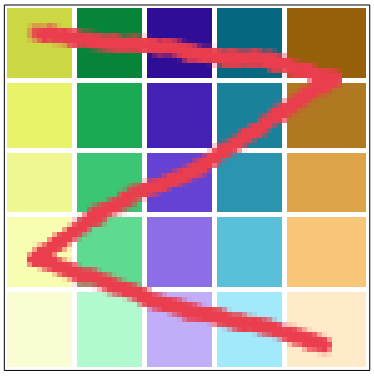
Treball Final de Màster

Contrasenyes Gràfiques



Màster Interuniversitari de Seguretat de les Tecnologies de la Informació i de les Comunicacions
- *MISTIC* -

Alumne: Josep Escoda Herrando
Directora: Cristina Pérez Solà
Universitat: UOC

Curs 2013- 2014, Sem. 1

Resum

Les contrasenyes gràfiques intenten aprofitar-se de la memòria visual dels humans per tal de compartir un secret basat en imatges o formes. Això és fàcil i econòmic d'implementar, evitant la invasió de la privacitat que poden generar alguns mètodes biomètrics i sense la necessitat de disposar de cap dispositiu físic (targeta, clau, NFC, etc.). Les contrasenyes gràfiques també aporten certes avantatges respecte les conegudes contrasenyes basades en text, com més facilitat per a ser recordades i un significatiu augment de la dificultat en atacs basats en predicció.

Partint d'aquesta idea el següent treball presenta un esquema de contrasenyes gràfiques que intenta fer èmfasi en la seguretat davant del robatori, ja sigui de les pròpies dades que formen la contrasenya (imatges de la pantalla, moviments dactilars, etc.), com del propi dispositiu. És en aquest últim punt on la nostra solució augmenta la seguretat dels esquemes basats en reconeixement d'imatges, encriptant amb un algorisme efectiu i fiable la base de dades.

Abstract

Graphical passwords use the human visual memory to share a secret based on images or shapes. They are easy and affordable to implement, avoiding the invasion of privacy that some biometric methods can produce and without the need of any physical device (card, key, NFC, etc.). Graphical passwords can also provide certain advantages over the usual text-based passwords: they are easier to remember, and guessing attacks become significantly more difficult to perform.

The following project presents a scheme of graphical password that tries to emphasize security against theft. The project deals both with data theft, where the data that form the password (images on the screen, fingerprints traces, etc.) is compromised, and with device theft, where the attacker has access to the entire device. It is on this last point where our solution increases the security of the schemes based on image recognition, encrypting the database with an effective and reliable algorithm.

Índex de continguts

1.- Introducció

Des de fa més de 100 anys, la psicologia sospitava que el cervell humà tenia més capacitat per a recordar informació visual que textual o verbal. Al 1971 *A. Paivio* va postular la teoria de la codificació dual (*dual-coding theory*) [25] on es demostrava que la informació visual i la informació verbal són processades i representades de forma diferent al cervell. Mentre que les imatges "*s'emmagatzemen*" directament com una forma o una percepció sensorial al cervell, el text ho fa de forma simbòlica. Així per exemple, una 'X' pot representar la grafia 'x' o bé el valor 10 en nombres romans, i calen més passos per a poder reconèixer-lo.

Atenent a això, podem deduir que les contrasenyes textuals han de ser més difícils de reconèixer i de recordar que no pas un tipus de contrasenya basada en informació visual [3]. Aquests tipus d'esquemes s'anomenen contrasenyes gràfiques i utilitzen imatges, gràfics o colors per tal de compartir un secret (la pròpia contrasenya) amb l'usuari.

Es parla d'una gran classificació de contrasenyes gràfiques basada en el tipus de procediment que ha de realitzar la memòria a l'hora d'utilitzar-la: **recordar** (*recall*), **reconèixer** (*recognition*) i **recordatori amb ajuda** (*cued-recall*) [2],[3]. El primer d'ells requereix que la persona recordi la contrasenya gràfica escollida per tal de poder-la reproduir. En canvi en el cas d'utilitzar el reconeixement és necessari que la persona hagi memoritzat la contrasenya gràfica prèviament i a l'hora de fer-ne ús realitzi un procés de *matching* entre la contrasenya memoritzada i la mostrada. Finalment, el tercer gran grup de metodologies basades en contrasenyes gràfiques oferirà un conjunt d'ajudes o pistes per tal de fer més fàcil el recordatori d'aquesta.

A més també es classifiquen les contrasenyes gràfiques en dos nivells: un primer nivell que s'anomena *pin-level*, on el nombre de possibilitats per a una longitud d'entre 12 i 15 bits es troba al voltant dels 10.000 elements, i el *password-level*, basat ens espais de 30 a 60 bits. Un exemple del primer nivell el podem trobar com a mecanisme de seguretat per al desbloqueig de les pantalles dels dispositius mòbils basats en Android OS.

En aquest treball s'analitzaran les solucions existents a nivell de contrasenyes gràfiques i s'intentarà definir un nou esquema amb un espai de possibilitats de tipus *password-level*, fent èmfasi en les principals característiques de seguretat que aquest ha d'aportar.

1.1.- Característiques principals de les contrasenyes gràfiques

Facilitat d'ús

Normalment per parlar de la facilitat d'ús d'un esquema de contrasenyes en general cal fer esment del temps necessari per a la creació i l'ús d'aquesta, de la capacitat de memoritza-la (*memorability*) i de les interferències (capacitat de recordar-la amb altres contrasenyes d'altres metodologies també en ús). Podem veure un bon resum a la taula 1 dels diferents esquemes de contrasenyes gràfiques i les seves propietats en relació a la usabilitat.

També cal fer esment en la capacitat de l'usuari d'escollir la contrasenya, on els tres grans tipus de contrasenyes gràfiques, que veurem a l'apartat 2, pateixen el mateix mal: l'usuari sol crear patrons predictibles que l'atacant pot explotar. A més se sol repetir la mateixa contrasenya per a diversos

comptes diferents, amb el risc que això comporta. Si en canvi la contrasenya és assignada pel sistema, se sol requerir un temps d'entrenament perquè l'usuari sigui capaç de memoritzar-ho, augmentant així el temps de creació i ús de la contrasenya. Cal cercar l'equilibri adequat entre aquests dos conceptes.

La seguretat

Per avaluar la seguretat de les contrasenyes parlarem de dos tipus d'atacs: els atacs de predicció (*guessing attacks*), on es realitza una cerca exhaustiva per tot l'espai de possibilitats de la contrasenya i els atacs de captura (*capture attacks*), on les credencials són obtingudes directament de l'usuari, mitjançant robatori o perquè ell mateix les ha divulgat (enginyeria social).

Graphical Password Schemes	Techniques		Usability Features on Graphical Password												
			Memorability												
	Recognition	Recall	Meaningfulness	Human faces	Organized by theme	User assign image	Icon based	Abstract image	Navigating through image	Drawing password	Efficiency	Input reliability & accuracy	Easy and fun to use	Grid based	Freedom of choice
Jansen et al. [10]	√		√		√						X		√	√	
Passfaces ™ [12]	√			√		√					X	√	√	√	
Triagle [18]	√						√				X		√		
Movable Frame [18]	√						√				X		√		
Intersection [18]	√						√				X		√		
Pict-O-Lock [19]	√						√				X	√		√	
Déjà Vu [3]	√							√			X		√	√	
Blonder [17]		√	√			√					X				√
VisKey SFR [6]		√	√			√					X	√			√
Passlogix v-Go [13]		√	√		√				√		X		√		
PassPoints [7]		√	√			√					√	√	√	√	√
DAS [9]		√	√							√		X		√	
√ = Yes X = No Blank = not mentioned															

Taula 1: Facilitat d'ús [2]

En termes generals, les contrasenyes gràfiques són molt més robustes que les textuals a atacs de força bruta, diccionaris i enginyeria social, principalment perquè és més difícil explicar una imatge / imatges que no pas dir un mot o un conjunt de caràcters. Per contra, són més dèbils davant dels atacs de captura, el més evident d'ells el *shoulder-surfing*, on l'atacant pot veure fàcilment quina imatge / imatges formen la nostra contrasenya, o fins i tot en casos més complexos, es pot fer ús conjunt d'un capturador de pantalla i un *logger* dels moviments de ratolí (atac per *man-in-the-middle* o MitM).

Si ens centrem en les contrasenyes gràfiques, podem extraure les següents conclusions:

- Per evitar els atacs d'enginyeria social és important que les imatges siguin difícils d'explicar i si poden ser generades de forma aleatòria molt millor.

- Per evitar els atacs de tipus *shoulder-surfing* sols podem utilitzar respostes dinàmiques, és a dir, que variïn en cada nova validació (*challenge-response*), però això fa que el temps dedicat a que l'usuari aprengui la contrasenya augmenti de forma desmesurada, convertint la metodologia en aplicable sols a situacions d'alta seguretat (on la biometria sol ser millor acollida).

- Per evitar atacs de força bruta o de diccionaris cal que l'usuari no defineixi la pròpia contrasenya. En el cas de que això no pugui ser, és important induir a l'usuari a no utilitzar patrons senzills.

- En general els atacs de diccionari sempre són més complicats de generar contra tècniques de reconeixement i de recordatori amb ajuda.

- Per evitar atacs de captura de contrasenyes, cal que s'eviti emmagatzemar-les als dispositius, utilitzar sempre les tècniques de *hash* per a les comunicacions i fer ús de *salts* o similar.

- Les contrasenyes de tipus reconeixement molts cops són un *pin-level*, per tant no aptes per a situacions on la seguretat sigui crítica.

- Una de les claus perquè l'usuari accepti la tècnica de les contrasenyes gràfiques és que siguin fàcils de memoritzar.

Un bon exemple de metodologia que intenta seguir bona part d'aquests principis és el que H.Gao, Z. Ren, X.Chang i U.Aickelin va presentar a [22]. En aquest cas es crea un esquema d'imatges per part de l'usuari seguint una història (veure *Story* [13]). Aquestes imatges, juntament amb unes altres, apareixen en posicions aleatòries a la pantalla. L'usuari ha de traçar un recorregut sobre les imatges escollides per ell (prèviament degradades a la pantalla) en l'ordre correcte (el de la seva història). Com que la traça passa per les seves imatges i per altres de col·locades aleatòriament, el possible atacant no podrà conèixer quines són exactament les seleccionades. A l'utilitzar una història creada per l'usuari, li serà fàcil de memoritzar.

Tot i això aquest esquema té dos problemes: el primer és que cal emmagatzemar al dispositiu quines imatges formen la història i el segon és que poden patir atacs d'intersecció si l'usuari no canvia sovint la contrasenya, ja que a partir d'un conjunt de captures de pantalles de validació es pot deduir quines són les imatges que sempre apareixen.

Igualment és una bona aportació i un sòlid punt d'inici per a perfilar quins són els problemes que cal resoldre en aquest treball de final de màster.

1.2.- Objectius i problemes

Cada vegada és més usual l'ús de dispositius mòbils, dotats de pantalla tàctil. Endarrere queden els temps de les pantalles de tipus PDA on era necessari l'ús d'un dispositiu de tipus *pen* per tal de poder interactuar amb el dispositiu. Aquests dispositius mòbils actuals disposen de potents

processadors amb gran capacitat multimèdia, i que fins i tot permeten executar un potent sistema operatiu complert. Alhora, el fet de que aquests dispositius siguin tan pròxims a l'usuari i emmagatzemin tanta informació privada (agenda, informes, accessos a diferents recursos on-line, xarxes socials, ...) fan d'ells un magnífic objectiu per als atacs d'usurpació d'informació. És més que evident que donat l'ús continuat que se'n fa i la informació crítica que emmagatzemen cal dotar-los d'un bon sistema d'autenticació. Si ajuntem totes aquestes característiques ens trobem en que els dispositius mòbils actuals, són un terreny perfecte per a l'ús de contrasenyes de tipus gràfic.

Després d'examinar un conjunt de solucions proposades en els últims anys (veure apartat 2) i de parlar de les característiques bàsiques que han de complir les contrasenyes, en general, i particularment les contrasenyes gràfiques, podem definir l'objectiu d'aquest treball.

L'**objectiu principal** és aportar una nova metodologia o esquema basat en contrasenyes gràfiques que eviti principalment els atacs de captura i alhora sigui prou robusta per evitar la majoria d'atacs de predicció. Tot això sense deixar de banda la facilitat d'ús per part de l'usuari i la facilitat de memorització.

Aquesta proposta estarà basada en el reconeixement d'un conjunt d'imatges aportades per l'usuari, però evitant emmagatzemar al dispositiu quin subconjunt forma la contrasenya gràfica. A més farà ús de la resposta dinàmica en el moment de la validació i la degradació d'imatges per tal d'evitar els atacs de captura de pantalla i *shoulder-surfing*. Respecte al tema de l'emmagatzematge, val a dir que és molt complicat no deixar cap mena de rastre de les imatges que formen la contrasenya al dispositiu, ja que la pròpia aplicació de validació ha de mostrar les imatges escollides perquè l'usuari es pugui autenticar, per tant l'aplicació ha de conèixer les imatges que formen la contrasenya tant sí com no. En el nostre cas, s'optarà per ocultar al màxim aquesta informació, mitjançant l'ús d'un algorisme d'encriptació fiable per a la base de dades.

