi, mille sisenditeks on tuletatud autentimissõnum ja paroolist tuletatud väärtus. IKEv2 jagatud saladusega autentimisprotokoll ei sisalda meetmeid parooli jõuga murdmise vastu. Kasutaja peab tagama parooli piisava keerukuse. Alternatiivselt kasutatav EAP autentimisprotokoll sisaldab meetmeid ka jõuründe vastu.

Pärast IKE turvaühendi kokkuleppimist loovad pooled vähemalt ühe alam-turvaühendi (*child security association*) konkreetsete liikluse-selektorite jaoks. Liikluse-selektor (*traffic selector*) on IP-aadresside vahemik, IP-protokoll ja pordivahemik, millele turvaühendiga määratud kaitset

rakendatakse.

Olemasoleva IKE turvaühendi kontekstis saavad pooled luua uusi turvaühendeid ja juba kokkulepitud turvaühendite võtmeid värskendada (*rekey*). On šifrite töörežiime, mille korral on sama võtmega krüpteeritav andmemaht piiratud (nt AES-GCM korral on see umbes 64GB) ja vajalik on võtmete regulaarne uuendamine. Mõned IPsec teostused ei uuenda võtmeid pärast ettenähtud andmemahu täitumist. Sellisel juhul peab kasutama šifreid, millel mahupiirangut pole või lühendama võtmevärskenduse perioodi nii, et võti uuendataks enne, kui kriitiline andmemaht täituda saab.

3.3.1 Soovitused

IKE on protokoll turvaühendite kokkuleppimiseks. Kõik kokkulepitavad algoritmid on nummerdatud, vastavaid registreid haldab IETF. Järgnevalt on lubatud algoritmid nimetatud IETFi registrites näidatud nimede ja numbritega.

Kõik IKE abil kokkulepitavad võtmed tuletatakse pseudojuhusliku funktsiooni (*Pseudo Random Function, PRF*) abil esimeste sõnumitega kehtestatud Diffie-Hellmani saladusest ja vahetatud nonssidest. See tähendab, et turvalisus sõltub oluliselt DH parameetritest ja kehtestatud saladuse turvatasemest.

3.3.1.1 DH rühmad

DH rühma abil kehtestatud saladust kasutatakse peasaladuse tuletamisel (IKE turvaühendis) ja võidakse kasutada lisasisendina alamturvaühendite loomisel.

Lubatud rühmad[1] on esitatud järgnevas loetelus. Kirjete juures sulgudes olev arv on rühma identifikaator IETF registris.

- 3072-bitine MODP rühm (15)
- 4096-bitine MODP rühm (16)
- 256-bitine juhuslik ECP rühm (19). See on IETF nimi ja identifikaator rühmale NIST P-256.
- 384-bitine juhuslik ECP rühm (20). See on IETF nimi ja identifikaator rühmale NIST P-384.
- 521-bitine juhuslik ECP rühm (21). See on IETF nimi ja identifikaator rühmale NIST P-521.
- brainpoolP256r1 (28)
- brainpoolP384r1 (29)
- brainpoolP512r1 (30)

Soovitame kasutada ECDHd koos Brainpooli elliptkõveraga, viimasel peab olema sobiv turvatase.

Alamturvaühendite saladused (kasutatavad võtmed) tuletatakse peasaladusest ja turvaühendi loomisel vahetatud uutest nonssidest. Protokoll lubab ka (valikulist) uut DH võtmekehtestust, mille abil tekitatakse lisasaladus alamturvaühendi saladuse tuletamiseks. Kui kasutatav IPsec teostus võimaldab sellist täiendavat võtmekehtestust, siis soovitame seda kasutada. Kasutama peaks sama või tugevamat DH rühma kui see, mida kasutati algse (IKE-)turvaühendi kokkuleppimisel.

[1] DH rühmade register: `https://www.iana.org/assignments/ikev2-parameters/ikev2-parameters.xhtml#ikev2-parameters-8`

Ilma täiendava saladuseta sõltub alamturvaühendi saladus ainult peasaladusest ning peasaladuse murdumisel on kõik alamsaladused võimalik tuletada pealtkuulatud sõnumitest – täiendav DH võtmekehtestus annab alamturvaühenditele tulevikuturvalisuse.

3.3.1.2 Pseudojuhuslik funktsioon

Pseudojuhuslikku funktsiooni kasutatakse Diffie-Hellmani protokolliga kehtestatud jagatud saladusest krüptovõtmete pärimiseks.

Lubatud funktsioonid[1] on:

- `PRF_AES128_XCBC` (4)
- `PRF_AES128_CMAC` (8)
- `PRF_HMAC_SHA2_256` (5)
- `PRF_HMAC_SHA2_384` (6)
- `PRF_HMAC_SHA2_512` (7)

Pseudojuhusliku funktsiooni väljundi pikkus peab olema vähemalt sama pikk, kui on kasutatava šifri võti. See tähendab, et 256-bitise võtmega šifreid kasutades ei tohi kasutada AES128-põhiseid funktsioone IETF registrinumbritega 4 ja 8.

3.3.1.3 Tervikluse kontrolli algoritmid

Tervikluse kontrolli algoritme[2] kasutatakse IKE, AH ja ESP protokollides sõnumi tervikluse tagamiseks. Nende kasutamine IKE ja ESP korral on vajalik vaid siis, kui kasutatav šiffer ise terviklust ei taga.

Lubatud sõnumiautentimise algoritmid on:

- `AUTH_AES_XCBC_96` (5)
- `AUTH_HMAC_SHA2_256_128` (12)
- `AUTH_HMAC_SHA2_384_192` (13)
- `AUTH_HMAC_SHA2_512_256` (14)

3.3.1.4 Šifrid

Šifreid kasutatakse protokollides IKE ja ESP lasti (*payload*) krüpteerimiseks (konfidentsiaalsuse saavutamiseks). Mõnedel šifritel/töörežiimidel on tervikluse tagamise omadus.

Lubatud šifrid[3] on:

- `ENCR_AES_CBC` (12) (koos sellega peab kasutama eraldi tervikluse tagamise algoritmi!)
- `ENCR_AES_CTR` (13) (koos sellega peab kasutama eraldi tervikluse tagamise algoritmi!)

[1] Pseudojuhuslike funktsioonide register: `https://www.iana.org/assignments/ikev2-parameters/ikev2-parameters.xhtml#ikev2-parameters-6`

[2] Tervikluse kontrolli algoritmide register: `https://www.iana.org/assignments/ikev2-parameters/ikev2-parameters.xhtml#ikev2-parameters-7`

[3] Šifrite register: `https://www.iana.org/assignments/ikev2-parameters/ikev2-parameters.xhtml#ikev2-parameters-5`

- ENCR_AES_GCM_16 (20)
- ENCR_AES_GCM_12 (19)
- ENCR_AES_CCM_16 (16)
- ENCR_AES_CCM_12 (15)
- ENCR_CHACHA20_POLY1305 (28)

Kõigi loetletud šifritega võib kasutada võtmeid pikkusega 128 ja 256 bitti, vastavalt šifri võimalustele.

3.3.1.5 Autentimisalgoritmid

Autentimisalgoritme kasutatakse protokollis IKE tõestamaks, et identiteediväidet esitav pool teab mingit jagatud saladust või deklareeritud avalikule võtmele vastavat salajast võtit.

Lubatud algoritmid[1] on:

- ECDSA kõveral secp256r1 (NIST P-256), SHA-256 (9)
- ECDSA kõveral secp384r1 (NIST P-384), SHA-384 (10)
- ECDSA kõveral secp521r1 (NIST P-521), SHA-512 (11)
- ECDSA kõveral brainpoolp256r1, SHA-256 (14)
- ECDSA kõveral brainpoolp384r1, SHA-384 (14)
- ECDSA kõveral brainpoolp521r1, SHA-512 (14)
- RSASSA-PSS, kasutades 4096-bitist RSA-võtit, SHA-384 (14)

3.3.1.6 Turvaühendi eluiga

IKE turvaühendi maksimaalne soovituslik eluiga on 24 tundi ja alamturvaühendi eluiga maksimaalselt 4 tundi [24]. Arvestama peab ka kasutatavast šifrist tingitud võimalikke andmemahupiiranguid.

3.4 WireGuard

WireGuard[2] on suhteliselt värske ja moodne VPN-protokoll. Alates märtsi lõpust 2020 sisaldub WireGuard Linuxi tuumas (tuuma versioon 5.6).

Kuigi ühest küljest kannab WireGuardi protokolli dokumentatsioon[3] märkust „*Draft revision*“, on protokoll siiski formaalselt verifitseeritud[4] ning protokolli krüptograafilistes omadustes võib seega suhteliselt kindel olla.

Kas WireGuardi kasutada või mitte, sõltub väga ümbritsevast keskkonnast, võrgutopoloogiast ja kättesaadavast oskusteabest – WireGuard on projekteeritud ennekõike kakspunkt-võrgutopoloogias kasutamise jaoks ning keerulisemate võrkude ülesseadmine võib olla

[1] https://www.iana.org/assignments/ikev2-parameters/ikev2-parameters.xhtml#ikev2-parameters-12

[2] https://www.wireguard.com/

[3] https://www.wireguard.com/papers/wireguard.pdf

[4] https://www.wireguard.com/formal-verification/

komplitseeritud. Ei tohi ka unustada, et konservatiivsemate organisatsioonide jaoks ei ole värske eesliinitehnoloogia peaaegu kunagi hea valik.

WireGuardi krüptograafiaprimitiivide konfiguratsiooni osas ei saa selle aruande kirjutamise ajal soovitusi anda, kuna WireGuardi konfiguratsioon on fikseeritud, aga see sisaldab siiski krüptograafilisi primitiive, mida aruande muudes osades on soovitatud (X25519, ChaCha20, Poly1305). Ühtegi mittesoovitatavat või keelatud primitiivi kasutuses ei ole.

Fikseeritud šifrikonfiguratsioon on WireGuardi üks suurematest nõrkustest, kuna mõne šifri murdumine ei võimalda sujuvat üleminekut, vaid tarkvara tuleb uuendada mõlemas otspunktis korraga. Suuremate installatsioonide puhul võib see olla problemaatiline.[1]

[1] https://blog.ipfire.org/post/why-not-wireguard

4 Krüptokonteinerite vormingud

Väga paljudes rakendustes ja protokollides on tarvis edastada krüptograafiliselt kaitstud infot mingis mitte-sidusas režiimis, tavaliselt kas failina või päringuparameetrina mingi suurema/üldisema protokolli sees. Kõige tuntumad näited ilmselt on Eestis laialdaselt kasutatavad ASiC-E ja CDOC konteinerid, vastavalt digitaalallkirja ning krüpteeritud sõnumite edastamiseks.

Neid vorminguid on aga rohkem, anname neist järgnevates jaotistes ülevaate koos soovitustega.

4.1 Cryptographic Message Syntax

Cryptographic Message Syntax (CMS) [47] on ASN.1 ja PKCS#7 põhine vorming, mida kasutakse ka standardsete CAdES [21] ja PAdES [81] signatuuride alusena. Probleemiks on vähene tuntus ning ASN.1 kodeeringu keerukus koos sellest omakorda tulenevate turvaprobleemidega. Samuti on CMS vanem versioon [46] haavatav asendusrünnetele (vt [47, *Abstract*] – sisuliselt tähendab see kõiki täna kasutusel olevaid realisatsioone, kuivõrd täiendus on väga uus).

CMS kasutamine rakenduste ehitusprimitiivina ei ole otseselt ebasoovitav, kuid arendaja peaks hoolikalt jälgima, kas tal on kogu vajalik kogemus ja oskusteave olemas, et niivõrd keerulist vormingut korrektselt rakendada.

4.2 ASiC

Associated Signature Container (ASiC) on ETSI poolt standarditud [32] konteinervorming digiallkirjade jaoks. ASiC oma erinevates variatsioonides ei ole midagi muud kui etteantud struktuuriga ZIP-fail. ASiC standard ise krüptograafilisi omadusi ei sätesta.

4.3 ASiC-E koos XAdES signatuuriga

Eestis on ilmselt kõige levinumaks krüptokonteineriks *Associated Signature Container Extended* (ASiC-E) koos XAdES (*XML Advanced Electronic Signatures*) signatuuriga, mille baasprofiili annab ETSI EN 319 162-1 [32] ning mis on täpsustatud Eesti kohaliku BDOC spetsifikatsiooniga [8].

Eestis kasutatakse seda vormingut kõigi igapäevaste digiallkirjade korral.

Kuivõrd tegu on tehniliselt keeruka vorminguga, on oluline ASiC-E digiallkirjadega töötamiseks kasutada mõnd head teeki valmisolevate hulgast,[1] vastavalt oma platvormile.

[1] https://github.com/open-eid/

4.4 ASiC-E koos CAdES signatuuriga ning PAdES

ASiC-E koos CAdES signatuuriga on vastava XAdES vorminguga analoogne, kuid XML asemel kasutatakse seal signatuuri tehnilise kandjana vormingut CMS. ETSI EN 319 162-1 [32] annab baasprofiili ka ASiC-E konteinerile koos CAdES (*CMS Advanced Electronic Signatures*) signatuurivorminguga.

PAdES (*PDF Advanced Electronic Signatures*) on laiendus PDF vorming, mis võimaldab lisada PDF-failile täiustatud digisignatuure. PAdES on standarditud ETSI poolt standardis EN 319 142-1 [80].

Eestis ei ole PAdES ja CAdES laialdaselt kasutusel, kuid nõue teistes Euroopa Liidu riikides kasutatavate digisignatuuri-vormingute aktsepteerimiseks tähendavad, et ka Eesti infosüsteemid peavad olema valmis sellised allkirju vähemalt valideerima, kasutades pigem heakvaliteedilist valmisteeki.

PAdES vormingul on olnud turvaprobleeme seoses spetsifikatsiooni mitmetimõistetavusega,[1] seetõttu tuleb veenduda, et PAdES-allkirjaga dokumentide verifitseerimiseks ja lugemiseks oleks kasutusel teenused, teegid ja PDF-vaaturid, milles on kõik teadaolevad turvavead parandatud. Vastasel korral on võimalik meelitada kasutaja aktsepteerima sellist kuvatud dokumendi ja digiallkirja kombinatsiooni, mida pole tegelikult allkirjastatud.

