Immagine che contiene disegnando, finestra, segnale

Descrizione generata automaticamente

Relazione Progetto: Studio di fattibilità per la definizione di tool crittografici riguardo un protocollo sicuro per il Preserving Contact Tracing

Immagine che contiene circuito

Descrizione generata automaticamente

Gruppo 30

Ludovico russo [0622700881]

Fabrizio lettieri [0622700882]

A.A. 2019 2020

Indice delle pagine

[1. Analisi del dominio di interesse e sviluppo dei task. 4](bookmark://_Toc45872633#_Toc45872633)

[2. Introduzione al progetto e studio dei tools 4](bookmark://_Toc45872634#_Toc45872634)

[3. Progettazione del protocollo 5](bookmark://_Toc45872635#_Toc45872635)

[3.1. Certificati 6](bookmark://_Toc45872636#_Toc45872636)

[4. Validità 6](bookmark://_Toc45872637#_Toc45872637)

[5. Implementazione di parti del protocollo 8](bookmark://_Toc45872638#_Toc45872638)

[5.1. Configurazione del web server apache. 8](bookmark://_Toc45872639#_Toc45872639)

[5.2. Creazione dei certificati con il tool OpenSSL. 14](bookmark://_Toc45872640#_Toc45872640)

[5.3. Generazione dei certificati client 21](bookmark://_Toc45872641#_Toc45872641)

[5.4. Implementazione del protocollo 22](bookmark://_Toc45872642#_Toc45872642)

**Indice delle figure**

[Figura 1 Definizione del protocollo per il rilascio dei codici monouso 3](#_Toc45870537)

[Figura 2 Albero delle C.A. 3](#_Toc45870538)

[Figura 3 Configurazione di SSL 16](#_Toc45870539)

[Figura 4 Screen delle cartelle per la generazione dei certificati 17](#_Toc45870540)

[Figura 5 Visualizzazione del certificato importato nel browser 18](#_Toc45870541)

[Figura 6 Accesso alla piattaforma tramite la selezione di un certificato client 19](#_Toc45870542)

[Figura 7 Accesso laboratorio analisi 19](#_Toc45870543)

[Figura 8 Pagina generacertificati 20](#_Toc45870544)

[Figura 9 Esempio di inserimento dei dati per la generazione di un certificato 20](#_Toc45870545)

[Figura 10 Creazione del certificato mediante lo script genssl.php 21](#_Toc45870546)

[Figura 11 Contenuto della directory client\_certificate 21](#_Toc45870547)

[Figura 12 Esempio della generazione di un codice monouso paziente positivo 23](#_Toc45870548)

[Figura 13 Generazione di esito esame negativo 24](#_Toc45870549)

**Indice delle configurazioni**

[Configurazione di Apache: 1 Edit del file ports.conf 5](#_Toc45871092)

[Configurazione di Apache: 2 Edit del file defaultssl.conf 7](#_Toc45871093)

[Configurazione di Apache: 3 Edit del file /etc/hosts 8](#_Toc45871094)

[Configurazione di Apache: 4 Edit del file apache2.conf 11](#_Toc45871095)

**Indice dei Comandi OpenSSL**

[Comando di OpenSSL 1 Edit del file di configurazione openssl.cnf 17](#_Toc45871122)

[Comando di OpenSSL 2 Definizione del comando per la richiesta del certificato rootCA 18](#_Toc45871123)

[Comando di OpenSSL 3: Comando per la richiesta di un certificato server 19](#_Toc45871124)

[Comando di OpenSSL 4: Comando per eseguire la firma di un certificato da parte della C.A. 19](#_Toc45871125)

[Comando di OpenSSL 5 Comando per l' esporazione del file binario del certificato 19](#_Toc45871126)

[Comando di OpenSSL 6 richiesta di un certificato client 19](#_Toc45871127)

[Comando di OpenSSL 7 Commando per la richiesta di firma alla CA 20](#_Toc45871128)

[Comando di OpenSSL 8 Comando per esportare il certificato pkcs12 con estensione p12 20](#_Toc45871129)

**Indice dell’implementazione**

[Codice php 1 genssl.php 23](#_Toc45871157)

[Codice php 2 gencode.php 24](#_Toc45871158)

# Analisi del dominio di interesse e sviluppo dei task.

Il tema del progetto è stato sviluppato sulla realizzazione di uno studio di fattibilità riguardante l'utilizzo dei tool crittografici per la protezione dei dati dei cittadini, con la possibilità di definire nuovi protocolli di comunicazione (supposizione, definizione di un nuovo protocollo o l'utilizzo di protocolli già presenti in letteratura) per tali dati (comunicati su un canale pubblico). In particolare lo studio si concentrerà sulla comunicazione tra il laboratorio analisi e il cittadino infetto, che comunicherà di essere infetto o meno al server (o un gruppo di essi).

Le entità coinvolte:

* Cittadino non infetto: è un cittadino al quale al momento non è arrivata comunicazione di essere infetto.
* Cittadino infetto: è un cittadino al quale è già arrivata la comunicazione di essere infetto.
* Laboratorio analisi: è il posto in cui può recarsi un cittadino per effettuare dei test e verificare la positività al virus.
* Cittadino malintenzionato: è il cittadino che vuole conoscere i dati del cittadino infetto, violando la sua privacy.
* Server governativo: Punto di raccolta dei dati delle persone riconosciute come infette. (Si considererà in una seconda analisi come l'utilizzo di un sistema centralizzato o decentralizzato possa inficiare sulla raccolta dei dati).

Gli obiettivi prefissati dal progetto sono stati quelli di:

Definire un protocollo di comunicazione sicuro tra il laboratorio analisi, il cittadino infetto e server, in modo da evitare che un cittadino malintenzionato possa venire a conoscenza dei dati di un presunto infetto, oppure cerca di inviare delle chiavi al server, senza conoscere il codice attribuitogli dal laboratorio analisi.

Task previsti:

1. Studio dei tool crittografici da utilizzare per capire quale schema di sicurezza adottare (o una composizione di essi).
2. Progettazione di un protocollo per la comunicazione dei dati tra il paziente infetto e il server, mediante un codice che il cittadino utilizza per comunicare la sua positività.
3. Analisi della validità dello schema progettato al punto 2)
4. Implementazione di parti del protocollo (si deciderà in corso d'opera dopo il completamento del primo task richiesto)

# Introduzione al progetto e studio dei tools

Il progetto svolto riguarda uno studio di fattibilità di un protocollo sicuro per il Preserving Contact Tracing con l’utilizzo di tool crittografici per la protezione dei dati dei cittadini, con la possibilità di definire nuovi protocolli di comunicazione per tali dati. In particolare, lo studio si concentra sulla comunicazione tra le tre entità: il laboratorio analisi, il cittadino infetto e il server. La Figura 1 espone la modifica del protocollo DP3T specializzato al caso di studio in esame.

La modifica sostanziale del protocollo è basata sull’aggiunta dell’entità dei laboratori analisi, i quali per accedere al backend ministeriale hanno bisogno di certificati firmati da rootCA (cioè il ministero della salute); essi richiedono il codice randomico per il cittadino risultante infetto, che viene restituito dal backend tramite un codice monouso non utilizzato. il laboratorio analisi consegna il codice al paziente infetto, allegandolo al referto, senza tenere traccia del codice, il quale viene stampato dal backend ministeriale; il paziente infetto invia il proprio codice al server per indicargli che esso è stato utilizzato.