Tot això serà implementat en mode d'aplicació *java* en un dispositiu mòbil amb sistema operatiu *Android OS*.

1.3.- Metodologia de desenvolupament

El desenvolupament del projecte ha constat de 5 etapes, que són les següents:

- **Definició del projecte**, document on es defineix de forma generalista l'objectiu del treball i es proposa una planificació temporal per a la seva realització.

- **Disseny i anàlisi del prototip d'aplicació**, on es realitza l'anàlisi de requeriments (funcionals i de seguretat), els casos d'ús, les estructures de dades i el disseny de la base de dades, la definició i disseny de les interfícies gràfiques i una primera aproximació a l'algorisme a implementar. Aquesta etapa també inclou un petit estudi sobre la viabilitat del projecte.

- **Desenvolupament del prototip d'aplicació**, on es treballa en la implementació de l'aplicació i les primeres proves de funcionament (fase *alpha*).

- **Proves i estudi estadístic**, on es realitzen les proves de l'aplicació amb usuaris reals (fase *beta*) i d'on s'extrauran un conjunt d'estadístiques per a la següent etapa.

- **Resultats i conclusions**, on s'analitzen els resultats obtinguts en l'anterior etapa, per poder avaluar

la funcionalitat de l'aplicació. També s'inclourà un anàlisi dels costos d'implementació.

No cal dir que durant totes les etapes s'ha creat la documentació necessària per al bon seguiment del projecte.

A nivell tecnològic, per al desenvolupament de l'aplicació s'ha fet servir un entorn basat en programació orientada a objectes, mitjançant el llenguatge d'alt nivell *Oracle Java*. A més de comptar amb el suport de les llibreries (*sdk*) de *Google* per al desenvolupament d'aplicacions sobre el sistema operatiu *Android OS*.

1.4.- Tasques portades a terme

Basat en les etapes anteriors, s'ha dividit el projecte en les tasques que podem veure al diagrama de Gantt de la següent pàgina.

1.5.- Organització de la Memòria

La memòria s'ha dividit en set grans apartats. El primer d'ells és la *Introducció*, on es presentaran els objectius del projecte portat a terme, i s'ubicarà la solució creada, així com la metodologia de desenvolupament seguida.

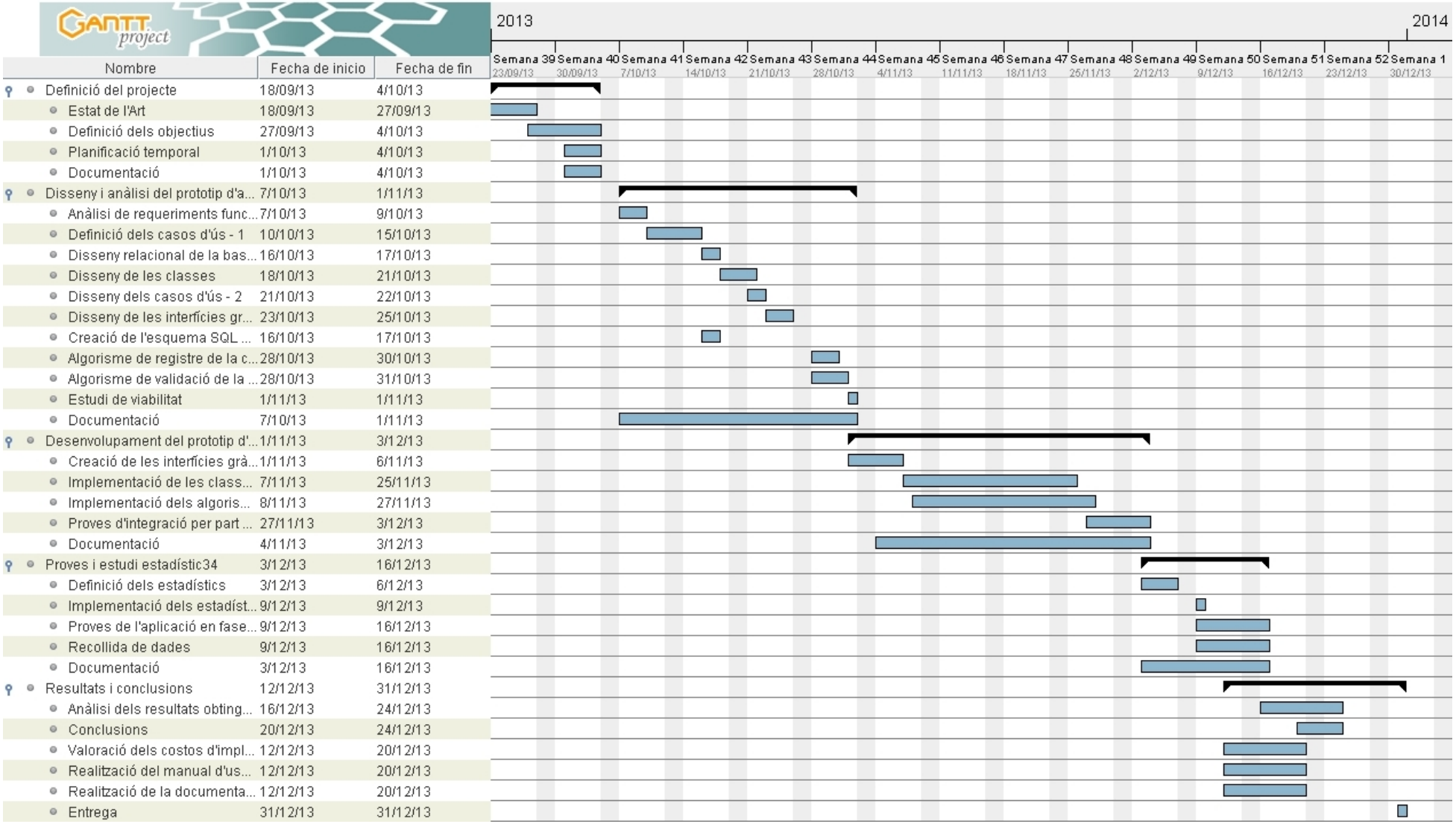
En el següent apartat, l'*Estat de l'art,* s'oferirà una visió de l'estat actual de les contrasenyes gràfiques, fent referència a les diverses solucions existents, i comentant els seus punts forts i els seus problemes més importants.

Per tal de poder articular una proposta, es realitzarà un petit *Anàlisi de requeriments,* on la informació generada en aquest apartat ens servirà per assentar les bases del *Disseny de l'aplicació*, que serà el següent apartat.

Finalment, un cop feta la *Implementació* de la proposta, es mostraran els detalls tècnics de la solució creada, tenint en compte tot el que s'ha comentat en l'apartat anterior. I com a *Conclusions,* es mostrarà el recull de les dades més importants pel que fa a l'ús de la nova aplicació.

El treball acaba amb la presentació de la *Bibliografia* utilitzada, índex de referència de totes les obres citades al treball.

Sols afegir que donada l'extensió i profunditat d'alguns dels apartats, a la memòria principal sols s'hi mostra un resum, deixant els detalls més tècnics per als annexos adjunts.



Nombre	Fecha de inicio	Fecha de fin
Definició del projecte	18/09/13	4/10/13
Estat de l'Art	18/09/13	27/09/13
Definició dels objectius	27/09/13	4/10/13
Planificació temporal	1/10/13	4/10/13
Documentació	1/10/13	4/10/13
Disseny i anàlisi del prototip d'a...	7/10/13	1/11/13
Anàlisi de requeriments func...	7/10/13	9/10/13
Definició dels casos d'ús - 1	10/10/13	15/10/13
Disseny relacional de la bas...	16/10/13	17/10/13
Disseny de les classes	18/10/13	21/10/13
Disseny dels casos d'ús - 2	21/10/13	22/10/13
Disseny de les interfícies gr...	23/10/13	25/10/13
Creació de l'esquema SQL ...	16/10/13	17/10/13
Algorisme de registre de la c...	28/10/13	30/10/13
Algorisme de validació de la ...	28/10/13	31/10/13
Estudi de viabilitat	1/11/13	1/11/13
Documentació	7/10/13	1/11/13
Desenvolupament del prototip d'...	1/11/13	3/12/13
Creació de les interfícies grà...	1/11/13	6/11/13
Implementació de les class...	7/11/13	25/11/13
Implementació dels algoris...	8/11/13	27/11/13
Proves d'integració per part ...	27/11/13	3/12/13
Documentació	4/11/13	3/12/13
Proves i estudi estadístic34	3/12/13	16/12/13
Definició dels estadístics	3/12/13	6/12/13
Implementació dels estadíst...	9/12/13	9/12/13
Proves de l'aplicació en fase...	9/12/13	16/12/13
Recollida de dades	9/12/13	16/12/13
Documentació	3/12/13	16/12/13
Resultats i conclusions	12/12/13	31/12/13
Anàlisi dels resultats obting...	16/12/13	24/12/13
Conclusions	20/12/13	24/12/13
Valoració dels costos d'impl...	12/12/13	20/12/13
Realització del manual d'us...	12/12/13	20/12/13
Realització de la documenta...	12/12/13	20/12/13
Entrega	31/12/13	31/12/13

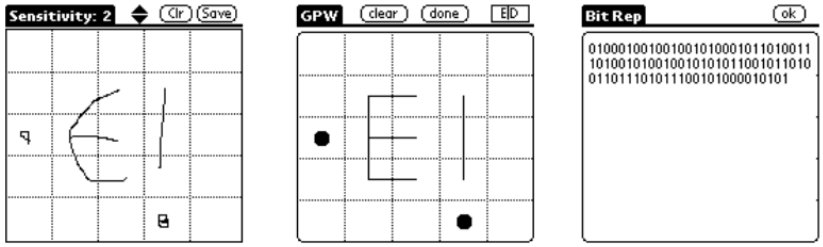
2.- Estat de l'Art

Basant-nos en la classificació presentada a l'inici de l'apartat anterior, es farà un petit recorregut per les tècniques d'implementació de les contrasenyes gràfiques sorgides en els últims anys.

2.1.- Sistemes basats en recordatori

Aquest conjunt de metodologies té en comú que l'usuari recorda i reprodueix un dibuix en una pantalla buida o amb l'ajuda d'una graella. Aquest dibuix ha de ser igual al que ell mateix ha definit com a dibuix secret. Notem que el recordatori sense cap mena d'ajuda és un procés costós per a la memòria. A més aquestes metodologies són molt susceptibles de patir atacs de captura, ja que la contrasenya ha de ser dibuixada en una pantalla visible i pot ser vista fàcilment pel possible atacant (*shoulder surfing*). Amb atacs de tipus *phishing* també és molt fàcil d'aconseguir, falsejant una interfície o pàgina. També hi ha problemes amb els capturadors de pantalla o els *mouse loggers*. Veiem algunes d'aquestes metodologies:

DAS: Una de les primeres aportacions a les contrasenyes gràfiques [4]. Aquesta metodologia consisteix en dividir una pantalla interactiva (on es pot dibuixar amb un *pen* o amb el dit) amb una graella (*grid*) de NxM caselles. Amb aquesta discretització de l'espai s'assignen un conjunt de coordenades que representen cadascuna de les caselles i quan l'usuari hi dibuixa s'emmagatzema l'ordre i la coordenada de cada casella on s'ha realitzat alguna traça. Notem que tota la traça interior a una casella acaba discretitzada en un sol punt dintre d'aquesta, veure fig.1.



(a) User inputs desired secret (b) Internal representation (c) Raw bit string

Fig. 1: Drawing A Secret (DAS)

Un *hash* SHA1 sobre aquesta llista de coordenades formarà la clau *k*. Amb aquesta clau s'encriptarà una frase *p* que està emmagatzemada al dispositiu o bé és pública i s'obtindrà el *ciphertext* que també s'emmagatzemarà al dispositiu obtenint $E_k(p)$[1]. Quan l'usuari vol validar-se al sistema es repeteixen els passos obtenint la clau k' i es comprova que $D_{k'}(E_k(p)) == p$. Amb aquest procediment s'evita emmagatzemar *k* al dispositiu.

Els autors també proposen el mecanisme per a calcular el nombre de possibles contrasenyes (espai de contrasenyes) i a la següent taula es pot observar el resultat amb una graella de 5x5:

$L_{\max}$	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*
$\log_2$(# passwords)	5	10	14	19	24	29	33	38	43	48
$L_{\max}$	*11*	*12*	*13*	*14*	*15*	*16*	*17*	*18*	*19*	*20*
$\log_2$(# passwords)	53	58	63	67	72	77	82	87	91	96

Fig. 2: Nombre de contrasenyes segons la longitud L

1 *E* representa el procés d'encriptació i *D* el de desencriptació.

A la fig.2, s'observa que amb una longitud (L_{max}) de 12 caselles, les possibilitats són de l'ordre de 2^{58}, superant les probabilitats de les contrasenyes textuals, on amb una taula ASCII de 95 codis i amb longitud 8 tindríem un espai de 95^8 que és aproximadament 2^{53}.

Però com es va demostrar a [5], l'ús per part de l'usuari de contrasenyes previsibles o simètriques, redueix dràsticament aquest espai. En aquest mateix treball es defineix el que és una contrasenya fàcil de recordar (*memorable password*), basant-se en l'ús de simetries, ja que està comprovat que l'ús d'aquestes ajuda a la memòria a recordar més fàcilment. Sols afegir que el DAS té problemes amb la resolució: un dibuix dintre d'un quadre acaba reduït a un punt.

