TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Aleksandra Budarina 186045IACB

Veebiserverite turvalisuse analüüs Telia Eesti AS näitel

Bakalaureusetöö

Juhendaja: Vladimir Viies
dotsent

Tallinn 2021

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Aleksandra Budarina

16.04.2021

Annotatsioon

Antud lõputöö eesmärk on veebiserverite turvalisuse analüüsida Telia Eesti AS näitel.

Lõputöö koosneb viiest peatükkidest.

Esimeses peatükis on ülevaade enim kasutatudest avatud lähtekoodiga veebiserveritest.

Teises peatükis on kirjeldatud veebiserverite turvalisus. Antud peatükis on välja toodud kõige levinud veebirünnakud, nende mõju serverile. Peatükis on välja toodud, millised meetodid kasutatakse turvalisuse kontrollimiseks.

Kolmandas peatükis on toodud ülevaade Telia Eesti AS veebiserveritest. Antud peatükis on väljatoodud, millised avatud veebiserverid kasutab Telia Eesti AS, millised rünnakud on veebiserveritele tehtud, milliste probleemidega Telia Eesti AS süsteemiadministraatorid puutusid kokku ning kuidas olid probleemide lahendused leitud.

Lõputöö praktiline osa on toodud neljandas peatükis. Antud peatükis on välja toodud metoodika, mis aitab serverite turvalisust kontrollida ja tõsta. Metoodika on toodud Telia Eesti AS näitel.

Viimases peatükis on tehtud kokkuvõte lõputööst.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 24 leheküljel, 4 peatükki, 8 joonist, 3 tabelit.

Abstract

Web server security analysis on the example of Telia Eesti AS

The aim of this thesis is to analyze the security of web servers on the example of Telia Eesti AS.

The main objectives of this thesis are:

1) Get acquainted with different open source web servers and find out the methods by which the security of the web server is ensured, compare and analyze them.

2) Examine the security possibilities of web servers and find out what future plans there are for increasing security.

The problem of the investigation is that the most common open source web servers today are not secure enough and there are more and more opportunities to attack the web server. In this regard, developers often add security measures that can increase the security of the web server.

In this work the author would like to analyze the work of web servers on the example of Telia Eesti AS, find out what security measures are in use by Telia Eesti AS and how Telia Eesti AS solves problems related to web server attacks.

The theoretical part of the dissertation is divided into three chapters. In the practical part, a methodology is presented to help control and increase the security of servers. The methodology is presented on the example of Telia Eesti AS.

The thesis is in estonian and contains 24 pages of text, 4 chapters, 8 figures, 3 tables.

Lühendite ja mõistete sõnastik

HTML	*HyperText Markup Language*, Hüperteksti märgistuskeel; keel, milles kirjutatakse veebilehti.
HTTP	*Hypertext Transfer Protocol*, Hüperteksti edastusprotokoll; protokoll andmete edastamiseks veebis.
HTTPS	*HyperText Transfer Protocol Secure*, Turvaline hüperteksti edastusprotokoll; turvaline protokoll andmete edastamiseks.
OS	*Operating system*, Operatsioonisüsteem
IIS	*Microsoft Internet Information Services*, Microsofti Interneti Infoteenused; veebiserver.
I/O	*Input/Output*, Sisend/Väljund
SQL	*Structured Query Language*, Struktuurpäringukeel; andmebaaside päringukeel.
DoS	*Denial of Service*, Teenusetõkestusrünne; rünnak veebiserveriltele.
DDoS	*Distributed denial-of-service attack*, Hajutatud teenusetõkestusrünne; rünnak veeriserveritele.
XSS	*Cross-Site Scripting*, Murdskriptimine; rünnak veebiserveritele.
F5	*F5*; teenusepakkuja
ACL	*Access Control List*, Pääsupiiramisloend; õiguste loend.
WAF	*Web application firewall*, Veebirakenduste tulemüür
SSL	*Secure Sockets Layer*, Turvasoklite kiht; andmete edastamise protokoll.
TLS	*Transport Layer Security*, Transpordikihi turbeprotokoll
SSH	*Secure Shell*, Turvakest; võrguprotokoll turvaliseks võrguteenuste opereerimiseks võrgu kaudu.
VPN	*Virtual Private Network*, Virtuaalne privaatvõrk; privaatne ja turvaline arvutivõrk.
URL	*Uniform resource locator*, Internetiaadress
IDS	*Intrusion Detection System*, Sisetungi tuvastamise süsteem
RSA	*Rivest-Shamir-Adleman*; krüpteerimise algoritm.
SMS	*Short Message Service*, Lühisõnum

IP	*Internet Protocol*, Internetiprotokoll
OWASP	*Open Web Application Security Project*, Avatud veebirakenduste turbeprojekt

Sisukord

Jooniste loetelu

Tabelite loetelu

Sissejuhatus

Tänapäeval üle 70% veebilehtedest kasutavad avatud lähtekoodiga veebiserverid oma lehtede kuvamiseks. [1] Avatud lähtekoodiga veebiserverid lihtsustavad suhtlemise lõppkasutaja ja veebilehtede failide vahel. Avatud lähtekoodiga veebiserverite kasutus kasvab iga aastaga, nende populaarsus kasvad ja veebiserverite turvalisus on tänapäeval väga oluliseks teemaks muutnud. Käesoleva töö peamised eesmärgid on:

1) Tutvuda erinevate avatud koodiga veebiserveritega ning saada teada meetodid, millega veebiserveri turvalisus on tagatud, neid võrrelda ja analüüsida.

2) Uurida veebiserverite turvalisuse võimalused ning saada teada millised on tuleviku plaanid turvalisuse tõstmise osas.

Uurimise probleem on selles, et tänapäeval enim levinud avatud koodiga veebiserverid ei ole piisavalt turvalsed ning tuleb rohkem võimalusi veebiserveri töö rünnamiseks. Seoses sellega arendajad lisavad turvameetmed, mis saavad veebiserveri turvalisust tõsta.

Antud töös soovin analüüsida veebiserverite töö Telia Eesti AS näitel. Selgitada välja millised turvameetmed on Telia Eesti AS kasutusel ning kuidas Telia Eesti AS lahendab veebiserverite rünnakutega seotud probleemid.

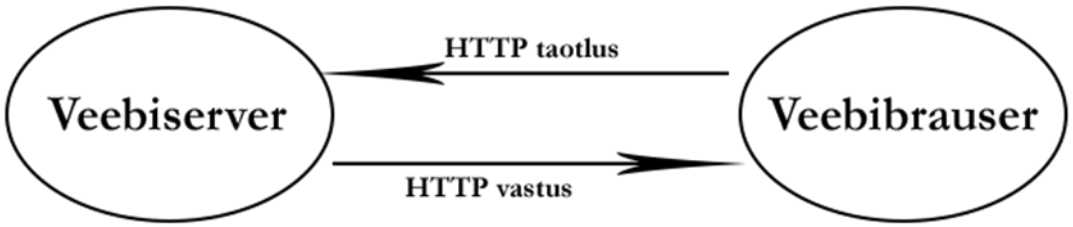
Lõputöö teoreetiline osa on jagatud kolmeks peatükiks. Praktilises osas on välja toodud metoodika, mis aitab serverite turvalisuse kontrollida ja tõsta. Metoodika on toodud Telia Eesti AS näitel.

1 Veebiserverid

Avatud lähtekoodiga tarkvara (veebiserver) on tarkvara, mille lähtekood on üldjuhul nähtav ja muutuv kasutajale. Kui veebiserveri lähtekood ei ole mingil viisil nähtav ja muutuv, siis selline tarkvara loetakse privaatseks ehk suletud lähtekoodiga tarkvara. Avatud lähtekoodiga veebiserverid on igapäevases elus kasutamisel. Avatud veebiserverid on üle 70% veebilehtedel kasutatav. [1]

Veebiserver on üldjuhul koht, kus hoitakse veebilehe failid (pildid, dokumendid, HTML-failid, JavaScript-failid, jne.). Veebiserver saadab need failid lõppkasutajale, mis on üldjuhul veebibrauser. Veebiserver samuti kontrollib, millised kasutajad saavad ligi failidele, mis on serveris olemas.

Kasutaja ehk veebibrauser saadab veebiserverile päring andmete saamiseks. Tavaliselt need andmed on pildid, failid, HTML-lehed ja muud andmed. Veebiserver võtab vastu HTTP päringud veebibrauseritelt ja annab HTTP vastused koos, näiteks, pildiga, failiga või muu andmetega.



Joonis 1 Suhtlus veebiserveri ja veebibrauseri vahel

Alljärgnevalt on välja toodud kõige enimkasutatud avatud lähtekoodiga veebiserveritest.

1.1 Apache HTTP server (Apache HTTP Server)

Kõige populaarsem veebiserver on Apache HTTP Server, mis on tuntud kui lihtsalt Apache. Apache on tulnud välja 1995. aastal ning sai kõige populaarsemaks veebiserveriks alates 1996. aastast. Viimase statistika järgi on näha, et Apache on kasutusel üle 25% kõikidest veebisaitidest, mis eksisteerivad maailmas. [1] Apache veebiserver töötab Linux, Mac OS ja Microsoft Windows platformide peal. Apache eelised on usaldusväärsus ja modulaarne arhitektuur (võimaldab lisada erinevad moodulid, millega saab serveri omadused laiendada, severi töö lihtsustada, turvalisuse tösta ja erinevate programeerimiskeelte toetada). [2]

Puudused	**Eelised**
Liiga palju konfiguratsiooni parameetrid võivad põhjustada turvaauke.	Tasuta.
Toimivuse probleemid suure liiklusega veebilehtedel.	Usaldusväärne ja stabiilne.
	Regulaarsed uuendused ja turvaaukude parandamised.
	Kergesti kohandatav.
	Multiplatvormiline.

Tabel 1 Apache serveri puudused ja eelised

1.2 Veebiserver Nginx

Teiseks populaarseks veebiserveriks on Nginx. Nginx tuli turule 2004. aastal. Nginx'i lähtekoodi kasutavad ligikaudu 23% veebisaitidest, kuid tänapäeval Nginx'x populaarsus kasvab. Viimaste kuude jooksul Netcraft andmetel Nginx'i kasutatakse 35% uutes veebisaitides. [1] Nginx'i eelised on kiirus ja usaldusväärsus. Nginx on veebiserver, millel ei ole liiga palju lisa funktsioone ning sobib rohkem staatiliste veebisaitide puhul. Nginx's üks tööprotsess saab paralleelselt teenindada mitu ühendust, mille tõttu ei teki iga päringu kohta eraldi tööprotsessi. Selline käitumine aitab serveri kiirus tõsta. Nginx'i veel üheks eeliseks on lihtsus. Nginx'i on lihtne kasutada ja muuta tema konfiguratsiooni failid. Muutmiseks ei ole vaja peatada käivitatud tõõprotsessid. Nginx on efektiivne vahend veebiserveri jaoks. [3]

	Apache	**Nginx**
Lihtsus	Lihtne arendamisel ja kasutamisel tänu sellele, et üks protsess teenindab ühte ühendust.	Raske arendamisel, sest üks protsess teenindab paralleelsest mitu ühendust.
Jõudlus staatilise lehe puhul	Näitab staatilised komponendid aeglaselt.	Töötab 2 korda kiiremini, kui Apache.
Jõudlus dünaamilise lehe puhul	Töötab hästi.	Töötab hästi.
Operatsioonisüsteemid	Toetab kõiki operatsioonisüsteemid.	Toetab kõiki operatsioonisüsteemid, kuid Windowsi peal töötab ebastabiilselt.
Turvalisus	Turvaline veebiserver.	Turvaline veebiserver.
Paindlikkus	On võimalik lisada juurde lisa moodulid.	Lisa moodulid saab lisada alatest 1.11.5 versioonist.
Dokumentatsioon	Detailne dokumentatsioon Apache serveri loomisest.	Detailne dokumentatsioon tuli välja ainult viimastel aastatel.

Tabel 2 Apache ja Nginx veebiserverite võrdlus [4]

1.3 Microsofti Interneti Infoteenused (Microsoft Internet Information Server (IIS))

IIS on kolmas kõige kasutatav veebiserveri rakendus ja moodul-laineduste kogum, mis on loodud Microsofti poolt Microsoft Windowsi kasutamiseks. IIS kasutavad umbes 13% veebisaitidest. [1] Turvalisuse süsteem on ehitatud Windows NT peale. Ligipääsu saamiseks kasutaja peab sisestama kasutajatunnust ja parooli, mis eksisteerivad Windows andmebaasides. See on mugav ettevõte sisemiste veebilehtede jaoks, kuid on mõtetu avatud veebilehtedet jaoks, kus ei ole võimalik igale kasutajale Windows kontot luua. [5]

	Apache	**IIS**
Turvalisus	Turvaline veebiserver.	Turvaline veebiserver. Peaaegu kõik turvaaugud on ainult operatsioonisüsteemi tasemel.
Jõudlus	Töötab hästi.	Töötab hästi.
Hind	Tasuta.	Saab soetada ainult koos Windows'iga.
Operatsioonisüsteemid	Toetab kõiki operatsioonisüsteemid.	Toetab ainult Windows.
Arhitektuur	Modulaarne.	Modulaarne.
Lihtsus	Lihtne arendamisel ja kasutamisel tänu sellele, et üks protsess teenindab ühte ühendust.	Lihtne administreerimis keskond.
.NET platvormiga ühendus	Nõrk ühendus .NET platvormiga.	Tugev ühendus .NET platvormiga.