Lo schema principale del protocollo si basa su una crittografia a chiave pubblica con firma digitale (***Gen,Sign,Vrfy***):

1. L’ algoritmo di generazione della chiave ***Gen***, prende come input un parametro sicuro 1n e genera in output una coppia di chiavi (*pk,sk*). La chiave *pk* è chiamata ***chiave pubblica***mentre la *sk* è chiamata ***chiave privata***. Noi assumiamo che *pk e sk* abbiano la stessa lunghezza n e che n può essere determinato da *pk* o *sk*.
2. L’ algoritmo di firma ***Sign*** prende come input una chiave privata *sk* e un messaggio m dallo spazio dei messaggi (che è indipendente da *pk*). Il suo output è una firma , infatti
3. L’ algoritmo di verifica deterministico chiamato ***Vrfy*** prende in input una chiave pubblica *pk* e un messaggio m e la firma . Il suo output sarà un bit b, quando *b=1* significa che il messaggio è valido, se *b=0* questo risulta essere invalido. Possiamo denotare questo come

La scelta di tale tool è stata fatta sull’analisi sia di costi computazionali e sia sugli attacchi possibili tipo il MIM attacks. Si suppone che il ministero della salute sia una rootCA, la quale può rilasciare certificati client ai laboratori analisi; in questo caso il laboratorio esegue l’accesso al server backend del ministero, il quale rilascia un nuovo codice. Tali codici sono ritenuti validi dal server perchè sono salvati sul database contenuto sul backend.

# Progettazione del protocollo

Il protocollo è stato sviluppato seguendo tale studio: l’ente laboratorio analisi si comporta come un client che, dopo aver richiesto un certificato e quindi garantito di essere un ente sicuro, invia al server un messaggio contenente la positività del paziente che ha effettuato il tampone nel suddetto laboratorio; il server risponde a tale messaggio inviando dal suo backend un codice usa e getta e univoco al laboratorio. La generazione di tale codice sfrutta la proprietà di forte crittograficamente, calcolando la funzione hash secondo l’algoritmo di SHA512 e una funzione randomica (crittograficamente forte) di numeri interi lunghi 128 bit. Questa scelta di progettazione rende molto difficile la rottura dello schema, in quanto la probabilità che un possibile avversario *PPT* A possa vincere nell’indovinare tale codice è negligibile, inoltre costringe il medesimo avversario a dover applicare un brute-force attack molto dispendioso e quasi impossibile con risorse limitate. La funzione hash è resistente alle collisioni poiché concatena il codice generato in maniera randomica e il timestamp generato all’atto della richiesta del laboratorio; in questo modo risulta difficile all’avversario poter trovare le collisioni all’interno del backend, quindi si garantisce la proprietà di resistenza alle collisioni. Il laboratorio, ottenuto il codice monouso, lo consegna al cittadino tramite la generazione di un pdf adibito a salvare l’esito e il codice nel caso egli sia positivo all’esame. Il codice consegnato rimane attivo, quindi segnalato come utilizzato nel backend, fino a quando il cittadino non risulta più infetto (parte non implementata ). Lo schema progettato può essere visto nella figura sottostante:

­Immagine che contiene testo

Descrizione generata automaticamente

Send Cod Monouso

Release Cod Monouso

Send positività

Laboratorio

Analisi

Cittadino

(Infetto)

Back-End

(SKt,t,Cod\_Monouso)

Figura 1 Definizione del protocollo per il rilascio dei codici monouso

Il nostro lavoro di sviluppo si è concentrato sulla comunicazione critica tra laboratorio analisi e back-end proprio per evitare il MIM attack, molto dannoso e pericoloso per il protocollo sviluppato. Nei paragrafi successivi sono presentati gli strumenti utilizzati nella progettazione.

## Certificati

L’ utilizzo dei certificati, ci ha permesso di eseguire una comunicazione sicura tra client (laboratori analisi) e la backend ministeriale tramite l’ utilizzo del protocollo TLS/SSL. L’ albero delle C.A. è stato definito nella Figura 2. Il ministero rilascia i certificati client a ciascun laboratorio analisi. Il laboratorio può generare i certificati che servono per gli altri operatori, oppure generare altri certificati ai laboratori analisi.

Figura 2 Albero delle C.A.

# Validità

La validità dello schema si basa su due dimostrazioni: la sicurezza della generazione del certificato secondo l’algoritmo *GenRSA e* la sicurezza rispetto al *MIM attack* garantita dai certificati sicuri.

Consideriamo ***GenRSA*** essere un algoritmo *PPT* che, su un input 1n, genera in uscita un modulo N che è il prodotto di due numeri primi a n bit (eccetto con probabilità negligibile), insieme a due numeri interi *e,* *d* che soddisfano il prodotto *ed=1 mod Ø (N)*. La generazione della chiave impiega semplicemente tale algoritmo e genera la tupla <*N,e*> come la chiave pubblica e *<N,d>* come la chiave privata. Per firmare un messaggio *m* ∈ Z\*N, il firmatario calcola *σ = [md mod N]*. La verifica della firma *σ* su un messaggio m rispetto alla chiave pubblica *<N,e>* è data dal controllo su se *m* è proprio uguale a *σe* mod *N*. l’algoritmo è formato dai seguenti passi*:*

* *Gen:* su un input 1n performa ***GenRSA***(1n)per ottenere la tripla (*N,e,d*). La chiave pubblica è *<N,e>* e la chiave privata è *<N,d>.*
* *Sign:* sull’input una chiave privata *sk* = *<N,d>* e un messaggio *m* ∈ Z\*N, calcola la firma *σ = [md mod N].*
* *Vrfy:* sull’input una chiave pubblica *pk* = *<N,e>*, un messaggio *m* ∈ Z\*N e una firma *σ* ∈ Z\*N, genera 1 se e solo se *m=?=[σe* mod *N]*

È facile da vedere che la verifica di una firma generata legittimamente è sempre fatta con successo poiché

*σe* = *(md)e* = *m[ed=1 mod Ø(N)]* = *m1* = *m mod N*

Questo schema potrebbe essere ritenuto sicuro perché un possibile avversario che conosce solo la chiave pubblica *<N,e>*, calcolando una firma valida su un messaggio *m*, deve risolvere il problema dello RSA (in quanto la firma è esattamente la radice e-esima di *m*). In realtà tale schema è vulnerabile ad un no-messagge attack e alla manomissione tramite messaggio arbitrario poiché si basa solo sull’uniformità del messaggio e non tratta la sua difformità o la sua arbitrarietà (secondo una scelta dell’avversario). Per tentare di prevenire tali attacchi, bisogna pensare ad una certa trasformazione dei messaggi prima di firmarli. In questo caso, un firmatario specifica una funzione deterministica *H* come parte della chiave pubblica del messaggio; tale funzione contiene certe proprietà crittografiche sulla mappatura dei messaggi nel gruppo Z\*N, quindi la firma del messaggio diventa *σ=[H(m)d*mod *N]* e la verifica della firma sul messaggio è definita con il controllo di *σe=?H(m)*. Tale idea prende nome di ***schema di firma RSA-FDH***. Questa idea è utilizzata nello standard ***RSA PKCS #1 v2.1*** che può essere visto come una variante dello schema sopra citato; tale standard definisce uno schema in cui la firma su un messaggio dipende da un sale (un valore randomico) scelto dal firmatario appena deve generare la firma. Se questo sale è fissato a NULL dal firmatario (ciò è permesso dallo standard), lo schema risultante è molto simile allo RSA-FDH. Nella nostra implementazione, l’utilizzo di tale standard è stato cruciale nel generare i certificati client e server e garantisce che un avversario qualsiasi, anche con una possibile manomissione di tali certificati, non può rompere lo schema in quanto la funzione *H* è una funzione hash resistente alle collisioni e deterministica; l’unico modo per vincere tale schema è di performare un brute-force attack che impiega moltissime risorse e moltissimo tempo. Tale standard è presente nella libreria di OpenSSL ed è chiamato nel bash con la parola chiave –*rsa:[numero di bit]*.