4.5 JAdES

Teadaolevalt tegeleb ETSI ka JSON-põhise signatuurikonteineri vormingu väljatöötamisega,[2] mis peaks avalikustatama 2021. aasta esimeses pooles.

4.6 CDOC

CDOC on *XML Encryption* [48] baasil koostatud Eesti kohalik vorming krüpteeritud andmete edastamiseks. Krüpteerimiseks kasutatakse avalikule võtme abil krüpteerimist (RSA PKCS#1 v1.5, ECC *Integrated Encryption Scheme* (IES)). Selliselt krüpteeritud sõnumite dekrüpteerimiseks vajalikke liideseid pakuvad vaid ID-kaart ja krüptopulgad (Mobiil-ID ja Smart-ID ei ole toetatud).

CDOC vorming ei paku sõnumi autori autentsust. Kui autentsuse tuvastamine on vajalik, tuleb edastatavad materjalid kas enne või pärast krüpteerimist digitaalselt allkirjastada. Samal ajal tuleb tähelepanu pöörata sellele, et selline kaheastmeline krüptograafiliste meetodite rakendamine ei pruugi anda soovitud kaitset: ründajale jääb tihtipeale võimalus tulemust manipuleerida [26, p. 4.3] [60]. Täpseid vastumeetmeid on keeruline ette kirjutada, kuivõrd need sõltuvad konkreetsest rakendusest kus CDOCi kasutatakse, kuid üldkasutatav soovitus on allkirjastada enne krüpteerimist ning hoolitseda selle eest, et allkirjastatud sisu hulgas oleks ka see info, kellelt kellele see sõnum (dokument) on mõeldud liikuma [2].

Oluline on märkida, et CDOC vorming ei paku tulevikuturvalisust, mistõttu ei tohi seda kasutada pika salastusajaga materjalide krüpteerimiseks. Hoiatav näide sellest, et tulevikuturvalisuse puudumine on probleem, oli 2017. aasta ROCA intsidendi kriis, mille käigus selgus, et ID-kaartide

[1] https://www.pdf-insecurity.org/

[2] https://portal.etsi.org/webapp/WorkProgram/Report_WorkItem.asp?WKI_ID=52897

RSA privaatvõtmed on murtavad. Muuhulgas tähendab see ka seda, et sellistele võtmetele saadetud CDOC konteinerid on piisavalt võimekate ründajate poolt avatavad [64].

CDOC uue versiooni väljatöötlust alustati Riigi Infosüsteemi Ameti projekti „CDOC2.0 analüüs (info pikaajalise salastamise lahenduse arhitektuur)“ käigus ning need tööd jätkuvad [79].

4.7 JSON Object Signing and Encryption

JSON Object Signing and Encryption (JOSE) [7] on üldnimetaja JSON-põhiste krüptograafiliste sõnumivormingute perekonnale[1] - JWS, JWE ja JWT.

Kuigi JOSE vorminguperekond on moodne ja lihtsasti kasutatav, on tema suureks puuduseks (JSONist pärinev) skeemikirjeldusvahendite puudumine, mis sunnib igal platvormil programmeerijat väga hoolikalt läbi mõtlema, kuidas JOSE struktuure turvaliselt parsida ning töödelda, samuti tekib probleeme ühilduvusega, kui on vaja vahetada JOSE struktuure erinevate platvormide vahel. On teada, et olemasolevad teegid ei taga kõiki vajalikke turvakontrolle, võimaldades näiteks asendusründeid konteinerite metaandmete vastu. Teiseks puuduseks on JOSE standardiga määratud krüptoalgoritmide suletud nimekiri, mis ei võimalda JOSE struktuure kasutada standardiga määratud registris[2] loetlemata algoritmidega (näiteks lävikrüptograafiat kasutades).

JOSE kasutamine (väljaspool avalikke standardeid, mis seda ette kirjutavad) ei ole otseselt ebasoovitav, kuid tuleb arvestada, et tegu on keerulise ja kõrge riskiastmega vorminguga, mille korrektseks rakendamiseks on vaja kaasata arendustegevusse turvainsener, kes koostab konkreetse rakenduse jaoks sobivad profiilid ning töötlusreeglid.

Riiklikes infosüsteemides kasutamisel (eriti kehtib see *OpenID Connecti* ja JWT kohta) soovitame killustumise vähendamiseks asutustevahelist kooskõlastamist ja profiilide koostamist.

4.7.1 JSON Web Signature

JSON Web Signature (JWS) [51] on vorming andmete tervikluse krüptograafiliseks kaitsmiseks kas digitaalsignatuuri või autentimiskoodi abil.

Oluline on märkida, et JWS ei kuulu eIDASe poolt tunnustatud allkirjavormingute hulka.

4.7.2 JSON Web Encryption

JSON Web Encryption (JWE) [54] on vorming andmete krüpteerimiseks (sh autentiva krüpteerimise abil).

4.7.3 JSON Web Token

JSON Web Token (JWT) [52] annab standardse viisi väidete (*claims*) edastamiseks kahe osapoole vahel JWS või JWE kujul ning määrab kindlaks mõned standardsed väited.

Erinevalt JWS ja JWE standarditest määrab JWT ära avateksti struktuuri (JSON objekt) ja jadastusreeglid, samuti reeglid vormingu lõppkujule, et seda oleks võimalik kasutada piiratud mahu ja tähestikuga vormingutes – näiteks HTTP URL osisena.

[1] Hoolimata sellest, et vormingu aluseks on JSON, võib kaitstav avatekst olla ka muule süntaksile vastav baidijada.

[2] `https://www.iana.org/assignments/jose/jose.xhtml`

Kõige tuntum ja levinum JWT standardrakendus on OpenID Connect protokollistik [91].

4.8 PGP

PGP (standarditud kui OpenPGP [36]) on paremate valikute puudumisel endiselt suhteliselt populaarne, kuid nagu krüptograafid ja infoturbeeksperdid on korduvalt[1, 2] tähelepanu juhtinud, on tema „parim enne“ tähtaeg ammu möödas. Ennekõike on probleemiks ülearu komplitseeritud ja pika aja jooksul *ad hoc* meetodil arendatud konteinervorming ning ajaproovile mitte vastu pidanud projekteerimisotsustel baseeruv etalonteostus (GnuPG). Seda illustreerib turvanõrkus *SigSpoof* numbriga CVE-2018-12020,[3] mis taas kord osutab sellele, kui ohtlik on andme- ja juhtkanalite omavaheline segamine tarkvarasüsteemides (tavaline *SQL injection* tuleneb samast veast).

Vormingu omapärade tõttu ei ole võimalik neid probleeme ka parandada [26, p. 3.1].

Realisatsiooni ja vormingu puudused üheskoos tekitavad probleeme nii vahetatavate sõnumite tervikluse kui konfidentsiaalsusega [76].

Sarnaselt CDOCiga puudub ka PGPl tulevikuturvalisuse omadus.

PGP kasutamine ei ole soovitatav.

[1] https://blog.cryptographyengineering.com/2014/08/13/whats-matter-with-pgp/
[2] https://latacora.micro.blog/2019/07/16/the-pgp-problem.html
[3] https://cert.europa.eu/static/SecurityAdvisories/2018/CERT-EU-SA2018-016.pdf

5 Autentimis-, volitus- ja delegeerimisprotokollid

5.1 OAuth 2.0

OAuth 2.0 [43, 27] on karkass (*framework*), mis defineerib volitusprotokolli (kuigi täpsem oleks öelda „delegeerimisprotokolli“). OAuth võimaldab kolmanda osapoole klientrakendusel saada piiratud pääsuõiguse HTTP teenustele. Kõige tavalisem kasutus on, et kasutaja annab ühele teenusele piiratud päringuõiguse mingis teises teenuses hallatavale ressursile.

Näiteks: kasutaja annab fototöötlusrakendusele piiratud ajaks õiguse lugeda ja muuta fotohoidlas asuvaid ja talle kuuluvaid pilte. Kasutaja on ressursi omanik. Fototöötlusrakendus on klientrakendus. Õigus lugeda ja kirjutada on delegeeritav piiratud õigus. Fotohoidla on teenus, millele klientrakendus juurdepääsu vajab. Pildid hoidlas on ressurss, mille kasutamise jaoks õiguseid delegeeriti (volitus anti).

On oluline silmas pidada, et OAuth 2.0 ei ole protokoll universaalseks volitustehalduseks, vaid ainult konkreetse kasutaja õiguste täielikuks või osaliseks delegeerimiseks mõnele rakendusele.

OAuth karkassile on ehitatud palju mitmesuguseid laiendusi nagu OpenID Connect[1] autentimisinfo standardiseeritud edastamiseks ja User Managed Access[2] pääsupoliitikate haldamiseks kasutajakeskselt, aga ka laiendused protokolli erinevatele aspektidele nagu parameetrite edastamine, klientrakenduse autentimine, volitusserveri parameetrite avastamine, reeglid pääsutõendite kasutamise kohta jms. OAuth volitusprotokolli nüansside ja kasutuste erinevates kontekstides kirjeldamisega tegeleb IETF *Web Authorization Protocol* töörühm[3]

OAuthi turvamudelit, st ohtusid millega on protokolli projekteerimisel arvestatud ning seda kuidas nende vastu kaitstakse, kirjeldab IETF RFC 6819 [69]. Oluline on märkida, et OAuth ei ole autentimisprotokoll[4] ja selle kasutamine autentimiseks ilma sobivate täiendusteta (nagu seda enne protokolli OpenID Connect standardimist sageli tehti) võimaldab mitmesuguseid ründeid.[5] Näiteks ei ole klientrakendusel üldjuhul võimalik kontrollida, et pääsutõend väljastati just talle või samas seansi vältel.

OAuthi abil väljastatud pääsutõendeid saab kasutada ka muudes protokollides peale HTTP, kuid OAuthi protokolliga kirjeldatud sõnumite vahendamine üle mingite muude protokollide peale HTTP pole OAuthi turvamudelis kirjeldatud ning sellised kasutusjuhud vajaksid eraldi turvaanalüüsi.

[1] https://openid.net/specs/openid-connect-core-1_0.html
[2] https://kantarainitiative.org/confluence/display/LC/User+Managed+Access
[3] https://tools.ietf.org/wg/oauth/
[4] https://oauth.net/articles/authentication/
[5] http://www.thread-safe.com/2012/01/problem-with-oauth-for-authentication.html

OAuth 2.0 projekteerimisel on arvesse võetud varasemate versioonide (1.0 ja 1.0a) kogemusi, kuid ta erineb teostuselt oluliselt. OAuth 2.0 parandas mitmed varasemates versioonides esinenud nõrkused; varasemad versioonid ei ole enam kasutamiseks turvalised. Standardimisel on ka versioon 2.1,[1] mis lihtsustab praegust kirjeldust ning koondab senised head turvatavad otse põhistandardisse. Olulisemad kavandatud uuendused, mis peaks standarditama versioonis 2.1 on järgmised.

- PKCE [92] on edaspidi volituskoodi kasutamisel kohustuslik.
- Naasmisaadresside võrdlemist on täpsustatud.
- Kaudne kinnitus (*Implicit Grant*) on standardist eemaldatud.
- Ressursiomaniku parooli kasutamine kinnitusena on standardist eemaldatud.
- Esitajatõendite (*bearer token*) edastamine URLi argumentidena on standardist eemaldatud.
- Värskendamispiletid peavad edaspidi olema seotud konkreetse kliendiga või olema vaid ühekordseks kasutamiseks.

5.1.1 Protokoll

OAuth2 protokolli järgi väljastab volitusserver klientrakendusele pääsutõendi(d) (*access token*), mida klientrakendus saab kasutada (HTTP) ressursiserverile päringute tegemiseks.

Abstraktsel tasemel koosneb protokoll kolmest sammust:

1. Volituskinnituse (*authorization grant*) hankimine. Klientrakendus hangib kasutajalt (võimalik, et volitusserveri vahendusel) volituskinnituse.
2. Pääsutõendi hankimine. Volitusserver väljastab volituskinnituse alusel klientrakendusele pääsutõendi.
3. Pääsutõendi kasutamine. Klient kasutab pääsutõendit päringus ressursiserverile.

Erinevate omaduste ja võimekustega klientrakenduste teenindamiseks defineerib OAuth erinevad viisid volituskinnituse ja pääsutõendi hankimiseks.

Tavaliselt kasutatakse volituskoodiga kinnitamist (*Authorization Code Grant*), kus pääsutõendi väljastamine algab veebibrauseri vahendusel edastatava volitustaotlusega. See võimaldab volitusserveril korraldada kasutaja autentimist ja/või küsida kasutaja nõusolekut pääsutõendi väljastamise kohta.

5.1.2 Osapooled

Loogiliselt on igas OAuth voos vähemalt kolm osapoolt: klientrakendus, volitusserver ja ressursiserver. Lisaks neile osaleb protokollis tavaliselt ka neljas osapool (ressursi omanik), kelleks on enamasti inimkasutaja.

Klientrakendus (*Client*) on pool, keda kasutaja volitab enda õigustes tegutsema. Klientrakenduste ehitamisel peab arvestama, et protokoll jagab need avalikeks (*public*) ja saladusevõimelisteks (*confidential*). Esimese olek on selle kasutajatele (ja võimalikele ründajatele) täielikult

[1] `https://oauth.net/2.1/`

nähtav ja võimalused volitamisserveri võimalused sellist klienti turvaliselt autentida seega piiratud. OAuth põhistandard kirjeldab olukorda, kus klientrakendus on veebirakendus, hilisemad laiendused [27] ja laienduste mustandid[1] spetsifitseerivad ka kasutamise omarakendustes (*native application*, sh mobiilirakendused) ja tervenisti brauseris käivates rakendustes.

Ressursiserver (*Resource Server*) pakub klientrakendusele kaitstud (HTTP) ressursi. OAuth kontekstis peab ressursiserveri arendaja mõtlema läbi viisid, kuidas selgitada välja ressursiserverile esitatud pääsutõendi tegelik tähendus ja kehtimine – kelle kinnitusel ja mille jaoks see tegelikult väljastati ning kas see on piisavalt värske, et selle alusel pääsuotsuseid teha. Turvakriitilistes süsteemides võib olla vajalik kehtivusinfo sage või igakordne kontrollimine volitusserverist. Arendamisel peaks arvestama pääsutõendite kasutamise spetsifikatsioonidega nagu nt RFC 6750 [53].