Tabel 3 Apache ja IIS veebiserverite võrdlus [6]

1.4 Veebiserver OpenResty

2021. aastal OpenResty on neljas populaarne veebiserver, mida kasutavad 6% veebilehtedest. [1] OpenResty on veebiserver, mis on ehitatud Nginx serveri põhjal. OpenResty arendajad integreerisid Nginxi sisse Lua programmeerimis keelt. See aitas luua kiire, stabiilse ja paindlikku veebiserveri. OpenResty on lihtne kasutusel ja arendamisel. OpenResty on asünkroonse I/O võimeline. OpenResty kasutatakse selle kiiruse, paindlikuse, minimalistlike struktuuri ja tuntud Nginx-i eelistusi – üks protsess teenindab paralleelselt mitu ühendust – pärast. OpenResty annab võimalust lisada lihtsad käsud ja moodulid, mis ei ole võimalik teha Nginx veebiserveris. [7] [8]

1.5 Veebiserver Lighttpd

Lighttpd on viies kõige kasutatav veebiserver, mida kasutavad umbes 0.05% kõikidest veebilehtedest maailmas. [9] Lighttpd on lihtne, väike, kiire, turvaline ja paindlik veebiserver. Lighttpd veebiserver sobib sellistes keskkonnades, kus protsesside kiirus on kõige oluline aspekt. Selline eelistus teistest veebiserveritest on põhjendatud sellega, et Lighttpd veebiserver kasutab vähem mälu. Lighttpd veebiserver aitab parandada veebilehe laadimise probleemid. Lighttpd server toetab kõik operatsioonisüsteemid. [10]

2 Serverite turvalisus

Läbi aastate serverite turvalisus muutis kõige olulisemaks teemaks veebikeskondade arendamisel. Selline muutus on tulnud selle pärast, et kaasaegsed veebikeskonnad ise muutsid populaarseks ning igapäevaselt nii ettevõted, kui ka tavalised inimesed kasutavad veebi. Selleks, et veebi kasutada, eksisteerivad serverid, mille peal on veeb ehitatud. Kuna veebi kasutamise populaarsus kasvab, kasvab ka veebi rünnakute arv. Antud peatükis on välja toodud kõige populaarsemad rünnakud veebiserveritele, veebiserverite turvalisuse tagamise võimalused ning mõned võimalused turvalisuse tasemr tõstmiseks.

2.1 Rünnakud veebiserveritele

Tänapäeval eksisteerivad peamised rünnakud: Murdskriptimine, Saitidevahelise taotluse võltsimine, Teenusetõkestusrünne, Hajutatud teenusetõkestusrünne ja SQL-i süstimine.

2.1.1 Murdskriptimine (Cross-site scripting)

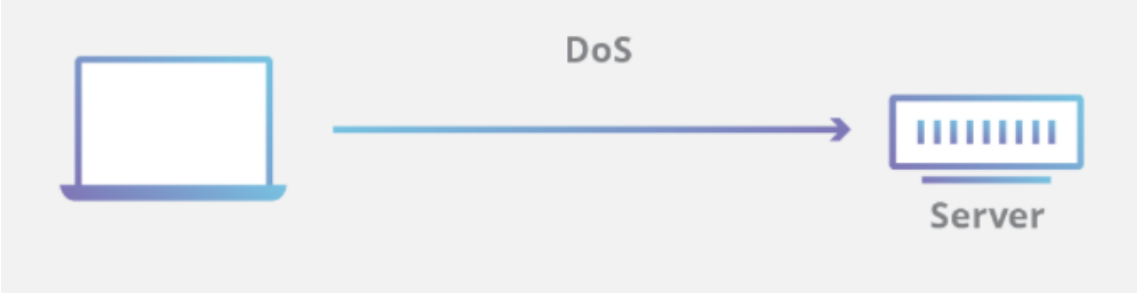
Murdskriptimine on rünnak, mille käigus pahatahtliku koodi sisestatakse konkreetsele veebilehele kasutades JavaScript-i, koodi ja ründajate kaugserveri vahel luuakse suhtlemine siis, kui kasutaja veebilehe avab. Murdskriptimise peamiseks eesmärgiks on varastada kasutaja küpsiseid, saada kätte ettevõte salastatud andmed, kasutades serverisse manustatud skripti koos vajalike andmete edasise proovivõtmisega ning kasutades neid järgnevateks rünnakuteks ja häkkimisteks. Kasutaja jaoks antud pahatahtlik kood kuvatakse saidi ühe osana. Üldiselt murdskriptimine serveri jaoks on ohutu, kuid kui ründaja saab kätte administraatori küpsised, saab ta juurdepääsu veebilehe juhtpaneelile ja selle sisule. [11]

2.1.2 Saitidevahelise taotluse võltsimine (Cross-site request forgery)

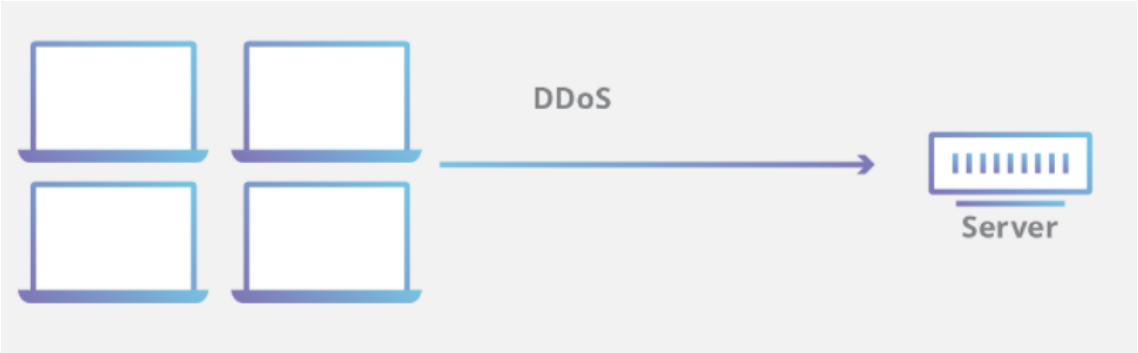
Saitidevahelise taotluse võltsimine on rünnak, mis sunnib lõppkasutajat soovimatiud toiminguid tegema veebilehel, milles ta on praegu autentitud. E-maili saatmise abil ründajad võivad sundida veebilehe kasutajad tegutsema nii, nagu ründaja ise on valinud. Sõltudes rünnatud kasutaja rollist ja õigustest, edukas saitidevahelise taotluse võltsimise rünnak annab ründajale võimaluse kas sundida kasutajat täitma olekumuutusi, näiteks rahaülekandeid, või ründaja saab täielikku kontrolli üle veebilehe ning saab kahjustada kogu veebiserveri, kui rünnatud kasutajal on administraatori õigused. [12]

2.1.3 Teenusetõkestusrünne (Denial of service, DoS) ja Hajutatud teenusetõkestusrünne (DDoS)

Teenusetõkestusrünne on rünnak, mille käigus ründaja eesmärk on muuta serveri kasutajatele kättesaamatuks, katkestades veebiserveri tavapärase toimimise ja koormata veebiserveri ressurssid. Teenusetõkestusrünne rünnakud tavaliselt üleujutavad sihtmasinat päringutega kuni tavalist liiklust ei õnnestu töödelda. Selle tulemuseks on kasutajatele teenuse keelamine. DoS rünnaku puhul ründajad kasutavad ühte internetiühendust ja ühte arvutit oma eesmärkide saavutamiseks. [13] Hajutatud teenusetõkestusrünne rünnak on sarnane teenusetõkestusrünnakuga. Hajutatud teenusetõkestusrünne puhul kasutatkase mitu arvutit ja mitu internetiühendust. Ründajad saadavad veebiserverile päringud, et serveri koormata. Kuna rünnaku puhul kasutatkase mitu seadmed, võib rünnakuliikluse eraldamine tavalisest serveri liiklusest olla keeruline ja rünnaku lahendamine võib võta rohkem aega ja jõudu. [14] Alljärgnevatel piltidel on näidatud erinevus DoS ja DDoS rünnakute vahel.



Joonis 2 DoS rünnak veebiserverile [14]



Joonis 3 DDoS rünnak veebiserverile [14]

2.1.4 SQL-i süstimine (SQL injections)

SQL-i süstimine on rünnak, mille käigus ründajad sekkuvad päringutesse veebilehe ja andmebaasi vahel. Üldiselt võimaldab see ründajal vaadata andmeid, mida nad tavaliselt ei saa. See võib hõlmata andmeid, mis kuuluvad teistele kasutajatele või muid andmeid, millele veebileht ise pääseb juurde. Paljudel juhtudel saab ründaja neid andmeid muuta või kustutada, põhjustades püsivaid muudatusi veebilehe sisus või käitumises. Mõnes olukorras võib ründaja laiendada SQL-i süstimise rünnakut, et kahjustada selle aluseks olevat veebiserverit või muud taustinfrastruktuuri või sooritada teenuse keeldumise rünnak. [15]

2.2 Veebiserverite turvalisuse tagamine

Veebiserverid oma struktuuri järgi on turvalisena loodud, kuid sellest ei piisa, kui rääkida täielikkust turvalisusest. Turvalisus sõltub erinevatest aspektidest nii veebiserveri enda poolt, kui ka operatsioonisüsteemi poolt. Operatsioonisüsteemi turvaaugud võivad tekkitada tõsised probleemid. Samuti veebiserveri enda konfiguratsioon võib põhjustada rünnakute ohtu. Kui veebiserver ei ole õigesti konfigureeritud, siis ründajatel tulevad rohkem võimalusi veebiserveri turvalisuse rikkuda.

Operatsioonisüsteemi turvaaugud on seotud kasutajate õigustega, mittevajalike teenuste kasutamisega, kaugpääsuga ja vana tarkvara versioonidega. Veebiserverite turvaaugud on seotud veebiserveri konfiguratsiooniga, kasutatud moodulitega ja serveri informatsiooni kuvamisega. Detailne analüüs veebiserveri turvalisusest on välja toodud peatükis 4.

2.3 Veebiserverite turvalisuse tõstmise võimalused

Tänapäeval veebiserverite turvalisuse tõstmine on üks olulisematest teemadest, millega igapäevaselt tegeletakse. Eksisteerivad erinevad tööriistad, mis aitavad turvelisust tõsta ja parandada.