BDAS: Aquesta metodologia utilitza el DAS com a base [6], però s'afegeixen imatges de *background* sota la interfície DAS perquè els usuaris facin contrasenyes més complexes, podent utilitzar les imatges de fons com a guia. És obvi que aquestes imatges intenten evitar l'ús de simetries o de patrons fàcils.

YAGP: S'utilitza un algorisme basat en la distància de Levenshtein a l'hora de fer el *matching* entre les contrasenyes, acceptant contrasenyes aproximadament correctes [7]. A més es perfecciona i afina el que ha traçat l'usuari sobre la pantalla (tenint en compte les derivades en la traçada del *pen*). Això permet afinar més el resultat i reduir la resolució de la graella (augmentant l'espai de contrasenyes), però per contra cal emmagatzemar la contrasenya original al dispositiu per poder reproduir l'algorisme i realitzar la validació.

PassDoodle: No deixa de ser un DAS sense la graella d'ajuda i amb altres característiques com el color del pen, les aixecades i baixades, el gruix del pen, etc [8]. El principal problema d'aquesta metodologia és que l'algorisme que afina la traça (idea ja exposada a YAGP per acceptar contrasenyes aproximadament correctes) necessita cert entrenament a l'hora de definir el model de la contrasenya, així l'usuari està obligat a realitzar-lo diverses vegades a l'hora de registrar-lo. I com totes les metodologies basades en l'afinament de la traçada, també necessita la contrasenya original emmagatzemada.

PassShapes: Aquesta metodologia està basada en convertir el dibuix a caràcters alfanumèrics mitjançant una discretització de la direcció de la traça en 8 angles (45º) [9]. Aquesta traducció permet aïllar-se del tamany i la forma de la interfície. Alhora el problema de la discretització és que l'espai de contrasenyes que s'obté l'acaba convertint en una mena de *pin-level*.

Pass-Go: L'usuari traça la contrasenya utilitzant les interseccions de la graella i no les caselles, així els moviments del *pen* o del dit queden discretitzats en interseccions i arestes entre elles, evitant el problema de resolució que té DAS [10]. Segons l'estudi d'efectivitat fet pels autors la tassa de bon funcionament a l'hora d'encertar la contrasenya per part dels usuaris és del 78%.

Notem que l'espai de contrasenyes d'aquesta metodologia és superior al del DAS ja que permet moviments en diagonal i adicionalment es pot canviar el color de la traçada. Amb el pas del temps s'ha millorat controlant la pressió exercida amb el pen (inviable amb el ratolí) evitant el *shoulder-surfing* i afegint imatges al *background* per millorar la capacitat de recordatori de l'usuari.

GrIDsure: Producte comercial [11] on a cada cel·la apareix un caràcter aleatori. Escollint aquests caràcters en forma de pin textual, es genera la traça que coneix l'usuari. A la fig. 3 podem observar que per a realitzar la traça que coneix l'usuari (en groc) cal escriure el PIN 8, 3, 6, 7, 3.

Fig. 3: Pantalla de GrID Sure

Per acabar aquest apartat referent les contrasenyes gràfiques basats en recordatori, afegir que actualment una versió reduïda de Pass-Go (graella de 3x3) forma part del sistema operatiu Android per a dispositius mòbils, activant-se com a mesura de seguretat per a la desprotecció de la pantalla un cop bloquejada. També ho podem trobar als dispositius *BlackBerry* sota el nom de *PatternLock*. Val a dir, però, que aquesta metodologia és susceptible de possibles atacs d'anàlisi de ditades a la pantalla (*smudge attacks*).

2.2.- Sistemes basats en reconeixement

Aquestes metodologies, també conegudes com *cognometrics systems* es basen en la capacitat dels humans de reconèixer imatges fàcilment. El mecanisme consisteix en escollir un seguit d'imatges en el moment de registrar la contrasenya i que després l'usuari haurà de seleccionar d'entre un grup per a validar-se.

La idea bàsica d'aquestes metodologies no és substituir les contrasenyes textuals, ja que l'espai de contrasenyes és equivalent a un *pin-level* de 4 o 5 dígits, però sí complementar-les. Si amb aquest tipus de sistemes es fan difícils els atacs per *phising* o *shoulder-surfing* (amb espais de contrasenyes grans), l'atac rei és el *Man In The Middle* (MITM), on es pot capturar informació ja que el sistema necessita intercanviar dades amb l'usuari en temps real (per exemple capturant els moviments del ratolí, *mouse logger*).

Una de les altres grans problemàtiques dels sistemes basats en reconeixement és que sempre s'ha de mantenir emmagatzemada la contrasenya al dispositiu, i per tant un atacant pot accedir als arxius del sistema i robar les claus. Veiem alguns d'aquests sistemes:

PassFace: Producte comercial [12] on cada contrasenya està formada per 4 cares humanes. En una primera fase s'assignen 4 cares a l'usuari, això formarà la seva contrasenya, i es realitza un entrenament de l'usuari (veure fig. 4), disposant panells de 9 cares on sols hi apareix una de les quatre cares que formen la seva contrasenya. Això es repeteix 3 vegades més, una per cada cara que forma la contrasenya. En el moment de validar-se apareixen durant 4 rondes els diferents panells amb 9 cares, on sols una forma part de la seva contrasenya. Això ens ofereix un espai de contrasenyes de 9^4, aproximadament 2^{13}.

A les proves que he fet he constat que la contrasenya té ordre, és a dir, en el moment de la validació les cares a escollir com a correctes segueixen sempre el mateix ordre. Desordenar l'elecció crearia una mica més de dificultat de cara a un possible atacant. A favor de PassFace sí que cal dir que l'assignació automàtica de la contrasenya, les 4 cares, facilita que l'usuari no seleccioni una contrasenya massa previsible, però per contra, aquesta tècnica és molt vulnerable als atacs que permetin capturar la pantalla i el ratolí, i per tant esbrinar quines són les cares seleccionades per l'usuari.

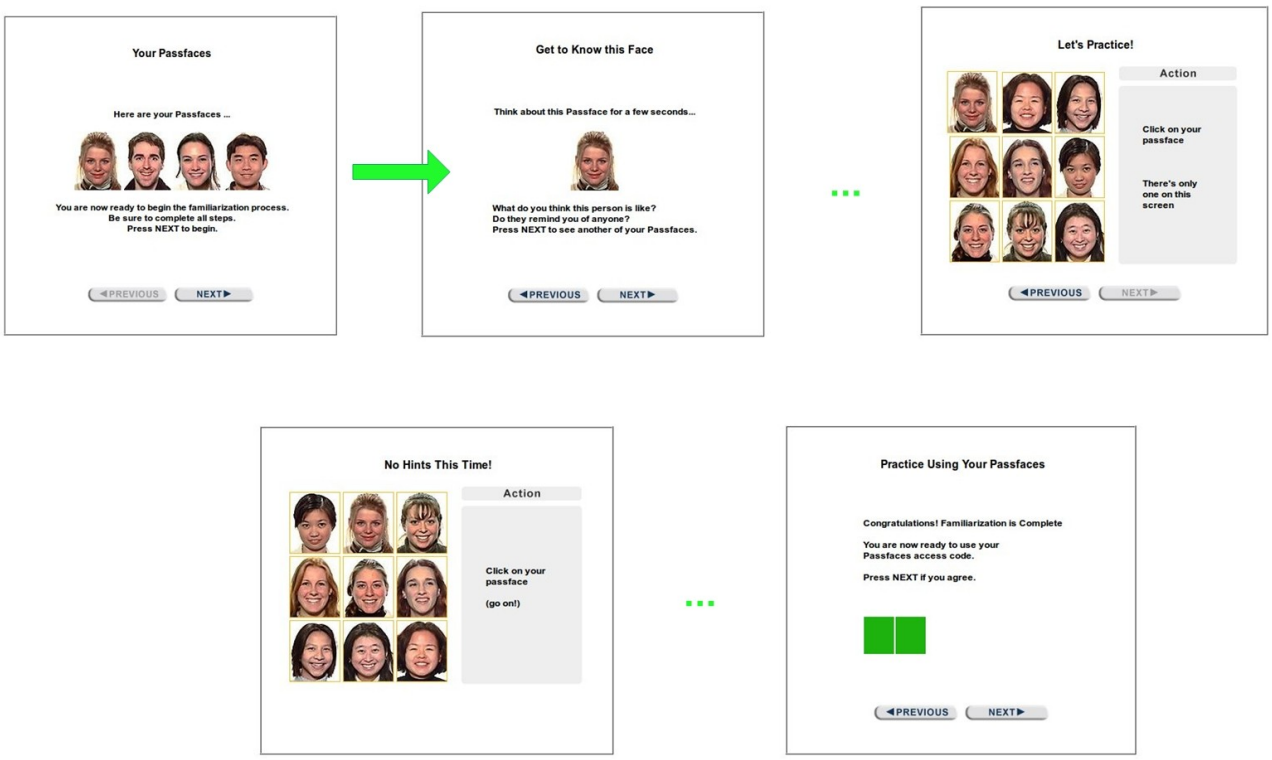


Fig. 4: Passos per al registre amb PassFace

Story: En aquest cas [13] l'usuari escull 4 imatges amb les que mentalment pot inventar una història. En el moment de validar-se, d'un panell de 9 imatges n'ha d'escollir les 4 que formen la seva història, és a dir, la contrasenya, en l'ordre correcte. Això ens ofereix un espai de contrasenyes 9·8·7·6, que és aproximadament 2^{12}.

Déjà Vu L'usuari escull un subconjunt d'imatges artístiques (*random-art*) que després haurà de seleccionar per poder validar-se [14]. L'elecció d'imatges de *random-art* fa que l'usuari no pugui explicar exactament com són les imatges i complica la divulgació de la contrasenya. En aquest cas l'espai de contrasenyes és el binomi basat en el nombre d'imatges que formen el conjunt de possibles eleccions (N) i el nombre d'imatges seleccionades per formar la contrasenya (M), així si N=25 i M=25, tenim que $\binom{N}{M}=\binom{25}{5}\simeq 2^{16}$

Cognitive Authentication: Metodologia que neix amb la idea d'evitar els atacs per robatori de contrasenya (*shoulder-surfing*) [15]. Prèvia selecció per part de l'usuari d'un conjunt d'imatges aquestes es mostren en una gran graella amb moltes imatges més (fig. 5). L'usuari ha de crear un camí per aquesta graella. Començant a la cantonada superior esquerra es va movent cap a la dreta per una mateixa filera fins trobar una imatge de les que formen la seva contrasenya, llavors baixa a la següent filera. Quan s'acaba la graella perquè s'ha arribat al marge dret o inferior, la casella final té un identificador, que és el que valida el procediment, mitjançant una resposta formulada a la mateixa pantalla (marge superior de la fig. 5). Això es repeteix en diversos torns.

El problema d'aquesta metodologia és que el conjunt d'imatges que l'usuari ha de memoritzar ha de ser força gran, per exemple 30. Si en cada panell es mostren 80 imatges, l'espai de contrasenyes és de $\binom{80}{30}\simeq 2^{73}$, però això és fictici perquè la validació real es fa amb la codificació de la casella i

això redueix dràsticament l'espai de contrasenyes, podent ser atacat fàcilment mitjançant força bruta.

Fig. 5: Panell d'exemple de Cognitive Authentication

També existeixen un conjunt de metodologies on l'usuari introdueix les seves pròpies imatges, que haurà de reconèixer per poder validar-se. Per fer-los més robustos, aquests mètodes modifiquen, deformen o distorsionen aquest conjunt d'imatges, per a dificultar un possible atac per enginyeria social. Així tenim *Convex Hull Click Schema* [16]*, Your Illusion* [17]*, Graphical Passwords with Icons (GPI)* [18], *etc*.

A mode de resum, sols afegir que les contrasenyes basades en reconeixement, tant engloben les metodologies on l'usuari simplement ha de seleccionar una imatge (equiparables a un *pin-level*), com les que l'usuari ha de realitzar un procés a partir de les imatges reconegudes per tal de validar-se (repte-resposta / secret compartit), com poden ser les metodologies amb imatges distorsionades.

2.3.- Sistemes basats en recordatori amb ajuda

Aquests sistemes es basen en fer que l'usuari recordi detalls o posicions d'una imatge concreta. Aquesta imatge és la pista (*cue*) que facilita el procés de recordatori. Aquestes metodologies presenten totes la mateixa vulnerabilitat: l'atacant pot capturar la pantalla i els moviments del ratolí per conèixer les posicions concretes que formen la contrasenya, així com utilitzar tècniques de *shoulder-surfing*. Veiem alguns exemples:

PassPoints: Donada una imatge l'usuari escull 5 punts o detalls amb un llindar de precisió concret (*box*) [19]. Per validar-se cal tornar a escollir els 5 punts registrats anteriorment seguint el mateix ordre. En aquest cas sols es compta amb una pista (*cue*) que és la imatge, sobre la que cal recordar 5 detalls. A més hem de definir quin és el llindar (*box*) sobre el que s'accepta que un punt és el punt registrat prèviament, per això se sol discretitzar l'espai amb diferents alternatives, com poden ser els polígons de Thiessen.