Pääsutõendi tähendus on tavaliselt see, et ressursi omanik delegeerib klientrakendusele mingid oma õigused ressursi suhtes. Pääsuotsuse tegemisel peab ressursiserver muuhulgas kontrollima, kas ressursiomanikul endal ikka olid delegeeritavad õigused (vt ka jaotis 5.1.3)

Volitusserver (*Authorization Server*) väljastab klientrakenduse soovil ja vajalike tingimuste täidetuse korral klientrakendusele pääsutõendi (*access token*). Tavaliselt on üheks tingimuseks ressursiomaniku kinnitus (*authorization grant*).

Klientrakendus peab olema volitusserveris eelnevalt registreeritud. Olemas on ka protokollilaiendused, mille abil saab klientrakendus end dünaamiliselt registreerida [88] ja sellist registreeringut hallata [87]. Klientrakenduste ja nendega seotud naasmisaadresside (*URL*) registreerimine ja kontrollimine on vajalik vältimaks kasutaja võimalikku suunamist ründaja ettesöödetud aadressile. Olukord, kus kasutaja saadetakse pärast edukat ja korrektsel aadressil toimunud autentimist ja volituskinnituse küsimist võltssaidile lisab sellisele võltssaidile teenimatut usaldusväärsust. Selle probleemiga peab arvestama ka klientrakenduste dünaamilise registreerimise võimaldamisel.

Pääsutõendite (ja muude volitusserveri väljastatud piletite ja tõendite) kehtivuse ja sisu kontrolliks ja edastamiseks on laiendus *Token Introspection*, RFC 7662 [86].

Ressursiomanik (*Resource Owner*) on kasutaja või süsteem, kellel või millel on õigus ja võimekus kaitstud ressursi kasutamise üle otsustada. Kui ressursiomanik on inimene, siis võib tema kohta öelda ka lõppkasutaja (*End-user*).

5.1.3 Skoop

Volitusserver võib defineerida ja pääsutõenditega siduda skoobid. Klientrakendus saab volituspäringus tellida mingi hulga skoope ning volitusserver võib (näiteks kasutaja valikute põhjal) väljastada pääsutõendi mingi teistsuguse skoopide komplekti jaoks. Kasutatavate skoopide hulk võib olla iga konkreetse klientrakenduse jaoks erinevalt piiratud. Samuti võib olla defineeritud vaikekomplekt, mille volitusserver seob pääsutõendiga, kui rakendus skoobitellimust üldse ei edastanud.

OAuth standard ei defineeri skoobi tähendust ega kasutusi, see on jäetud konkreetsete teostuste ülesandeks – skoop on silt, mille tähenduse defineerib volitusserver. Ressursiomanikult

[1] `https://tools.ietf.org/html/draft-ietf-oauth-browser-based-apps-07`

pääsutõendi väljastamiseks loa küsimisel kuvab volitusserver tavaliselt mingil viisil ülevaate skoopidest, mille kavatseb pääsutõendiga siduda ning võib võimaldada kasutajal skoopide hulka muuta.

Skoope saab kasutada ja on kasutatud näiteks OAuth protokolli laienduste kokkuleppimiseks – OpenID Connect'i kasutamiseks peab klient kasutama skoopi `openid`. Levinud on ka skoopide kasutamine ressursiserveris rakendatavate pääsuõiguste kodeerimiseks, näiteks näitamaks, millistele ressurssidele (meiliaadress, nimi, elukoht jne) ning milliste tegevuste jaoks (lugemine, kirjutamine, kustutamine jne) pääsutõend õiguse annab.

Hea tava on, et klientrakendus küsib ja volitusserver annab pääsutõendi minimaalse vajaliku skoopidekomplekti jaoks. Sh võib kasutada skeeme, kus kasutajalt võetakse kinnitus rohkemate skoopide kohta, kuid pääsutõend väljastatakse vaid küsitud skoopidele. Sedasi saab klientrakendus vajadusel hiljem kasutajat segamata küsida uue, teistsuguse skoobiga pääsutõendi, samas kui iga konkreetse pääsutõendi lekkimisel tekkiv kahju on minimaalne.

Eraldi väärib rõhutamist, et ressursiomanik (kasutaja) saab delegeerida vaid neid õiguseid ja juurdepääse, mis tal endal juba olemas on. Skoopide lisamise või eemaldamisega ei saa (ega tohi saada) seda õiguste hulka laiendada. Näiteks kui kasutajal on mingitele andmetele ainult lugemisõigus, aga volitusserver kirjeldab ka skoobi, mille tähendus on samade andmete kirjutamisõigus, siis on ressursiserveri ülesanne tagada, et sellise, kasutaja kinnitatud ja kirjutamis-skoobiga seotud pääsutõend siiski ei tähendaks kirjutamisõigust.

5.1.4 Volituskinnituse hankimise viisid

Oluline osa OAuth karkassist kirjeldab volituskinnituse hankimist. Sõltuvalt ressursiomaniku ja klientrakenduse võimekusest pakub algne karkass välja neli erinevat viisi [43, jaotis 4] ning olemas on ka mitmesuguseid sellekohaseid laiendusi – näiteks UMA (*User-Managed Access*, vt jaotis 5.3) jt.

OAuth2 kasutamisel volitamisprotokollina oma algsel kujul (st mitte mõne muu otstarbega laienduse osana) piisab tegelikult volituskoodiga kinnituse kasutamisest. Kaudne kinnitus on praeguseks oma algse mõtte kaotanud ja muud kirjeldatud kinnitusviisid on algusest peale olnud mittesoovitatavad üleminekutööriistad.

Volituskoodiga kinnitus (*Authorization Code Grant*) on kõige tavalisem ja enimkasutatud viis kinnituse hankimiseks. Selle algsel kujul kasutamise eeldus on saladusevõimeline (*confidential*) klient, mis suudab end volituskoodi pääsutõendiks vahetamise jaoks turvaliselt volitusserverile autentida.

Avalike klientide jaoks loodi laiendus PKCE [92], mis võimaldab siduda volituskoodipäringu ja hilisema pääsutõendi päringu. Selline sidumine on kasulik ka saladusevõimelise kliendi korral ja selle laienduse kasutamine on soovitav igal juhul.[1]

Klientrakendus ja volitusserver peavad hoolitsema, et volituskood ei lekiks – see peaks olema ühekordseks kasutamiseks ja lühikese kehtivusajaga, klientrakendus peab selle ära kasutama esimesel võimalusel. Kuna volituskood edastatakse klientrakendusele parameetrina ümbersuunamisaadressis, siis satub see sellisena kasutaja brausimisajalukku ning volitusserveri päringulogisse.

[1] OAuth 2.1 teeb PKCE kasutamise kohutuslikuks

Kaudne kinnitus (*Implicit Grant*) on lihtsustatud ning piiratud turvaomadustega autentimisvoog, kasutamiseks avalikes klientides, peamiselt kasutaja brauseris käivitatavates JavaScripti rakendustes. Selline lahendus (kaudne kinnitus) oli vajalik, sest vanemate brauserite turvamudel ei võimaldanud teha pääsutõendi väljastamiseks vajalikku varjatud POST-päringut volitusserveritele, mille allikas (*origin*) erines rakenduse enda omast. Tänapäeval on see AJAXi ja CORSi abil võimalik ning kaudse kinnituse asemel peab kasutama volituskoodi koos laiendusega PKCE.[1]

Kaudse kinnituse korral ei kasutata pääsutõendi väljastamiseks eraldi päringut ega autendita klientrakendust. Kaudse kinnituse korral ei ole volitusserveril lubatud tagastada värskendamispiletit.

Ressursiomaniku parool kinnitusena (*Resource Owner Password Credentials Grant*) ja Kliendi mandaadiga kinnitus (*Client Credentials Grant*) on ajaloolised kinnituse hankimise viisid, mille eesmärk oli soodustada üleminekut pääsutõendite-põhisele pääsuhaldusele. Nende kirjelduse leiab OAuth2 spetsifikatsioonist [43]. Nende kasutamine ei ole tänapäeval soovitatav.

OAuth võimaldab ka muid kinnituse viise, näiteks RFC 7522 [19] defineerib viisi SAML 2.0 esitajakinnituste (*SAML Bearer Assertion*) kasutamiseks volituskinnitusena ja kliendi autentimiseks. Standarditud laiendusi volituskinnitusele registreeritakse muuhulgas näiteks IETF URN OAuthiga seotud kasutuste alamnimeruumis [20].

5.1.5 Pääsutõendid

OAuth protokolli eesmärk on väljastada klientrakendusele sobivate omadustega pääsutõend. Klientrakenduse vaatest on pääsutõend üldjuhul läbipaistmatu (tähenduseta) sõne, mis on mõeldud esitamiseks ressursiserveri(te)le.

Kasutatakse kahte lähenemist – pääsutõend võib olla kas juhuslikult genereeritud sõne või mingi andmestruktuuri esitus baidijadana.

Juhuslikult genereeritud pääsutõendit oskab tõlgendada vaid volitusserver ise. Seega peab server kogu volituskonteksti (kellele, millise ressursi kohta, millal, mis asjaoludel, mille jaoks ja kui kauaks tõend väljastati) säilitama kuni tõendi kehtivusaja lõpuni.

Struktuuriga pääsutõendisse saab saab salvestada volituskonteksti või selle osad ning sellise tõendi kontrollimiseks vajalikud metaandmed. Sellisel otstarbel saab kasutada nt signeeritud ja/või krüpteeritud JWT-struktuuri. Sellist tõendit ei pea volitusserver tingimata ise meeles pidama – vajadusel saab ta ise (või signeerimise korral ka nt ressursiserver) pääsutõendi verifitseerida ja vajalikud andmed tõendist välja lugeda.

Millist lähenemist kasutada, sõltub konkreetsest rakendusest. Juhuslikult genereeritud pääsutõend on tavaliselt lühem ja sellest ei saa lekkida volituskonteksti asjaolusid. Sellist tõendit on ka lihtsam tühistada või muul moel muuta sellega seotud andmeid – ressursiserver peab tõendi kasutamisel selle kehtimise ja kasutamise asjaolude kohta volitusserverile päringu tegema ja sealt saadavad vastused võivad ajas erineda.

Struktuuriga tõendi eelisteks on volitusserveri poolt hoitava andmestiku väiksem maht ning ressursiserveri võimekus tõendit iseseisvalt valideerida. Samas on keerulisem nii tõendiga seotud andmete muutmine pärast tõendi väljastamist kui ka tõendi enneaegne tühistamine (st kehtetuks muutmine varem kui tõendis endas sisalduv tõendi kehtivuse lõpu ajahetk).

[1] https://tools.ietf.org/html/draft-ietf-oauth-browser-based-apps-07

5.2 OpenID Connect

OpenID Connect [91] on OAuth 2.0 karkassile ehitatud identiteediinfo kiht. OpenID Connect spetsifitseerib viisid, kuidas delegeerida kasutaja autentimine autentimisteenusele ja autentimise tulemus klientrakendusele tagastada. OpenID Connect defineerib ka viisid kasutaja profiili-info standardseks edastamiseks.

OpenID Connect defineerib viisid kuidas identiteediinfot kodeerida, kuidas leppida kokku edastava identiteediinfo sisus ning kirjeldab turva- ja privaatsusaspektid protokolli kasutamisel. Sarnaselt OAuthile ei kehtesta OpenID Connect kuidas täpselt volitamisserver kasutaja autentima peab, küll aga võimaldab standardsel viisil kommunikeerida toimunud autentimise asjaolusid ja klientrakenduse soove autentimise omadustele.

OpenID Connect kirjeldust haldab OpenID Foundation.[1] Sama organisatsioon on varasemalt avaldanud standardi OpenID versioonid 1.0, 1.1 ja 2.0, mille arendamisest on praeguseks loobutud. Kuna need varasemad standardid ei võta aluseks OAuthi, vaid kirjeldavad iseseisvat autentimisprotokolli, on oluline veenduda, millisest protokollist räägitakse, kui viidatakse OpenIDle.

5.2.1 Terminoloogia

Ajaloost ja traditsioonidest tulenevalt nimetatakse autentimisprotokollides (sh varasemates OpenID protokollides) protokolli osapooli teisiti kui OAuthis tavaks:

OpenID pakkuja, OP (*OpenID Provider*) volitusserver, mis toetab OpenID Connect protokolli – st korraldab lõppkasutaja autentimise ning esitab selle tulemuse identiteeditõendina.

toenduja, RP (*Relying Party*) klientrakendus, mis kasutab OpenID Connect protokolli kasutaja identiteedi tuvastamiseks ja muude OP pakutavate teenuste tarbimiseks.[2]

autentimispäring (*Authentication request*) OAuth 2.0 volituspäring OPle, milles kasutatakse OpenID Connect skoopi ja muid laiendatud parameetreid, eesmärgiga autentida lõppkasutaja ning edastada identiteediinfo RPle.

lõppkasutaja (*End-user*) inimkasutaja, kelle OP autendib ja kelle identiteediinfo toendujale edastatakse.

5.2.2 Identiteeditõend (*ID Token*)

OpenID Connect defineerib viisi lõppkasutaja identiteedi, st lõppkasutaja kohta käivate identifikaatorite ja muude atribuutide edastamiseks OPlt toendujale. Selleks laiendatakse pääsutõendi väljastamise teenust, mis OpenID Connect standardi kohaselt väljastab lisaks tavalisele pääsutõendile ka JWT-kujul identiteeditõendi (*ID Token*). Identiteeditõendi terviklus ja autentsus tagatakse signeerimise või HMAC (räsipõhise autentimiskoodi) kasutamisega – toenduja peab saama veenduda, et tõendi väljastas õige OP.

Identiteeditõendis seotakse ära omavahel kasutaja identifikaatorid ja atribuudid ning konkreetne autentimisjuhtum ja autentimispäring.

[1] `https://openid.net/foundation/`

[2] „Toenduma“ on A. V. Kõrvi 1936. aasta uudissõnastikus antud tähendusega „(millelegi) kellegi v. millegi pääle toetuma (näiteks kõnes).“ Vastava nimisõna sobib suurepäraselt *Relying Party* vasteks, väljendades osalist ja usaldavat toetumist kellelegi teisele.