Nel MIM attack, l’avversario si inserisce nella comunicazione e può generare e scambiare dati tra un utente e l’altro della connessione senza essere scoperto da entrambi, addirittura venendo considerato da entrambi gli utenti come la controparte affidabile con cui stanno scambiando reciprocamente dati sensibili e informazioni. Il protocollo sviluppato è valido su tale attacco in quanto è garantito dalla generazione dei certificati sicura, effettuata secondo l’infrastruttura a chiave pubblica PKI e la connessione sicura di HTTP.

# Implementazione di parti del protocollo

L’ implementazione del protocollo si è concentrato sullo sviluppo di una piattaforma web per il rilascio dei codici monouso. La prima cosa che è stata fatta, è stata installare sul sistema operativo Ubuntu 20.04 LTS un LAMP server (cioè installare i demoni Apache MySQL PHP/Python/Perl) ed è stato anche installato l’applicativo *PhpMyAdmin* per l’accesso al database.

Come linguaggio di sviluppo della back end è stato scelto PHP.

## Configurazione del web server apache.

La configurazione del web server è avvenuta configurando i seguenti file nella cartella /ect/apache2. Per prima cosa è stato disabilitato l’ascolto sulla porta 80 del server web. Per fare questo è stato modificato il file ***ports.conf***.Si riporta il listato del file:

# If you just change the port or add more ports here, you will likely also

# have to change the VirtualHost statement in

# /etc/apache2/sites-enabled/000-default.conf

#Listen 80

<IfModule ssl\_module>

Listen 443

</IfModule>

<IfModule mod\_gnutls.c>

Listen 443

</IfModule>

# vim: syntax=apache ts=4 sw=4 sts=4 sr noet

Configurazione di Apache: 1 Edit del file ports.conf

L’ utente che vuole accedere al web server, deve farlo per forza tramite l’utilizzo del protocollo TLS/SSL. Questa scelta è stata fatta perché, un utente generico che voleva accedere al server se utilizzava il protocollo http, lo faceva senza che il server richiedeste il certificato di accesso.

Sono state apportate delle modifiche al file ***default-ssl.conf***.