2.3.1 Avatud veebirakenduste turbeprojekt (Open Web Application Security Project, OWASP)

Avatud veebirakenduste turbeprojekt (OWASP) on organisatsioon, mis tegeleb erinevate projektidega. Organisatsiooni eesmärgiks on avatud lähtekoodiga tarkvara turvalisuse parandada. OWASP kogub informatsioon turvaprobleemidest ja turvameetmetest ülemaailma, koostab selle informatsiooni põhjal dokumentatsiooni ja korraldab küberturvalisuse valdkonnas koolitusprogramme, et arendajad oleksid kursis turvaohtudega. OWASP pakub kasutajatele tasuta dokumentatsiooni, tööriistaid, foorumeid ja uuringuid. [16]

2.3.2 Netsparker turvalisuse kontrollimise rakendus

Netsparker on automatiseeritud rakenduste turvalisuse testimise töörist, mis võimaldab ettevõtele serveri turvalisust kontrollida, parandada, tõsta ja rünnakuohtu vähendada. Netsparker on täielikult automatiseeritud isegi keerulistes keskkonnades. Netsparker on lihtne kasutada ja rakendada oma süsteemis. Netsparker aitab automatiseerida turvaülesandeid ja sästada töötajate aega, omab kõrge haavatuse tuvastamise taset ja kontrollib juurdepääsu ja mitmekasutaja keskkonda. Tööristal puuduvad valepositiivsed tulemused. [17] [18] [19]

2.3.3 ModSecurity plugin

ModSecurity plugin on veebirakenduste tulemüür (WAF). ModSecurity eesmärgiks on kontrollida kõiki veebiserverisse saabuvaid päringud ja filtreerida need, mis vastavad turvaeeskirjadele. ModSecurity töötab koos pahatahtlike mallidega. Kui URL-päring vastab mõnele mallile, siis see päring blokeeritakse. ModSecurity toetab palju veebiserveri, selle hulgast Apache, Nginx ja IIS. ModSecurity eelistusteks on standardrünnakute blokeerimine ja kohalik seadistus, mis võimaldab kirja panna konkreetse ettevõte jaoks konkreetsed reeglid ja mallid. ModSecurity puuduseks on suure reeglistiku kompleksne haldamine. [20]

2.3.4 NAXSI plugin

NAXSI plugin on samuti veebirakenduste tulemüür, mis on spetsiaalselt loodud Nginx veebiserveri jaoks. NAXSI võimaldab kaitsta ründajate vastu, kes lisavad veebilehtedele haavatavaid skripte. NAXSI kaitseb veebiserveri lihtsa reeglistikuga, kasutades hinnangupõhist turberaamistikku. Nendele rangetele reeglitele tuginedes saab iga saadud URL-i päring hinde. Kui see hinne ületab teatud konfiguratsioonis konfigureeritud taset, blokeerib NAXSI selle päringu automaatselt. NAXSI eelistusteks on lihtne reeglistik, kiire ja lihtne administreerimine, lubatud loendisse lisamise tugi. NAXSI puuduseks on sõltuvus veebiserveri tüübist kuna plugin töötab ainult Nginx veebiserveri põhjal. [20]

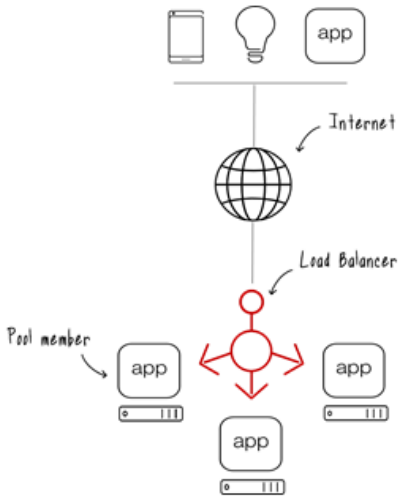
3 Telia Eesti AS veebiserverite kirjeldus

Antud peatükis on kirjeldatud Telia Eesti AS veebiserverite kasutus, rünnakute lahendamine ning tehtud rünnakud.

3.1 Kasutatavad veebiserverid Telia Eesti AS serverites ja nende turvalisuse tagamine

Peamised veebiserverid, mida kasutab Telia Eesti AS, on Apache, Nginx ja IIS. [21]

Telia Eesti AS serveritel on rakendatud koormuse tasakaalustaja (load balancer) meetod. [21] Koormuse tasakaalustaja on seade, mis toimib vastupidise puhverserverina ja jagab võrgu- või rakenduste liiklust paljude serverite vahel.Koormuse tasakaalustaja vähendab serverite koormust. Koormuse tasakaalustajad tagavad töökindluse ja kättesaadavuse, jälgides rakenduste tervislikkust ja saates päringuid ainult serveritele ja rakendustele, mis suudavad õigeaegselt vastata. [22]



Joonis 4 Koormuse tasakaalustaja [22]

Koormuse tasakaalustaja ees on F5 täiustatud veebirakenduste tulemüür (F5 Advanced Web Application Firewall) turvalisuse lahendus. [21] F5 pakub kaitset kooditaseme

haavatavuste, nagu süstimis- või XSS-rünnakute, samuti peaaegu kõigi tarkvarapakettide komponentidest leitud tarkvara haavatavuste eest. [23]

F5 turvaline lahendus võimaldab veebiserveri turvalisuse tagada ACL-de ja WAF tasemel. Juurdepääsukontrolli loend (Access Control List, ACL) on tabel, mis ütleb arvuti oeratsioonisüsteemile, millised juurdepääsuõigused on igal kasutajal konkreetsele süsteemiobjektile, näiteks failikataloogile või üksikule failile. [24] Veebirakenduste tulemüür (Web application firewall, WAF) kaitseb veebirakendusi mitmesuguste rakenduskihi rünnakute eest, nagu saididevaheline skriptimine (XSS), SQL-i süstimine ja küpsiste mürgitamine. WAF kaitseb veebirakendused, filtreerides, jälgides ja blokeerides veebirakendusse suunduva pahatahtliku HTTPS-liikluse ning takistab volitamata andmete lahkumist rakendusest. [25] Samuti veebiserverite turvalisuse tagamiseks tehakse tarkvara parandused ja turva skanneerimine.

3.2 Turva probleemide lahendamine

Veebiserverite turva probleemide lahendamiseks peamine tööriist on andmete liikluse filtreerimine. Andmete liiklus on lubatud ainult juhul, kui kasutusel on turvaline ühendus ehk on rakendatud krüpteerimise protokoll TLS. Samuti Telia Eesti AS igapäevaselt monitoorib turvanõrkusi ning vajadusel parandab neid.

3.3 Tehtud rünnakud

Selle aasta veebruaril Telia Eesti AS veebiserveritele oli tehtud kaks DDoS rünnakut ühe päeva jooksul. Selleks, et rünnaku lahendada oli rakendatud andmete liikluse filtreerimine. Samuti oli tehtud ka rünnak, mille käigus ründajad üritasid veebiserverisse sisse logida. [21]

4 Telia Eesti AS veebiserverite turvalisuse tagamine

Turvalisuse tagamiseks on vaja teha kindlad sammud – on vaja välja anda kindel metoodika. Minu poolt pakutavas metoodikas on välja toodud sammud, mida on vaja rakendada nii veebiserveri, kui ka operatsioonisüsteemi poolt. Käesoleva töö autor ei nõua, et ettevõte poolt kasutatud turvameetmed, mis toimivad efektiivselt, vahetatakse teiste vastu. Kui aga tekib vajadus uue turvameetme valimiseks, ettevõte (antud töös Telia Eesti AS) peab kasutama vastavat metoodikat oma veebiserverite turvalisuse tagamiseks.

Metoodika koostmisel on oluline küsida: millised on peamised sammud, mida kindlasti on vaja läbida, et veebiserveri turvalisuse tagada? Kuna veebiserverite turvalisus nõuab igapäevaselt arendamist, on vaja eelnimetatud küsimus püstitada iga päev ning iga päev jälgida antud metoodika nõudet.

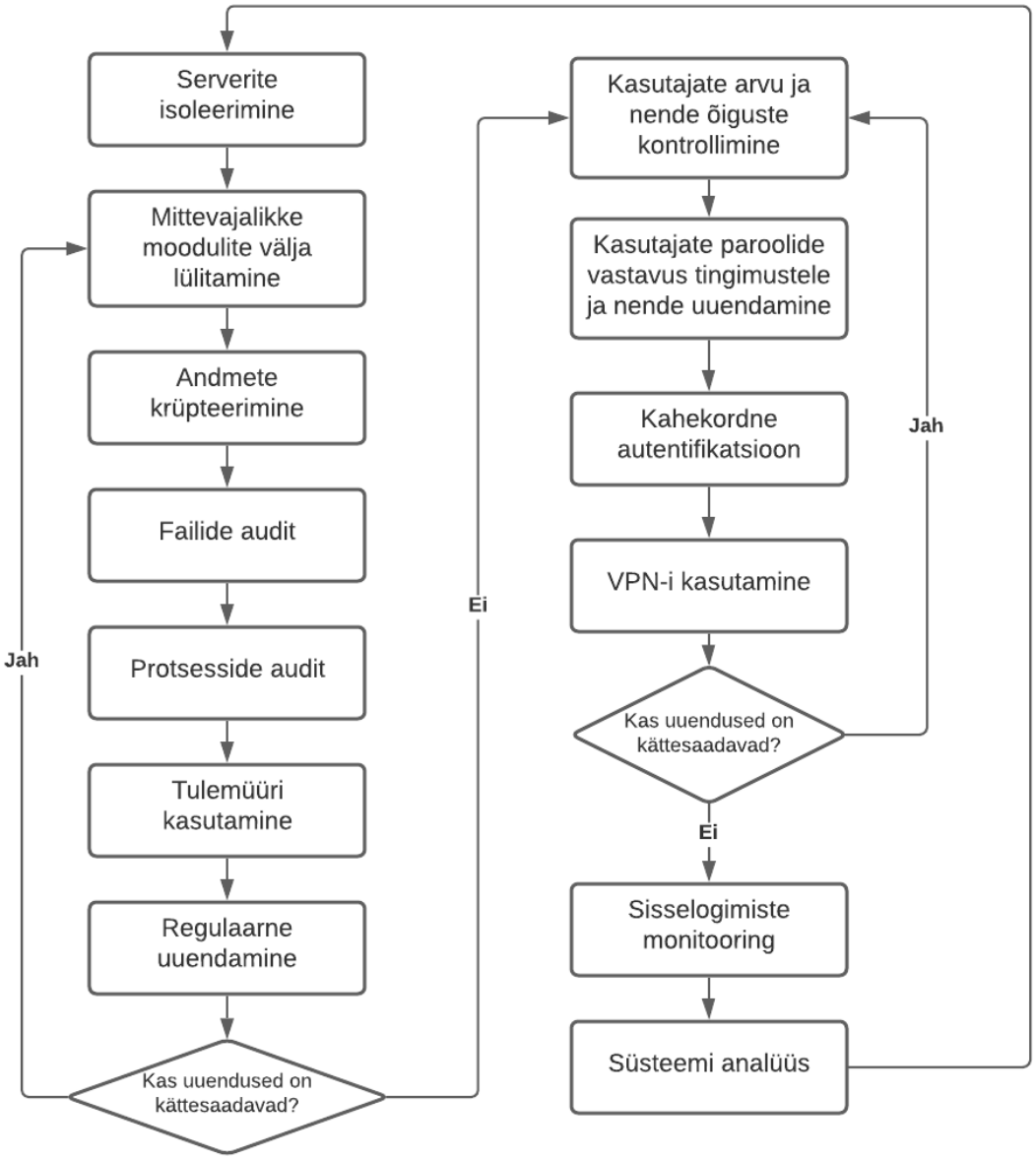
Käesolev metoodika veebiserverite turvalisuse tagamiseks koosneb kolmeteistkümnest peamisest sammudest.

1. Serverite isoleerimine
2. Mittevajalikke moodulite välja lülitamine
3. Andmete krüpteerimine
4. Failide audit
5. Protsesside audit
6. Tulemüüri kasutamine
7. Regulaarne uuendamine
8. Kasutajate arvu ja nende õiguste kontrollimine
9. Kasutajate salasõna vastavus tingimustele ja nende uuendamine
10. SSH autentimine ja kahekordne autentimine

11. VPN-i kasutamine

12. Sisselogimiste monitooring

13. Süsteemi analüüs

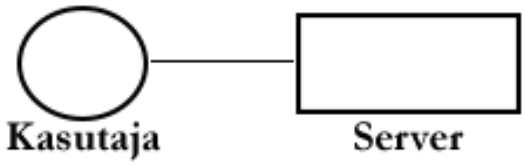


Joonis 5 Metoodika veebiserverite turvalisuse tagamiseks

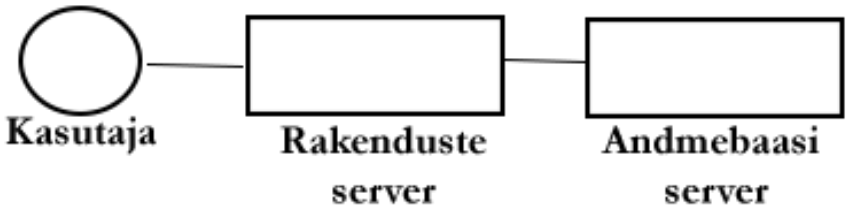
4.1 Serverite isoleerimine

Iga veebileht baseerub mingi serveri peal. Esialgu turvalisuse tagamiseks on vaja iga kasutatav server füüsiliselt isoleerida, kasutada turvasüsteemid selleks, et kontrollida, kes pääseb ruumi sisse. Ligipääs füüsilistele serveritele peab olema ühel või kahel inimestel ja nende kasutaja kontod ja salasõnad peavad olema unikaalsed.