Tähtis on tähele panna, et identiteeditõend edastab infot ennekõike autentimisjuhtumi, mitte lõppkasutaja kohta [91, jaotis 2]. Identiteeditõendi ülekoormamine lõppkasutaja kohta käiva infoga ei ole standardikohane lahendus. Lõppkasutaja kohta detailse informatsiooni edastamiseks on ette nähtud kasutajainfo teenus [91, jaotis 5.3].

Identiteeditõendis peavad kindlasti sisalduma

1) väljastaja (OP) identifikaator,
2) lõppkasutaja muutumatu ja mõne teise lõppkasutaja jaoks mitte (taas)kasutatav identifikaator tõendi väljastaja juures,
3) loetelu tõendi võimalike kasutajate identifikaatoritega (st: „kellele tõend väljastati“), toenduja identifikaator OPs peab selles hulgas sisalduma,
4) kehtivuse lõpuaeg sekundi täpsusega,
5) väljastamise aeg sekundi täpsusega.

Kokkuleppel toendujaga võib OP identiteeditõendisse lisada ka suvalisi muid toendujale kasulikke väiteid (*claims*).

Lõppkasutaja identifikaatorina võib kasutada ka mõne kolmanda osapoole (näiteks riigi) väljastatud identifikaatorit, kui sellel on nõutud omadused. OpenID Connect defineerib ka valikulise skeemi, kus lõppkasutaja identifikaator iga konkreetse toenduja jaoks on unikaalne (*pairwise identifier type*). See väldib toendujate omavahelise koopereerumise ilma kasutaja ilmutatud loata.

Soovi korral võib identeeditõendi ka krüpteerida. See võib olla kasulik, kui identiteeditõend liigub avalikes või lekkimisohuga kanalites, näiteks kaudse kinnitamisega voos või hübriidautentimisvoos (vt 5.2.6 ja 5.1.4).

5.2.3 Skoobid

OpenID Connect defineerib skoobi `openid`. Selle kasutamine OpenID Connect autentimispäringutes on kohustuslik, andmaks OPle teada, et kehtivad OpenID Connect sätestatud reeglid.

Täiendavalt defineerib OpenID Connect valiku skoobinimesid, mille abil toenduja saab tellida erinevate atribuudikomplektide lisamise identiteeditõendisse või nende tagastamise kasutajainfo teenuse kaudu. Standardi versioon 1.0 defineerib skoobid `profile`, `email`, `address` ja `phone`.

OpenID Connect kirjeldab ka skoobi `offline_access` kasutamise värskendamispileti tellimiseks ning sellega seotud soovituslikud reeglid.

5.2.4 Kasutajainfo teenus

Muude kasutajaga seotud atribuutide hankimiseks defineerib OpenID Connect (OPle) kasutajainfo teenuse (*UserInfo Endpoint*). See on tavaline OAuth kaitstud teenus, mille kasutamiseks peab toenduja kasutama talle väljastatud pääsutõendit ja mis tagastab hulga kasutaja kohta käivaid väiteid, kas tavalise JSON-objekti või signeeritud ja/või krüpteeritud JWT kujul.

Kasutajainfo teenust saab kasutada selliste kasutaja kohta käivate väidete esitamiseks, mida pole identiteeditõendis mõistlik esitada nende ajas muutumise, mahu või privaatsuskaalutluste tõttu.

Kasutajaatribuutide edastamiseks standardsel viisil kirjeldab OpenID Connect JWT väidete registrisse[1] hulga võimalikke väiteid koos nende esitamise kuju ja tähendusega.

5.2.5 Autenditud päringud

Mitmed OAuth ja OpenID varasemate versioonide nõrkused on tingitud asjaolust, et päringud volitamisteenusele ei ole autenditud. OpenID Connect defineerib viisi, kuidas päringu parameetreid esitada JWT-kujul, mis võimaldab nende signeerimist ja soovi korral ka krüpteerimist.

JWT-kujul päringu parameetrid esitatakse väidetena JWTs. JWT lisatakse tavalisele OAuth päringule kas otse või viida kaudu (URLiga). Parameetrid `response_type`, `client_id` ja `scope` peab edastama (ka) tavaliste OAuth parameetritena, et OP tunneks ära OAuth päringu ja OpenID Connect kasutamise. Muude parameetrite kasutamine JWT kõrval on lubatud. Tavalisel viisil ja JWTs esitatud sama nimega parameetri korral eelistatakse JWTs sisalduvat väärtust.

5.2.6 Autentimisvood

OpenID Connecti eesmärk on võimaldada toendujal hankida kasutaja kohta usaldusväärne identiteediinfo. Selle saavutamiseks tagastatakse koos pääsutõendiga ka identiteeditõend (ning pakutakse täiendava identiteediinfo jaoks kasutajainfo teenust). Kuna OpenID Connect tegeleb lõppkasutaja autentimisinfoga, siis kasutatakse ainult volituskoodiga ja kaudset kinnitamist, mis mõlemad algavad kasutaja brauseri suunamisega volitusteenusele. OpenID Connect laiendab ja täpsustab OAuth defineeritud volitusteenuse parameetreid. Näiteks on parameetrite `scope` ja `redirect_uri` kasutamine autentimispäringus kohustuslik. Täiendavalt saab toenduja täpsustada OP käitumist. Nii näiteks saab olemasolevaid (OP) seansse taaskasutada või tellida veateate kui seanssi pole, nõuda spetsiifilisi autentimistasemeid, anda vihjeid toendujale teadaoleva varasema autentimise kohta ja anda vihjeid kasutaja kasutajaliidesega seotud eelistuste osas.

OpenID Connect defineerib kolm autentimisvoogu:

1. Volituskoodiga voog (*Authorization Code Flow*). Volituskoodi saamine ning pääsutõendiks ja identiteeditõendiks vahetamine sarnaselt volituskoodiga kinnitusele OAuth protokollis. Volitusserver tagastab koos pääsutõendiga ka identiteeditõendi. See on kõige levinum voog autentimiseks veebirakendustes.[2]
2. Kaudne voog (*Implicit Flow*). Volitusteenust kasutatakse nagu kaudse kinnitamise korral OAuth protokollis. Sarnaselt OAuthi kaudsele kinnitusele pole see voog enam soovitatav ja selle asemel peaks kasutama volituskoodiga voogu või hübriidvoogu koos laiendusega PKCE.
3. Hübriidvoog (*Hybrid Flow*). Volitusteenust kasutatakse nagu volituskoodiga kinnituse korral, kuid OP tagastab naasmisaadressile lisatud fragmendis volituskoodi ja ühe või mitu tõendit, sarnaselt kaudsele kinnitamisele. Seda voogu peaks kasutama vaid laiendusega PKCE [92].

[1] https://www.iana.org/assignments/jwt/jwt.xhtml#claims
[2] https://e-gov.github.io/TARA-Doku/TehnilineKirjeldus

5.2.7 OpenID Connect laiendused

Lisaks OpenID Connect põhispetsifikatsioonile on OpenID Foundation juba kirjeldanud või on kirjeldamas mitmesuguseid laiendusi.[1] Laiendustega lisatakse nii täiendavaid funktsioone kui kirjeldatakse profiile erinevate kasutusviiside jaoks.

5.2.7.1 SSO

OpenID Connect laiendused defineerivad viisid, kuidas toenduja saab algatada OP seansi lõpetamise ning meetodid, kuidas OP saab teavitada kõiki toendujaid, keda seansi lõpetamine mõjutab (st kõiki, kes lõpetatavast seansist sõltuvad).

Seansi olemasolul autentimise mittenõudmine võimaldab OPl pakkuda ainulogimist (*single sign-on*, SSO), st skeemi, milles kasutaja peab autentima vaid korra ning pääseb ühe autentimise alusel kasutama kõiki OPga seotud toendujaid. Sellisel viisil kasutab OpenID Connecti näiteks Google. OpenID Connect seansihaldus pakub võimaluse teostada ka *single logout*, mille korral ühest toenduvast infosüsteemist väljalogimine põhjustab seansi lõpetamise OPs ja kõigis teistes toendujatest.

OpenID Connect seansihalduse ja/või SSO kasutamisel on oluline jälgida, et kõigi osapoolte (OP ja kõigi toendujate) seansimudel oleks ühesugune. Sh peaks rakendusest väljalogimise alustamine kõigi toendujate korral tähendama sama asja – kas SSO-seansi ja sellega koos kõigi toendujate seansside lõpetamist või ainult selle konkreetse toenduja seansi lõpetamist. Kui erinevad rakendused käituvad ses osas erinevalt, võib kergesti juhtuda, et kasutaja meelest lõpetatud seanss kestab tegelikult edasi ja seda saab väärkasutada.

5.2.7.2 Födereerimine

OpenID Connect Federation[2] on spetsifikatsioon, mis võimaldab siduda hulga OpenID Connect ja OAuth protokollide olemeid (OPd, RPd, volitusserverid, klientrakendused, ressursiserverid ja födereerimisserverid) ühte või mitmesse usaldushierarhiasse. Praeguses versioonis kirjeldatakse kasutusjuhud OpenID pakkujate ja toendujate jaoks ning uut tüüpi olemite, födereerimisserverite käitumine.

Födereerimine on kasutatav nt juhul, kui autenditavaid kasutajaid (ja nendega seotud atribuute) haldavad paljud erinevad asutused ning kasutajate autentimiseks kasutatakse kohalikke autentimismeetodeid, näiteks paroolikontrolli kohaliku andmebaasiga. Eesti tingimustes on enamasti võimalik kasutada kõigile kättesaadavaid tugevaid autentimisvahendeid ja vajadus sedasorti födereerimise vastu on väiksem.

5.2.8 Muud OpenID Foundationi töögrupid

OpenID Foundation töögrupid tegelevad lisaks OpenID Connect põhispetsifikatsioonile ka OAuth ja OpenID Connect profiilide ja laienduste loomise ja haldamisega.

HEART (*Heart Relationship Trust*) kirjeldab OAuth 2.0, OpenID Connect, FHIR (*Fast Healthcare Interoperability Resources*) kasutamise profiilid. Töögrupi eesmärk on võimaldada lõppkasutajal (patsiendil) ise otsustada enda meditsiiniandmete jagamise üle ning võimaldada

[1] https://openid.net/developers/specs/

[2] https://openid.net/specs/openid-connect-federation-1_0.html

meditsiiniandmeid säilitavatel ja kasutavatel osapooltel nende andmete kasutamine turvaliselt ja kasutaja soovidele vastavalt.

FAPI (*Financial-grade API*) kirjeldab profiile, mille turvatase on piisav pankade jt finantsasutuste pakutavate liideste turvamiseks.[1] Neid profiile kasutataksegi Open Banking APIde ja PSD2 juures.[2,3]

MODRNA (*Mobile Operator Discovery, Registration & autheNticAtion*) kirjeldab protokolle ja profiile, mis laiendavad OAuth ja OpenID Connect kasutamise mobiilivõrkudele. Kirjeldatakse viise, kuidas mobiilioperaator saab olla identiteedipakkujaks (*identity provider*). Mobiilikasutaja identifikaator on selle laienduse kontekstis tema mobiiltelefoni number ja autentimisseadmeks mobiiltelefon. Samuti kirjeldatakse CIBA (*Client Initiated Backchannel Authentication*) autentimisvoog, mis on sarnane sellele, kuidas töötavad Mobiil-ID ja Smart-ID. RP ei suuna kasutaja brauserit OP aadressidele, vaid selle asemel vahendab OP autentimisprotokolli RP ja kasutaja valduses oleva mobiilseadme vahel.

EAP (*Enhanced Authentication Profile*) töögrupp kirjeldab muuhulgas viisi, kuidas kasutada *Token Binding*'ut OpenID Connectiga.[4]

iGOV (*International Government Assurance Profile*) töörühm kirjeldab profiili, mis võimaldab avaliku sektori teenusepakkujatel üle maailma (st ka jurisdiktsioonide üleselt) kasutajaid autentida ning nõusolekupõhiselt kasutajate atribuute vahetada.

5.3 User-Managed Access

User-Managed Access (UMA) on Kantara Initiative[5] hallatav spetsifikatsioonidekomplekt. UMA laiendab OAuth karkassi uue rolliga. Päringuesitaja (*requesting party*) on füüsiline või juriidiline isik, kelle nimel klientrakendus tegutseb. Päringuesitaja rollis võib olla (aga tavaliselt ei ole) ressursiomanikuga sama isik. UMA lisab uue volituskinnituse tüübi, mis võimaldab asünkroonset (ressursiomaniku vahetu sekkumiseta) volitamist ja pääsutõendi väljastamist. Ressursi omanik (samuti füüsiline või juriidiline isik) kehtestab päringuesitaja ja klientrakenduse suhtes pääsupoliitika, mille alusel volitusserver otsustab pääsutõendi väljastamise konkreetsele klientrakendusele.

5.3.1 Protokoll

Protokolli eeldus on, et ressursiomanik on mingil (UMAs) täpsustamata viisil kehtestanud volitusserveris pääsupoliitika. Protokoll kirjeldab klientrakenduse, ressursiserveri, volitusserveri ja päringuesitaja suhtluse; ressursiomaniku suhtlus volitusserveri ja ressursiserveriga ei ole UMA käsitlusalas. Samuti ei käsitle UMA viise, kuidas klientrakendus saab teadlikuks ressursiserveri pakutavatest teenustest.

[1] http://www.slideshare.net/nat_sakimura/openid-foundation-foundation-financial-api-fapi-wg

[2] https://standards.openbanking.org.uk/security-profiles/

[3] https://curity.io/resources/whitepapers/financial-grade-apis-using-oauth

[4] https://openid.net/specs/openid-connect-token-bound-authentication-1_0.html

[5] https://kantarainitiative.org/

1. Klientrakendus teeb päringu ressursiserverile, kuid ei pane kaasa pääsutõendit.
2. Ressursiserver vastab klientrakendusele (pääsuõiguste puudumise) veaga ja lisab veateatele viite volitusserverile ja algse pääsupileti.
3. Klientrakendus küsib volitusserverilt (OAuth pääsutõendite väljastamise teenuse kaudu) päringuesitaja pääsutõendi (*RPT, requesting party token*). Klientrakendus esitab volitusserverile ressursiserverilt saadud algse pääsupileti ja võib esitada täiendavaid väiteid (*claims*) päringuesitaja, klientrakenduse või millegi pääsuotsuse tegemiseks relevantse kohta.
4. Volitusserver hindab esitatud väiteid.
5. Volitusserver võib vastata veateatega väidete ebapiisavuse kohta ning viidata liidesele, mille kaudu päringuesitaja saab interaktiivselt esitada täiendavaid/puuduvaid väiteid.
6. Väidete vastuseks väljastab volitusserver pääsupileti.
7. Klientrakendus küsib volitusserverilt pääsupileti alusel uuesti pääsutõendit (RPT).
8. Volitusserver väljastab klientrakendusele pääsutõendi (RPT).
9. Klientrakendus pöördub ressursiserveri poole, esitades ka pääsutõendi.
10. Ressursiserver kontrollib pääsutõendi ja positiivse tulemuse korral jätkab töötlusega.