<IfModule mod\_ssl.c>

<VirtualHost \_default\_:443>

#ServerAdmin webmaster@localhost

ServerName covidpositivi.gov.it

DocumentRoot /var/www/html

# Available loglevels: trace8, ..., trace1, debug, info, notice, warn,

# error, crit, alert, emerg.

# It is also possible to configure the loglevel for particular

# modules, e.g.

#LogLevel info ssl:warn

ErrorLog ${APACHE\_LOG\_DIR}/error.log

CustomLog ${APACHE\_LOG\_DIR}/access.log combined

# For most configuration files from conf-available/, which are

# enabled or disabled at a global level, it is possible to

# include a line for only one particular virtual host. For example the

# following line enables the CGI configuration for this host only

# after it has been globally disabled with "a2disconf".

#Include conf-available/serve-cgi-bin.conf

# SSL Engine Switch:

# Enable/Disable SSL for this virtual host.

SSLEngine on

# A self-signed (snakeoil) certificate can be created by installing

# the ssl-cert package. See

# /usr/share/doc/apache2/README.Debian.gz for more info.

# If both key and certificate are stored in the same file, only the

# SSLCertificateFile directive is needed.

#SSLCertificateFile /etc/ssl/certs/ssl-cert-snakeoil.pem

#SSLCertificateKeyFile /etc/ssl/private/ssl-cert-snakeoil.key

SSLCertificateFile /home/ludovicorusso/certificate/mycert.pem

SSLCertificateKeyFile /home/ludovicorusso/certificate/mykey.pem

# Server Certificate Chain:

# Point SSLCertificateChainFile at a file containing the

# concatenation of PEM encoded CA certificates which form the

# certificate chain for the server certificate. Alternatively

# the referenced file can be the same as SSLCertificateFile

# when the CA certificates are directly appended to the server

# certificate for convenience.

#SSLCertificateChainFile /etc/apache2/ssl.crt/server-ca.crt

# Certificate Authority (CA):

# Set the CA certificate verification path where to find CA

# certificates for client authentication or alternatively one

# huge file containing all of them (file must be PEM encoded)

# Note: Inside SSLCACertificatePath you need hash symlinks

# to point to the certificate files. Use the provided

# Makefile to update the hash symlinks after changes.

#SSLCACertificatePath /etc/ssl/certs/

SSLCACertificateFile /etc/ssl/certs/CAcovid.pem

# Certificate Revocation Lists (CRL):

# Set the CA revocation path where to find CA CRLs for client

# authentication or alternatively one huge file containing all

# of them (file must be PEM encoded)

# Note: Inside SSLCARevocationPath you need hash symlinks

# to point to the certificate files. Use the provided

# Makefile to update the hash symlinks after changes.

#SSLCARevocationPath /etc/apache2/ssl.crl/

#SSLCARevocationFile /etc/apache2/ssl.crl/ca-bundle.crl

# Client Authentication (Type):

# Client certificate verification type and depth. Types are

# none, optional, require and optional\_no\_ca. Depth is a

# number which specifies how deeply to verify the certificate

# issuer chain before deciding the certificate is not valid.

#SSLVerifyClient require

#SSLVerifyDepth 10

# SSL Engine Options:

# Set various options for the SSL engine.

# o FakeBasicAuth:

# Translate the client X.509 into a Basic Authorisation. This means that

# the standard Auth/DBMAuth methods can be used for access control. The

# user name is the `one line' version of the client's X.509 certificate.

# Note that no password is obtained from the user. Every entry in the user

# file needs this password: `xxj31ZMTZzkVA'.

# o ExportCertData:

# This exports two additional environment variables: SSL\_CLIENT\_CERT and

# SSL\_SERVER\_CERT. These contain the PEM-encoded certificates of the

# server (always existing) and the client (only existing when client

# authentication is used). This can be used to import the certificates

# into CGI scripts.

# o StdEnvVars:

# This exports the standard SSL/TLS related `SSL\_\*' environment variables.

# Per default this exportation is switched off for performance reasons,

# because the extraction step is an expensive operation and is usually

# useless for serving static content. So one usually enables the

# exportation for CGI and SSI requests only.

# o OptRenegotiate:

# This enables optimized SSL connection renegotiation handling when SSL

# directives are used in per-directory context.

#SSLOptions +FakeBasicAuth +ExportCertData +StrictRequire

<FilesMatch "\.(cgi|shtml|phtml|php)$">

SSLOptions +StdEnvVars

</FilesMatch>

#<Directory /usr/lib/cgi-bin>

# SSLOptions +StdEnvVars

#</Directory>

<Directory "/var/www/html/secret/">

SSLVerifyClient require

SSLOptions +StdEnvVars +ExportCertData

</Directory>

# SSL Protocol Adjustments:

# The safe and default but still SSL/TLS standard compliant shutdown

# approach is that mod\_ssl sends the close notify alert but doesn't wait for

# the close notify alert from client. When you need a different shutdown

# approach you can use one of the following variables:

# o ssl-unclean-shutdown:

# This forces an unclean shutdown when the connection is closed, i.e. no

# SSL close notify alert is send or allowed to received. This violates

# the SSL/TLS standard but is needed for some brain-dead browsers. Use

# this when you receive I/O errors because of the standard approach where

# mod\_ssl sends the close notify alert.

# o ssl-accurate-shutdown:

# This forces an accurate shutdown when the connection is closed, i.e. a

# SSL close notify alert is send and mod\_ssl waits for the close notify

# alert of the client. This is 100% SSL/TLS standard compliant, but in

# practice often causes hanging connections with brain-dead browsers. Use

# this only for browsers where you know that their SSL implementation

# works correctly.

# Notice: Most problems of broken clients are also related to the HTTP

# keep-alive facility, so you usually additionally want to disable

# keep-alive for those clients, too. Use variable "nokeepalive" for this.

# Similarly, one has to force some clients to use HTTP/1.0 to workaround

# their broken HTTP/1.1 implementation. Use variables "downgrade-1.0" and

# "force-response-1.0" for this.

# BrowserMatch "MSIE [2-6]" \

# nokeepalive ssl-unclean-shutdown \

# downgrade-1.0 force-response-1.0

</VirtualHost>

</IfModule>

Configurazione di Apache: 2 Edit del file defaultssl.conf

Dopo aver abilitato l’engine di SSL, sono inseriti la direttiva *ServerName* per indicare il nome del server e *DocumentRoot* per indicare la directory dove il server va a consultare i file.

Le direttive *SSLCertificateFile* e *SSLCertificateKeyFile* contengono rispettivamente i path del file del certificato e i path della chiave privata, mentre nella directory /var/www/html/secret è stata inserita la richiesta di certificato client per poter accedere alla directory; la direttiva che ha permesso ciò è: *SSLVerifyClient require*.

La direttiva di apache *SSLCACertificateFile* permette di indicare dove si trova il file del certificato della rootCA. È stato inserito un link simbolico nel path /etc/ssl/certs/CAcovid.pem che contiene il percorso del file della CA.

Infine, è stato modificato il file *etc/hosts* in modo che il server risponda con l’indirizzo covidpositivi.gov.it si riporta il file:

127.0.0.1 localhost

127.0.1.1 ubuntu

127.0.0.1 www.covidpositivi.gov.it

127.0.0.1 covidpositivi.gov.it

# The following lines are desirable for IPv6 capable hosts

::1 ip6-localhost ip6-loopback

fe00::0 ip6-localnet

ff00::0 ip6-mcastprefix

ff02::1 ip6-allnodes

ff02::2 ip6-allrouters

Configurazione di Apache: 3 Edit del file /etc/hosts

Vengono aggiunti al file le due righe evidenziate in rosso.

Un ultimo step di setting è stato quello di non far eseguire l’accesso ad alcune particolari cartelle al web server, e soprattutto ad alcuni file presenti in esso, modificando il file *apache.conf* presente nella directory /ect/apache2/. Si riporta il listato del file:

ServerName covidpositivi.gov.it

DefaultRuntimeDir ${APACHE\_RUN\_DIR}

#

# PidFile: The file in which the server should record its process

# identification number when it starts.

# This needs to be set in /etc/apache2/envvars

#

PidFile ${APACHE\_PID\_FILE}

#

# Timeout: The number of seconds before receives and sends time out.

#

Timeout 300

#

# KeepAlive: Whether or not to allow persistent connections (more than

# one request per connection). Set to "Off" to deactivate.

#

KeepAlive On

#

# MaxKeepAliveRequests: The maximum number of requests to allow

# during a persistent connection. Set to 0 to allow an unlimited amount.

# We recommend you leave this number high, for maximum performance.

#

MaxKeepAliveRequests 100

#

# KeepAliveTimeout: Number of seconds to wait for the next request from the

# same client on the same connection.

#

KeepAliveTimeout 5

# These need to be set in /etc/apache2/envvars

User ${APACHE\_RUN\_USER}

Group ${APACHE\_RUN\_GROUP}

#

# HostnameLookups: Log the names of clients or just their IP addresses

# e.g., www.apache.org (on) or 204.62.129.132 (off).

# The default is off because it'd be overall better for the net if people

# had to knowingly turn this feature on, since enabling it means that

# each client request will result in AT LEAST one lookup request to the

# nameserver.

#

HostnameLookups Off

# ErrorLog: The location of the error log file.

# If you do not specify an ErrorLog directive within a <VirtualHost>

# container, error messages relating to that virtual host will be

# logged here. If you \*do\* define an error logfile for a <VirtualHost>

# container, that host's errors will be logged there and not here.

#

ErrorLog ${APACHE\_LOG\_DIR}/error.log

#

# LogLevel: Control the severity of messages logged to the error\_log.

# Available values: trace8, ..., trace1, debug, info, notice, warn,

# error, crit, alert, emerg.

# It is also possible to configure the log level for particular modules, e.g.

# "LogLevel info ssl:warn"

#

LogLevel warn

# Include module configuration:

IncludeOptional mods-enabled/\*.load

IncludeOptional mods-enabled/\*.conf

# Include list of ports to listen on

Include ports.conf

# Sets the default security model of the Apache2 HTTPD server. It does

# not allow access to the root filesystem outside of /usr/share and /var/www.

# The former is used by web applications packaged in Debian,

# the latter may be used for local directories served by the web server. If

# your system is serving content from a sub-directory in /srv you must allow

# access here, or in any related virtual host.

<Directory />

Options FollowSymLinks

AllowOverride None

Require all denied

</Directory>

<Directory /usr/share>

AllowOverride None

Require all granted

</Directory>

<Directory /var/www/>

Options Indexes FollowSymLinks

AllowOverride None

Require all granted

</Directory>

<Directory /var/www/html/secret/demoCA/>

Options -Indexes +FollowSymLinks

AllowOverride None

Require all granted

<FilesMatch "cacert.pem|index.txt|index.txt.attr|index.txt.attr.old|index.txt.old|serial|serial.old">

Require all denied

</FilesMatch>

</Directory>

<Directory /var/www/html/secret/demoCA/private/>

Options -Indexes +FollowSymLinks

AllowOverride None

Require all granted

<FilesMatch "cakey.pem">

Require all denied

</FilesMatch>

</Directory>

<Directory /var/www/html/secret/demoCA/newcerts/>

Options -Indexes +FollowSymLinks

AllowOverride None

Require all granted

<FilesMatch "\.pem">

Require all denied

</FilesMatch>

</Directory>

<Directory /var/www/html/secret/certificate/>

Options -Indexes +FollowSymLinks

AllowOverride None

Require all granted

<FilesMatch "opensslclient.cnf">

Require all denied

</FilesMatch>

</Directory>

#<Directory /srv/>

# Options Indexes FollowSymLinks

# AllowOverride None

# Require all granted

#</Directory>

# AccessFileName: The name of the file to look for in each directory

# for additional configuration directives. See also the AllowOverride

# directive.

#

AccessFileName .htaccess

#

# The following lines prevent .htaccess and .htpasswd files from being

# viewed by Web clients.

#

<FilesMatch "^\.ht">

Require all denied

</FilesMatch>

#

# The following directives define some format nicknames for use with

# a CustomLog directive.

#

# These deviate from the Common Log Format definitions in that they use %O

# (the actual bytes sent including headers) instead of %b (the size of the

# requested file), because the latter makes it impossible to detect partial

# requests.

#

# Note that the use of %{X-Forwarded-For}i instead of %h is not recommended.

# Use mod\_remoteip instead.

#

LogFormat "%v:%p %h %l %u %t \"%r\" %>s %O \"%{Referer}i\" \"%{User-Agent}i\"" vhost\_combined

LogFormat "%h %l %u %t \"%r\" %>s %O \"%{Referer}i\" \"%{User-Agent}i\"" combined

LogFormat "%h %l %u %t \"%r\" %>s %O" common

LogFormat "%{Referer}i -> %U" referer

LogFormat "%{User-agent}i" agent

# Include of directories ignores editors' and dpkg's backup files,

# see README.Debian for details.

# Include generic snippets of statements

IncludeOptional conf-enabled/\*.conf

# Include the virtual host configurations:

IncludeOptional sites-enabled/\*.conf

Configurazione di Apache: 4 Edit del file apache2.conf

Sono state aggiunte le direttive <*Directory path*>, dove sono state settate le opzioni -index per evitare l’accesso all’indice dei file; con la <*Files Match regex*> è stata definita un’espressione regolare che applica la regola *Require all denited* riconoscendo l’espressione stessa, il che permette di non far eseguire l’accesso al client a questi file.