Existia una versió comercial per a desbloquejar la pantalla dels extints PocketPC, que s'anomenava *visKey*, on l'usuari podia escollir el nombre de punts, la imatge i la precisió. També existeixen modificacions on es presenta part de la imatge ombrejada i l'usuari sols ha d'indicar si el detall està a la part visible o no (evita mínimament el *shouder-surfing*), o bé variacions on sols s'ha d'escollir un detall d'una seqüència d'imatges (*Cued Click Points* ó *CCP*).

Una bona variant del CCP, anomenada *Persuasive-CCP* [20] indica possibles zones on l'usuari pot escollir els detalls que crearan la contrasenya. Això evita que l'usuari pugui escollir punts massa previsibles o seguir patrons geomètrics excessivament fàcils.

Inkblot Auth: En aquest cas s'utilitza una visualització de taques de tinta (*inkblot*) on l'usuari defineix de forma textual el que li recorda aquella taca, per exemple utilitzant la lletra inicial i final de l'objecte recordat [21]. En el moment de validar-se se li presenten un seguit de taques en ordre aleatori i cal que reescrigui el text que ha associat a cada taca. Notem que la contrasenya textual resultant és forta ja que no es solen crear paraules amb significat, dificultant enormement els atacs per diccionari. Tot i això, aquest últim tipus de tècnica estaria situada a mig camí entre les contrasenyes gràfiques i les textuals, en el que podríem anomenar *gràfics d'ajuda a les contrasenyes textuals*.

Podem veure una taula resum de totes les tècniques presentades al document [3] .

3.- Anàlisi de Requeriments

Abans d'iniciar aquest apartat caldrà que definim uns quants conceptes que ens ajudaran a identificar millor els requeriments de l'aplicació:

- Facilitat de memorització: (*Memorability*): Direm que un esquema de contrasenyes és fàcil de memoritzar quan la contrasenya o el mecanisme d'autenticació sigui fàcil de recordar per part de l'usuari. Aquesta és una propietat decreixent en el temps.

- Facilitat d'ús: Com hem vist en apartats anteriors, parlarem de facilitat d'ús quan l'usuari no tingui problemes per utilitzar el mecanisme d'autenticació d'un esquema de contrasenyes concret. La facilitat d'ús és un factor que sols s'incrementa amb el temps d'utilització o amb entrenament.

- Atac per *shoulder-surfing*: Atac consistent en observar com l'usuari aplica el mecanisme d'autenticació, sense que se n'adoni, i després reproduir-lo sense autorització.

- Atac per captura: (MitM): Atac consistent en interposar-se en el canal de comunicació per a capturar les dades que hi circulen. En el nostre cas ens referirem als atacs que permeten capturar el mecanisme utilitzat per l'usuari en un esquema de contrasenyes concret. Per exemple els capturadors dels moviments del punter per la pantalla, o imatges de la pròpia pantalla.

- Atac per robatori: Quan s'esmenti un atac per robatori ens referirem a la usurpació o furt del dispositiu mòbil o portàtil.

- *Rootejar*: La paraula *rootejar* és l'argot que indica l'ús d'un dispositiu mòbil com a usuari amb privilegis, és a dir, com a administrador o *root* (en entorns Android).

Com bé hem dit a la introducció, l'objectiu final del treball proposat és la implementació d'un esquema de contrasenyes gràfiques per a un dispositiu mòbil. Aquest esquema parteix amb la idea inicial de protegir l'accés a una aplicació crítica, àmbit en el que es pot permetre un mecanisme d'autenticació més complex, que tot i ser ràpid, pot no ser-ho tant com l'esquema utilitzat a l'hora de desbloquejar una pantalla, per exemple.

Classificarem els requeriments segons el seu ordre d'importància o prioritat de cara a cercar una solució que els satisfaci.

Prioritaris

- Es vol un esquema de contrasenyes de tipus gràfic que millori la facilitat de memorització dels esquemes de tipus textual.

- Es vol un esquema de contrasenyes basat en l'ús d'una interfície tàctil, en un dispositiu mòbil.

- L'esquema de contrasenyes ha de ser robust davant d'atacs per captura.

- L'esquema ha de ser robust davant d'un atac per robatori d'un dispositiu *rootejat*.

- L'esquema ha de constar d'una etapa d'entrenament prèvia el més curta possible, intentant, fins i tot, que no sigui necessària.

Altres requeriments

- La seguretat prevaldrà per sobre de la facilitat de memorització de la contrasenya (*memorability*), però mantenint la senzillesa en el mecanisme d'autenticació.

- La implementació s'ha de portar a terme mitjançant eines gratuïtes.

- La implementació ha de funcionar en dispositius amb pantalles a partir de 4" o més.

- No es pot suposar que l'usuari, d'alguna forma, no tingui accés al codi de l'aplicació o conegui el funcionament de l'esquema presentat.

- L'esquema hauria de ser robust als atacs de força bruta o predicció, i tenir algun tipus de protecció contra l'enginyeria social.

Veiem una taula resum dels requeriments de la solució a desenvolupar:

	Requeriments Prioritaris	**Altres Requeriments**
Funcionals	- Ús de contrasenyes gràfiques - Dispositius mòbils amb pantalla tàctil	- Ús d'eines gratuïtes de desenvolupament - Minimització de l'etapa d'entrenament - Pantalles de 4” o més
De Seguretat	- Robust davant d'atacs de captura. - Robust davant d'atacs per robatori (*rootejat*)	- Robust davant d'atacs de predicció. - Mecanismes contra l'enginyeria social. - Suposem que l'atacant coneix el codi de l'aplicació

Taula 2: Requeriments de la solució a desenvolupar

Notem que en els requeriments prioritaris que l'esquema ha de complir hi trobem la robustesa davant de dos tipus d'atacs concrets. Pel que fa als atacs de captura, cal lluitar contra la facilitat que pot tenir un atacant de veure com l'usuari utilitza el mecanisme d'autenticació (per exemple amb *shoulder-surfing* o fent ús de capturadors de pantalla). En aquests cas cal optar per un tipus de mecanisme que faci que l'usuari no hagi de repetir sempre les mateixes accions, és el que s'anomena resposta dinàmica.

En els mecanismes amb resposta dinàmica es comparteix un secret entre l'aplicació i l'usuari i és indiferent de la seva forma de presentació en pantalla, això ràpidament ens farà pensar en un esquema de tipus reconeixement. Els esquemes de recordatori fan difícil utilitzar una resposta dinàmica, ja que l'usuari recorda exactament una imatge o forma i per tant no es pot anar variant a cada autenticació sense confondre'l i perdre la facilitat de memorització.

El problema dels esquemes basats en reconeixement és que sempre s'ha d'emmagatzemar al dispositiu els elements a reconèixer durant el mecanisme d'autenticació (si es vol evitar l'ús d'una arquitectura client-servidor). Això fa que en cas de robatori d'un dispositiu, on l'atacant pot arribar a obtenir privilegis d'administrador, aquests esquemes siguin molt fàcils d'evadir. L'atacant sols haurà de fer una cerca pels elements emmagatzemats fins a trobar els que identifiquen a l'usuari.

Seguint amb aquests dos raonaments, sembla evident que la millor solució serà trobar un esquema on s'utilitzi el recordatori i a l'hora el reconeixement, per tal de dotar de seguretat l'aplicació.

De cara a clarificar els problemes inherents a la seguretat i per tal d'identificar les possibles solucions per a implementar en el nostre esquema, es mostra una taula basada en els coneixements o les accions que pot emprendre un suposat atacant:

Atacant	***Possibles Solucions***
Ha capturat la pantalla de l'usuari en el moment de l'autenticació	Utilitzar resposta dinàmica i no marcar l'elecció de l'usuari en el moment d'introduir la contrasenya.
Ha capturat el punter de l'usuari en el moment de l'autenticació	Utilitzar resposta dinàmica.
Ha capturat la pantalla i el punter de l'usuari en el moment de l'autenticació	Dificultar l'acció mitjançant *multitouch*, ja que pocs capturadors són capaços de fer-hi front. Notem que aquesta situació és desastrosa, gairebé com donar-li la clau a l'atacant.
Ha robat el dispositiu físic i no és *root*	Les pròpies capacitats a nivell de permisos del sistema operatiu haurien de ser suficients.
Ha robat el dispositiu físic i té accés a les dades	Encriptació robusta de les dades (AES) mitjançant secret compartit amb l'usuari, és a dir, la contrasenya que encripta les dades no pot estar emmagatzemada al propi dispositiu.
Ha capturat prèviament el punter de l'usuari en el moment de l'autenticació i ara té el dispositiu físic	Dissenyar un esquema on l'autenticació no depengui solament del punter de l'usuari, per exemple amb resposta dinàmica.
Ha capturat prèviament la pantalla de l'usuari en el moment de l'autenticació i ara té el dispositiu físic	Utilitzar resposta dinàmica per a validar l'encriptació de les dades.
Ha capturat prèviament la pantalla i el punter de l'usuari en el moment de l'autenticació i ara té el dispositiu físic	No hi ha res a fer.
Ha robat el dispositiu físic i ataca per força bruta dades encriptades	Utilitzar un algoritme robust per a encriptar i generar contrasenyes no basades en diccionari.
Ha robat el dispositiu físic i ataca amb diccionari dades encriptades	Utilitzar un algoritme robust per a encriptar i generar contrasenyes no basades en diccionari amb l'afegit d'utilitzar un *salt*.
L'atacant intenta descobrir la contrasenya per enginyeria social	Utilitzar mecanismes que dificultin la descripció de la contrasenya. Per exemple distorsionant les imatges en mecanismes de reconeixement de forma que siguin relativament fàcils de reconèixer però molt difícils d'explicar.
L'atacant intenta utilitzar *shoulder-surfing* per veure que seleccionem o dibuixem a la pantalla	Resposta dinàmica sense marcar l'elecció i mecanismes que dificultin l'observació (colors poc brillants, imatges de fons per distraure, traçades parcials, ...)

Taula 3: Accions a esperar d'un possible atacant

4.- Decisions de disseny

Tenint en compte els requeriments exposats a l'apartat anterior i les possibles solucions presentades, s'ha decidit crear un sistema de contrasenyes gràfiques fusionant un mètode de recordatori i un de reconeixement. D'aquesta manera, s'eviten els inconvenients de cadascun per separat. Un bon exemple d'aquesta fusió el podem trobar a [22], comentat ja al revisar l'estat de l'art.

En relació a la primera fase, s'ha escollit fer servir un DAS [4] amb una única traça o cop (*stroke*), generada cada vegada que s'aixeca o s'abaixa el punter (tot i que l'aplicació podria funcionar igualment si permetem més d'una traça com a contrasenya). Aquesta traça ens permetrà desencriptar les imatges que formaran part de la segona fase.

En relació a la segona fase, l'usuari haurà de reconèixer dues parelles d'imatges, repetint el procediment per a cada parella i amb un conjunt de característiques que reforçaran la seguretat de l'esquema i seran presentades a l'apartat 4.2.

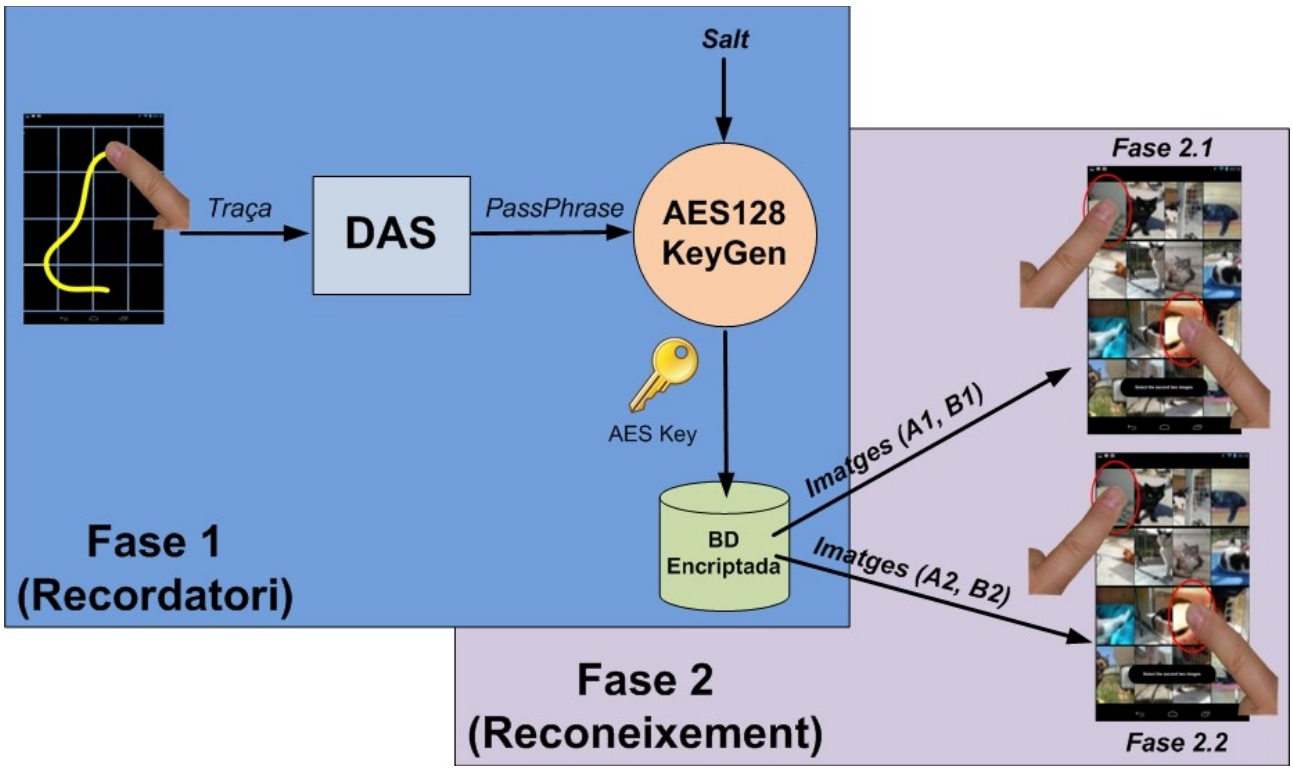


Fig. 6: Diagrama de blocs de l'esquema presentat

4.1.- Primera fase - DAS

La matriu utilitzada en aquesta primera fase tindrà unes dimensions de 4x4 cel·les *(GxG)*. Recordem que l'algorisme DAS redueix tots els punts d'una mateixa cel·la a un únic punt, però té en compte l'ordre de pas per cadascuna de les cel·les. Així amb 16 cel·les, l'espai de contrasenyes dependrà sols de la longitud de la traça que l'usuari faci. Es restringeix la traça a un sol cop, és a dir, la traça ha de

ser contínua. A mode de resum[2] podem veure la següent taula on apareix el nombre de possibilitats agafant com a longitud màxima acceptada, en cel·les, de la traça L_{MAX} .