Tehniliselt defineerib UMA uue volituskinnituse tüübi – UMA pääsupileti ning viisid selle hankimiseks, uuendamiseks ja kasutamiseks pääsutõendi väljastamisel.

Sarnaselt OAuth põhikarkassile ei kirjelda UMA volituskinnituse spetsifikatsioon ressursiserveri ja volitusserveri vahel toimuvat kommunikatsiooni.

5.3.2 Födereeritud volitamine

UMA 2.0 valikuline lisaspetsifikatsioon kirjeldab födereeritud volitamist (*federated authorization*). Lisaspetsifikatsioon võimaldab ressursiserveril kasutada sõltumatu (UMA) volitamisserveri teenuseid. Selleks defineeritakse viis, kuidas ressursiserver saab enda hallatavad (pääsureguleerimist vajavad) ressursid ning nendega seotud skoobid volitamisserveris registreerida ning viis, kuidas ressursiserver saab volitamisserverilt küsida esialgse pääsupileti (klientrakendusele tagastamiseks).

Födereeritud volitamise abil saab ressursiomanik delegeerida endale kuuluvate ressursside haldamise mingile usaldatavale või sobival viisil poliitikakirjeldamist võimaldavale volitusserverile.

5.4 FIDO ja Web Authentication

FIDO Alliance[1] on tööstusühendus, mille missiooniks on autentimisstandardite abil vähendada maailma sõltuvust paroolidega autentimisest. Organisatsioon edendab autentimis- ja atesteerimisstandardite loomist, kasutamist ja nende kasutamise ühilduvuskontrolli.

FIDO Alliance töötab välja standardeid, mis võimaldavad kasutajatel ühe autentimistegurina (*authentication factor*) kasutada asümmeetrilise krüptosüsteemi võtmepaari. Kogu protokollistiku oluline erinevus näiteks Eestis harjumuspärasest ID-kaardi taristust seisneb selles, et igal kasutajal on palju võtmepaare ning iga võtmepaar on kasutusel autentimiseks vaid ühe

[1] `https://fidoalliance.org/`

rakendusega. See on oluline meede mitmesuguste asendus- ja vahendusrünnete vältimiseks, samuti teeb see raskemaks kasutaja tegevuste korreleerimise erinevate toendujate vahel.

Üldjuhul haldab selliseid võtmepaare mingi iseseisev riistvaraline seade (autentur, *authenticator*) – *FIDO Alliance* standardid kirjeldavad madala taseme kommunikatsiooni selliste seadmetega ning kaugautentimiseks kasutatavaid autentimis- ning atesteerimisprotokolle, mille käigus noid võtmepaare kasutatakse. Tüüpiline selline seade on USB ja NFC liidesega „pulk“.[1]

Täielikult spetsifikatsioonile vastavad autenturid toetavad ka atesteerimist, ehk seadme enese füüsilise ja funktsionaalse tervikluse kontrolli krüptograafiliste meetmetega. Atesteerimine võimaldab toendujal veenduda, et kasutaja privaatvõti on kogu oma elukaare jooksul nõuetekohaselt kaitstud.

Senised spetsifikatsioonid jagunevad kolme gruppi.

- FIDO U2F (*FIDO Universal 2nd Factor*; tuntud ka kui CTAP1) – vanem pärandprotokollistik, millele tuleks võimalusel alati eelistada Webauthn [3] ja CTAP (*Client To Authenticator Protocol*) kasutamist. Defineerib ennekõike krüpotvõtmeid haldava füüsilise autentimisseadme ja selle kasutuse tugeva teise autentimistegurina.
- FIDO UAF (*FIDO Universal Authentication Framework*) – kui U2F eesmärk on pakkuda turvalist teist autentimistegurit, siis UAF kirjeldab suhtluse ka mitmikautentimist võimaldava autenturiga, mis eemaldab vajaduse parooliga autentimise järele. Lisaks U2F autenturitele võimaldab see kasutada autentimisseadmeid, mis enne võtme kasutamist kontrollivad kasutajasamasust ka muul viisil: näiteks biomeetrilise kontrolliga.
- FIDO2 (katab standardid CTAP ja WebAuthn) – täna kehtiv standard tugeva autentimise toetamiseks veebirakendustes.

CTAP spetsifikatsioon [17] kirjeldab protokolli kaht eri versioon: CTAP1/U2F ja CTAP2.

Tänapäeval müüdavad FIDO2 (WebAuthn) autentimisseadmed realiseerivad CTAP2 protokolli ning osa seadmeid realiseerib tagasiühilduvuse huvides ka CTAP1/U2F protokolli.

WebAuthn on W3C standard [3], mis defineerib autentimise veebirakenduses koos asjakohasete protokollide ja API-dega. WebAuthn abil on võimalik autentimiseks kasutada nii U2F kui UAF autentimisseadmeid.

Uutes süsteemides tuleb igal juhul järgida standardit WebAuthn *Level 1* [3]. Ettevalmistamisel on ka standard WebAuthn *Level 2* [4].

WebAuthn toe lisamine infosüsteemidele ei ole paraku triviaalne, see nõuab tuge nii veebirakenduse raamistiku autentimisfunktsioonidelt kui halvemal juhul ka rakenduse kasutajate halduse meetodite täiendamist WebAuthn jaoks vajaliku dünaamilise võtmehaldusega.

5.4.1 FIDO U2F

FIDO U2F on esimene komplekt *FIDO Alliance* poolt välja antud standardeid. FIDO U2F kirjeldab autentimisseadet (*U2F device*), mis oskab genereerida ja atesteerida võtmepaare ning neid hiljem kasutaja autentimiseks kasutada. Toimingud (võtmete genereerimine ja kasutamine) peavad olema kaitstud „kasutaja kohaloleku kontrolliga“ (*user presence*), kasutaja peab vajutama nuppu või muul moel tehingud kinnitama. Igale veebisaidile või rakendusele genereeritakse

[1] `https://www.yubico.com/ee/product/yubikey-5-nfc/`

oma sihtotstarbeline võtmepaar, standard kirjeldab U2F autentimisseadet metafooriga „kasutaja füüsiline võtmerõngas“.

Autentimisseade pakub lihtsat liidest autentimisvõtme loomiseks (kasutaja registreerimiseks) ning hilisemaks autentimiseks. FIDO U2F spetsifikatsioonid kirjeldavad ka veebibrauseris kasutatavad liidesed kummagi operatsiooni jaoks.[1]

Toenduja (*relying party*) on üldjuhul veebirakendus.

U2F mudelis on võtmepaar vaid üks lisa-autentimistegur. Eeldatakse, et kasutaja on eelnevalt mingil muul viisil (nt lihtsa PINiga) autenditud ning toenduja võib igal hetkel nõuda täiendava tegurina võtmepaari kasutamist.

U2F autentimisseadme kasutuselevõtmiseks konkreetses rakenduses kasutatakse registreerimisoperatsiooni. Selle käigus genereerib autentimisseade uue võtme koos võtmepidemega (*key handle*), seob selle toenduja rakenduseidentifikaatori (*application id, origin*) ja kasutajaidentifikaatoriga.

Autentimise õnnestumiseks on vaja, et sobiksid omavahel nii rakenduse poolt autentimiseks küsitud võtmepide, kasutajaidentifikaator kui rakenduseidentifikaator.

5.4.2 FIDO UAF

FIDO UAF spetsifikatsioonid laiendavad ja üldistavad U2F standardeid. Defineeritakse üldine abstraktne autentur (*authenticator*), mis on võimekam variant U2F autentimisseadmest.

FIDO UAF autentur on abstraktne liides seadmele, mis võimaldab kasutaja autentimist FIDO UAF standardis defineeritud viisidel.

UAF võimaldab kasutaja identiteedi kinnitamise erinevatel turvatasemetel:

- kasutaja kohalolu ega samasust ei kontrollita (*Silent Authenticator*);
- kasutaja kohalolu kontroll (*User Presence Check*) – kasutaja peab kinnitama oma kohalolu, vajutades nuppu, viibates NFC-seadmega vms;
- kasutajasamasuse kontrollimine (*User Verification*) – autentur kontrollib kasutaja õigust registreeritud võtit kasutada, tuvastades kasutaja näiteks PINi või sõrmejälge abil. Sisuliselt tähendab see koos võtme kasutamisega kaksikautentimist (kontrollitakse midagi, mida kasutaja teab või milline kasutaja on; võtmepaar on miski, mis kasutajal on).

UAF autentur ja sellega suhtlemise protokollid on kirjeldatud viisil, mis võimaldavad kasutada ka U2F autentimisseadmeid spetsiifilise võimekusega UAF autenturina.

5.4.3 FIDO2 – WebAuthn ja CTAP

Kolmas komplekt FIDO spetsifikatsioone ühtlustab ja üldistab sõnumivahetuse autenturi ja UAF kliendi vahel (CTAP, *Client To Authenticator Protocol*) ning kirjeldab liidese veebibrauserist FIDO autenturite kasutamiseks (*Web Authentication, WebAuthn*).

WebAuthn [3] on projekti FIDO2 peamine tulem ja ta defineerib kasutajate autentimise veebirakendustes. WebAuthn spetsifikatsioon on valminud *FIDO Alliance*'i ja *World Wide Web Consortiumi* (W3C) koostöös ja on avaldatud W3C soovitusena (*recommendation*).

[1] Need liidesed on praeguseks mittesoovitatavad, kasutama peaks liideseid, mille spetsifitseerib WebAuthn.

WebAuthn defineerib oma autenturi mudeli, CTAP-ühilduv autentur on selle mudeli üks võimalik teostus. WebAuthn jagab autentureid:

- Ühendatuse järgi: platvormiautentur on seotud klientseadmega ega pole sellest eraldatav. Platvormiülene autentur (*cross-platform authenticator*) võib suhelda teise seadmega. Kommunikatsiooni vahendava sobiva kihi olemasolul on võimalik (nt mobiiltelefoni sisseehitatud) platvormiautenturit kasutada mingis teises kontekstis platvormiülese autenturina.
- Salajaste võtmete hoidmise viisi järgi: kohalike mandaatidega (*resident credential*, võtit ja sellega seotud atribuute hoitakse autenturi kohalikus hoidlas) autentur ja võtmeid võtmepidemesse kodeeriv autentur. autentur võib toetada mõlemat mandaatide (võtmete) hoidmise viisi, mandaadi loomisel saab valida, kas mandaatide hoidmine autenturis on kohustuslik.
- Kasutajasamasuse kontrollimise (*user verification*) toetatuse järgi: mitmikautentimist (*multi factor authentication*) toetavad autenturid ja seda mittetoetavad autenturid. Mitmikautentimist toetav autentur võib toetada ka rohkem kui kahte autentimistegurit.

CTAP omakorda defineerib sõnumid FIDO kliendi (ehk lõppkasutajaseadmes oleva FIDO tarkvarapinu) ja tegeliku autenturi vahel toimuvaks suhtluseks. CTAP1 on U2F ühilduv, CTAP2 on FIDO2 muude standarditega kooskõlaline uuem protokoll. Reeglina võimaldavad FIDO2 kliendid kasutada ka U2F autentureid. Madala taseme transpordiprotokoll jääb mõlema CTAP versiooni jaoks samaks, kirjeldatakse USB, NFC ja BLE (*Bluetooth Low Energy*) kasutamine.

5.4.4 Soovitused

WebAuthn/CTAP kasutatavate krüptoalgoritmide loetelu on alamosa COSE algoritmidest.[1] Reaalselt kasutatavate algoritmide loetelu piirab ühelt poolt konkreetsete autenturite poolt toetatud krüptalgoritmide valik, teiselt poolt peab aga iga toenduja enda süsteemis piirama kasutatud krüptoalgoritmid ja nende parameetrid vastavalt oma turvaprofiilile.

Meie soovitame igal juhul järgida selle aruande soovitusi krüptoprimitiivide osas. Mõnikord võib see osutuda problemaatiliseks, sest paljud füüsilised tookenid toetavad ebapiisava turvatasemega 2048-bitiseid RSA võtmeid, kuid osa vabavaralisi WebAuthn realisatsioone ei võimalda RSA võtmetele piiranguid konfigureerida.

WebAuthn ökosüsteemi autenturite puudus on nendega seotud käideldavusprobleem: kui autentur või selles olev võtmematerjal peaks hävima, võib see tekitada autenturi omanikule probleeme infosüsteemidesse ligipääsemisel. Kui näiteks mõne asutuse sisekasutuse kontekstis ei ole nii suur mure, sest IT-abi saab kasutajale pärast isiku tuvastamist uue autentimisvõtme registreerida, siis avalike teenuste puhul võib isiku sidumine uue autenturiga olla keeruline – erinevalt näiteks ID-kaardist, mille puhul on olemas sissetöötatud protseduurid uue kaardi väljastamiseks ja isikustamiseks. Seda tuleb WebAuthn rakendamisel kindlasti silmas pidada.

Igal juhul tuleb WebAuthn protokollistikku lugeda selle aruande kirjutamise ajal veel pigem eksperimentaalseks – seda just keerukuse, mahuka dokumentatsiooni ja väheste realisatsioonide tõttu. See on ka põhjus, miks me ei anna täpseid soovitusi krüptoalgoritmide osas – WebAuthn juurutamiseks on igal juhul vaja kaasata spetsialist, kes hoolitseb terviklahenduse eest infosüsteemi enese, selle kasutajate ning potentsiaalselt pruugitavate autenturite koosmõju arvestades.