## Creazione dei certificati con il tool OpenSSL.

La creazione della CA è stata implementata mediante il tool ***OpenSSL***, modificando il file *openssl.conf* presente in etc/ssl/openssl.cnf. Le modifiche eseguite nel file sono evidenziate dal colore rosso.

# OpenSSL example configuration file.

# This is mostly being used for generation of certificate requests.

# Note that you can include other files from the main configuration

# file using the.include directive.

#.include filename

# This definition stops the following lines choking if HOME isn't

# defined.

HOME = .

RANDFILE = $ENV::HOME/.rnd

# Extra OBJECT IDENTIFIER info:

#oid\_file = $ENV::HOME/.oid

oid\_section = new\_oids

# To use this configuration file with the "-extfile" option of the

# "openssl x509" utility, name here the section containing the

# X.509v3 extensions to use:

# extensions =

# (Alternatively, use a configuration file that has only

# X.509v3 extensions in its main [= default] section.)

[ new\_oids ]

# We can add new OIDs in here for use by 'ca', 'req' and 'ts'.

# Add a simple OID like this:

# testoid1=1.2.3.4

# Or use config file substitution like this:

# testoid2=${testoid1}.5.6

# Policies used by the TSA examples.

tsa\_policy1 = 1.2.3.4.1

tsa\_policy2 = 1.2.3.4.5.6

tsa\_policy3 = 1.2.3.4.5.7

####################################################################

[ ca ]

default\_ca = CA\_default # The default ca section

####################################################################

[ CA\_default ]

dir = ./demoCA # Where everything is kept

certs = $dir/certs # Where the issued certs are kept

crl\_dir = $dir/crl # Where the issued crl are kept

database = $dir/index.txt # database index file.

#unique\_subject = no # Set to 'no' to allow creation of

# several certs with same subject.

new\_certs\_dir = $dir/newcerts # default place for new certs.

certificate = $dir/cacert.pem # The CA certificate

serial = $dir/serial # The current serial number

crlnumber = $dir/crlnumber # the current crl number

# must be commented out to leave a V1 CRL

crl = $dir/crl.pem # The current CRL

private\_key = $dir/private/cakey.pem# The private key

RANDFILE = $dir/private/.rand # private random number file

x509\_extensions = usr\_cert # The extensions to add to the cert

# Comment out the following two lines for the "traditional"

# (and highly broken) format.

name\_opt = ca\_default # Subject Name options

cert\_opt = ca\_default # Certificate field options

# Extension copying option: use with caution.

copy\_extensions = copy

# Extensions to add to a CRL. Note: Netscape communicator chokes on V2 CRLs

# so this is commented out by default to leave a V1 CRL.

# crlnumber must also be commented out to leave a V1 CRL.

# crl\_extensions = crl\_ext

default\_days = 365 # how long to certify for

default\_crl\_days= 30 # how long before next CRL

default\_md = default # use public key default MD

preserve = no # keep passed DN ordering

# A few difference way of specifying how similar the request should look

# For type CA, the listed attributes must be the same, and the optional

# and supplied fields are just that :-)

policy = policy\_match

# For the CA policy

[ policy\_match ]

countryName = match

stateOrProvinceName = match

organizationName = match

organizationalUnitName = optional

commonName = supplied

emailAddress = optional

# For the 'anything' policy

# At this point in time, you must list all acceptable 'object'

# types.

[ policy\_anything ]

countryName = optional

stateOrProvinceName = optional

localityName = optional

organizationName = optional

organizationalUnitName = optional

commonName = supplied

emailAddress = optional

####################################################################

[ req ]

default\_bits = 2048

default\_keyfile = privkey.pem

distinguished\_name = req\_distinguished\_name

attributes = req\_attributes

x509\_extensions = v3\_ca # The extensions to add to the self signed cert

# Passwords for private keys if not present they will be prompted for

# input\_password = secret

# output\_password = secret

# This sets a mask for permitted string types. There are several options.

# default: PrintableString, T61String, BMPString.

# pkix : PrintableString, BMPString (PKIX recommendation before 2004)

# utf8only: only UTF8Strings (PKIX recommendation after 2004).

# nombstr : PrintableString, T61String (no BMPStrings or UTF8Strings).

# MASK:XXXX a literal mask value.

# WARNING: ancient versions of Netscape crash on BMPStrings or UTF8Strings.

string\_mask = utf8only

req\_extensions = v3\_req #The extensions to add to a certificate request

[ req\_distinguished\_name ]

countryName = Country Name (2 letter code)

countryName\_default = AU

countryName\_min = 2

countryName\_max = 2

stateOrProvinceName = State or Province Name (full name)

stateOrProvinceName\_default = Some-State

localityName = Locality Name (eg, city)

0.organizationName = Organization Name (eg, company)

0.organizationName\_default = Internet Widgits Pty Ltd

# we can do this but it is not needed normally :-)

#1.organizationName = Second Organization Name (eg, company)

#1.organizationName\_default = World Wide Web Pty Ltd

organizationalUnitName = Organizational Unit Name (eg, section)

#organizationalUnitName\_default =

commonName = Common Name (e.g. server FQDN or YOUR name)

commonName\_max = 64

emailAddress = Email Address

emailAddress\_max = 64

# SET-ex3 = SET extension number 3

[ req\_attributes ]

challengePassword = A challenge password

challengePassword\_min = 4

challengePassword\_max = 20

unstructuredName = An optional company name

[ usr\_cert ]

# These extensions are added when 'ca' signs a request.

# This goes against PKIX guidelines but some CAs do it and some software

# requires this to avoid interpreting an end user certificate as a CA.

basicConstraints=CA:FALSE

# Here are some examples of the usage of nsCertType. If it is omitted

# the certificate can be used for anything \*except\* object signing.

# This is OK for an SSL server.

nsCertType = server

# For an object signing certificate this would be used.

# nsCertType = objsign

# For normal client use this is typical

# nsCertType = client, email

# and for everything including object signing:

# nsCertType = client, email, objsign

# This is typical in keyUsage for a client certificate.

keyUsage = nonRepudiation, digitalSignature, keyEncipherment

# This will be displayed in Netscape's comment listbox.

nsComment = "OpenSSL Generated Certificate"

# PKIX recommendations harmless if included in all certificates.

subjectKeyIdentifier=hash

authorityKeyIdentifier=keyid,issuer

# This stuff is for subjectAltName and issuerAltname.

# Import the email address.

# subjectAltName=email:copy

# An alternative to produce certificates that aren't

# deprecated according to PKIX.

# subjectAltName=email:move

# Copy subject details

# issuerAltName=issuer:copy

#nsCaRevocationUrl = http://www.domain.dom/ca-crl.pem

#nsBaseUrl

#nsRevocationUrl

#nsRenewalUrl

#nsCaPolicyUrl

#nsSslServerName

# This is required for TSA certificates.

# extendedKeyUsage = critical,timeStamping

[ v3\_req ]