L_{MAX}	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\log_2$	*4*	*6*	*7,75*	*9,46*	*11,15*	*12,84*	*14,54*	*16,73*	*17,93*	*19,62*	*21,32*	*23*	*24,7*	*26,4*	28,1

Taula 4: Espai de contrasenyes del DAS 4x4

En principi l'aplicació no restringirà la longitud de la traça, però a partir de 16 cel·les caldrà passar més d'un cop pel mateix lloc. Això està permès però fa la traça complexa i és poc probable que l'usuari la defineixi així. Podem agafar com a mesura més probable una longitud de la traça de 8 cel·les, el que donaria un espai de contrasenyes de 76863, és a dir aproximadament 2^{17}.

Encara que a priori pugui semblar un espai de contrasenyes petit, cal esmentar que fins al final de la segona fase no es pot saber si l'autenticació ha estat correcta o no, i per tant aquestes 2^{17} possibilitats cal multiplicar-les per les possibilitats de la segona fase.

És important que l'usuari eviti les traçades simètriques, tal com s'explica a [5]. Així si, a mode d'exemple, de l'anterior espai de contrasenyes ens fixem sols en les generades basades en rectangles[3], podem veure que el nombre es redueix dràsticament,

$$\left(\left(\binom{G}{2}^2\right)\cdot 8\right)\cdot 2=\left(\left(\frac{1}{4}G^2(G-1)^2\right)\cdot 8\right)\cdot 2=576$$

També es fa palès que generar traçades gràfiques de forma automàtica sobre un dispositiu mòbil tampoc és una tasca fàcil, dificultant la feina d'un possible atacant.

A més, per evitar possibles atacs mitjançant *shoulder-surfing* es farà servir un color de baixa intensitat al dibuixar la traça a la pantalla. A més, conforme s'avanci en el dibuix de la traça la part inicial es farà desaparèixer parcialment de la pantalla.

Si l'atacant utilitza un capturador de pantalla serà capaç de generar la mateixa traça que l'usuari, en aquest cas caldrà confiar en la segona fase del procés d'autenticació.

La traçada generada, mitjançant una funció resum SHA-512, es convertirà en una *passphrase* de 128 caràcters, de la que s'obtindrà una clau que permetrà encriptar les dades referents a la segona fase, és a dir, les imatges que formen la contrasenya de l'usuari en la fase de reconeixement. Quant més llarga sigui la traçada més complexa serà la *passphrase* generada i més complex serà trobar-la mitjançant algun mecanisme de força bruta. Es podria pensar que l'atacant pot generar alguna mena d'atac de diccionari sabent el tipus de codificació de les cel·les per tal de trobar la *passphrase* que genera la clau per desencriptar la base de dades, reduint així l'espai de cerca. Per aquesta raó s'utilitzarà un *salt* afegit a la traçada i un altre de diferent per a generar la clau a partir del resum de la traçada.

Aquesta manera de procedir fa que les imatges que formen la contrasenya en la segona fase no estiguin emmagatzemades en clar al dispositiu, dificultant la feina de l'atacant que el roba i disposa de privilegis d'accés a aquestes dades. En el cas que l'atacant intentés realitzar atacs purs de força bruta, necessitaria una capacitat computacional difícilment assumible, ja que per a encriptar les

2 Veure Annex – 1 per saber com s'ha realitzat aquest càlcul
3 https://www.usenix.org/legacy/events/sec99/full_papers/jermyn/jermyn_html/node10.html#SECTION00033210000000000000

dades referents a les imatges de la contrasenya s'utilitzarà un algorisme robust de clau simètrica, com pot ser AES-128 (veure la justificació d'aquesta decisió a l'*Annex-2*).

4.2.- Segona fase - recognition

Tornant a l'aplicació, un cop s'han desencriptat les dades referents a les imatges que l'usuari fa servir com a contrasenya, aquestes han de ser mostrades a la pantalla juntament amb altres imatges, on tant les imatges com les posicions han de ser aleatòries i canviants, generant una resposta dinàmica que anul·li els atacs de *shoulder-surfing*. La millor disposició de cara a que l'usuari pugui seleccionar les imatges de la contrasenya és en forma de matriu, on a cada cel·la hi trobarem una imatge distorsionada, per evitar una fàcil descripció mitjançant enginyeria social.

Les imatges escollides per l'usuari com a contrasenya també seran distorsionades. Això no hauria de representar cap problema si imposem que les imatges escollides siguin generades pel propi usuari (per exemple amb la càmera que gairebé tots els dispositius mòbils tenen actualment). Com que l'usuari reconeix bé les imatges que ell mateix ha captat (són escenes que ha viscut o vist en directe) les pot reconèixer encara que tinguin certa distorsió. Aquesta idea va ser presentada a [17] com a una eina per a dificultar els atacs per enginyeria social, ja que és difícil que un atacant pugui reconèixer una imatge distorsionada tot i que li expliquin que hi apareix.

La matriu d'imatges tindrà unes dimensions de 4x4, per mantenir el mateix model que en la fase anterior. Així es disposa a priori de 16 possibilitats a l'hora d'escollir una imatge.

Per dificultar l'ús de capturadors de pulsacions a la pantalla es farà servir la característica *multitouch* de les pantalles tàctils actuals. Com ja s'ha comentat a l'apartat d'anàlisi de requeriments, no hi ha gaires capturadors de pulsacions capaços de detectar més d'una pulsació alhora. Per una mera qüestió ergonòmica, el nombre de pulsacions que l'usuari haurà de fer alhora seran dues, una per a cada dit gros de cada mà. Això fa que l'usuari hagi de reconèixer dues imatges, augmentat l'espai de contrasenyes a 16·15 = 240 possibilitats.

Per fer més gran aquest espai de contrasenyes, les dues imatges seran dividides en dues parts, així l'usuari sols caldrà que recordi dues imatges, però a l'hora de reconèixer-les en reconeixerà quatre. Per fer-ho fàcil, simplement es repetirà aquesta segona fase dues vegades. I en cada repetició s'escolliran les parts que formen les dues parelles. Ara tenim 240 possibilitats per cada fase, el que augmenta l'espai de contrasenyes fins a 240^2 (57600). Afegint les probabilitats generades a la primera fase, amb una traça de longitud 8, obtenim un espai de contrasenyes final de $240^2 \cdot 76863$, per sobre d'un *pin-level*[4].

Si es permeten contrasenyes amb múltiples traçades a la primera fase, l'espai de contrasenyes, amb longitud de traça 8, s'eleva a $240^2 \cdot 2^{34}$ (Veure Annex-1).

L'ús de capturadors de pantalla no té sentit en l'esquema presentat ja que en cap moment es marcaran les imatges escollides com a contrasenya quan aquestes siguin seleccionades.

En cas de realització d'un atac d'intersecció per a esbrinar quines imatges són les que apareixen

4 Amb una traça de longitud 8 obtenim un espai de contrasenyes de 2^{32}. Si suposem un *pin-level* de 4 caràcters *ascii* imprimibles obtenim un espai de contrasenyes de 95^4 que és aproximadament 2^{26}, per tant $2^{32} > 2^{26}$.

Si ho comparem amb l'esquema de desbloqueig de pantalla que implementa *Android OS*, versió 3x3 simplificada de *Pass-Go*, que permet diagonals, l'espai de contrasenyes amb una traça de longitud 8 és d'aproximadament 2^{19}.

sempre (les que formen la contrasenya), caldria prèviament haver aconseguit realitzar correctament la fase primera, ja que si no no es mostraran les imatges que formen la contrasenya en les matrius d'imatges. Per evitar-ho cal augmentar la freqüència en que l'usuari hauria de canviar la contrasenya.

4.3.- Resum de les decisions de disseny

Taula a mode de resum de les decisions de disseny presses respecte als requeriments de seguretat.

Decisió de disseny	Aportació a la seguretat
Encriptació de les dades mitjançant secret compartit	Evita que un atacant que roba el mòbil i té privilegis per accedir a la base de dades pugui conèixer les imatges que formen la contrasenya
Ús de AES-128 per encriptar les dades	Ús d'un algorisme robust i eficient per tal d'encriptar les dades. Dificulta els atacs per força bruta.
Utilització del DAS en la primera fase	Contrasenya de tipus gràfic fàcil d'utilitzar i que genera un espai de contrasenyes acceptable, essent el secret compartit amb l'usuari
Resum de la traça DAS amb SHA-512	La funció de *hash* SHA-512 genera un resum de 128 caràcters utilitzat com a *passphrase* per a encriptar les dades de la base de dades. Assegurem una longitud constant.
Ús d'un *salt* amb la traça DAS i al generar la clau de l'AES	Utilitzar un *salt* evita els atacs de diccionari si es coneix prèviament la codificació que es fa de la traça.
Baixa intensitat de color a la traça i visibilitat parcial d'aquesta	S'evita en la mesura del possible els atacs per *shoulder-surfing* dificultat que l'atacat pugui observar fàcilment la traça
Reconeixement d'imatges distorsionades	Evita atacs per enginyeria social ja que és difícil explicar i reconèixer el contingut de la imatge
Reconeixement d'imatges amb *multitouch*	Evita l'ús de capturadors de pulsacions
No indicació d'encert al seleccionar les imatges	Evita l'ús de capturadors de pantalla
Col·locació aleatòria de les imatges (resposta dinàmica)	Evita que un atacant pugui veure fàcilment quines imatges són les seleccionades com a contrasenya, ja que cada vegada apareixeran en una posició diferent
Repetició de la segona fase de l'esquema presentat	Augmenta l'espai de contrasenyes per evitar els atacs de força bruta

Taula 5: Resum de les decisions de disseny

4.4.- Estudi de Viabilitat i de Costos

Si cerquem esquemes de contrasenyes comercials es fa difícil trobar un gran ventall. De tots els que s'han presentat a l'estat de l'art, sols hi trobem GrIDsure [11] i PassFace [12] que puguin ser adquirits com a solució independent. La resta dels esquemes sempre formen part d'una aplicació o del propi sistema operatiu. Per tant es fa difícil de comparar l'esquema presentat, a nivell econòmic, amb altres propostes.

A nivell tècnic, l'aportació de la proposta presentada té els seus punts forts en l'èmfasi en la seguretat davant del robatori del dispositiu, utilitzant una contrasenya gràfica de tipus mixt reconeixement – recordatori, sense deixar de banda la robustesa davant d'atacs de captura.
Crec que a dia d'avui és important comptar amb la seguretat necessària per fer front a un robatori del dispositiu i amb permisos d'administrador per part del lladre, cosa que permet esquivar les mesures de seguretat del propi sistema operatiu. De res serveix disposar d'un esquema molt robust de contrasenyes, si l'atacant pot accedir a la base de dades i modificar-la, o bé obtenir la codificació de la contrasenya i facilitar un atac per força bruta.

Així com els mètodes basats en recordatori solen utilitzar funcions resum o secrets compartits per emmagatzemar la contrasenya al dispositiu (atacables mitjançant força bruta o diccionari), els mètodes basats en reconeixement necessiten algun mecanisme que permeti conèixer les imatges que l'usuari ha triat com a contrasenya abans que res, ja que aquestes s'hauran de mostrar per pantalla. Això fa que o bé s'emmagatzemi en clar aquesta informació o bé la possible encriptació sigui dèbil, ja que no hi ha interacció prèvia amb l'usuari.

Per contra les contrasenyes dels mètodes basats en recordatori són difícil de memoritzar (*memorability*) al llarg del temps, i les que si que no presenten aquest problema, solen ser dèbils o directament són *pin-levels*. Així que considero que la unió d'aquests dos mecanismes com a esquema de contrasenya és un projecte viable i que intenta realitzar una petita aportació en l'àmbit de les contrasenyes gràfiques per a dispositius mòbils.