[1] https://www.iana.org/assignments/cose/cose.xhtml#algorithms

5.5 SAML

SAML (*Security Assertion Markup Language*)[1] on XML-põhine turvaväidete – ennekõike autentimis- ja volitusinfo – edastamise standard, mis määrab asjakohased vormingud ja protokollid.

SAMList on olemas kaks generatsiooni: SAML 1 (versioonid 1.0 ja 1.1) ning SAML 2.0. SAMLi versioone 1.0 ega 1.1 ei tohiks mujal kui pärandsüsteemides kasutada, sest nende turvatase on sisuliselt teadmata. Ka OWASP SAML *Security Cheat Sheet*[2] annab soovitusi vaid SAML 2.0 põhjal.

Nimetatud OWASP soovitus peab olema iga SAMLi rakendaja regulaarne lugemisvara, lisaks tuleb pidavalt jälgida kasutatava tarkvara turvauuenduste infot. Ainuüksi 2020. aastal registreeriti vähemalt 27 CVE-baasi kirjet, mis puudutasid kuidagi SAMLi, s.h SAML dokumendi signatuurivalideerimise viga laialt kasutatud raamistikus *Spring Security*.[3]

SAMLi nõrkuste hulka tuleb kahtlemata lugeda tema baseerumine XMLil: XML korrektne valideerimine on väga keeruline[4] ning selle teostamisel on väga lihtne teha vigu [50].

Viimane ametlik täiendus SAML 2.0 tuumikstandardile pärineb aastast 2012, kuigi ka hiljem on avaldatud mõningaid mustandeid ja täpsustusi. SAML 2.1 koostamise tööd on peatunud aastal 2013. Võimalusel tuleks SAMLile eelistada aktiivsemalt arendatud protokollistikke, näiteks OpenID Connect, kuivõrd kogu asjakohane tehnoloogiline keskkond on viimastest uuendustest möödunud aja jooksul radikaalselt muutunud ja see toob iseenesest kaasa riske.

[1] https://wiki.oasis-open.org/security/FrontPage
[2] https://cheatsheetseries.owasp.org/cheatsheets/SAML_Security_Cheat_Sheet.html
[3] https://tanzu.vmware.com/security/cve-2020-5407
[4] https://cheatsheetseries.owasp.org/cheatsheets/XML_Security_Cheat_Sheet.html

6 Teegid ja rakendused

Kui rakenduses on tarvis kasutada mingeid krüptograafilisi primitiive, struktuure või protokolle, siis on alati vajalik valida rakenduse keskkonda sobiv kõrge kvaliteediga krüptoteek – keerulisematel juhtudel ka sobitada omavahel krüptoteegi ning arendus- ja käidukeskkondade valikut, sest keskkonnad ei ole krüptograafiliste teekide kättesaadavuses osas võrdsed (vt jaotis 6.2). Paljudel juhtudel ei ole valitud keskkonna jaoks piisavalt universaalseid krüptoteeke üldse olemas ning sel juhul tuleb valida mõni teek, millel on olemas sobiv mähkur (tavaliselt on selleks teegiks OpenSSL).

Krüptograafiliste primitiivide või konstruktsioonide teostamine mingit äriprotsessi toetava rakenduse arenduse käigus ei ole üldiselt hea plaan, kuivõrd selline arendus kipub keskenduma vaid ärirakenduse funktsiooniga seotud vajadustele, mitte aga turvalise krüptoteegi puhul isegi olulisemate funktsioonivälistе nõuete täitmisele (selline nõue võib olla näiteks kõrvalkanalikindlus).

Kindlasti tuleb arendusprojekti jaoks krüptoteegi valikul ja kasutamisel teha järgnevat.

- Tutvuda enne kasutuselevõttu teegi versioonidega ning võimalike paralleelselt eksisteerivate peaversioonidega (*major version*). Käesolev aruanne ei soovita meelega ühtegi kindlat tarkvaraversiooni, sest olukord muutub väga kiiresti ning põhjapanevaid teadustöid sel teemal ei ole.
- Hinnata, kui suur on teegi üldine populaarsus (*Google Trends* on selleks väga hea vahend), vähepopulaarsetel teekidel on kalduvus muutuda jäätvaraks.
- Hinnata, millise kiirusega on arendusmeeskond seni teegile turvaparandusi väljastanud.
- Hinnata, kui suur meeskond teegi arendamisega tegeleb ning kas võib eeldada meeskonna jätkusuutlikkust arendatava projekti eluea piires. Paratamatult on siinkohal eelistatud teegid, millel on mõne suurkorporatsiooni, valitsuse või suurema MTÜ tugi.
- Kasutada alati teegi viimast versiooni ning hoolitseda, et oleks olemas protseduurid kasutatud teekide korraliseks ülevaatuseks ja uuendamiseks. Tähelepanu tuleb pöörata ka sellele, et süsteemne pakihaldus (näiteks APT, RPM) ei rikuks ära arendaja plaane ega paigaldaks toodangukeskkonda dünaamiliselt lingituna vanemat ning vigast versiooni krüptoteegist.
- Pidada silmas, et teegid sisaldavad ühilduvuse huvides ka mitmesuguste selliste primitiivide ja protokollide teostusi, mida ei peaks üheski uues ega uuendatavas süsteemis enam kasutama. Mingi konstruktsiooni olemasolu teegis ei ole põhjus selle kasutamiseks, iga kasutamine tuleb otsustada pädevatele ja värsketele turvahinnangutele tuginedes.

6.1 Ülevaade levinumatest krüptoteekidest

Kokkuvõtlik ülevaade levinumatest krüptoteekidest on tabelis 5.

Tabel 5. Krüptoteekide ülevaade

Nimi	Keel(ed)	Soovitatud
OpenSSL	C	Jah
LibreSSL	C	Jah
BoringSSL	C	Ei
GnuTLS	C	Ei
BouncyCastle	Java, C#	Jah
Tink	Java, C++, Objective-C, Go, Python	Jah
libsodium	C	Jah

6.1.1 OpenSSL

OpenSSL[1] on kasutuselolevatest laiatarbe krüptoteekidest pikima ajalooga. Alustatud 1998. aastal selleks hetkeks juba kolm aastat vana teegi SSLeay baasil, on ta tänaseks 22 aastat vana. Kahjuks on see iga nähtav ka programmikoodi kvaliteedis, näiteks kurikuulsa Heartbleedi[2] nõrkuse näol. Õnneks on OpenSSL-i kvaliteet viimastel aastatel oluliselt paranenud, vähemalt CVE statistika alusel,[3] kuid ei ole päris selge, kas see tuleneb ründajate huvi vähenemisest või ongi koodikvaliteet sedavõrd palju paremaks muutunud.

OpenSSL teostab suure hulga krüptoprimitiive (räsialgoritmid, sümmeetrilised ja asümmeetrilised krüptoalgoritmid), vorminguid (näiteks CMS) ja protokolle (TLS, DTLS).

6.1.2 LibreSSL

LibreSSL[4] on OpenSSL-i kloon, mis tehti 2014. aastal pärast Heartbleedi nõrkuse avastamist, põhjuseks rahulolematus OpenSSL-i lähtekoodi kvaliteediga. LibreSSL-i arendajad tegid mitmeid olulisi turvaparandusi, millest osa viidi ka OpenSSL-i koodibaasi tagasi, kuid tänaseks on LibreSSL toetatud funktsioonide osas OpenSSL-ist maha jäänud[5] ja see tekitab tema kasutuselevõtuga probleeme – näiteks TLS 1.3 täielik tugi tekkis alles käesoleva aruande kirjutamise ajal, 2020. aasta oktoobris (OpenSSL-ist kaks aastat hiljem). Enne LibreSSL-i kasutuselevõttu peaks hoolikalt kaaluma, kas pakutavad funktsioonid on piisavad ning seda, kas aeglasem arendustsükkel on probleemiks või mitte.

6.1.3 BoringSSL

BoringSSL on Google'i sisemiseks tarbeks loodud OpenSSL-i derivaat. Selle lähtekood on küll avalik, aga selle kasutamise kohta Google'i-välistes projektides ütleb Google ise, et see ei ole soovitatav.[6]

[1] https://www.openssl.org/
[2] https://heartbleed.com/
[3] https://www.cvedetails.com/product/383/Openssl-Openssl.html
[4] https://www.libressl.org/
[5] https://lwn.net/Articles/841664/
[6] https://boringssl.googlesource.com/boringssl/

6.1.4 GnuTLS

GnuTLS[1] on vabavaraline TLSi ja DTLSi teostus, kuid oluliselt kitsama katvusega kui OpenSSL. Tema miinuseks võib pidada suhteliselt vähest populaarsust võrreldes OpenSSL-iga. Aegade jooksul on GnuTLSi nimega seotud märksa vähem CVE-kirjeidsid kui OpenSSL-iga,[2] kuid ei saa olla kindel, et see oleks märk kõrgemast kvaliteedist, sest puuduvad viited, et keegi oleks GnuTLSi koodi süstemaatiliselt auditeerinud (GnuTLS vs OpenSSL CVE-de arvu suhe on ligikaudu 1:4, samas kui lähtekoodi ridade arvu suhe on ligikaudu 1:2).

6.1.5 Bouncy Castle

Bouncy Castle[3] on vabavaraline Java ja C# krüptoteek. Mõlemas keeles või õigemini mõlemal platvormil (JVM ja .NET) on tegu *de facto* põhiliste krüptoteekidega, kui süsteemsed teegid välja arvata.

6.1.6 Tink

Tink[4] on kõige uuem teek siin nimekirjas (versioon 1.0.0 avaldati septembris 2017) ning ka kõige tähelepanuväärsem, arvestades, et ta on arendatud tänaseks teadaolevaid häid tavasid järgides ning ühtlasti teostatud mitme populaarsema keele jaoks (Java, C++, Objective-C, Go ja Python, eelvaate kvaliteediga on olemas ka JavaScript/TypeScript tugi).

Tink teostab suure hulga krüptoprimitiive ning võtmehalduse mehhanisme, kuid tema käsitlusalasse ei kuulu kõrgema taseme protokollid ega vormingud.

6.1.7 libsodium

Libsodium[5] on edasiarendus Daniel J. Bernsteini krüptoteegist NaCl, mis ise on tänaseks jäätvaraks muutunud.

Libsodium pakub põhilisi madala taseme krüptoprimitiive ning tema projekteerimiseesmärkideks on turvalisus ja lihtne kasutatavus. Libsodium on ristkompileeritav paljudele platvormidele, sh JavaScript ja WebAssembly, ning mähitav levinud programmeerimiskeelte jaoks.[6]

Oluline on tähele panna, et libsodium pakub vaid madalama taseme primitiive, millest keerulisemate konstruktsioonide loomine nõuab vastavat pädevust.

6.2 Arenduskeskkonnad

6.2.1 Java

Krüptograafiliste meetodite kasutamine Java (JVM) platvormil toetub *Java Cryptography Architecture* (JCA) standardile, mis on Java platvormi normatiivne osa. JCA sätestab krüptoprimitiivide kasutamise liidesed ning sisaldab ka osa liideste teostusi.

[1] https://www.gnutls.org/
[2] https://www.cvedetails.com/product/4433/GNU-Gnutls.html
[3] https://www.bouncycastle.org/
[4] https://github.com/google/tink
[5] https://doc.libsodium.org/
[6] https://doc.libsodium.org/bindings_for_other_languages

Kuni Java versioonini 1.3 olid krüpteerimise ja võtmekehtestusega tegelevad liidesed defineeritud eraldi *Java Cryptography Extension* (JCE) nime all, kuid hilisemates versioonides seda lahusust enam ei ole. JCE nimi on endiselt kasutuses, kuid tihtipeale lihtsalt JCA sünonüümina.

JCA defineerib oluliselt suurema hulga primitiive kui ta ise teostab ning seetõttu on tihtipeale vaja kasutada kolmanda osapoole pistikteeke (*provider*), näiteks Bouncy Castle (vt jaotis 6.1.5). Iga JCA versiooni poolt defineeritud primitiivide loetelu on võimalik leida JCA vastava versiooni dokumentatsiooni hulgast (*Java Cryptography Architecture Standard Algorithm Name Documentation*, *Java Security Standard Algorithm Names*).

JCA juurde käib eraldi arhitektuurse moodulina *Java Secure Socket Extension* (JSSE), mis sätestab TLSi kasutamise liidesed Java platvormil. JSSE projekteerimisprintsiibid on samad kui JCA-l.

JCA ning JSSE dokumentatsioon on põhiline alguspunkt arendajale, kellel on vaja Java (JVM) rakendustes krüptograafilisi primitiive või turvalisi sideprotokolle kasutada.

Tuleb hoolitseda, et nii Java käigukeskkond (*runtime environment*) kui pistikteegid oleks alati uuendatud ja turvanõrkused eemaldatud.

Selle aruande kirjutamise ajal on aktuaalsed Java LTS versioonid 8 (toetatud veel vaid osaliselt, kuid siiski kõige populaarsem[1]) ja 11 (ainus täiesti toetatud LTS versioon).

6.2.2 Go

Go standardteek sisaldab paketti `crypto` koos selle alampakettidega, milles on realiseeritud valik krüptoprimitiive- ja protokolle. Lisaks on täiendav komplekt primitiive ja protokolle realiseeritud eraldi moodulis, mis asub nimeruumis `golang.org/x/crypto`.

Google enda poolt on Go jaoks lisaks arendatud krüptoteek nimega Tink (vt jaotis 6.1.6).

Ühtegi OpenSSL avalikku mähkurit, mis töötaks praeguste (vaadeldud) Go versioonidega ning OpenSSL viimase versiooniga 1.1, ei ole teada. Samas on C teekide liidestamine Go programmidega lihtne (see on keele Go osa) ja seega pole valmisteegi puudumine probleem.

Aruande kirjutamise ajal vaatlesime vaid Go versioone 1.14 ja 1.15.

6.2.3 Node.js

Node.js on JavaScripti rakenduste käivituskeskkond kasutamiseks väljaspool veebibrausereid.

Node.js standardteek sisaldab moodulit `crypto`, mis realiseerib arbitraarse valiku krüptoprimitiive, ning moodulit `tls`, mis teostab TLS versioonid 1.2 ja 1.3. Mõlemad moodulid on tegelikult mähkurid Node.js käivituskeskkonnaga vaikimisi koos paigaldatava OpenSSL-i ümber.