# Extensions to add to a certificate request

basicConstraints = CA:FALSE

keyUsage = nonRepudiation, digitalSignature, keyEncipherment

#This extentions enabled alternative name of server

subjectAltName = @alt\_names

[alt\_names]

DNS.1 = www.covidpositivi.gov.it

DNS.2 = covidpositivi.gov.it

IP.1 = 127.0.0.1

[ v3\_ca ]

# Extensions for a typical CA

# PKIX recommendation.

subjectKeyIdentifier=hash

authorityKeyIdentifier=keyid:always,issuer

basicConstraints = critical,CA:true

# Key usage: this is typical for a CA certificate. However since it will

# prevent it being used as an test self-signed certificate it is best

# left out by default.

# keyUsage = cRLSign, keyCertSign

# Some might want this also

# nsCertType = sslCA, emailCA

# Include email address in subject alt name: another PKIX recommendation

# subjectAltName=email:copy

# Copy issuer details

# issuerAltName=issuer:copy

# DER hex encoding of an extension: beware experts only!

# obj=DER:02:03

# Where 'obj' is a standard or added object

# You can even override a supported extension:

# basicConstraints= critical, DER:30:03:01:01:FF

[ crl\_ext ]

# CRL extensions.

# Only issuerAltName and authorityKeyIdentifier make any sense in a CRL.

# issuerAltName=issuer:copy

authorityKeyIdentifier=keyid:always

[ proxy\_cert\_ext ]

# These extensions should be added when creating a proxy certificate

# This goes against PKIX guidelines but some CAs do it and some software

# requires this to avoid interpreting an end user certificate as a CA.

basicConstraints=CA:FALSE

# Here are some examples of the usage of nsCertType. If it is omitted

# the certificate can be used for anything \*except\* object signing.

# This is OK for an SSL server.

# nsCertType = server

# For an object signing certificate this would be used.

# nsCertType = objsign

# For normal client use this is typical

# nsCertType = client, email

# and for everything including object signing:

# nsCertType = client, email, objsign

# This is typical in keyUsage for a client certificate.

# keyUsage = nonRepudiation, digitalSignature, keyEncipherment

# This will be displayed in Netscape's comment listbox.

nsComment = "OpenSSL Generated Certificate"

# PKIX recommendations harmless if included in all certificates.

subjectKeyIdentifier=hash

authorityKeyIdentifier=keyid,issuer

# This stuff is for subjectAltName and issuerAltname.

# Import the email address.

# subjectAltName=email:copy

# An alternative to produce certificates that aren't

# deprecated according to PKIX.

# subjectAltName=email:move

# Copy subject details

# issuerAltName=issuer:copy

#nsCaRevocationUrl = http://www.domain.dom/ca-crl.pem

#nsBaseUrl

#nsRevocationUrl

#nsRenewalUrl

#nsCaPolicyUrl

#nsSslServerName

# This really needs to be in place for it to be a proxy certificate.

proxyCertInfo=critical,language:id-ppl-anyLanguage,pathlen:3,policy:foo

####################################################################

[ tsa ]

default\_tsa = tsa\_config1 # the default TSA section

[ tsa\_config1 ]

# These are used by the TSA reply generation only.

dir = ./demoCA # TSA root directory

serial = $dir/tsaserial # The current serial number (mandatory)

crypto\_device = builtin # OpenSSL engine to use for signing

signer\_cert = $dir/tsacert.pem # The TSA signing certificate

# (optional)

certs = $dir/cacert.pem # Certificate chain to include in reply

# (optional)

signer\_key = $dir/private/tsakey.pem # The TSA private key (optional)

signer\_digest = sha256 # Signing digest to use. (Optional)

default\_policy = tsa\_policy1 # Policy if request did not specify it

# (optional)

other\_policies = tsa\_policy2, tsa\_policy3 # acceptable policies (optional)

digests = sha1, sha256, sha384, sha512 # Acceptable message digests (mandatory)

accuracy = secs:1, millisecs:500, microsecs:100 # (optional)

clock\_precision\_digits = 0 # number of digits after dot. (optional)

ordering = yes # Is ordering defined for timestamps?

# (optional, default: no)

tsa\_name = yes # Must the TSA name be included in the reply?

# (optional, default: no)

ess\_cert\_id\_chain = no # Must the ESS cert id chain be included?

# (optional, default: no)

ess\_cert\_id\_alg = sha1 # algorithm to compute certificate

# identifier (optional, default: sha1)

Comando di OpenSSL 1 Edit del file di configurazione openssl.cnf

Le direttive *SubjectAlternativeName (SAN)* sono state aggiunte per definire all’interno del certificato il nome DNS e l’indirizzo IP a cui il server risponde; infatti ciò è utile per evitare problemi di accesso del web browser.

Un ulteriore modifica è stata quella di forzare la versione del protocollo utilizzata dalla backend. Questo è stato fatto, aggiungendo all’ interno del file /etc/apache2/mods-enabled/ssl.conf la seguente direttiva:

SSLProtocol all -SSLv3 -TLSv1.3

Questo ha permesso di poter utilizzare sul back end, l’ultima versione del protocollo TLS/SSL, cioè la versione 3.

Immagine che contiene screenshot, monitor, schermo

Descrizione generata automaticamente

Figura 3 Configurazione di SSL

Come mostrato dalla Figura 3, la connessione sicura con il server è avvenuta in modo soddisfacente, dimostrato dal lucchetto chiuso nella barra degli indirizzi. Ovviamente sono stati necessari degli opportuni settings nel web browser:

* Poiché il browser ha al suo interno un database delle rootCA fidate, possiamo impostare come sicura la rootCA *MinSalute*.
* Si importa il certificato client all’interno del browser generato con un file chiamato *opensslclient.cnf*, nel quale è stata settata la linea ***nsCertType=server***, in modo da generare certificati client.

Per quanto riguarda la generazione dei certificati della rootCA, sono stati usati i seguenti comandi di OpenSSL. La richiesta di certificato è stata richiesta definendo un file *.rnd* che contiene dei valori randomici, all’interno della cartella padre. La richiesta di certificato della è cosi formata:

**openssl req -new -x509 -newkey rsa:2048 -out cacert.pem -keyout cakey.pem -nodes -days 365**

Comando di OpenSSL 2 Definizione del comando per la richiesta del certificato rootCA

Il parametro *req* della richiesta indica che si vuole una richiesta di certificato; Il *-new* indica una nuova richiesta di certificato; mentre il parametro *-x509* indica una richiesta *self signed;* *-newkey* indica che bisogna definire una nuova chiave di tipo *rsa:2048* (dove 2048 indica la grandezza della chiave e *rsa* indica il tipo); *-keyout* indica il nome del file con l’estensione; *-out* indica il nome del file del certificato; *-nodes* indica che la chiave non deve essere cifrata sul disco; *-days 365* indica la durata in giorni della chiave e del certificato, giunta la data di scadenza bisogna aggiornare chiave e certificato.

È stata creata una directory chiamata *demoCA,* specificata nel file di configurazione *openssl.cnf*, che contiene al suo interno (come specificato in Figura 4) il certificato della CA e la chiave privata.