4.4.1.- *Costos de desenvolupament del projecte*

Veiem quins són els costos previstos per al desenvolupament de l'aplicació i per al disseny fet prèviament.

Concepte	Realitzador	Hores	Preu/Hora	Cost
Anàlisi de requeriments	Analista	10h	35€	350€
Disseny de l'aplicació	Analista	12h	35€	420€
Implementació de l'aplicació	Programador	40h	20€	800€
Cost eines de desenvolupament	-	-	-	0€
Compra de galeries d'imatges[5]	-	-	-	0€
Tria i gestió de les galeries	Dissenyador	10h	20€	200€
			Total	**1770€**

5 Hi han galeries GPL com pot ser http://es.fotolia.com

5.- Disseny de l'aplicació

Podem trobar tot el procés de disseny de l'aplicació documentat de forma molt més extensa a l'*Annex-3*. En els següents apartats es mostra una versió resumida per qüestions d'espai i de simplicitat en la documentació. Per a una visió més complerta us emplacem a la lectura d'aquest annex.

5.1.- Casos d'ús

La nostra aplicació interactuarà amb dos actors. El primer d'ells serà l'usuari encarregat d'autenticar-se, i el segon, l'aplicació que requerirà l'autenticació de l'usuari.

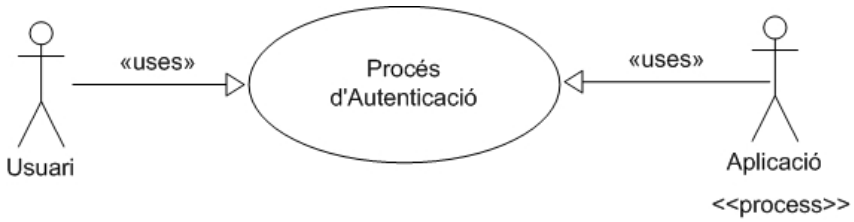


Fig. 7: **Diagrama de casos d'ús de nivell 1**

Si baixem de nivell en l'esquema anterior, podem dividir les accions i processos amb els que interacciona cadascun dels actors en dos: procés de registre i procés d'autenticació.

Veiem els casos d'ús del procés de registre:

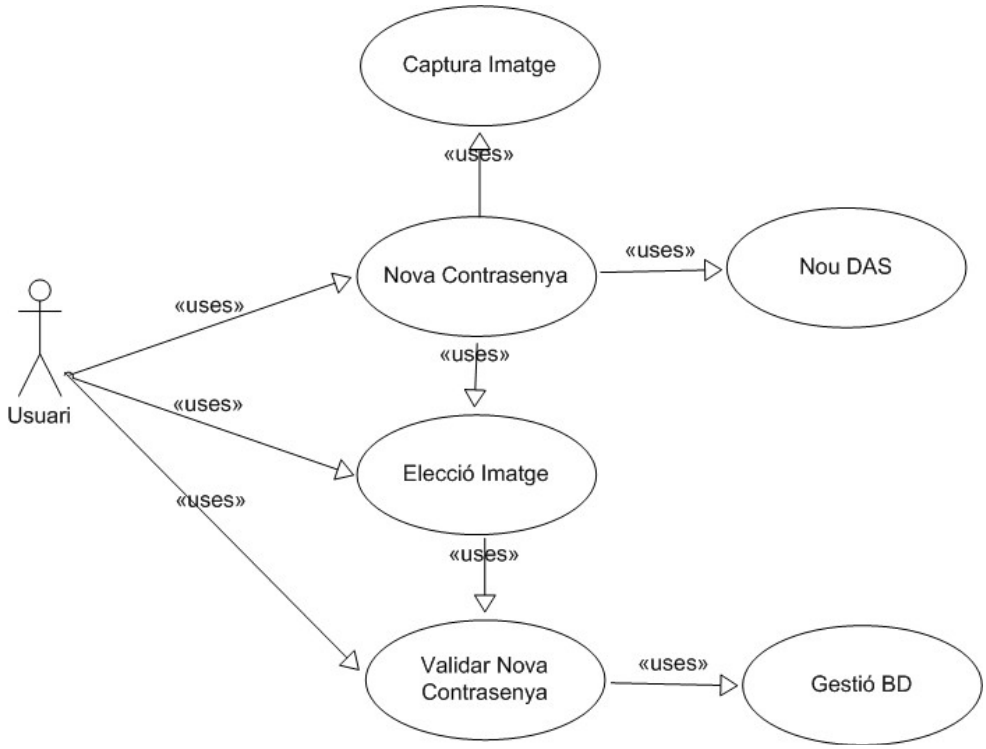


Fig. 8: **Casos d'ús del procediment de registre**

L'usuari iniciarà el procés de registre amb l'opció de "*Nova Contrasenya*". Aquest procés li farà reescriure una nova traça, que serà el nou secret compartit DAS ("*Nou DAS*"), o bé li permetrà capturar fotografies per incloure-les a la galeria d'imatges i poder-les seleccionar com a imatges de la seva contrasenya.

Un cop s'hagi dibuixat el nou DAS, l'aplicació cal que mostri un llistat amb totes les imatges que formen part de la galeria d'imatges capturades per l'usuari, que n'ha d'escollir dues ("*Escollir Imatges*"). Aquestes imatges seran distorsionades i separades en dues parelles cadascuna, com s'explica a l'apartat 2. Sols caldrà que l'usuari accepti el resultat ("*Validar Nova Contrasenya*") perquè totes aquestes dades siguin escrites a la base de dades.

Notem que el procediment de registre no es pot iniciar si l'usuari no s'ha autenticat correctament. Per facilitar les proves del prototipus a crear, si es detecta que la taula de la base de dades on s'emmagatzema la contrasenya resta buida, directament s'iniciarà el procés de registre. En un entorn real caldria dissenyar una aplicació que sols realitzés el procediment de registre de forma separada de l'autenticació[6].

Si veiem ara el procediment d'autenticació, tenim el següent diagrama de casos d'ús:

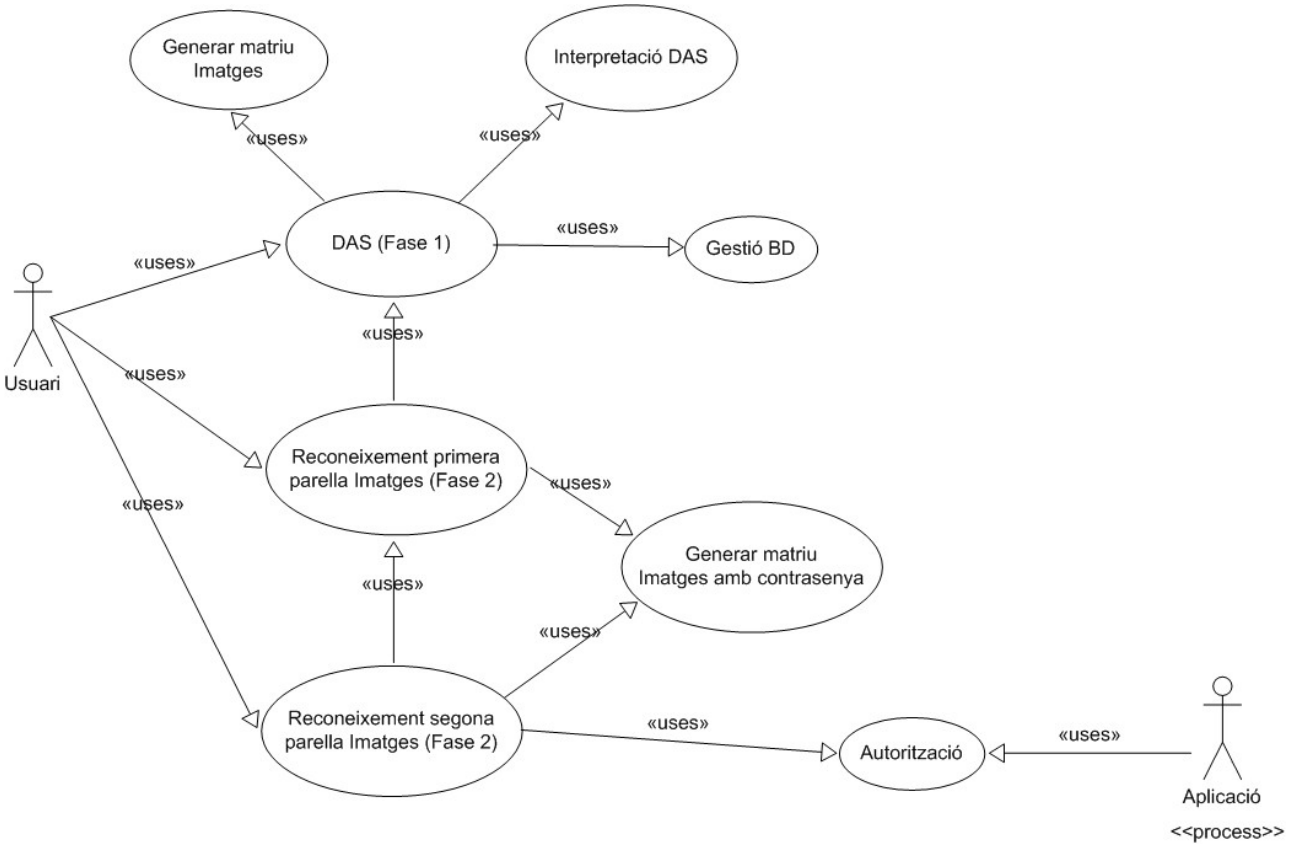


Fig. 9: **Casos d'ús del procediment d'autenticació**

En aquest diagrama es pot veure que l'usuari iniciarà el procediment realitzant una traça amb el seu DAS, que és el que defineix la fase 1. Per a realitzar la traçada s'utilitzarà una pantalla amb imatges aleatòries de fons per dificultar que un atacant pugui veure fàcilment la traça i a més aquestes serviran per delimitar cadascuna de les cel·les del DAS ("*Genera Matriu d'Imatges*").

6 Si un atacant amb permisos al sistema de fitxers elimina la base de dades, directament s'iniciarà el procés de registre, és obvi que això no ha de poder-se realitzar en un entorn real.

Un cop s'hagi realitzat la traça i s'aixequi el punter/dit de la pantalla es procedirà a extraure els punts que formaran la seqüència del DAS ("*Interpreta DAS*") i a l'hora la *passphrase* per a generar la clau que encripta la base de dades ("*Gestió BD*"), tal com s'ha explicat a l'apartat de Decisions de Disseny (Ap. 2).

Després es passarà a la fase 2, composta per dos procediments iguals, on s'escolliran les parelles que formen les dues imatges. Aquestes parelles són la partició de les dues imatges generades a l'acceptar un canvi de contrasenya en el procediment de registre ("*Validar nova Contrasenya*", vist a l'anterior diagrama). Per a poder escollir les parelles, primer cal generar les matrius d'imatges que es mostraran per pantalla, tenint en compte que si el DAS és correcte, s'hauran d'incloure les parelles d'imatges que formen la contrasenya ("*Generar Matriu d'Imatges amb Contrasenya*"). Recordem que l'elecció de les imatges s'ha de fer alhora (*multitouch*).

Si l'usuari realitza correctament la traça, s'haurà pogut desencriptar la base de dades, si a més ha escollit les dues imatges partides correctes, es farà una crida a la interfície de comunicació amb l'aplicació a protegir, indicant-li que l'autenticació ha estat satisfactòria.

5.2.- Disseny de la Base de Dades

En aquesta aplicació la base de dades serà molt simple, ja que sols cal mantenir les dades referents a l'autenticació de l'usuari. En el nostre cas tindrem dues seqüències d'imatges, per a les dues parts que formen el reconeixement de les imatges de la segona fase.

Notem que les imatges no s'emmagatzemen a la base de dades, ja que l'aplicació està pensada per a dispositius mòbils i les bases de dades típiques per aquests entorns no fan una gestió òptima de les dades multimèdia (*blob*). A més de cara a l'esquema de contrasenyes presentat és indiferent si les imatges estan emmagatzemades a la base de dades o bé al sistema de fitxers. Es pot pensar que emmagatzemant a la base de dades les imatges que formen la contrasenya, prèviament encriptades, l'esquema seria més segur, però llavors l'atacant que té accés al sistema de fitxers del dispositiu sols tindria que comparar les imatges que formen part del sistema de fitxers (galeria) i les que apareguin en la fase de reconeixement i no estiguin en la llista. Aquestes últimes serien les de la contrasenya (emmagatzemades a la base de dades), generant així un atac per intercepció.

Per aquesta raó s'ha optat per codificar a la base de dades simplement l'identificador del fitxer de les imatges escollides, i aquestes restaran al sistema de fitxers juntament amb la resta de les imatges que formen la galeria d'imatges.

També tindrem una taula on s'emmagatzemaran diferents estadístics per a la fase de proves de l'aplicació.