Samuti on vaja isoleerida veebiserverid. Kui veebilehel on kasutusel nii tarkvara serverid, kui ka andmebaaside serverid, siis on vaja neid eraldada. Jagades erinevad protsessid erinevate serveri vahel tõstetakse serverite turvalisus ning tuleb võimalus isoleerida potentsiaalse turvaauku. [26]



Joonis 6 Serverid ilma isolatsiooni



Joonis 7 Serverid isolatsiooniga

4.2 Mittevajalikke moodulite välja lülitamine

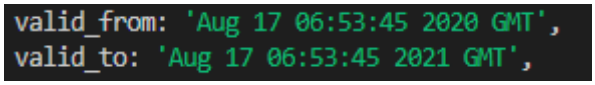
Teiseks sammuks tuleb mittevajalikke moodulid välja lülitada. See aitab suurendada serveri turvalisust vähendades ründevektorit. On vaja alles jääda ainult need teenused ja need moodulid, mis vastavad miinimumnõuetele süsteemi installimiseks ja hooldamiseks. Lubada on vaja ainult need võrgupordid, mis on kasutusel operatsioonisüsteemis ja installitud komponentides. Ennem, kui hakkatakse uue mooduli

ehk tarkvara installima, on vaja veenduda, et see moodul tõesti on vajalik ja ei tekki lisa turvaaugud süsteemis. [27]

4.3 Andmete krüpteerimine

Andmete krüpteerimine teeb informatsiooni liikumine kahe süsteemi vahel turvaliselt kasutades Secure Socket Layer (SSL) ja Transport Layer Security (TLS) turva protokolle. SSL ja TLS protokollid krüpteerivad andmed nii, et tundlikku infot (näiteks, nimed, salasõnad, pangakontod ja PIN koodid) ei varastataks ülekanne ajal. [28]

Kui veebilehel on SSL või TLS protokoll rakendatud, siis veebilehe URL-is on HTTPS märk olemas, mis näitab veebilehe turvalisus. Kas veebilehel on HTTPS olemas või mitte, on võimalik kontrollida koodiga, mis on toodud LISA 2 all.



Joonis 8 HTTPS tugi kontroll online.ee veebilehe näitel

Andmete krüpteerimise näide kasutades TLS protokolli on välja toodud LISA 3 all.

4.4 Failide audit

Failide audit on protsess, mille käigus võrreldatakse praeguse süsteemi olekut failisalvestiste ja süsteemi omadustega, kui süsteem on korras. Peamiseks riistaks auditi viimiseks on Sisetungi tuvastamise süsteem (Intrusion Detection System, IDS), mis on osa operatsioonisüsteemi tarkvarast. IDS jälgib süsteemi volitamata toimingute suhtes. Failide audit on üks väheseid viise, kuidas tagada, et ükski kasutaja või protsess pole failisüsteemi muutnud. Failide audit annab teada, millised failid on muutnud ning selle abiga on võimalik kiiresti turvaauke parandada. [28]

4.5 Protsesside audit

Protsesside audit on protsess, mis näitab severi infrastruktuuris töötavaid teenuseid. Protsesside audit annab analüüsi sellest, milliseid porte süsteem kasutab ning milliseid protokolle aktsepteeritakse. Kuna server käivitub erinevad protsessid paralleelselt, siis iga selline protsess on potentsiaalne rünnaku oht serverile. Audidi andmete järgi on võimalik arusaada, millised protsessid on mittevajalikud, millised tegevused saab välja lülitada ning kus on potentsiaalsed turvaaukud. [28]

4.6 Tulemüüri kasutamine

Tulemüür kontrollib ja piirab juurdepääsu süsteemile. Tulemüüri kasutamine on serveri turvalisuse tagamiseks hädavajalik kuna tulemüür võimaldab ainult konkreetseid olulisi ühendusi, lukustades juurdepääsu teisele teenustele. Tulemüür on serveri konfiguratsiooni lahutamatu osa. Isegi kui kasutatav tarkvara võimaldab turvalisuse tagada, tulemüür tekitab täiendav kaitsekiht. [28]

4.7 Regulaarne uuendamine

Turvalisuse tagamiseks on vaja regulaarselt uuendada kasutatud tarkvara – operatsioonisüsteemi ja andmebaaside serverid. On vaja igapäevaselt uuenduste saadavuse kontrollida. Vananenud tarkvara jääb väga kiiresti läbipaistvaks ehk turvaaukude eksisteerimine muutub paljudele kasutajatele kättesaadavaks ning iga auk on potentsiaalse rünnaku jaoks eelis.

Üheks variandiks on automaatne uuendamine. Selline lahendus aitab inimfaktorit vältida, kui süsteemiadministraator unustab tarkvara uuendada.

Teiseks variandiks on käsitsi uuendamine. Kuna automaatne uuendamine võib olla riskantne ja tuua enda kaasas süsteemsed tõrked, peaks kontrollima, kuidas süsteem tervikuna hakkab reageerima uuendustele. Selleks tuleb uuendused käsitsi teha.

Värskendada tuleb serveri juhtpaneeli, sisuhaldussüsteeme, kõik pluginaid ja operatsioonisüsteemi tarkvara. Iga uus uunedus sisaldab teadaolevate turvaprobleemide lahendused.

4.8 Kasutajate arvu ja nende õiguste kontrollimine

Selleks, et rünnaku oht vähendada, tuleb aktiivselt monitoorida kasutajat, kes saavad ligi serverile ning teha kindlaks, kellel on administraatori õigused.

Tuleb keelata liigsed külaliskontod ja kontrollida kasutajate õigused. Ei ole vaja anda administraatori õigused nendele kasutajatele, kelle töö seda ei nõua. Kuna administraator saab täita mis tahes käske, ründajad keskenduvad sellele, et teada saada adminisraatori salasõna ning rünnata selle spetsiifilise kasutajat. Kui ettevõtes on palju kasutajaid administraatori õigustega, siis see võib tekitada täiendavad turvaaugud.

Kui serveri haldamisega on seotud rohkem, kui üks inimene, siis tuleb luua igaühele eraldi konto. See aitab täpselt jälgida, kes selle või teise toimingu süsteemis tegi ning annab võimaluse kohe rünnakut peatada.

Samuti tuleb kasutada kohalikke ja rühma turvapoliitikaid selleks, et käivitada automaatse väljalogimist teatud aja pärast, kui kasutaja ei ole midagi selle aja jooksul teinud. Näiteks, seadistada kasutajate väljalogimine iga 15 minutit pärast. [27]

4.9 Kasutajate salasõna vastavus tingimustele ja nende uuendamine

Turvalisuse tõstmiseks tuleb seadistada vastavad tingimused salasõna loomiseks. Need tingimused peavad rakendama iga ettevõte kasutajale, kellel on serverile ligipääs. [27]

1. Salasõna ei saa siseldada tühikud.
2. Salasõna peab olema teatud pikkusega. Parem panna 16 märki.
3. Salasõna peab olema salvestatud eraldi andmebaasis ja olema krüpteeritud.
4. On vaja seadistada automaatne väljalogimine.
5. Salasõna aegumisel kasutaja peab vahetama parooli. Uueks salasõnaks ei saa kasutada eelnev salasõna. Aegumise periood saab olla erinev – paar nädalat või paar kuud, sõltub turvapoliitikast.
6. Salasõna ei tohi sisaldada kasutaja nime, telefoninumbri, sünipäeva või muu isiklikku infot.

7. Salasõna loomisel peab olema seadistatud raskusaste.

8. Salasõna peab olema unikaalne.

4.10 SSH autentimine ja kahekordne autentimine

Salasõna asemel on võimalik ka autentida kasutajat SSH võtmete paari abil. Võtmetel on rohkem bitte kui tavalisel salasõnal ja enamik kaasaegseid arvuteid ei saa neid lõhkuda. Näiteks, RSA 2048-bitine krüpteerimine, mis on kasutatud SSH võtmete loomisel, on samaväärne kui 617-kohaline salasõna.

SSH võtmepaar koosneb avalikust ja privaatvõtmetest. Avalik võti on kättesaadav mitmes eksemplaris, millest üks jääb serverisse, teised jagatakse kasutajate vahel. Kasutajatel, kellel on avalik võti, on õigus andmeid krüpteerida kuid need andmed saavad lugeda ainult kasutajad, kellel on vastav privaatvõti. Privaatvõtit ei tohi jagada ja võti peab olema turvaline. Ühenduse loomisel server küsib enne privilegeeritud juurdepääsu lubamist tõendeid selle kohta, et kasutajal on privaatne võti. [27]

Kahekordne autentimine on barjäär, milles enamik küberkurjategijad üle ei saa. Kahekordse autentimise puhul salasõnale või SSH võtmele juurde tuleb kinnitus kood SMS-iga kasutaja telefonile. Kasutaja peab sisestama selle koodi, et tõendada serveri ligipääsu õigust. [29]

4.11 VPN-I kasutamine

Virtuaalne privaatne võrk ehk VPN on viis turvalise ühenduse loomiseks kaugarvutite ja kasutatud ühenduse vahel. VPN võimaldab konfigureerida oma privaatse võrgu nii, nagu kasutusel on turvaline kohalik võrk. Privaatsed võrkud kasutavad privaatsed IP aadressid selleks, et luua eraldatud sidekanalid serverite vahel. See võimaldab mitmel serveril vahetada teavet ja andmeid nii, et avalik võrk jääb puutamata. [27]

4.12 Sisselogimiste monitooring

Sissetungi vältimise tarkvara kasutamine sisselogimiskatsete jälgimiseks on viis kaitsta oma serverit rünnakute eest. Sissetungi vältimise tarkvara jälgib kõiki logifaile ja tuvastab kahtlased sisselogimiskatsed. Kui katsete arv ületab seatud normi, blokeerib sissetungimise vältimise tarkvara IP-aadressi teatud ajaperioodiks või isegi määramata ajaks. [27]

4.13 Süsteemi analüüs

Kui kõik eelnevalt kirjeldatud sammud on tehtud, on vaja teostada süsteemi analüüsi. Süsteemi analüüs näitab, kas rakendatud meetmed töötavad, kas veebiserveri turvalisus on tagatud ning kas on vaja mõned sammud uuesti teha selleks, et turvalisuse tõsta.

Kokkuvõte

Käesolevas töös vastavalt püstitud eesmärgile analüüsida avatud lähtekoodiga veebiserverid ja nende turvalisuse tagamise meetodid on välja toodud ülevaade kõige populaarsematest veebiserveritest, võimalikutest veebiserverite rünnakutest ja veebiserverite turvalisuse tagamise võimalustest. Veebiserverid on analüüsitud ja on tehtud nende võrdlemine.

Antud töös on välja toodud Telia Eesti AS kasutatud veebiserverid, millised turvameetmed kasutab Telia Eesti AS ning kuidas Telia Eesti AS lahendab veebiserverite rünnakutega seotud probleemid. Samuti antud töös on välja toodud rünnakud, mis olid tehtud Telia Eesti AS veebiserveritele.

Antud töös on tehtud metoodika veebiserverite turvalisuse tagamiseks, mis koosneb kolmeteistkümnest sammudest. Metoodikat saab kasutada uue serveri loomise ajal selleks, et oma veebiserverite turvalisuse tagada. Tulevikus antud metoodika esitatakse Telia Eesti AS-ile selleks, et metoodikat rakendada.

Kasutatud kirjandus

[1] W. S. Survey, „Netcraft,“ 29 Märts 2021. [Võrgumaterjal]. Available: https://news.netcraft.com/archives/category/web-server-survey/. [Kasutatud Aprill 2021].

[2] V. Buhteev, „Apache,“ Jaanuar 2021. [Võrgumaterjal]. Available: https://timeweb.com/ru/community/articles/chto-takoe-veb-server-apache-i-kak-im-polzovatsya. [Kasutatud Aprill 2021].

[3] „Nginx,“ Nginx, [Võrgumaterjal]. Available: https://nginx.org/ru/. [Kasutatud Aprill 2021].

[4] M. Networks, „Apache vs Nginx,“ Merion Networks, Jaanuar 2020. [Võrgumaterjal]. Available: https://wiki.merionet.ru/servernye-resheniya/34/apache-vs-nginx-sravnenie-i-preimushhestva/. [Kasutatud Aprill 2021].

[5] P. Vuollet, „What is IIS?,“ Stackify, 8 Mai 2018. [Võrgumaterjal]. Available: https://stackify.com/iis-web-server/. [Kasutatud Aprill 2021].

[6] M. Networks, „Apache või IIS,“ Merion Networks, 6 Märts 2016. [Võrgumaterjal]. Available: https://wiki.merionet.ru/servernye-resheniya/3/apache-ili-iis/. [Kasutatud Aprill 2021].

[7] V. Protasov, „OpenResty,“ 17 Veebruar 2017. [Võrgumaterjal]. Available: https://habr.com/ru/post/321864/. [Kasutatud Aprill 2021].

[8] „OpenResty,“ [Võrgumaterjal]. Available: https://openresty.org/en/. [Kasutatud Aprill 2021].

[9] „Apache vs Lighttpd,“ [Võrgumaterjal]. Available: https://www.similartech.com/compare/apache-vs-lighttpd. [Kasutatud Aprill 2021].

[10] „Lighttpd,“ 14 Aprill 2012. [Võrgumaterjal]. Available: http://gooblogerman.blogspot.com/2012/04/lighttpd-un-servidor-web-muy-agil-y.html. [Kasutatud Aprill 2021].

[11] „XSS rünnak,“ 8 September 2020. [Võrgumaterjal]. Available: https://wiki.rookee.ru/cross-site-scripting/. [Kasutatud Aprill 2021].