Immagine che contiene screenshot

Descrizione generata automaticamente

Figura 4 Screen delle cartelle per la generazione dei certificati

Nella Figura 4 vediamo le directory:

* *newcerts*: che contiene i certificati emessi in *formato.pem*
* *private*: che contiene il file della chiave della rootCA.
* *cacert.pem*: il certificato della rootCA.
* *Index.txt* è il database locale dei certificati rilasciati dalla RootCA. Nel formato:

V 210712074705Z 01 unknown /C=IT/ST=Roma/O=Gov/OU=MinSalute/CN=covidpositivi.gov.it

* *Serial*: che contiene il numero seriale del certificato che è stato rilasciato.

Eseguite tutte queste fasi di set up, si genera il primo certificato del server. Per poter fare ciò viene lanciato il comando:

**openssl req -new -newkey rsa:2048 -out myreq.pem -keyout mykey.pem -nodes -days 365**

Comando di OpenSSL 3: Comando per la richiesta di un certificato server

Notiamo che nel Comando di OpenSSL 2 è omesso il parametro *-x509*, poiché in questo caso il certificato server non deve essere self-signed.

Chiediamo ora alla CA di firmare il certificato; quindi eseguiamo il seguente comando:

**openssl ca -in myreq.pem -out mycert.pem**

Comando di OpenSSL 4: Comando per eseguire la firma di un certificato da parte della C.A.

Ora bisogna creare il certificato da importare nel browser, in modo che quest’ultimo riconosca il nostro server e la rootCA come elementi affidabili. Per fare ciò viene eseguito il comando:

**openssl pkcs12 -export -chain -CAfile demoCA/cacert.pem -inkey mykey.pem -name laboratorioanalisi -in mycert.pem -out mycert.p12**

Comando di OpenSSL 5 Comando per l' esporazione del file binario del certificato

il parametro pkcs12 serve per avere il formato binario:

* *-export* per esportarlo;
* *-CAfile* indica il certificato della rootCA;
* *-chain* indica l’intera catena del certificato utente;
* *-inkey* indica la chiave del certificato da utilizzare;
* *-name* indica il nome “amichevole” del file;
* *-in* indica il file del certificato da esportare;
* *-out* indica il nome del file esportato con estensione.p12;

Il certificato server è ben formato secondo lo standard, come si evince dalla Figura 5.

Immagine che contiene screenshot

Descrizione generata automaticamente

Figura 5 Visualizzazione del certificato importato nel browser

## Generazione dei certificati client

La creazione di questi certificati è stata utilizzata per permettere ai laboratori analisi l’accesso alla piattaforma di generazione degli esiti degli esami. Per fare ciò sono stati utilizzati i seguenti comandi.

La richiesta di certificato è stata definita come segue:

**openssl req -new -newkey rsa:2048 -out myreq\_client.pem -keyout mykey\_client.pem -nodes -days 365 -config opensslclient.cnf**

Comando di OpenSSL 6 richiesta di un certificato client

È stato utilizzato un nuovo parametro *-config* che indica quale file di configurazione deve essere utilizzato, in questo caso è stato definito un file differente da quello di default per la creazione dei certificati client.

**openssl ca -in myreq\_client.pem -out mycert\_client.pem -config opensslclient.cnf**

Comando di OpenSSL 7 Commando per la richiesta di firma alla CA

Con il Comando di OpenSSL 5, è stata chiesto di firmare il certificato alla C.A

**openssl pkcs12 -export -chain -CAfile demoCA/cacert.pem -inkey mykey\_client.pem -name laboratorioanalisisrl -in mycert\_client.pem -out mycert\_client.p12**

Comando di OpenSSL 8 Comando per esportare il certificato pkcs12 con estensione p12

Il Comando di OpenSSL 6 ha permesso di eseguire l’export del certificato client nel formato .p12.

## Implementazione del protocollo

La piattaforma è stata sviluppata per il rilascio dei codici usa e getta da parte dei laboratori analisi. Essa presenta una homepage scritta in linguaggio HTML, come si può evincere dalla Figura 3. Utilizzando la nav bar, è possibile accedere all’area riservata della piattaforma. Quando si accede all’area riservata, il browser richiede di selezionare un certificato client valido per accedere al server, come mostrato in Figura 6 .

Immagine che contiene screenshot

Descrizione generata automaticamente

Figura 6 Accesso alla piattaforma tramite la selezione di un certificato client

Cliccando su OK, è possibile selezionare il certificato per accedere alla risorsa secret; Nel caso in cui non si possieda un certificato utente valido, il server non permette l’accesso alla cartella *index* contenuta in secret, utilizzata dal laboratorio analisi.

Immagine che contiene screenshot, monitor, sedendo, parcheggiato

Descrizione generata automaticamente

Figura 7 Accesso laboratorio analisi

Nella Figura 7 è presente l’accesso all’area riservata per i laboratori analisi. Navigando la nav bar è possibile che un laboratorio analisi possa generare un certificato per un altro laboratorio analisi o per un altro operatore del laboratorio, mentre la pagina “genera esito esame” permette di generare l’esito dell’esame da collegare al referto consegnato al cittadino infetto, viene generato un pdf con un codice casuale univoco da consegnare al paziente.

Cliccando sul link “Genera certificato”, si cade sulla pagina *generacertificato.php*, la quale contiene una form per l’inserimento dei dati (vedi Figura 8). Vengono infatti richiesti tramite il popolamento di una form, l’inserimento della provincia, della città dove risiede il laboratorio analisi, il nome del laboratorio analisi (senza la regione sociale) e il nome dell’operatore che inserisce i dati.

Immagine che contiene screenshot, monitor

Descrizione generata automaticamente

Figura 8 Pagina generacertificati

Ogni singolo campo della form è obbligatorio, questo è stato implementato grazie all’opzione *required* del linguaggio HTML (vedi Figura 9).

Immagine che contiene screenshot, monitor, schermo

Descrizione generata automaticamente

Figura 9 Esempio di inserimento dei dati per la generazione di un certificato

Quando viene cliccato il pulsante “submit”, l’utente è reindirizzato alla pagina *genssl.php*. Vediamo che vengono creati dei link contenente il download della *req*, della chiave privata, del certificato client e del certificato client.p12 da importare nel browser (come sei evince dalla Figura 10).

Immagine che contiene screenshot, monitor, parcheggiato

Descrizione generata automaticamente

Figura 10 Creazione del certificato mediante lo script genssl.php

Notiamo che sulla backend nella directory client\_certificate (vedi Figura 11) sono creati i certificati che l’utente ha scaricato nella precedente figura.

Immagine che contiene screenshot, parcheggio, metro, macchina

Descrizione generata automaticamente

Figura 11 Contenuto della directory client\_certificate

Si suppone che sul server sia presente un event trigger che alle ore 00.00 di ogni giorno, elimini il contenuto della cartella client\_certificate, così che questa resti sempre vuota (ciò non è stato implementato).