Algoritmide jaoks, mida standardmoodulid ei paku, leiduvad pakihaldussüsteemi *npm* pakettidena levitatavad moodulid.

Node.js pakub ka N-API nimelist laiendusliidest C ja C++ teekide liidestamiseks, kuid selle kasutamine on keeruline. Kuivõrd OpenSSL on juba vaikimisi liidestatud, ei ole selle järele krüptograafia kontekstis ka suurt vajadust.

Aruande kirjutamise ajal vaatlesime ainult Node.js versiooni 15 ja selle ühilduvust.

[1] `https://www.jrebel.com/blog/2020-java-technology-report`

6.2.4 Rust

Rusti standardteek ei sisalda krüptograafilisi algoritme ning nende teostamine on jäetud välistele teekidele Rusti pakihaldussüsteemis *Cargo*. Eraldi võib ära märkida OpenSSL mähkuri.[1]

Standardteegi ja sellega koordineeritud arendusprotsessi puudumise tulemusena on krüptalgoritmide teostusi palju[2] ning nende kvaliteet ei ole ühtlane [74].

Arendusjärgus on ka Tinki (vt jaotis 6.1.6) mitteametlik Rusti versioon,[3] kuid see on veel väga värske – projekt algas septembris 2020.

Kuigi Rusti kui keele projekteerimiseesmärk on turvalisus, väljendub see ennekõike mäluturvalisuses. Rusti kasutamist projektides, kus krüptograafiliste algoritmide kasutamine moodustab olulise osa funktsioonist, tuleks hoolikalt kaaluda.

Aruande kirjutamise ajal vaatlesime ainult Rusti versiooni 1.48.0 ja selle ühilduvust.

6.2.5 PHP

PHPl on mõned krüptograafilised laiendused,[4] millest kõige märkimisväärsem on OpenSSL mähkur, mis võimaldab PHPst kasutada kõiki OpenSSL funktsioone.

Juhul, kui on vaja PHP rakenduses krüptograafilisi funktsioone kasutada, soovitame pöörduda otse OpenSSL mähkuri poole.

Aruande kirjutamise ajal vaatlesime ainult PHP versioone 7 ja 8 ning nende ühilduvust.

6.2.6 .NET

.NET platvormil (mis praktikas tähendab enamaltjaolt C#-keelt) on süsteemne nimeruum `System.Security.Cryptography`, milles sisalduvad põhiliste krüptograafiliste primitiivide teostused. Kui vaja on midagi enamat, tasub ennekõike uurida, kas see on toetatud vabavaralises BouncyCastle teegis (vt jaotis 6.1.5).

Aruande kirjutamise ajal vaatlesime ainult .NET versiooni 5.0 ja .NET Framework versiooni 4.8.

6.2.7 Python

Pythoni põhiline krüptoteek on eraldi paigaldatav `cryptography`, mis pakub krüptoprimitiivide teostusi. TLSi kasutamiseks on vaja paigaldada OpenSSL mähkur `pyOpenSSL` ning selle autorid soovitavad, et kõigi muude kasutusjuhtude puhul tuleks pöörduda `cryptography` poole.

Soovitus kehtib nii Python 2.7 kui Python 3 perekonna värskemate versioonide kohta (aruande kirjutamise ajal – 3.6 kuni 3.9).

[1] https://crates.io/crates/openssl
[2] https://blog.logrocket.com/rust-cryptography-libraries-a-comprehensive-list/
[3] https://github.com/project-oak/tink-rust/
[4] https://www.php.net/manual/en/refs.crypto.php

Summary in English

This is the seventh report in the series of reports on cryptographic algorithms and their life cycle. The report covers five topics. Its intended audience are managers and practitioners both in IT and in software development. As usual, the report describes several algorithms, protocols, formats and their secure usage. This time it also looks closer at some of the protocols and the development tools for applications that use cryptography. The most important point is still that for practical evaluation of use of cryptography in any single context, one should hire a security engineer.

The second chapter focusses on cryptographic primitives and contains current recommendations on key length and other relevant parameters that need consideration to keep systems secure. It also contains an overview of the main types of cryptographic primitives and relevant algorithms that one could use. The last part of that chapter is an overview of the state of standardization of post-quantum cryptographic algorithms.

The third chapter describes several families of widely used communication protocols. Where relevant, it gives recommendations about using them securely. Section on TLS also describes other measures that are relevant when protecting one's website or other services with TLS. Other protocols covered are SSH, IPsec and WireGuard.

The fourh chapter describes several container formats used to cryptographically secure one-time messages between parties or documents at rest. Covered formats include CMS, ASiC-E and its variants, JOSE formats, PGP and Estonian local format CDOC.

The fifth chapter, on authentication and authorization protocols describes OAuth 2.0 and several potential problems that might occur while implementing it. It also mentions newer specifications such as PKCE and profile for native application usage. Upcoming standards such as OAuth 2.1 and profile for browser-based applications are also covered. The chapter goes on to describe OpenID Connect and UMA as well as refer to some other profiles and extensions to OAuth/OpenID Connect framework. The chapter also describes specifications produced by FIDO Alliance, such as FIDO U2F, FIDO UAF and WebAuthn.

The last chapter describes tools to implement cryptographic measures in software. It begins with an overview of popular libraries that implement cryptographic primitives or protocols, such as OpenSSL. The rest of the chapter describes support for cryptography in popular development environments and programming languages, such as Java, Go, Rust and PHP.

Kirjandus

[1] *Agreed Cryptographic Mechanisms*. SOG-IS Crypto Evaluation Scheme, https://www.sogis.eu/documents/cc/crypto/SOGIS-Agreed-Cryptographic-Mechanisms-1.2.pdf. Jaanuar 2020.

[2] Ross J. Anderson ja Roger M. Needham. “Robustness Principles for Public Key Protocols”. Teoses: *Advances in Cryptology - CRYPTO ’95, 15th Annual International Cryptology Conference, Santa Barbara, California, USA, August 27-31, 1995, Proceedings*. Toim. Don Coppersmith. Köide 963. Lecture Notes in Computer Science. Springer, 1995, lk. 236–247.

[3] Dirk Balfanz ja Alexei Czeskis et al. *Web Authentication: An API for accessing Public Key Credentials*. Level 1 https://www.w3.org/TR/webauthn-1/. Märts 2019.

[4] Dirk Balfanz ja Alexei Czeskis et al. *Web Authentication: An API for accessing Public Key Credentials*. Level 2 Candidate Snapshot https://www.w3.org/TR/webauthn-2/. Detsember 2020.

[5] Elaine Barker. *Recommendation for Key Management. Part 1: General*. NIST Special Publication 800-57 Part 1, Revision 5, https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt1r5.pdf. Mai 2020.

[6] Elaine Barker, Quynh Dang ja Sheila Frankel. *Guide to IPsec VPNs*. NIST Special Publication 800-77, https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-77r1.pdf. Juuni 2020.

[7] Richard Barnes. *Use Cases and Requirements for JSON Object Signing and Encryption (JOSE)*. RFC 7165. Aprill 2014. URL: https://rfc-editor.org/rfc/rfc7165.txt.

[8] *BDOC – digiallkirja vorming*. https://www.id.ee/wp-content/uploads/2020/01/bdoc-spec212-est.pdf. Versioon 2.1.2. Cybernetica report no. A-101-10 (in Estonian). 2014.

[9] Mike Belshe, Roberto Peon ja Martin Thomson. *Hypertext Transfer Protocol Version 2 (HTTP/2)*. RFC 7540. Mai 2015. URL: https://rfc-editor.org/rfc/rfc7540.txt.

[10] David Benjamin. *Using TLS 1.3 with HTTP/2*. RFC 8740. Veebruar 2020. URL: https://rfc-editor.org/rfc/rfc8740.txt.

[11] Daniel J. Bernstein. *ChaCha, a variant of Salsa20*. https://cr.yp.to/chacha/chacha-20080128.pdf. Jaanuar 2008.

[12] Daniel J. Bernstein. *The Salsa20 family of stream ciphers*. https://cr.yp.to/snuffle/salsafamily-20071225.pdf. Detsember 2007.

[13] Alex Biryukov *et al. The memory-hard Argon2 password hash and proof-of-work function*. Internet-Draft draft-irtf-cfrg-argon2-12. Work in Progress. Internet Engineering Task Force, september 2020. 21 lk. kokku. URL: https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-argon2-12.

[14] Sharon Boeyen *et al. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. RFC 5280. Mai 2008. URL: https://rfc-editor.org/rfc/rfc5280.txt.

[15] Dan Boneh, Henry Corrigan-Gibbs ja Stuart Schechter. *Balloon Hashing: A Memory-Hard Function Providing Provable Protection Against Sequential Attacks*. Cryptology ePrint Archive, Report 2016/027. https://eprint.iacr.org/2016/027. 2016.

[16] Colin Boyd, Anish Mathuria ja Douglas Stebila. *Protocols for Authentication and Key Establishment, Second Edition*. Information Security and Cryptography. Springer, 2020. ISBN: 978-3-662-58145-2.

[17] Christiaan Brand *et al. Client to Authenticator Protocol (CTAP)*. Jaanuar 2019. URL: https://fidoalliance.org/specs/fido-v2.0-ps-20190130/fido-client-to-authenticator-protocol-v2.0-ps-20190130.html.

[18] Ahto Buldas *et al.* "Server-Supported RSA Signatures for Mobile Devices". Teoses: *Computer Security - ESORICS 2017 - 22nd European Symposium on Research in Computer Security, Oslo, Norway, September 11-15, 2017, Proceedings, Part I*. Toim. Simon N. Foley, Dieter Gollmann ja Einar Snekkenes. Lecture Notes in Computer Science. https://home.cyber.ee/~peeter/research/esorics2017.pdf. 2017.

[19] Brian Campbell, Chuck Mortimore ja Michael Jones. *Security Assertion Markup Language (SAML) 2.0 Profile for OAuth 2.0 Client Authentication and Authorization Grants*. RFC 7522. Mai 2015. URL: https://rfc-editor.org/rfc/rfc7522.txt.

[20] Brian Campbell ja Hannes Tschofenig. *An IETF URN Sub-Namespace for OAuth*. RFC 6755. Oktoober 2012. URL: https://rfc-editor.org/rfc/rfc6755.txt.

[21] *CMS Advanced Electronic Signatures (CAdES)*. Versioon 2.2.1. Electronic Signatures and Infrastructures (ESI) https://www.etsi.org/deliver/etsi_ts/101700_101799/101733/02.02.01_60/ts_101733v020201p.pdf. 2013.

[22] *Cryptographic algorithms lifecycle report 2016)*. https://www.ria.ee/public/RIA/Cryptographic_Algorithms_Lifecycle_Report_2016.pdf. Cybernetica report no. A-101-3. 2016.

[23] *Cryptographic Mechanisms: Recommendations and Key Lengths*. BSI – Technical Guideline TR-02102-1, https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/TechGuidelines/TG02102/BSI-TR-02102-1.pdf. Märts 2020.

[24] *Cryptographic Mechanisms: Recommendations and Key Lengths – Use of Internet Protocol Security (IPsec) and Internet Key Exchange (IKEv2)*. BSI – Technical Guideline TR-02102-3, https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/TechGuidelines/TG02102/BSI-TR-02102-3.pdf. Jaanuar 2020.

[25] *Cryptographic Mechanisms: Recommendations and Key Lengths – Use of Secure Shell (SSH)*. BSI – Technical Guideline TR-02102-4, https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/TechGuidelines/TG02102/BSI-TR-02102-4.pdf. Jaanuar 2020.

[26] Don Davis. "Defective Sign & Encrypt in S/MIME, PKCS#7, MOSS, PEM, PGP, and XML". Teoses: *Proceedings of the General Track: 2001 USENIX Annual Technical Conference, June 25-30, 2001, Boston, Massachusetts, USA*. Toim. Yoonho Park. USENIX, 2001, lk. 65–78. URL: http://www.usenix.org/publications/library/proceedings/usenix01/davis.html.

[27] William Denniss ja John Bradley. *OAuth 2.0 for Native Apps*. RFC 8252. Oktoober 2017. URL: https://rfc-editor.org/rfc/rfc8252.txt.

[28] *Digital Signature Standard (DSS)*. FIPS PUB 186-4, https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf. Juuli 2013.

[29] *Digital Signature Standard (DSS)*. FIPS PUB 186-5 (Draft), https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-5-draft.pdf. Oktoober 2019.

[30] Benjamin Dowling *et al.* "A cryptographic analysis of the TLS 1.3 handshake protocol". *Journal of Cryptology* (2021). To appear. Cryptology ePrint Archive, Report 2020/1044, https://eprint.iacr.org/2020/1044.

[31] Viktor Dukhovni ja Wes Hardaker. *SMTP Security via Opportunistic DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS)*. RFC 7672. Oktoober 2015. URL: https://rfc-editor.org/rfc/rfc7672.txt.

[32] *Electronic Signatures and Infrastructures (ESI); PAdES digital signatures; Part 1: Building blocks and PAdES baseline signatures*. Versioon 1.1.1. Electronic Signatures and Infrastructures (ESI) https://www.etsi.org/deliver/etsi_en/319100_319199/31914201/01.01.01_60/en_31914201v010101p.pdf. Aprill 2016.

[33] *Eliminating Obsolete Transport Layer Security (TLS) Protocol Configurations*. Jaanuar 2021. URL: https://media.defense.gov/2021/Jan/05/2002560140/-1/-1/0/ELIMINATING_OBSOLETE_TLS_UO0197443-20.PDF.

[34] Chris Evans, Chris Palmer ja Ryan Sleevi. *Public Key Pinning Extension for HTTP*. RFC 7469. Aprill 2015. URL: https://rfc-editor.org/rfc/rfc7469.txt.

[35] Tony Finch, Matthew A. Miller ja Peter Saint-Andre. *Using DNS-Based Authentication of Named Entities (DANE) TLSA Records with SRV Records*. RFC 7673. Oktoober 2015. URL: https://rfc-editor.org/rfc/rfc7673.txt.

[36] Hal Finney *et al. OpenPGP Message Format*. RFC 4880. November 2007. URL: https://rfc-editor.org/rfc/rfc4880.txt.