[12] KristenS, „Cross Site Reguest Forgery (CSRF),“ [Võrgumaterjal]. Available: https://owasp.org/www-community/attacks/csrf. [Kasutatud Aprill 2021].

[13] „What is a Denial-of-Service (DoS) Attack?,“ [Võrgumaterjal]. Available: https://www.cloudflare.com/learning/ddos/glossary/denial-of-service/. [Kasutatud Aprill 2021].

[14] „What is a DDoS Attack?,“ [Võrgumaterjal]. Available: https://www.cloudflare.com/learning/ddos/what-is-a-ddos-attack/. [Kasutatud Aprill 2021].

[15] „SQL injection,“ [Võrgumaterjal]. Available: https://portswigger.net/web-security/sql-injection. [Kasutatud Aprill 2021].

[16] T. O. Foundation, „OWASP,“ [Võrgumaterjal]. Available: https://owasp.org/. [Kasutatud Aprill 2021].

[17] „Netsparker,“ [Võrgumaterjal]. Available: https://www.netsparker.com/product/#. [Kasutatud Aprill 2021].

[18] „Netsparker Enterprise,“ [Võrgumaterjal]. Available: https://www.anti-malware.ru/products/Netsparker-Enterprise. [Kasutatud Aprill 2021].

[19] „Netsparker reviews,“ [Võrgumaterjal]. Available: https://www.g2.com/products/netsparker/reviews. [Kasutatud Aprill 2021].

[20] A. Kotsukov, „NAXSI vs ModSecurity,“ [Võrgumaterjal]. Available: https://networkguru.ru/waf-sistem%D0%B0-dlya-zashchity-veb-prilozhenij-naxsi-ili-modsecurity/. [Kasutatud Aprill 2021].

[21] S. Surva, Interviewee, *Telia Eesti AS.* [Intervjuu]. Aprill 2021.

[22] F. Glossary, „Load Balancer,“ [Võrgumaterjal]. Available: https://www.f5.com/services/resources/glossary/load-balancer. [Kasutatud Aprill 2021].

[23] „Application Security Solutions,“ [Võrgumaterjal]. Available: https://www.f5.com/solutions/application-security. [Kasutatud Aprill 2021].

[24] T. Contributor, „Access control list,“ Jaanuar 2006. [Võrgumaterjal]. Available: https://searchsoftwarequality.techtarget.com/definition/access-control-list#:~:text=An%20access%20control%20list%20(ACL,identifies%20its%20access%20control%20list.. [Kasutatud Aprill 2021].

[25] „What is a Web Application Firewall (WAF)?,“ [Võrgumaterjal]. Available: https://www.f5.com/services/resources/glossary/web-application-firewall#:~:text=A%20web%20application%20firewall%20(WAF)%20protects%20web%20applications%20from%20a,gateway%20to%20your%20valuable%20data.. [Kasutatud Aprill 2021].

[26] S. turvalisus, Mai 2018. [Võrgumaterjal]. Available: https://integrus.ru/blog/zashhita-servera-ot-vzloma.html. [Kasutatud Aprill 2021].

[27] S. Simic, „Server security tips,“ Aprill 2019. [Võrgumaterjal]. Available: https://phoenixnap.com/kb/server-security-tips. [Kasutatud Aprill 2021].

[28] Valeriy_Squadra, „7 võimalusi serveri turvalisuse jaoks,“ November 2016. [Võrgumaterjal]. Available: https://habr.com/ru/company/galtsystems/blog/314344/. [Kasutatud Aprill 2021].

[29] „Kuidas kaitsta veebiserver veebis,“ Märts 2019. [Võrgumaterjal]. Available: https://1cloud.ru/blog/kak-zaschitit-server-v-internete. [Kasutatud Aprill 2021].

[30] S.-2. C. H. Algorithm, „Movable Type Scripts,“ [Võrgumaterjal]. Available: https://www.movable-type.co.uk/scripts/sha256.html. [Kasutatud Aprill 2021].

[31] A. A. E. Standard, „Movable Type Scripts,“ [Võrgumaterjal]. Available: https://www.movable-type.co.uk/scripts/aes.html. [Kasutatud Aprill 2021].

[32] H. Spiegelberg, „Linuxi kasutatavate veebiserverite turvalisuse analüüs,“ Tallinn, 2017.

Lisa 1 – Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks[1]

Mina, Aleksandra Budarina

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „Veebiserverite turvalisuse analüüs Telia Eesti AS näitel“, mille juhendaja on Vladimir Viies
   1.1. reprodutseerimiseks lõputöö säilitamise ja elektroonse avaldamise eesmärgil, sh Tallinna Tehnikaülikooli raamatukogu digikogusse lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;
   1.2. üldsusele kättesaadavaks tegemiseks Tallinna Tehnikaülikooli veebikeskkonna kaudu, sealhulgas Tallinna Tehnikaülikooli raamatukogu digikogu kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.
2. Olen teadlik, et käesoleva lihtlitsentsi punktis 1 nimetatud õigused jäävad alles ka autorile.
3. Kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest ning muudest õigusaktidest tulenevaid õigusi.

16.04.2021

1 Lihtlitsents ei kehti juurdepääsupiirangu kehtivuse ajal vastavalt üliõpilase taotlusele lõputööle juurdepääsupiirangu kehtestamiseks, mis on allkirjastatud teaduskonna dekaani poolt, välja arvatud ülikooli õigus lõputööd reprodutseerida üksnes säilitamise eesmärgil. Kui lõputöö on loonud kaks või enam isikut oma ühise loomingulise tegevusega ning lõputöö kaas- või ühisautor(id) ei ole andnud lõputööd kaitsvale üliõpilasele kindlaksmääratud tähtajaks nõusolekut lõputöö reprodutseerimiseks ja avalikustamiseks vastavalt lihtlitsentsi punktidele 1.1. ja 1.2, siis lihtlitsents nimetatud tähtaja jooksul ei kehti.

Lisa 2 – HTTPS tugi kontroll

```
var https = require('https');
var options = {
    host: 'online.ee', //veebileht
    port: 443,
    method: 'GET'
};

var req = https.request(options, function(res) {
    console.log(res.socket.getPeerCertificate());
});

req.end();
```

Lisa 3 – Andmete krüpteerimine TLS protokolli abiga

```
const AES_GCM_SHA256 = require('./Encryption_module/AES_GCM_SHA256');

let message = 'test';
let privateKey = 'Very secret password';
let authentication_data = 'ip:192.168.1.1';
let AES_specification = 128; // 128/192/256

console.log('Is used AES_' + AES_specification + '_GCM_SHA256');
console.log('Initial message is: ' + message    );
let encrypted_message = AES_GCM_SHA256.encrypt(message, privateKey,
authentication_data, AES_specification);
console.log('Encrypted message is: ' + encrypted_message);

let decrypted_message = AES_GCM_SHA256.decrypt(encrypted_message, privateKey,
authentication_data, AES_specification);
console.log('Decrypted message is: ' + decrypted_message);
```