?php

require('libreriapagine.php');

function gencert(){

$dn = array( "c" => "IT",

"st" => str\_replace(" ","",$\_POST["stateOrProvinceName"]),

"l" => str\_replace(" ","",$\_POST["localityName"]),

"o" => str\_replace(" ","",$\_POST["organizationName"]),

"ou"=>str\_replace(" ","",$\_POST["organizationName"]),

"cn" => str\_replace(" ","",$\_POST["commonName"]),

);

$request\_cert='openssl req -new -newkey rsa:2048 -out client\_certificate/myreq\_client\_'.$dn["o"].'\_'.$dn["cn"].'.pem -keyout client\_certificate/mykey\_client\_'.$dn["o"].'\_'.$dn["cn"].'.pem -subj "/C='.$dn["c"].'/ST='.$dn["st"].'/L='.$dn["l"].'/O='.$dn["o"].'/OU='.$dn["ou"].'/CN='.$dn["cn"].'" -nodes -days 365 -config certificate/opensslclient.cnf';

shell\_exec($request\_cert);

$ca\_request='openssl ca -batch -in client\_certificate/myreq\_client\_'.$dn["o"].'\_'.$dn["cn"].'.pem -out client\_certificate/mycert\_client\_'.$dn["o"].'\_'.$dn["cn"].'.pem -config certificate/opensslclient.cnf';

shell\_exec($ca\_request);

$export\_cert='openssl pkcs12 -export -chain -CAfile demoCA/cacert.pem -inkey client\_certificate/mykey\_client\_'.$dn["o"].'\_'.$dn["cn"].'.pem -name '.$dn["cn"].' -in client\_certificate/mycert\_client\_'.$dn["o"].'\_'.$dn["cn"].'.pem -out client\_certificate/mycert\_client\_'.$dn["o"].'\_'.$dn["cn"].'.p12 -password pass:';

shell\_exec($export\_cert);

if(file\_exists('client\_certificate/myreq\_client\_'.$dn["o"].'\_'.$dn["cn"].'.pem') & file\_exists('client\_certificate/mykey\_client\_'.$dn["o"].'\_'.$dn["cn"].'.pem') & file\_exists('client\_certificate/mycert\_client\_'.$dn["o"].'\_'.$dn["cn"].'.pem') & file\_exists('client\_certificate/mycert\_client\_'.$dn["o"].'\_'.$dn["cn"].'.p12')){

echo '<a class="iconaDownloadNeraNEW" href="client\_certificate/myreq\_client\_'.$dn["o"].'\_'.$dn["cn"].'.pem" download="client\_certificate/myreq\_client\_'.$dn["o"].'\_'.$dn["cn"].'.pem">Download Richiesta Certificato</a></br>';

echo '<a class="iconaDownloadNeraNEW" href="client\_certificate/mykey\_client\_'.$dn["o"].'\_'.$dn["cn"].'.pem" download="client\_certificate/mykey\_client\_'.$dn["o"].'\_'.$dn["cn"].'.pem">Download Chiave Certificato</a></br>';

echo '<a class="iconaDownloadNeraNEW" href="client\_certificate/mycert\_client\_'.$dn["o"].'\_'.$dn["cn"].'.pem" download="client\_certificate/mycert\_client\_'.$dn["o"].'\_'.$dn["cn"].'.pem">Download Certificato fomrato pem</a></br>';

echo '<a class="iconaDownloadNeraNEW" href="client\_certificate/mycert\_client\_'.$dn["o"].'\_'.$dn["cn"].'.p12" download="client\_certificate/mycert\_client\_'.$dn["o"].'\_'.$dn["cn"].'.p12">Download Certificato per l\'accesso</a></br>';

}

}

partesuperiore('Generazione Certificati client');

body('Generazione dei certificati client');

navbar();

corpo();

if(!empty($\_POST))

gencert();

endcorpo();

?>

Codice php 1 genssl.php

Nel Codice php 1, è stata utilizzata la funzione *shell\_exec*, per eseguire la generazione in bash dei certificati client. Analizzando il codice, viene preparato un array chiamato $dn, che contiene i campi del certificato da generare. Poiché non è stato possibile inserire questi parametri sfruttando una bash ubuntu, è stato settato il parametro *-subject* nella richiesta del certificato; tale parametro contiene gli stessi parametri che l’utente immette da input nella shell, invece i name dei file sono generati concatenando l’organizzazione e il common name, che contiene il nome dell’operatore che sta inserendo i dati.

Nel comando ca è stato introdotto il parametro *-batch*, il quale permette di generare automaticamente la richiesta CA senza avere ulteriore input da parte dell’utente.

Invece per il comando *pkcs12*, è stato introdotto il parametro *-password pass:*; tale parametro permette di settare una password per cifrare il file pkcs12, nel caso specifico questa password è stata lasciata vuota.

La pagina “esitoesame.php” contiene al suo interno due radio button; una volta che l’operatore ha comunicato una positività, viene generato un codice randomico da inserire nella piattaforma. Ciò è possibile perché la form ricade sulla pagina *gencode.php*, il cui codice è descritto nel modo seguente:

<?php

include ('fpdf/fpdf.php');

function gencode(){

if($\_POST["risposta"]=='si'){

$date=new DateTime();

$date->getTimestamp();

$lenghtkey=128;

$rand=random\_bytes($lenghtkey);

$code=hash('sha512',$rand.hash('sha512',$date\_hash));

print\_pdf($code);

}

else{

print\_pdf\_neg();

}

}

function print\_pdf($code){

$connessione=mysqli\_connect('localhost','root','covidserver','positivi') or die ("Connessione non avvenuta");

$query="INSERT INTO codici\_non\_usati(code) VALUES('$code')";

mysqli\_query($connessione,$query);

print(mysqli\_error($connessione));

$pdf=new FPDF('P',"mm","A4");

$pdf->AddPage();

$pdf->SetFont('Arial','B',16);

$pdf->Image("fpdf/testata.png",null,null,195,50);

$pdf->Ln(10);

$pdf->SetX(75);

$pdf->Cell(65,10,"Risultato analisi: Covid Positivo");

$pdf->Ln();

$pdf->SetX(0);

$pdf->SetFont('Arial','I',8);

$pdf->Cell(22,10,"Code:".$code);

$pdf->Output('D',"Positivo".$code.".pdf");

mysqli\_close($connessione);

}

function print\_pdf\_neg(){

$pdf=new FPDF('P',"mm","A4");

$pdf->AddPage();

$pdf->SetFont('Arial','B',16);

$pdf->Image("fpdf/testata.png",null,null,195,50);

$pdf->Ln(10);

$pdf->SetX(75);

$pdf->Cell(65,10,"Risultato analisi: Covid Negativo");

$pdf->Ln();

$pdf->Output('D',"Negativo.pdf");

}

gencode();

echo "<script>window.location.replace(\"esitoesame.php\");</script>"

?>

Codice php 2 gencode.php

La funzione *gencode* permette di generare il codice monouso per la comunicazione di positività al server. Per la generazione del codice, è inizializzata una variabile date che contiene il timestap della richiesta. La funzione *random\_bytes* genera in uscita dei random bytes con una chiave di 128 bit. È utilizzata in seguito una funzione hash dove è calcolato l’hash del timestamp e concatenato con l’uscita della funzione *random\_bytes,* ritenuta crittograficamente forte. Fatto ciò, viene generato il codice e salvato in una table di un database chiamata ***codici\_non\_usati***. Il campo code di questa table è *unique*, quindi non ci possono essere codici duplicati. Un esempio di funzionamento del sistema è riportato nella Figura 12.

Immagine che contiene screenshot

Descrizione generata automaticamente

Figura 12 Esempio della generazione di un codice monouso paziente positivo

Se il paziente risulta essere negativo, verrà generato un pdf con il risultato dell’esame negativo e senza il codice monouso come in Figura 13.

Immagine che contiene screenshot

Descrizione generata automaticamente

Figura 13 Generazione di esito esame negativo

Quando un cittadino risulta essere positivo, può comunicare il suo esito nella pagina *insertcode.php*. Tale pagina prende in input il codice randomico e salva questo codice nella tabella ***codici\_usati***. Poichè il campo code della table è univoco, non si può inserire due volte lo stesso codice.

Alla fine, è stampato a video un’alert-box, nella quale è mostrato il messaggio che il codice è stato inserito con successo oppure che il codice inserito non è valido.

Per le altre implementazioni si faccia riferimento al codice, allegato al pdf.