La nostra aplicació utilitzarà un sistema gestor de bases de dades relacional típic dels dispositius mòbils, com pot ser SQLite3. Aquests sistema gestor no proporciona cap mesura de seguretat, per això és la pròpia aplicació l'encarregada d'encriptar les dades amb AES-128, com ja s'ha comentat a l'apartat de Decisions de Disseny (Ap. 2). Existeixen extensions del propi sistema gestor que permeten treballar amb dades encriptades de forma transparent, però o bé són de pagament o bé compliquen la instal·lació dels *drivers* per a controlar la base de dades des de l'aplicació (a més de deixar la seguretat en mans de tercers).

Veiem el disseny de la nostra base de dades:

Fig. 10: **Disseny de la base de dades**

La taula *up*[7] s'encarregarà de mantenir les imatges i l'identificador de cada usuari. Estarà composta pels següents camps:

- **_id**: Camp autonumèric utilitzat com a clau primària. Aquest camp és necessari perquè el *driver* encarregat de comunicar el sistema gestor amb l'aplicació funcioni correctament.

- **up**: Camp de tipus *varchar* amb l'identificador de l'usuari, aquest camp tindrà la restricció *unique*, és a dir, que no es pot repetir el seu valor a la taula. Notem que aquest camp és realment el que hauria d'actuar com a clau primària.

- **imSeq**: Camp de tipus *varchar* on emmagatzemarem el nom dels dos fitxers que formen la primera parella d'imatges de la fase 2 (reconeixement). El format de la cadena serà [*nomFitxer1A* | *nomFitxer2A*].

- **imSeq2**: Camp de tipus *varchar* on emmagatzemarem el nom dels dos fitxers que formen la segona parella d'imatges de la fase 2 (reconeixement). El format de la cadena serà [*nomFitxer1B* | *nomFitxer2B*].

L'aplicació estarà pensada per a gestionar un nombre indeterminat d'usuaris, però sembla evident que en un dispositiu mòbil és difícil que existeixi més d'un usuari real. L'aplicació presentada no realitzarà la identificació de l'usuari, ja que això es podria fer amb algun tipus de dada personal resident al propi dispositiu (targeta de comunicació, dades del sistema *Android OS*, ...)

La taula *statistic* manté dades estadístiques referents a cada autenticació amb l'esquema presentat:

- **_id**: Camp autonumèric utilitzat com a clau primària. Aquest camp és necessari perquè el *driver* encarregat de comunicar el sistema gestor amb l'aplicació funcioni correctament.

- **date**: Camp de tipus *varchar* amb la data en que s'ha capturat la dada estadística.

- **time**: Camp de tipus *float* amb la duració en segons (precisió de ms) de tot el procés d'autenticació.

- **auth**: Camp de tipus *smallint* que indica si l'usuari s'ha pogut autenticar satisfactòriament (1) o no (0).

- **longTrace**: Camp de tipus *smallint* que indica la longitud de la traça realitzada.

Podrem trobar l'esquema SQL complert de la base de dades a l'*Annex-3*.

7 *User-Password*

5.3.- Disseny de les pantalles

En aquest apartat es descriurà el funcionament de les pantalles de que consta l'aplicació. Al següent apartat es podrà veure un diagrama de seqüència per comprovar l'ordre d'ús de cadascuna. Cal esmentar que totes les pantalles implementades sobre *Android OS* estan formades per dos components: una classe *java* que hereta d'*Activity* i fa de controlador, i una vista definida sobre XML, que es coneix com a *layout*. El *layout* és mostrat per l'*Activity* amb una acció pròpia que s'anomena *inflate*, i conté la definició i distribució de tots els components que apareixeran a la pantalla del dispositiu.

En aquest *layout* es poden utilitzar classes estàndard com a controladors de components d'interfície gràfica o bé controladors i components propis, com poden ser les definicions basades en la classe *surfaceView*. Tot això s'explicitarà amb més detall en la secció del disseny de les classes (Ap. 5.5).

Com sempre, per a un major detall veure l'*Annex-3*.

5.3.1.- *p1Activity*

Aquesta primera pantalla ens mostrarà una matriu d'imatges que no tenen cap altra funcionalitat que distraure a un possible atacant per *shoulder-surfing*. Sobre aquesta matriu, l'usuari haurà de traçar, mitjançant el desplaçament del dit sobre la pantalla, un recorregut per les diferents cel·les de la matriu mostrada. Aquest recorregut serà la contrasenya *DAS* que es farà servir per desencriptar la base de dades, com ja s'ha comentat anteriorment. Notem que el recorregut sols pot contenir una traça, és a dir, en el moment en que s'aixequi el dit de la pantalla es donarà per acabat el traç i es passarà a executar *p2Activity*.

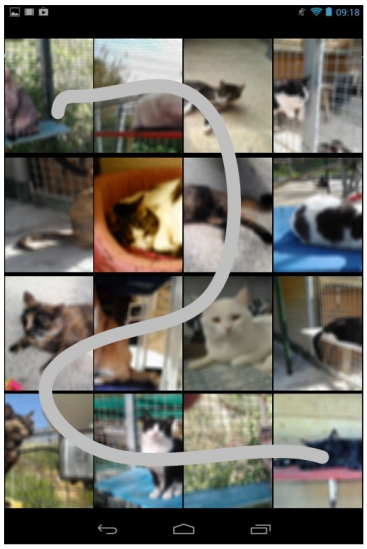


Fig. 11: Pantalla *p1Activity*

Per a dificultar el possible atac per *shoulder-surfing* la pantalla farà la traçada amb un color i una intensitat baixa, a més de fer desaparèixer la part inicial de la traça conforme s'avança en la realització d'aquesta.

Aquesta pantalla estarà definida al *p1_layout.xml* i farà un ús intensiu de les classes de creació pròpia *interficieCanvas* i *interficieCanvasThread*.

5.3.2.- *p2Activity*

A la pantalla de *p2Activity* apareixerà de nou una matriu amb imatges, de les quals s'hauran de seleccionar dues a l'hora, fent ús de la propietat *multitouch* del dispositiu. A aquesta pantalla hi arribarem encara que el *DAS* no sigui el correcte, ja que així un possible atacant no sabrà en quina part de l'esquema de contrasenya falla.

Les imatges són aleatòries, igual que la seva posició a la pantalla. No es mostrarà cap indicació de quines imatges han estat escollides per evitar l'atac de *shoulder-surfing*. També cal esmentar que les imatges es mostren amb cert nivell de distorsió per tal d'evitar que es puguin descriure fàcilment i així evitar l'enginyeria social (veure [17]).

A la imatge següent es mostra la pantalla i quines han estat les dues imatges seleccionades:

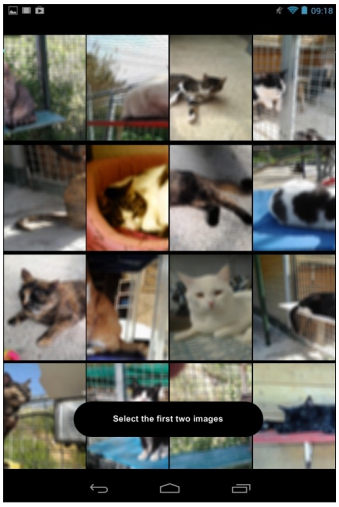


Fig. 12: Pantalla *p2Activity*

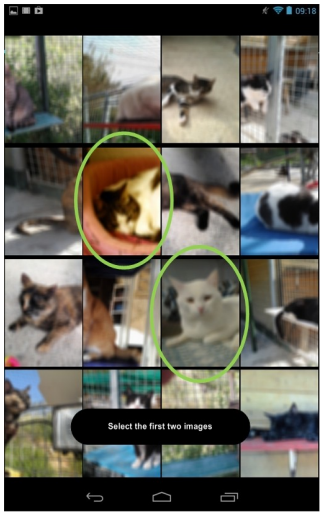


Fig. 13: Pantalla *p2Activity* amb imatges seleccionades

Aquesta pantalla estarà definida al *p2_layout.xml* i farà un ús intensiu de les classes de creació pròpia *interficieCanvas2* i *interficieCanvasThread2*.

5.3.3.- *p3Activity*

A la pantalla de *p3Activity* tornaran a aparèixer noves imatges, de les quals s'hauran de seleccionar les dues parelles de les seleccionades a *p2Activity*. S'haurà de tornar a fer alhora, fent ús de la propietat *multitouch* del dispositiu. Ja s'ha comentat en el procediment d'autenticació que el fet de dividir les imatges sent pròximes i conegudes per l'usuari no afegeix dificultat per recordar-les. També a aquesta pantalla hi arribarem encara que el *DAS* no sigui el correcte, ja que així un possible atacant no sabrà en quina part de l'esquema de contrasenya falla.

No es mostrarà cap indicació de quines imatges han estat escollides per evitar l'atac de *shoulder-surfing*.

Aquesta pantalla estarà definida al *p3_layout.xml* i farà un ús intensiu de les classes de creació pròpia *interficieCanvas3* i *interficieCanvasThread3*.

5.3.4.- *p4Activity*

Si arribem a la pantalla de p4Activity voldrà dir que el procés d'autenticació s'ha realitzat amb èxit. En aquest moment s'hauria de presentar la primera pantalla de l'aplicació a la que s'hi aplica l'esquema de contrasenya dissenyat. En el nostre cas hi apareixerà *p4Activity* que és la pantalla encarregada de gestionar l'assignació d'una nova contrasenya a l'usuari (tema tractat en el disseny dels casos d'ús, Ap. 5.2).

A part del botó per a sortir de l'aplicació, també hi trobarem un botó que ens permetrà afegir fotos nostres per a ser usades com a contrasenya (*Add Photo*) i el botó que ens permetrà canviar la contrasenya (*New Password*). El primer d'ells ens portarà a la *photoActivity* mentre que el segon ho farà a *reg1Activity*.

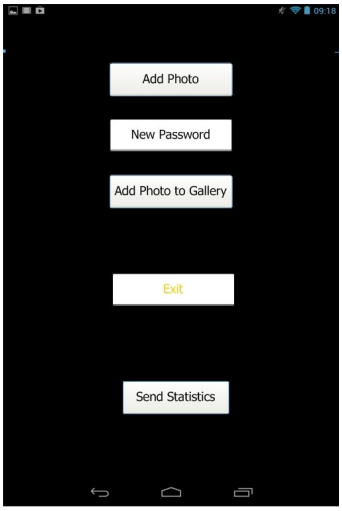


Fig. 14: Pantalla de *p4Activity*

S'han afegit, respecte la fase de disseny, dos nous botons: *Add photo to Gallery*, que ens permetrà afegir fotos nostres a la galeria d'imatges, mitjançant *galleryActivity*, i un botó per a enviar dades estadístiques, *Send Statistics*. Aquesta pantalla estarà definida al *p4_layout.xml*.

5.3.5.- *photoActivity*

Aquesta pantalla estarà pensada per a mostrar la imatge captada per alguna de les càmeres que té el dispositiu (si no en té mostrarà un missatge indicatiu). Sobre aquesta imatge es definirà un nou *layout* que contindrà sols un botó (*camera_layer.xml*). Aquest apareixerà sobre la imatge captada per la càmera i servirà per indicar quan capturar la fotografia.

Fig. 15: **Pantalla de *p4Activity***

Aquesta pantalla estarà definida al *photo_layout.xml*. Un cop captada la fotografia es tornarà automàticament a *p4Activity*.

5.3.6.- *reg1Activity*

La pantalla reg1Activity permet que l'usuari registri o torni a definir la traça que farà de *DAS*. Molt similar en quant a característiques a la *p1Activity* però en aquest cas sense la necessitat de mostrar imatges de fons per a distreure un possible atacant. S'entén que l'usuari ja posarà mitjans que evitin la captura de la seva contrasenya, en el moment de la definició. Un cop acabada la traça, aquesta pantalla ens portarà fins a *reg2Activity*. La pantalla *reg1Activity* restarà definida al *r1_layout.xml* i farà un ús intensiu de les classes pròpies *interficieCanvas4 i interficieCanvasThread4*.

5.3.7.- *reg2Activity*

La pantalla reg2Activity permet escollir les imatges que formaran part de la contrasenya. En ella simplement hi apareixerà una llista amb totes les imatges que es troben al directori d'imatges de càmera. Notem que aquesta llista no inclourà les imatges que formen part de la galeria d'imatges de la pròpia aplicació.

Això es fa així perquè l'usuari sols pugui utilitzar imatges captades per ell amb la càmera com a part de la contrasenya. El fet d'utilitzar una imatge generada i coneguda per ell augmenta la facilitat de memorització d'aquesta (veure Decisions de Disseny, Ap.2).

Fig. 16: Pantalla *reg2Activity*

En aquesta pantalla caldrà escollir dues imatges, ja que recordem que la contrasenya està composta per dues parelles d'imatges dividides. Un cop seleccionades, a la següent pantalla (*reg3Activity*), l'usuari podrà decidir si realment accepta l'elecció com a nova contrasenya.

La pantalla *reg2Activity* estarà definida al *r2_layout* i farà ús de les llistes amb *scroll* pròpies del SDK d'*Android OS*. Per a poder utilitzar aquest tipus de control caldrà definir un nou *layout* per a cada filera. En el nostre cas, aquest *layout*, anomenat *row_item*, estarà compost per una imatge i pel nom del fitxer que la conté.