Joonis 9 index.js

```
const SHA256 = require('./SHA256');
const AES = require('./AES');

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - -  */

class AES_GCM_SHA256 extends AES {

    /**
     * Encrypt a text using AES encryption in Counter mode of operation.
     *
     * Unicode multi-byte character safe.
     *
     * @param   {string} plaintext - Source text to be encrypted.
     * @param   {string} password - The password to use to generate a key for
encryption.
     * @param   {number | 128} nBits - Number of bits (Default 128) to be
used in the key; 128 / 192 / 256.
     * @param   {string} authData - Authentication data of sender
     * @returns {string} Encrypted text, base-64 encoded.
     *
     * @example
     *   const encr = AES_GCM_SHA256.encrypt('big secret', 'pāşšŵōřđ',
'ip:192.168.1.1', 128); // 'TwI6amwIcmAClp64l52n4KlTYAeTMVfNTCKDj9NMBRgEBA=='
     */
    static encrypt(plaintext, password, authData, nBits = 128) {
        if (![ 128, 192, 256 ].includes(nBits)) throw new Error('Key size is
not 128 / 192 / 256');
        plaintext = AES_GCM_SHA256.utf8Encode(String(plaintext));
        password = AES_GCM_SHA256.utf8Encode(String(password));
        authData = AES_GCM_SHA256.utf8Encode(String(authData));

        //use SHA256 Hash function to encrypt and expand password to 32 bytes
long
        let key = SHA256.encrypt(password); // create 32-byte key
        key = key.slice(0, nBits/8); // set fixed 16/24/32 byte key

        // initialise 1st 8 bytes of counter block with nonce (NIST SP 800-
38A §B.2): [0-1] = millisec,
        // [2-3] = random, [4-7] = seconds, together giving full sub-millisec
uniqueness up to Feb 2106
        const timestamp = (new Date()).getTime(); // milliseconds since 1-
Jan-1970
        const nonceMs = timestamp%1000;
        const nonceSec = Math.floor(timestamp/1000);
        const nonceRnd = Math.floor(Math.random()*0xffff);
        // for debugging: const [ nonceMs, nonceSec, nonceRnd ] = [ 0, 0, 0
];
        const counterBlock = [ // 16-byte array; blocksize is fixed at 16 for
AES
            nonceMs  & 0xff, nonceMs >>>8 & 0xff,
            nonceRnd & 0xff, nonceRnd>>>8 & 0xff,
```

```
            nonceSec & 0xff, nonceSec>>>8 & 0xff, nonceSec>>>16 & 0xff,
nonceSec>>>24 & 0xff,
            0, 0, 0, 0, 0, 0, 0, 0,
        ];

        // and convert nonce to a string to go on the front of the ciphertext
        const nonceStr = counterBlock.slice(0, 8).map(i =>
String.fromCharCode(i)).join('');

        // convert (utf-8) plaintext to byte array
        const plaintextBytes = plaintext.split('').map(ch =>
ch.charCodeAt(0));

        // convert (utf-8) authentication data to byte array
        const authDataBytes = authData.split('').map(ch => ch.charCodeAt(0));

        // ------------ perform encryption ------------
        const [ciphertextBytes, cipherTagBytes] =
AES_GCM_SHA256.GCM_Encryption(plaintextBytes, key, counterBlock,
authDataBytes);

        // convert byte array to (utf-8) ciphertext string
        const ciphertextUtf8 = ciphertextBytes.map(i =>
String.fromCharCode(i)).join('');

        // convert tag byte array to (utf-8) tag string
        const cipherTagUtf8 = cipherTagBytes.map(i =>
String.fromCharCode(i)).join('');

        // base-64 encode ciphertext
        const ciphertextB64 =  AES_GCM_SHA256.base64Encode(nonceStr +
ciphertextUtf8 + cipherTagUtf8);

        return ciphertextB64;
    }

    /**
     *  Galois/Counter Mode (GCM) is a mode of operation for symmetric-key
cryptographic block ciphers which is widely adopted for its performance.
     *  - All operation are done in GF(2^128) field.
     *  - 0^128 - means a string of 128 bits
     *
     *  Steps:
     *  1. Get H hash key using encryption algorithm H = AES(0^128, K)
     *  2. Get Special Y, which is actually a counter block for CTR mode.
     *      1. if length of IV is 96 bit, add 0^31 and last 1 bit to IV to
creat Y0
     *      2. if length of IV is not 96 bit, use GHASH function to create Y0
= GHASH(H, {}, IV ).
     *  3. Encrypt first block Y0 using encryption algorithm, and save it for
futher
     *  4. Encrypt plainttext using encrypted blocks Y(1 + n) and counter
mode method, where n - one 16 byte block
```

```
     *      1. Ci = Pi ⊕ E(K, Yi) for i = 1, . . . , n − 1
     *  5. Create Tag of encryption using GHASH function and concatenated
byte string of authentication data, encrypted text and their length
     *      1. T = GHASH(H, A, C, len(A), len(C)) ⊕ E(K, Y0)
     *
     *
     * @param   {number[]} plaintext - Plaintext to be encrypted, as byte
array.
     * @param   {number[]} key - Key to be used to encrypt plaintext.
     * @param   {number[]} IV - Initial 16-byte vector (with nonce & 0
counter).
     * @param   {number[]} A - Authentication data of sender
     * @returns {[number[], number[]]} Ciphertext and Crypted Tag as byte
array.
     *
     * @private
     */
    static GCM_Encryption(plaintext, key, IV, A) {

        // It is considered that each element of the array contains 1 byte or
8 bits.
        const bits = 8;

        // generate key schedule - an expansion of the key into distinct Key
Rounds for each round
        const keySchedule = AES.keyExpansion(key);

        //Step 1
        const zeroBlock = new Array(16)
        for (let i = 0; i < zeroBlock.length; i++) {
            zeroBlock[i] = 0;
        }
        const H = AES.cipher(zeroBlock, keySchedule); // Combine key with
0^128 block of bits

        //Step 2
        let Y, additionalBlock;
        if (IV.length * bits === 96) { //If len(IV) is 96 bits, we can avoid
using GHASH function
            additionalBlock = [0, 0, 0, 1]
            Y = IV.concat(additionalBlock);
        } else { //If len(IV) is not 96 bits, we have to use GHASH function

            //This step is needed to expand IV to block, which is devided by
128 for GHASH function
            let s = 128 * (Math.ceil(IV.length * bits / 128)) - IV.length *
bits
            additionalBlock = new Array(8+(s/8))
            for (let i = 0; i < additionalBlock.length; i++) {
                additionalBlock[i] = 0;
            }
```

```
            let IVBlock64BitLength =
AES_GCM_SHA256.create64BitBlockLength(IV);

            Y = AES_GCM_SHA256.GHASH(IV.concat(additionalBlock,
IVBlock64BitLength), H);
        }

        //Step 3 & Step 4
        const [Y0, ciphertext] =
AES_GCM_SHA256.counterMode_encrypt(plaintext, key, Y);

        //Step 5
        //This step is needed to expand Authentication data and ciphertext to
block, which is devided by 128 for GHASH function
        let u = 128*(Math.ceil(ciphertext.length*bits/128)) -
ciphertext.length*bits
        let v = 128*(Math.ceil(A.length*bits/128)) - A.length*bits

        const zeroBlock_u = new Array(u/8)
        for (let i = 0; i < zeroBlock_u.length; i++) {
            zeroBlock_u[i] = 0;
        }

        const zeroBlock_v = new Array(v/8)
        for (let i = 0; i < zeroBlock_v.length; i++) {
            zeroBlock_v[i] = 0;
        }

        const ABlock64BitLength = AES_GCM_SHA256.create64BitBlockLength(A);
        const CBlock64BitLength =
AES_GCM_SHA256.create64BitBlockLength(ciphertext);

        const S = AES_GCM_SHA256.GHASH(A.concat(zeroBlock_v, ciphertext,
zeroBlock_u, ABlock64BitLength, CBlock64BitLength), H);

        const Tag = AES_GCM_SHA256.GF_add(S, Y0);

        return [ciphertext, Tag]
    }

    /**
     * NIST SP 800-38A sets out recommendations for block cipher modes of
operation in terms of byte
     * operations. This implements the §6.5 Counter Mode (CTR).
     *
     *     O_j = CIPH_k(T_j)      for j = 1, 2 … n
     *     C_j = P_j ⊕ O_j        for j = 1, 2 … n-1
     *     C*_n = P* ⊕ MSB_u(O_n) final (partial?) block
     *   where CIPH_k is the forward cipher function, O output blocks, P
plaintext blocks, C
     *   ciphertext blocks
     *
```

```
     * @param   {number[]} plaintext - Plaintext to be encrypted, as byte
array.
     * @param   {number[]} key - Key to be used to encrypt plaintext.
     * @param   {number[]} counterBlock - Initial 16-byte CTR counter block
(with nonce & 0 counter).
     * @returns {[number[], number[]]} Y0 and Ciphertext as byte array.
     *
     * @private
     */
    static counterMode_encrypt(plaintext, key, counterBlock) {
        const blockSize = 16; // block size fixed at 16 bytes / 128 bits
(Nb=4) for AES

        // generate key schedule - an expansion of the key into distinct Key
Rounds for each round
        const keySchedule = AES.keyExpansion(key);

        const blockCount = Math.ceil(plaintext.length/blockSize);
        const ciphertext = new Array(plaintext.length);

        //Create first special block Y0 for encryption of Tag
        const Y0 = AES.cipher(counterBlock, keySchedule);

        // increment counter block (counter in 2nd 8 bytes of counter block,
big-endian)
        counterBlock[blockSize-1]++;

        // and propagate carry digits
        for (let i=blockSize-1; i>=8; i--) {
            counterBlock[i-1] += counterBlock[i] >> 8;
            counterBlock[i] &= 0xff;
        }

        for (let b=0; b<blockCount; b++) {
            // ---- encrypt counter block; Oⱼ = CIPHₖ(Tⱼ) ----
            const cipherCntr = AES.cipher(counterBlock, keySchedule);
            // block size is reduced on final block
            const blockLength = b<blockCount-1 ? blockSize :
(plaintext.length-1)%blockSize + 1;

            // ---- xor plaintext with ciphered counter byte-by-byte; Cⱼ = Pⱼ
⊕ Oⱼ ----
            for (let i=0; i<blockLength; i++) {
                ciphertext[b*blockSize + i] = cipherCntr[i] ^
plaintext[b*blockSize + i];
            }

            // increment counter block (counter in 2nd 8 bytes of counter
block, big-endian)
            counterBlock[blockSize-1]++;

            // and propagate carry digits
```

```
            for (let i=blockSize-1; i>=8; i--) {
                counterBlock[i-1] += counterBlock[i] >> 8;
                counterBlock[i] &= 0xff;
            }

        }

        return [Y0, ciphertext]
    }

    /**
     * Decrypt a text encrypted by AES in counter mode of operation.
     *
     * @param   {string} ciphertext - Cipher text to be decrypted.
     * @param   {string} password - Password to use to generate a key for
decryption.
     * @param   {number | 128} nBits - Number of bits (Default 128) to be
used in the key; 128 / 192 / 256.
     * @param   {string} authData - Authentication data of sender
     * @returns {string} Decrypted text
     *
     * @example
     *   const decr =
AES_GCM_SHA256.decrypt('TwI6amwIcmAClp64l52n4KlTYAeTMVfNTCKDj9NMBRgEBA==',
'pāşšŵōřđ', 'ip:192.168.1.1', 128); // 'big secret'
     */
    static decrypt(ciphertext, password, authData, nBits = 128) {
        if (![ 128, 192, 256 ].includes(nBits)) throw new Error('Key size is
not 128 / 192 / 256');
        ciphertext = AES_GCM_SHA256.base64Decode(String(ciphertext));
        password = AES_GCM_SHA256.utf8Encode(String(password));
        authData = AES_GCM_SHA256.utf8Encode(String(authData));

        //use SHA256 Hash function to encrypt and expand password to 32 bytes
long
        let key = SHA256.encrypt(password); // create 32-byte key
        key = key.slice(0, nBits/8); // set fixed 16/24/32 byte key

        // recover nonce from 1st 8 bytes of ciphertext into 1st 8 bytes of
counter block

        const counterBlock = [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
];
        for (let i=0; i<8; i++) counterBlock[i] = ciphertext.charCodeAt(i);

        // recover crypted Tag from last 16 bytes of ciphertext
        const cipherTagBytes = new Array(16);
        for (let i=0; i < 16; i++) cipherTagBytes[i] =
ciphertext.charCodeAt(ciphertext.length - i - 1);
        cipherTagBytes.reverse()

        // convert ciphertext to byte array (skipping past initial 8 bytes)
```

```
        const ciphertextBytes = new Array(ciphertext.length-24);
        for (let i=8; i<ciphertextBytes.length + 8; i++) ciphertextBytes[i-8]
= ciphertext.charCodeAt(i);

        //Convert authentication data of sender to byte array (skipping past
initial 8 bytes)
        const authDataBytes = authData.split('').map(ch => ch.charCodeAt(0));

        // ------------ perform decryption ------------
        const plaintextBytes = AES_GCM_SHA256.GCM_decryption(ciphertextBytes,
key, counterBlock, cipherTagBytes, authDataBytes);

        // convert byte array to (utf-8) plaintext string
        const plaintextUtf8 = plaintextBytes.map(i =>
String.fromCharCode(i)).join('');

        // decode from UTF8 back to Unicode multi-byte chars
        const plaintext = AES_GCM_SHA256.utf8Decode(plaintextUtf8);

        return plaintext;
    }

    /**
     *  Galois/Counter Mode (GCM) is a mode of operation for symmetric-key
cryptographic block ciphers which is widely adopted for its performance.
     *  - All operation are done in GF(2^128) field.
     *  - 0^128 - means a string of 128 bits
     *
     *  Steps:
     *  1. Get H hash key using encryption algorithm H = AES(0^128, K)
     *  2. Get Special Y, which is actually a counter block for CTR mode.
     *      1. if length of IV is 96 bit, add 0^31 and last 1 bit to IV to
creat Y0
     *      2. if length of IV is not 96 bit, use GHASH function to create Y0
= GHASH(H, {}, IV ).
     *  3. Decrypt first block Y0 using decryption algorithm, and save it for
the futher
     *  4. Decrypt plainttext using decrypted blocks Y(1 + n) and counter
mode method, where n - one 16 byte block
     *      1. Pi = Ci ⊕ E(K, Yi) for i = 1, . . . , n − 1
     *  5. Create Tag of encryption using GHASH function and concatenated
byte string of authentication data, ciphertext and their length
     *      1. T = GHASH(H, A, C, len(A), len(C)) ⊕ E(K, Y0)
     *  6. Compare Encrypted Tag of sender with the new encrypted Tag
     *      1. If they are equal, return plainttext
     *      2. If they are not equal, throw Error
     *
     *
     * @param   {number[]} plaintext - Plaintext to be encrypted, as byte
array.
     * @param   {number[]} key - Key to be used to encrypt plaintext.
```

```
     * @param   {number[]} IV - Initial 16-byte vector (with nonce & 0
counter).
     * @param   {number[]} A - Authentication data of sender
     * @returns {[number[], number[]]} Ciphertext and Crypted Tag as byte
array.
     *
     * @private
     */
    static GCM_decryption(ciphertext, key, IV, senderCryptTag, A) {
        // It is considered that each element of the array contains 1 byte or
8 bits.
        const bits = 8;

        // generate key schedule - an expansion of the key into distinct Key
Rounds for each round
        const keySchedule = AES.keyExpansion(key);

        //Step 1
        const zeroBlock = new Array(16)
        for (let i = 0; i < zeroBlock.length; i++) {
            zeroBlock[i] = 0;
        }
        const H = AES.cipher(zeroBlock, keySchedule); // Combine key with
0^128 block of bits

        //Step 2
        let Y, additionalBlock;
        if (IV.length * bits === 96) { //If len(IV) is 96 bits, we can avoid
using GHASH function
            additionalBlock = [0, 0, 0, 1]
            Y = IV.concat(additionalBlock);
        } else { //If len(IV) is not 96 bits, we have to use GHASH function

            //This step is needed to expand IV to block, which is devided by
128 for GHASH function
            let s = 128 * (Math.ceil(IV.length * bits / 128)) - IV.length *
bits
            additionalBlock = new Array(8+(s/8))
            for (let i = 0; i < additionalBlock.length; i++) {
                additionalBlock[i] = 0;
            }

            let IVBlock64BitLength =
AES_GCM_SHA256.create64BitBlockLength(IV);

            Y = AES_GCM_SHA256.GHASH(IV.concat(additionalBlock,
IVBlock64BitLength), H);
        }

        //Step 3 & Step 4
        const [plaintext, Y0] =
AES_GCM_SHA256.counterMode_decrypt(ciphertext, key, Y);
```

```
        //Step 5
        //This step is needed to expand Authentication data and ciphertext to
block, which is devided by 128 for GHASH function
        let u = 128*(Math.ceil(ciphertext.length*bits/128)) -
ciphertext.length*bits
        let v = 128*(Math.ceil(A.length*bits/128)) - A.length*bits

        const zeroBlock_u = new Array(u/8)
        for (let i = 0; i < zeroBlock_u.length; i++) {
            zeroBlock_u[i] = 0;
        }

        const zeroBlock_v = new Array(v/8)
        for (let i = 0; i < zeroBlock_v.length; i++) {
            zeroBlock_v[i] = 0;
        }

        const ABlock64BitLength = AES_GCM_SHA256.create64BitBlockLength(A);
        const CBlock64BitLength =
AES_GCM_SHA256.create64BitBlockLength(ciphertext);

        const S = AES_GCM_SHA256.GHASH(A.concat(zeroBlock_v, ciphertext,
zeroBlock_u, ABlock64BitLength, CBlock64BitLength), H);

        const Tag = AES_GCM_SHA256.GF_add(S, Y0);

        //Step 6
        for (let i = 0; i < senderCryptTag.length; i++) {
            if (senderCryptTag[i] !== Tag[i]) throw new Error("Crypted Tag of
sender was corrupted");
        }

        return plaintext
    }

    /**
     * NIST SP 800-38A sets out recommendations for block cipher modes of
operation in terms of byte
     * operations. This implements the §6.5 Counter Mode (CTR).
     *
     *     O_j = CIPH_k(T_j)      for j = 1, 2 … n
     *     P_j = C_j ⊕ O_j        for j = 1, 2 … n-1
     *     P*_n = C* ⊕ MSB_u(O_n) final (partial?) block
     *   where CIPH_k is the forward cipher function, O output blocks, C
ciphertext blocks, P
     *   plaintext blocks
     *
     * @param   {number[]} ciphertext - Ciphertext to be decrypted, as byte
array.
     * @param   {number[]} key - Key to be used to decrypt ciphertext.
```

```
     * @param   {number[]} counterBlock - Initial 16-byte CTR counter block
(with nonce & 0 counter).
     * @returns {number[]} Plaintext as byte array.
     *
     * @private
     */
    static counterMode_decrypt(ciphertext, key, counterBlock) {
        const blockSize = 16; // block size fixed at 16 bytes / 128 bits
(Nb=4) for AES

        // generate key schedule - an expansion of the key into distinct Key
Rounds for each round
        const keySchedule = AES.keyExpansion(key);

        const blockCount = Math.ceil(ciphertext.length/blockSize);
        const plaintext = new Array(ciphertext.length);

        //Create first special block Y0 for encryption of Tag
        const Y0 = AES.cipher(counterBlock, keySchedule);
        // increment counter block (counter in 2nd 8 bytes of counter block,
big-endian)
        counterBlock[blockSize-1]++;
        // and propagate carry digits
        for (let i=blockSize-1; i>=8; i--) {
            counterBlock[i-1] += counterBlock[i] >> 8;
            counterBlock[i] &= 0xff;
        }

        for (let b=0; b<blockCount; b++) {
            // ---- decrypt counter block; Oⱼ = CIPHₖ(Tⱼ) ----
            const cipherCntr = AES.cipher(counterBlock, keySchedule);

            // block size is reduced on final block
            const blockLength = b<blockCount-1 ? blockSize :
(ciphertext.length-1)%blockSize + 1;

            // ---- xor ciphertext with ciphered counter byte-by-byte; Pⱼ =
Cⱼ ⊕ Oⱼ ----
            for (let i=0; i<blockLength; i++) {
                plaintext[b*blockSize + i] = cipherCntr[i] ^
ciphertext[b*blockSize + i];
            }

            // increment counter block (counter in 2nd 8 bytes of counter
block, big-endian)
            counterBlock[blockSize-1]++;
            // and propagate carry digits
            for (let i=blockSize-1; i>=8; i--) {
                counterBlock[i-1] += counterBlock[i] >> 8;
                counterBlock[i] &= 0xff;
            }
```

```
        }

        return [plaintext, Y0];
    }

    /* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - -  */

    /**
     * The GHASH algorithm is a special form of the
        Carter-Wegman polynomial evaluation MAC. Each 16-bytes block of the
        authenticated data is multiplied by a different power of the hash key
(H), where
        computations occur in some specific binary finite field that we
denote here by
        GF_GCM(2^128)
     *
     *
     * @param {number[]} DATA - Byte array, which containes data to hash (the
bit length of DATA is assumed to be divisible by 128)
     * @param {number[]} H - hash key of GCM algorithm
     * @returns {number[]} Hashed 128 byte-array of DATA byte-array
     *
     * @private
     */
    static GHASH(DATA, H) {
        //The number of bits in one byte array element
        const bits = 8;

        //Devide long DATA byte array to 16-byte blocks
        const N = DATA.length * bits / 128
        const M = new Array(N)

        for (let i = 0; i < N; i++) {
            M[i] = DATA.slice(i*16, i*16 + 16)
        }

        //Calculate hash function using formula: M(1)×H + M(2)×H + … + M(N)×H
        let ans = AES_GCM_SHA256.GF_mul(M[0], H)
        for (let i = 1; i < N; i++) {
            ans = AES_GCM_SHA256.GF_add(ans, AES_GCM_SHA256.GF_mul(M[i], H))
        }
        return ans
    }


    /**
     * Addition operation in GF_GCM(2^128)
     *
     *
     * @param {number[]} arr1 //First byte-array
```

```
     * @param {number[]} arr2 //Second byte-array
     * @returns {number[]} Product of addition
     *
     * @private
     */
    static GF_add (arr1, arr2) {
        const X = [...arr1];
        const Y = [...arr2];
        for (let i = 0; i < X.lenght; i++) {
            X[i] ^= Y[i]
        }
        return X
    }

    /**
     * Multiplication operation in GF_GCM(2^128)
     *
     *
     *
     * @param {number[]} arr1 //First byte-array
     * @param {number[]} arr2 //Second byte-array
     * @returns {number[]} Product of multiplication
     *
     * @private
     */

    static GF_mul (arr1, arr2) {
        let X = [...arr1]
        const Y = [...arr2]
        const Z = new Array(16); // Product of two byte-arrays
multiplications

        for (let i = 0; i < Z.length; i++) {
            Z[i] = 0;
        }

        const R = new Array(16) // Special polynomial byte-array for
multiplication, which is using value R = 11100001||0^120
        R[0] = 225
        for(let i = 1; i < R.length; i++) {
            R[i] = 0;
        }

        for (let i = 0; i < 128; i++) {
            if (Y[parseInt(i/8)] & (1 << (i % 8))) {
                for (let j = 0; j < 16; j++) {
                    Z[j] ^= X[j];
                }
            }

            if((X[15] & 128) === 0) {
```

```
                X = AES_GCM_SHA256.Rsh_byte_array(X);
            } else {
                X = AES_GCM_SHA256.Rsh_byte_array(X);
                for (let j = 0; j < 16; j++) {
                    X[j] ^= R[j];
                }
            }
        }
        return Z
    }

    /**
     * Special Right Shift function, this implimintation is using byte-array
     *
     * @param {number[]} A - Byte-array
     * @returns {number[]} Rightshifted byte-array
     *
     * @private
    */
    static Rsh_byte_array (A) {
        const X = [...A];
        for (let i = 0, prev_carry = 0, curr_carry = 0; i < X.length; i++) {
            if (X[i] & 1) curr_carry = 1;
            X[i] >>= 1;

            if (prev_carry) {
                X[i] += 128;
                prev_carry = 0
            }

            if (curr_carry) {
                prev_carry = 1;
                curr_carry = 0;
            }

        }
        return X
    }
    /**
     * Special function to create 64 bit byte-array, which is containes
length of the original byte-array
     *
     * @param {number[]} A - Initial byte-array
     * @returns {number[]} The block of 64 bits, which is contains A element
length
     *
     * @private
     */
    static create64BitBlockLength(A) {
        let Block64BitLength = new Array(8);
        Block64BitLength = [0, 0, 0, 0, 0, 0, 0, A.length * 8]
```

```
        for (let i = 7; i > 0;) {
            if (Block64BitLength[i] === 0) break;

            while (Block64BitLength[i] > 256) {
                Block64BitLength[i] -= 256;
                Block64BitLength[i-1]++;
            }

            i--;
        }
        return Block64BitLength
    }

    /**
     * Encodes multi-byte string to utf8.
     *
     * Note utf8Encode is an identity function with 7-bit ascii strings, but
not with 8-bit strings;
     * utf8Encode('x') = 'x', but utf8Encode('ça') = 'Ã§a', and
utf8Encode('Ã§a') = 'Ãƒâ§a'.
     *
     * @private
     */
    static utf8Encode(str) {
        try {
            return new TextEncoder().encode(str, 'utf-8').reduce((prev, curr)
=> prev + String.fromCharCode(curr), '');
        } catch (e) { // no TextEncoder available?
            return unescape(encodeURIComponent(str)); //
monsur.hossa.in/2012/07/20/utf-8-in-javascript.html
        }
    }

    /**
     * Decodes utf8 string to multi-byte.
     *
     * @private
     */
    static utf8Decode(str) {
        try {
            return new TextEncoder().decode(str, 'utf-8').reduce((prev, curr)
=> prev + String.fromCharCode(curr), '');
        } catch (e) { // no TextEncoder available?
            return decodeURIComponent(escape(str)); //
monsur.hossa.in/2012/07/20/utf-8-in-javascript.html
        }
    }

    /**
     * Encodes string as base-64.
     *
```

```
     * - developer.mozilla.org/en-US/docs/Web/API/window.btoa,
nodejs.org/api/buffer.html
     * - note: btoa & Buffer/binary work on single-byte Unicode (C0/C1), so
ok for utf8 strings, not for general Unicode...
     * - note: if btoa()/atob() are not available (eg IE9-), try
github.com/davidchambers/Base64.js
     *
     * @private
     */
    static base64Encode(str) {
        if (typeof btoa != 'undefined') return btoa(str); // browser
        if (typeof Buffer != 'undefined') return new Buffer(str,
'binary').toString('base64'); // Node.js
        throw new Error('No Base64 Encode');
    }

    /**
     * Decodes base-64 encoded string.
     *
     * @private
     */
    static base64Decode(str) {
        if (typeof atob != 'undefined') return atob(str); // browser
        if (typeof Buffer != 'undefined') return new Buffer(str,
'base64').toString('binary'); // Node.js
        throw new Error('No Base64 Decode');
    }

}


/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - - - - - - - -  */
module.exports = AES_GCM_SHA256;
```