[37] Taher El Gamal. "A public key cryptosystem and a signature scheme based on discrete logarithms". *IEEE Trans. Inf. Theory* 31.4 (1985), lk. 469–472.

[38] Daniel Kahn Gillmor. *Negotiated Finite Field Diffie-Hellman Ephemeral Parameters for Transport Layer Security (TLS)*. RFC 7919. August 2016. URL: https://rfc-editor.org/rfc/rfc7919.txt.

[39] Yoel Gluck, Neal Harris ja Angelo Prado. “BREACH: reviving the CRIME attack”. *Unpublished manuscript* (2013). URL: http://breachattack.com/resources/BREACH%20-%20SSL,%20gone%20in%2030%20seconds.pdf.

[40] Paul A. Grassi, James L. Fenton ja Elaine M. Newton et al. *Authentication and Lifecycle Management*. Digital Identity Guidelines, NIST Special Publication 800-63B, https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-63b.pdf. Juuni 2017.

[41] Ólafur Guðmundsson. *Adding Acronyms to Simplify Conversations about DNS-Based Authentication of Named Entities (DANE)*. RFC 7218. Aprill 2014. URL: https://rfc-editor.org/rfc/rfc7218.txt.

[42] Phillip Hallam-Baker, Rob Stradling ja Jacob Hoffman-Andrews. *DNS Certification Authority Authorization (CAA) Resource Record*. RFC 8659. November 2019. URL: https://rfc-editor.org/rfc/rfc8659.txt.

[43] Dick Hardt. *The OAuth 2.0 Authorization Framework*. RFC 6749. Oktoober 2012. URL: https://rfc-editor.org/rfc/rfc6749.txt.

[44] Jeff Hodges, Collin Jackson ja Adam Barth. *HTTP Strict Transport Security (HSTS)*. RFC 6797. November 2012. URL: https://rfc-editor.org/rfc/rfc6797.txt.

[45] Paul E. Hoffman ja Jakob Schlyter. *The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA*. RFC 6698. August 2012. URL: https://rfc-editor.org/rfc/rfc6698.txt.

[46] Russ Housley. *Cryptographic Message Syntax (CMS)*. RFC 5652. September 2009. URL: https://rfc-editor.org/rfc/rfc5652.txt.

[47] Russ Housley. *Update to the Cryptographic Message Syntax (CMS) for Algorithm Identifier Protection*. RFC 8933. Oktoober 2020. URL: https://rfc-editor.org/rfc/rfc8933.txt.

[48] Takeshi Imamura *et al. XML Encryption Syntax and Processing Version 1.1*. https://www.w3.org/TR/2013/REC-xmlenc-core1-20130411/. Aprill 2013.

[49] *IVXV protokollide kirjeldus*. https://www.valimised.ee/sites/default/files/uploads/eh/IVXV-protokollid.pdf. Vabariigi Valimiskomisjon IVXV-PR-1.5.0 (in Estonian). 2019.

[50] M. Jensen *et al.* “On the effectiveness of XML Schema validation for countering XML Signature Wrapping attacks”. Teoses: *1st International Workshop on Securing Services on the Cloud (IWSSC)*. 2011.

[51] Michael Jones, John Bradley ja Nat Sakimura. *JSON Web Signature (JWS)*. RFC 7515. Mai 2015. URL: https://rfc-editor.org/rfc/rfc7515.txt.

[52] Michael Jones, John Bradley ja Nat Sakimura. *JSON Web Token (JWT)*. RFC 7519. Mai 2015. URL: https://rfc-editor.org/rfc/rfc7519.txt.

[53] Michael Jones ja Dick Hardt. *The OAuth 2.0 Authorization Framework: Bearer Token Usage*. RFC 6750. Oktoober 2012. URL: https://rfc-editor.org/rfc/rfc6750.txt.

[54] Michael Jones ja Joe Hildebrand. *JSON Web Encryption (JWE)*. RFC 7516. Mai 2015. URL: https://rfc-editor.org/rfc/rfc7516.txt.

[55] Jakob Jonsson ja Burt Kaliski. *Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1*. RFC 3447. Veebruar 2003. URL: https://rfc-editor.org/rfc/rfc3447.txt.

[56] Charlie Kaufman *et al. Internet Key Exchange Protocol Version 2 (IKEv2)*. RFC 7296. Oktoober 2014. URL: https://rfc-editor.org/rfc/rfc7296.txt.

[57] Stephen Kent. *IP Authentication Header*. RFC 4302. Detsember 2005. URL: https://rfc-editor.org/rfc/rfc4302.txt.

[58] Stephen Kent. *IP Encapsulating Security Payload (ESP)*. RFC 4303. Detsember 2005. URL: https://rfc-editor.org/rfc/rfc4303.txt.

[59] 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.

[60] Hugo Krawczyk. "The Order of Encryption and Authentication for Protecting Communications (or: How Secure Is SSL?)" Teoses: *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings*. Toim. Joe Kilian. Köide 2139. Lecture Notes in Computer Science. Springer, 2001, lk. 310–331.

[61] *Krüptograafiliste algoritmide kasutusvaldkondade ja elutsükli uuring (Report on the life cycle of cryptographic algorithms)*. https://www.ria.ee/sites/default/files/content-editors/publikatsioonid/kryptoalgoritmide_elutsykli_uuring_15-07-2011.pdf. Cybernetica report no. A-60-1 (in Estonian). 2011.

[62] *Krüptograafiliste algoritmide kasutusvaldkondade ja elutsükli uuring (Report on the life cycle of cryptographic algorithms)*. https://www.ria.ee/public/PKI/kruptograafiliste_algoritmide_elutsukli_uuring_II.pdf. Cybernetica report no. A-77-5 (in Estonian). 2013.

[63] *Krüptograafiliste algoritmide kasutusvaldkondade ja elutsükli uuring (Report on the life cycle of cryptographic algorithms)*. https://www.ria.ee/public/RIA/Kruptograafiliste_algoritmide_uuring_2015.pdf. Cybernetica report no. A-101-1 (in Estonian). 2015.

[64] *Krüptograafiliste algoritmide kasutusvaldkondade ja elutsükli uuring (Report on the life cycle of cryptographic algorithms)*. https://www.ria.ee/sites/default/files/content-editors/publikatsioonid/kruptograafiliste_algoritmide_elutsukli_uuring_2017.pdf. Cybernetica report no. A-101-9 (in Estonian). 2017.

[65] Adam Langley, Mike Hamburg ja Sean Turner. *Elliptic Curves for Security*. RFC 7748. Jaanuar 2016. URL: https://rfc-editor.org/rfc/rfc7748.txt.

[66] Peeter Laud ja Meelis Roos. "Formal Analysis of the Estonian Mobile-ID Protocol". Teoses: *Identity and Privacy in the Internet Age, 14th Nordic Conference on Secure IT Systems, NordSec 2009, Oslo, Norway, 14-16 October 2009. Proceedings*. Toim. Audun Jøsang, Torleiv Maseng ja Svein J. Knapskog. Köide 5838. Lecture Notes in Computer Science. Springer, 2009, lk. 271–286. URL: https://home.cyber.ee/~peeter/research/nordsec09.pdf.

[67] Ben Laurie, Adam Langley ja Emilia Kasper. *Certificate Transparency*. RFC 6962. Juuni 2013. URL: https://rfc-editor.org/rfc/rfc6962.txt.

[68] Bingyu Li *et al.* “Certificate Transparency in the Wild: Exploring the Reliability of Monitors”. Teoses: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*. Toim. Lorenzo Cavallaro *et al.* ACM, 2019, lk. 2505–2520.

[69] Torsten Lodderstedt, Mark McGloin ja Phil Hunt. *OAuth 2.0 Threat Model and Security Considerations*. RFC 6819. Jaanuar 2013. URL: https://rfc-editor.org/rfc/rfc6819.txt.

[70] Chris M. Lonvick ja Tatu Ylonen. *The Secure Shell (SSH) Transport Layer Protocol*. RFC 4253. Jaanuar 2006. URL: https://rfc-editor.org/rfc/rfc4253.txt.

[71] Daniel Margolis *et al. SMTP MTA Strict Transport Security (MTA-STS)*. RFC 8461. September 2018. URL: https://rfc-editor.org/rfc/rfc8461.txt.

[72] Daniel Margolis *et al. SMTP TLS Reporting*. RFC 8460. September 2018. URL: https://rfc-editor.org/rfc/rfc8460.txt.

[73] Johannes Merkle ja Manfred Lochter. *Elliptic Curve Cryptography (ECC) Brainpool Standard Curves and Curve Generation*. RFC 5639. Märts 2010. URL: https://rfc-editor.org/rfc/rfc5639.txt.

[74] Kai Mindermann, Philipp Keck ja Stefan Wagner. “How Usable Are Rust Cryptography APIs?” Teoses: *2018 IEEE International Conference on Software Quality, Reliability and Security, QRS 2018, Lisbon, Portugal, July 16-20, 2018*. IEEE, 2018, lk. 143–154.

[75] Kathleen Moriarty, Burt Kaliski ja Andreas Rusch. *PKCS #5: Password-Based Cryptography Specification Version 2.1*. RFC 8018. Jaanuar 2017. URL: https://rfc-editor.org/rfc/rfc8018.txt.

[76] Jens Müller *et al.* “Re: What’s Up Johnny? - Covert Content Attacks on Email End-to-End Encryption”. Teoses: *Applied Cryptography and Network Security - 17th International Conference, ACNS 2019, Bogota, Colombia, June 5-7, 2019, Proceedings*. Toim. Robert H. Deng *et al.* Köide 11464. Lecture Notes in Computer Science. Springer, 2019, lk. 24–42.

[77] David Mazières Niels Provos. *Stronger key derivation via sequential memory-hard functions*. Juuni 1999.

[78] Yoav Nir ja Adam Langley. *ChaCha20 and Poly1305 for IETF Protocols*. RFC 7539. Mai 2015. URL: https://rfc-editor.org/rfc/rfc7539.txt.

[79] Mart Oruaas ja Jan Willemson. “Developing Requirements for the New Encryption Mechanisms in the Estonian eID Infrastructure”. Teoses: *Databases and Information Systems - 14th International Baltic Conference, DB&IS 2020, Tallinn, Estonia, June 16-19, 2020, Proceedings*. Toim. Tarmo Robal *et al.* Köide 1243. Communications in Computer and Information Science. Springer, 2020, lk. 13–20.

[80] *PAdES digital signatures; Part 1: Building blocks and PAdES baseline signatures*. Electronic Signatures and Infrastructures (ESI) https://www.etsi.org/deliver/etsi_en/319100_319199/31914201/01.01.01_60/en_31914201v010101p.pdf. 2016.

[81] *PDF Advanced Electronic Signature Profiles; Part 1: PAdES Overview - a framework document for PAdES*. Electronic Signatures and Infrastructures (ESI) https://www.etsi.org/deliver/etsi_ts/102700_102799/10277801/01.01.01_60/ts_10277801v010101p.pdf. 2016.

[82] Colin Percival. *Stronger key derivation via sequential memory-hard functions*. Jaanuar 2009.

[83] Colin Percival ja Simon Josefsson. *The scrypt Password-Based Key Derivation Function*. RFC 7914. August 2016. URL: https://rfc-editor.org/rfc/rfc7914.txt.

[84] *Postkvant-krüptograafia ülevaade (Report on the post-quantum cryptographic algorithms)*. https://www.ria.ee/sites/default/files/content-editors/publikatsioonid/postkvant-kruptograafia-ulevaade-2018.pdf. Cybernetica report no. A-101-10 (in Estonian). 2018.

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

[86] Justin Richer. *OAuth 2.0 Token Introspection*. RFC 7662. Oktoober 2015. URL: https://rfc-editor.org/rfc/rfc7662.txt.

[87] Justin Richer *et al. OAuth 2.0 Dynamic Client Registration Management Protocol*. RFC 7592. Juuli 2015. URL: https://rfc-editor.org/rfc/rfc7592.txt.

[88] Justin Richer *et al. OAuth 2.0 Dynamic Client Registration Protocol*. RFC 7591. Juuli 2015. URL: https://rfc-editor.org/rfc/rfc7591.txt.

[89] Phillip Rogaway ja Thomas Shrimpton. “Cryptographic Hash-Function Basics: Definitions, Implications, and Separations for Preimage Resistance, Second-Preimage Resistance, and Collision Resistance”. Teoses: *Fast Software Encryption, 11th International Workshop, FSE 2004, Delhi, India, February 5-7, 2004, Revised Papers*. Toim. Bimal K. Roy ja Willi Meier. Köide 3017. Lecture Notes in Computer Science. Springer, 2004, lk. 371–388.

[90] Eyal Ronen *et al.* “The 9 Lives of Bleichenbacher’s CAT: New Cache ATtacks on TLS Implementations”. Teoses: *IEEE Symposium on Security and Privacy*. San Francisco: IEEE, 20. mai 2019, lk. 966–983.

[91] N. Sakimura *et al. OpenID Connect Core 1.0 incorporating errata set 1*. https://openid.net/specs/openid-connect-core-1_0.html. November 2014.

[92] Nat Sakimura, John Bradley ja Naveen Agarwal. *Proof Key for Code Exchange by OAuth Public Clients*. RFC 7636. September 2015. URL: https://rfc-editor.org/rfc/rfc7636.txt.

[93] *Secure Hash Standard (SHS)*. FIPS PUB 180-4, https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf. August 2015.

[94] Karen Seo ja Stephen Kent. *Security Architecture for the Internet Protocol*. RFC 4301. Detsember 2005. URL: https://rfc-editor.org/rfc/rfc4301.txt.

[95] *SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*. FIPS PUB 202, https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf. August 2015.

[96] Yaron Sheffer, Ralph Holz ja Peter Saint-Andre. *Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)*. RFC 7525. Mai 2015. URL: https://rfc-editor.org/rfc/rfc7525.txt.

[97] Meltem Sönmez Turan *et al. Recommendation for Password-Based Key Derivation Part 1: Storage Applications*. NIST Special Publication 800-132, https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-132.pdf. Detsember 2010.

[98] Friedrich Wiemer ja Ralf Zimmermann. “High-speed implementation of bcrypt password search using special-purpose hardware”. Teoses: *2014 International Conference on ReConFigurable Computing and FPGAs, ReConFig14, Cancun, Mexico, December 8-10, 2014*. IEEE, 2014, lk. 1–6.