Joonis 10 AES_GCM_SHA256.js

```
class AES {

    /**
     * AES Cipher function: encrypt 'input' state with Rijndael algorithm
[§5.1];
     *   applies Nr rounds (10/12/14) using key schedule w for 'add round
key' stage.
     *
     * @param   {number[]}   input - 16-byte (128-bit) input state array.
     * @param   {number[][]} w - Key schedule as 2D byte-array (Nr+1 × Nb
bytes).
     * @returns {number[]}   Encrypted output state array.
     */
    static cipher(input, w) {
        const Nb = 4;               // block size (in words): no of columns
in state (fixed at 4 for AES)
        const Nr = w.length/Nb - 1; // no of rounds: 10/12/14 for
128/192/256-bit keys

        let state = [ [], [], [], [] ];  // initialise 4×Nb byte-array
'state' with input [§3.4]
        for (let i=0; i<4*Nb; i++) state[i%4][Math.floor(i/4)] = input[i];

        state = AES.addRoundKey(state, w, 0, Nb);

        for (let round=1; round<Nr; round++) {
            state = AES.subBytes(state, Nb);
            state = AES.shiftRows(state, Nb);
            state = AES.mixColumns(state, Nb);
            state = AES.addRoundKey(state, w, round, Nb);
        }

        state = AES.subBytes(state, Nb);
        state = AES.shiftRows(state, Nb);
        state = AES.addRoundKey(state, w, Nr, Nb);

        const output = new Array(4*Nb);  // convert state to 1-d array before
returning [§3.4]
        for (let i=0; i<4*Nb; i++) output[i] = state[i%4][Math.floor(i/4)];

        return output;
    }


    /**
     * Perform key expansion to generate a key schedule from a cipher key
[§5.2].
     *
     * @param   {number[]}   key - Cipher key as 16/24/32-byte array.
     * @returns {number[][]} Expanded key schedule as 2D byte-array (Nr+1 ×
Nb bytes).
     */
```

```
    static keyExpansion(key) {
        const Nb = 4;            // block size (in words): no of columns in
state (fixed at 4 for AES)
        const Nk = key.length/4; // key length (in words): 4/6/8 for
128/192/256-bit keys
        const Nr = Nk + 6;       // no of rounds: 10/12/14 for 128/192/256-
bit keys

        const w = new Array(Nb*(Nr+1));
        let temp = new Array(4);

        // initialise first Nk words of expanded key with cipher key
        for (let i=0; i<Nk; i++) {
            const r = [ key[4*i], key[4*i+1], key[4*i+2], key[4*i+3] ];
            w[i] = r;
        }

        // expand the key into the remainder of the schedule
        for (let i=Nk; i<(Nb*(Nr+1)); i++) {
            w[i] = new Array(4);
            for (let t=0; t<4; t++) temp[t] = w[i-1][t];
            // each Nk'th word has extra transformation
            if (i % Nk == 0) {
                temp = AES.subWord(AES.rotWord(temp));
                for (let t=0; t<4; t++) temp[t] ^= AES.rCon[i/Nk][t];
            }
            // 256-bit key has subWord applied every 4th word
            else if (Nk > 6 && i%Nk == 4) {
                temp = AES.subWord(temp);
            }
            // xor w[i] with w[i-1] and w[i-Nk]
            for (let t=0; t<4; t++) w[i][t] = w[i-Nk][t] ^ temp[t];
        }

        return w;
    }


    /**
     * Apply SBox to state S [§5.1.1].
     *
     * @private
     */
    static subBytes(s, Nb) {
        for (let r=0; r<4; r++) {
            for (let c=0; c<Nb; c++) s[r][c] = AES.sBox[s[r][c]];
        }
        return s;
    }
```

```
    /**
     * Shift row r of state S left by r bytes [§5.1.2].
     *
     * @private
     */
    static shiftRows(s, Nb) {
        const t = new Array(4);
        for (let r=1; r<4; r++) {
            for (let c=0; c<4; c++) t[c] = s[r][(c+r)%Nb];  // shift into
temp copy
            for (let c=0; c<4; c++) s[r][c] = t[c];         // and copy back
        }          // note that this will work for Nb=4,5,6, but not 7,8
(always 4 for AES):
        return s;  // see
asmAES.sourceforge.net/rijndael/rijndaelImplementation.pdf
    }


    /**
     * Combine bytes of each col of state S [§5.1.3].
     *
     * @private
     */
    static mixColumns(s, Nb) {
        for (let c=0; c<Nb; c++) {
            const a = new Array(Nb);  // 'a' is a copy of the current column
from 's'
            const b = new Array(Nb);  // 'b' is a•{02} in GF(2^8)
            for (let r=0; r<4; r++) {
                a[r] = s[r][c];
                b[r] = s[r][c]&0x80 ? s[r][c]<<1 ^ 0x011b : s[r][c]<<1;
            }
            // a[n] ^ b[n] is a•{03} in GF(2^8)
            s[0][c] = b[0] ^ a[1] ^ b[1] ^ a[2] ^ a[3]; // {02}•a0 + {03}•a1
+ a2 + a3
            s[1][c] = a[0] ^ b[1] ^ a[2] ^ b[2] ^ a[3]; // a0 • {02}•a1 +
{03}•a2 + a3
            s[2][c] = a[0] ^ a[1] ^ b[2] ^ a[3] ^ b[3]; // a0 + a1 + {02}•a2
+ {03}•a3
            s[3][c] = a[0] ^ b[0] ^ a[1] ^ a[2] ^ b[3]; // {03}•a0 + a1 + a2
+ {02}•a3
        }
        return s;
    }


    /**
     * Xor Round Key into state S [§5.1.4].
     *
     * @private
     */
    static addRoundKey(state, w, rnd, Nb) {
```

```
        for (let r=0; r<4; r++) {
            for (let c=0; c<Nb; c++) state[r][c] ^= w[rnd*4+c][r];
        }
        return state;
    }


    /**
     * Apply SBox to 4-byte word w.
     *
     * @private
     */
    static subWord(w) {
        for (let i=0; i<4; i++) w[i] = AES.sBox[w[i]];
        return w;
    }


    /**
     * Rotate 4-byte word w left by one byte.
     *
     * @private
     */
    static rotWord(w) {
        const tmp = w[0];
        for (let i=0; i<3; i++) w[i] = w[i+1];
        w[3] = tmp;
        return w;
    }


}


// sBox is pre-computed multiplicative inverse in GF(2^8) used in subBytes
and keyExpansion [§5.1.1]
AES.sBox = [
    0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01, 0x67, 0x2b,
0xfe, 0xd7, 0xab, 0x76,
    0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4, 0xa2, 0xaf,
0x9c, 0xa4, 0x72, 0xc0,
    0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5, 0xe5, 0xf1,
0x71, 0xd8, 0x31, 0x15,
    0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a, 0x07, 0x12, 0x80, 0xe2,
0xeb, 0x27, 0xb2, 0x75,
    0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0, 0x52, 0x3b, 0xd6, 0xb3,
0x29, 0xe3, 0x2f, 0x84,
    0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0xcb, 0xbe, 0x39,
0x4a, 0x4c, 0x58, 0xcf,
    0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45, 0xf9, 0x02, 0x7f,
0x50, 0x3c, 0x9f, 0xa8,
```

```
    0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5, 0xbc, 0xb6, 0xda, 0x21,
0x10, 0xff, 0xf3, 0xd2,
    0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4, 0xa7, 0x7e, 0x3d,
0x64, 0x5d, 0x19, 0x73,
    0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0xee, 0xb8, 0x14,
0xde, 0x5e, 0x0b, 0xdb,
    0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c, 0xc2, 0xd3, 0xac, 0x62,
0x91, 0x95, 0xe4, 0x79,
    0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x56, 0xf4, 0xea,
0x65, 0x7a, 0xae, 0x08,
    0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8, 0xdd, 0x74, 0x1f,
0x4b, 0xbd, 0x8b, 0x8a,
    0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e, 0x61, 0x35, 0x57, 0xb9,
0x86, 0xc1, 0x1d, 0x9e,
    0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94, 0x9b, 0x1e, 0x87, 0xe9,
0xce, 0x55, 0x28, 0xdf,
    0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99, 0x2d, 0x0f,
0xb0, 0x54, 0xbb, 0x16,
];


// rCon is Round Constant used for the Key Expansion [1st col is 2^(r-1) in
GF(2^8)] [§5.2]
AES.rCon = [
    [ 0x00, 0x00, 0x00, 0x00 ],
    [ 0x01, 0x00, 0x00, 0x00 ],
    [ 0x02, 0x00, 0x00, 0x00 ],
    [ 0x04, 0x00, 0x00, 0x00 ],
    [ 0x08, 0x00, 0x00, 0x00 ],
    [ 0x10, 0x00, 0x00, 0x00 ],
    [ 0x20, 0x00, 0x00, 0x00 ],
    [ 0x40, 0x00, 0x00, 0x00 ],
    [ 0x80, 0x00, 0x00, 0x00 ],
    [ 0x1b, 0x00, 0x00, 0x00 ],
    [ 0x36, 0x00, 0x00, 0x00 ],
];

module.exports = AES;
```

Joonis 11 AES.js [30]

```
class SHA256 {

    /**
     * Array of round constants:
     * (first 32 bits of the fractional parts of the cube roots of the first
64 primes 2...311)
     *
     * @param {number[]} K - array of round constants
     *
     * @privete
     */
    static K = [0x428A2F98, 0x71374491, 0xB5C0FBCF, 0xE9B5DBA5, 0x3956C25B,
0x59F111F1, 0x923F82A4, 0xAB1C5ED5,
                0xD807AA98, 0x12835B01, 0x243185BE, 0x550C7DC3, 0x72BE5D74,
0x80DEB1FE, 0x9BDC06A7, 0xC19BF174,
                0xE49B69C1, 0xEFBE4786, 0x0FC19DC6, 0x240CA1CC, 0x2DE92C6F,
0x4A7484AA, 0x5CB0A9DC, 0x76F988DA,
                0x983E5152, 0xA831C66D, 0xB00327C8, 0xBF597FC7, 0xC6E00BF3,
0xD5A79147, 0x06CA6351, 0x14292967,
                0x27B70A85, 0x2E1B2138, 0x4D2C6DFC, 0x53380D13, 0x650A7354,
0x766A0ABB, 0x81C2C92E, 0x92722C85,
                0xA2BFE8A1, 0xA81A664B, 0xC24B8B70, 0xC76C51A3, 0xD192E819,
0xD6990624, 0xF40E3585, 0x106AA070,
                0x19A4C116, 0x1E376C08, 0x2748774C, 0x34B0BCB5, 0x391C0CB3,
0x4ED8AA4A, 0x5B9CCA4F, 0x682E6FF3,
                0x748F82EE, 0x78A5636F, 0x84C87814, 0x8CC70208, 0x90BEFFFA,
0xA4506CEB, 0xBEF9A3F7, 0xC67178F2]



    /**
     * Initialize hash values:
     * (first 32 bits of the fractional parts of the square roots of the
first 8 primes 2...19):
     *
     * @param {number[]} H - Array of hash values
     *
     * @privete
     */
    static H = [0x6A09E667, 0xBB67AE85, 0x3C6EF372, 0xA54FF53A, 0x510E527F,
0x9B05688C, 0x1F83D9AB, 0x5BE0CD19]

    /**
     * Generates SHA-256 hash of string.
     *
     * @param   {string} msg - (Unicode) string to be hashed.
     * @returns {string} Hash of msg as hex character string.
     *
     * @example
     *   import Sha256 from './sha256.js';
     *   const hash = Sha256.hash('abc'); //
'ba7816bf8f01cfea414140de5dae2223b00361a396177a9cb410ff61f20015ad'
     */
```

```
    static encrypt(message) {
        //convert string to UTF-8, as SHA only deals with byte-streams
        let m = new TextEncoder().encode(message, 'utf-8').reduce((prev,
curr) => prev + String.fromCharCode(curr), '');

        let H = [...this.H];
        let K = [...this.K];

        // add trailing '1' bit (+ 0's padding) to string
        m += String.fromCharCode(0x80);

        // convert string msg into 512-bit blocks (array of 16 32-bit
integers)
        const l = m.length/4 + 2; // length (in 32-bit integers) of msg + ‘1’
+ appended length
        const N = Math.ceil(l/16);  // number of 16-integer (512-bit) blocks
required to hold 'l' ints
        const mArr = new Array(N);     // message mArr is N×16 array of 32-
bit integers

        for (let i=0; i<N; i++) {
            mArr[i] = new Array(16);
            for (let j=0; j<16; j++) { // encode 4 chars per integer (64 per
block), big-endian encoding
                mArr[i][j] = (m.charCodeAt(i*64+j*4+0)<<24) |
(m.charCodeAt(i*64+j*4+1)<<16)
                          | (m.charCodeAt(i*64+j*4+2)<< 8) |
(m.charCodeAt(i*64+j*4+3)<< 0);
            } // note running off the end of msg is ok 'cos bitwise ops on
NaN return 0
        }

        // add length (in bits) into final pair of 32-bit integers (big-
endian)
        // note: most significant word would be (len-1)*8 >>> 32, but since
JS converts
        // bitwise-op args to 32 bits, we need to simulate this by arithmetic
operators
        const lenHi = ((m.length-1)*8) / Math.pow(2, 32);
        const lenLo = ((m.length-1)*8) >>> 0;
        mArr[N-1][14] = Math.floor(lenHi);
        mArr[N-1][15] = lenLo;

        //Process the message in successive 512-bit chunks:
        for (let i=0; i<N; i++) {
            const W = new Array(64);

            //(The initial values in w[0..63] don't matter, so many
implementations zero them here)
            for (let t=0;  t<16; t++) W[t] = mArr[i][t];

            //Extend the first 16 words into the remaining 48 words w[16..63]
of the message schedule array:
```

```
            for (let t=16; t<64; t++) {
                W[t] = (SHA256.σ1(W[t-2]) + W[t-7] + SHA256.σ0(W[t-15]) +
W[t-16]) >>> 0;
            }

            //Initialize working variables to current hash value:
            let a = H[0], b = H[1], c = H[2], d = H[3], e = H[4], f = H[5], g
= H[6], h = H[7];

            //Compression function main loop:
            for (let t=0; t<64; t++) {
                const T1 = h + SHA256.Σ1(e) + SHA256.Ch(e, f, g) + K[t] +
W[t];
                const T2 =     SHA256.Σ0(a) + SHA256.Maj(a, b, c);
                h = g;
                g = f;
                f = e;
                e = (d + T1) >>> 0;
                d = c;
                c = b;
                b = a;
                a = (T1 + T2) >>> 0;
            }

            //Add the compressed chunk to the current hash value:
            H[0] = (H[0]+a) >>> 0;
            H[1] = (H[1]+b) >>> 0;
            H[2] = (H[2]+c) >>> 0;
            H[3] = (H[3]+d) >>> 0;
            H[4] = (H[4]+e) >>> 0;
            H[5] = (H[5]+f) >>> 0;
            H[6] = (H[6]+g) >>> 0;
            H[7] = (H[7]+h) >>> 0;
        }

        for (let h=0; h<H.length; h++) H[h] =
('00000000'+H[h].toString(16)).slice(-8);

        const separator = '';

        return H.join(separator);
    }

    /**
     * Rotates right (circular right shift) value x by n positions
     *
     * @param {number} n
     * @param {number} x
     * @returns {number}
     *
     * @private
```

```
     */
    static _ROTR = (n, x) => (x >>> n) | (x << (32-n));

    /**
     *
     * Logical function.
     * @private
     */
    static σ0 = x => SHA256._ROTR(7,  x) ^ SHA256._ROTR(18, x) ^ (x>>>3);
    /**
     *
     * Logical function.
     * @private
    */
    static σ1 = x => SHA256._ROTR(17, x) ^ SHA256._ROTR(19, x) ^ (x>>>10);
    /**
     *
     * Logical function.
     * @private
     */
    static Σ0 = x => SHA256._ROTR(2,  x) ^ SHA256._ROTR(13, x) ^
SHA256._ROTR(22, x);
    /**
     *
     * Logical function.
     * @private
     */
    static Σ1 = x => SHA256._ROTR(6,  x) ^ SHA256._ROTR(11, x) ^
SHA256._ROTR(25, x);
    /**
     *
     * Logical function.
     * @private
     */
    static Ch = (x, y, z) =>  (x & y) ^ (~x & z);
    /**
     *
     * Logical function.
     * @private
     */
    static Maj = (x, y, z) => (x & y) ^ (x & z) ^ (y & z);
}

module.exports = SHA256;
```

Joonis 12 SHA256.js [31]