5.3.8.- *reg3Activity*

La pantalla reg3Activity mostrarà el resultat de difuminar (*blur*) i dividir en dues parts les imatges escollides a l'anterior pantalla. Si l'usuari està conforme amb l'elecció podrà acceptar el canvi de contrasenya o bé cancel·lar-lo. Tant en un cas com en un altre tornarem a la pantalla d'inici del registre (*p4Activity*). Aquesta pantalla estarà definida a *r3_layout.xml*.

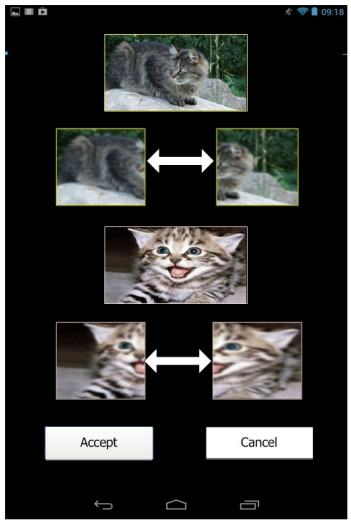


Fig. 17: Pantalla *r3Activity*

5.3.9.- gallery*Activity*

La pantalla galleryActivity és idèntica a la mostrada a *reg2Activity*, i ens permetrà escollir quina fotografia feta amb la càmera volem que passi a formar part de la galeria d'imatges. Sols caldrà escollir una en aquest cas.

5.4.- Seqüència de pantalles

En la següent imatge (fig. 18), es mostra la seqüència en la que apareixeran les diferents pantalles de l'aplicació, ja comentades en l'apartat anterior.

Al gràfic es poden diferenciar dues fileres: a la superior hi apareix la seqüència d'autenticació, on l'usuari introdueix la seva traça (primera fase) i després selecciona els dos parells d'imatges que formen part de la seva contrasenya (*p2Activity* i *p3Activity,* formen la segona fase).

A la filera inferior hi apareix el procés de registre o de canvi de contrasenya, on es poden captar amb la càmera les noves imatges candidates a ser contrasenya o bé es pot passar a generar una nova contrasenya, definint un nou *DAS* i escollint les dues parelles d'imatges noves.

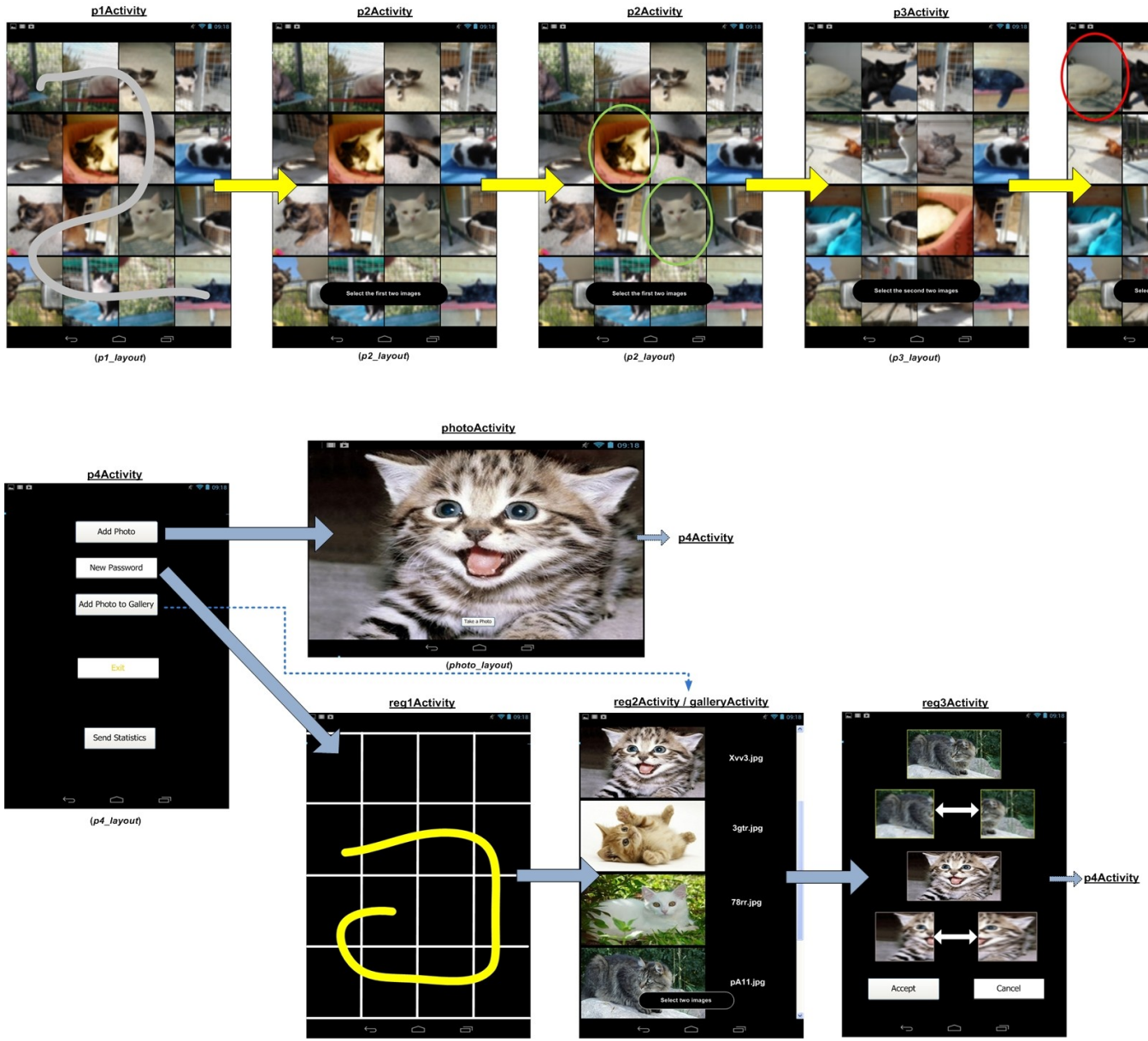


Fig. 18: Seqüència de pantalles

5.5.- Disseny de les classes

L'aplicació estarà dividida en 6 paquets, cadascun amb una funcionalitat concreta. Abans de passar a descriure'ls per separat veiem quins són (explicats molt detalladament a l'*Annex-3*):

tfmApplication: Paquet on hi trobarem les classes que formen el flux de l'aplicació i que actuen majoritàriament com a controladors.
bdController: Classes encarregades de comunicar l'aplicació amb la base de dades.
sfController: Classes encarregades de gestionar i interactuar amb el sistema de fitxers.
cripto: Classes encarregades d'implementar els algoritmes criptogràfics utilitzats per l'aplicació.
imageTools: Classes encarregades del tractament de les imatges.
canvas: Classes que implementen interfícies gràfiques no estàndards i pròpies de l'aplicació.

Al mateix *Annex-3* també hi podrem trobar detallats els diagrames de seqüència per veure com interaccionen els diferents objectes que conformen l'aplicació, entre si.

5.5.1.- *tfmApplication*

Dintre d'aquest paquet hi trobarem les classes que representen el flux de l'aplicació. El SDK de *java* per a *Android OS* està basat en *Intents*. Els *Intents* no són més que la voluntat de realitzar una acció. Una d'aquestes accions és l'inici d'una nova *Activity*. Les *Activity*'s són els controladors de cadascuna de les vistes (pantalles) d'una aplicació *Android OS*, com bé s'ha comentat a l'apartat 5.3. Recordem que una *Activity* està associada a una vista que resta definida en un fitxer XML anomenat *layout*.

Gran part de les classes del paquet *tfmApplication* seran herència de la classe *Activity* del SDK d'*Android OS,* i representaran les diferents pantalles de l'aplicació. Moltes d'aquestes pantalles tenen components gràfics que es crearan especialment per a aquesta aplicació (paquet *canvas*).

Dintre d'aquest paquet cal destacar la classe *configuration*. Aquesta classe serà l'encarregada de mantenir en memòria els paràmetres globals necessaris per a l'aplicació i seguirà un patró de tipus *singleton*. Aquest patró permet crear una instància de la classe en el seu primer ús i mantenir-se en memòria mentre s'estigui executant l'aplicació.

La resta de classes que formaran el paquet heretaran de la ja comentada classe *Activity* i per tant heretaran tots els mètodes encarregats de gestionar el flux d'execució així com les vistes associades.

A l'*Annex-3* podem veure el diagrama de classes complert del paquet *tfmApplication* i la seva interacció. La nostra aplicació tindrà 8 activitats, les quatre primeres destinades al procés d'autenticació, una altra destinada a la captura d'imatges amb la càmera i les tres restants del procés de registre, tal com s'ha explicat en el Disseny de les Pantalles (Ap. 5.3).

El flux d'execució també s'ha explicat en el disseny de les pantalles i estarà gestionat per les crides *nextActivity,* a excepció de les *Activity*'s on la pròpia vista gestiona el pas a una altra pantalla (*photoActivity* i *p4Activity*).

5.5.2.- *canvas*

Dintre d'aquest paquet hi trobarem les classes que implementaran components gràfics específics de la nostra aplicació.

La classe principal del paquet és la *interficieCanvasGenerica*, aquesta classe heretarà de la classe *SurfaceView* del SDK, que permet crear components gràfics dibuixables (*drawable*) des de la pròpia aplicació (anomenats tècnicament *canvas*). També implementarà la interfície *SurfaceHolder.Callback* que li permet detectar i capturar events generats per la pantalla tàctil.

Els atributs d'aquesta classe són referents al nombre d'imatges que cal mostrar per pantalla (*imagesDisplay*), el nombre d'imatges que forma la galeria (*imagesNumber*) o bé el tamany de la pantalla (*widthCanvas* i *heightCanvas*).

Les classes que implementen els components utilitzats en cada vista (veure apartat anterior) heretaran d'*interficieCanvasGenerica* i hauran d'implementar els mètodes següents:

- *surfaceCreated*: Encarregat de crear els gràfics que apareixeran al *canvas*
- *onDraw*: Encarregat de dibuixar el *canvas* i mostrar els components gràfics necessaris
- *onTouchEvent*: Captura i tractment dels events generats per la pantalla tàctil

Notem que cada classe de tipus *interficieCanvasX* té un *thread* (*interficieCanvasThreadGenerica*) associat. Aquest procés de baix pes serà activat dintre del mètode *surfaceCreated* un cop s'hagin creat tots els components del *canvas*. La seva funció és cridar de forma cíclica al mètode *onDraw* per tal de refrescar la pantalla. Al ser un *thread* independent s'evita que en cas d'error es col·lapsi tot el sistema operatiu.

Comentem algunes de les classes que són peces claus en la implementació directa de l'esquema de contrasenyes proposat.

interficieCanvas

Aquesta classe implementarà el *canvas* encarregat de mostrar una matriu aleatòria d'imatges d'entre totes les que es troben al *galleryDir*. Com ja s'ha comentat aquestes imatges sols serviran per marcar les cel·les per les que l'usuari farà la traça del DAS i alhora per dificultar que un atacant pugui veure fàcilment el recorregut d'aquesta traça.

Quan es detecti l'event *action_up*, és a dir, que es deixi de fer la traça, el vector de punts serà enviat al *dasController*, que és la classe encarregada d'obtenir l'ordre dels *quadrants* per on ha passat la traça de l'usuari. El mètode *dasController.obtePassword* ens retornarà directament una funció resum de tipus *sha-512* amb la llista cronològica dels quadrants que formen la traça.

Quan la traça hagi estat feta, es farà la consulta a la base de dades, ja que amb la traça ja podem obtenir la clau AES-128 que l'encripta (veure Decisions de Disseny, Ap. 2). Si l'autenticació és correcta obtindrem la seqüència d'imatges que formen la contrasenya i que hauran de ser mostrades en la segona fase.

interficieCanvas2* i *interficieCanvas3

Aquests dos components gràfics ens permetran mostrar la matriu d'imatges per tal que l'usuari pugui escollir les que formen la contrasenya. Si la fase 1 de l'autenticació ha estat correcta, carregarem en posicions aleatòries les parelles d'imatges que formen la contrasenya, acompanyades d'altres imatges de la galeria, sense repetició. En cas de que la traça de la fase 1 no fos correcta es carregarien només imatges de la galeria d'imatges.

Programarem la captura d'events de la pantalla per a detectar l'event *multitouch*. En aquest cas es verificarà que les imatges seleccionades siguin les que formen la contrasenya. Si el procés d'autenticació és correcte caldria avisar a l'aplicació que utilitza l'esquema de contrasenyes perquè actués. Si l'autenticació no és correcta s'avisarà a l'usuari i es finalitzarà l'aplicació. Aquesta classe s'encarregarà d'emmagatzemar les dades estadístiques a la base de dades, al finalitzar el procés d'autenticació.

Afegir que trobarem el disseny de les classes amb tot detall a l'*Annex-3*.

5.5.3.- *imageTools*

Dintre d'aquest paquet trobarem les classes encarregades d'implementar els tractaments gràfics que rebran les imatges. Normalment els mètodes seran estàtics. Rebran com a paràmetre un objecte de tipus *Bitmap* que és capaç d'emmagatzemar informació gràfica i recodificar-la en diferents formats. Aquesta classe forma part del SDK d'*Android OS*. A destacar la classe *boxBlur* que implementa la distorsió de la imatge mitjançant el mecanisme de *box blur*[8]. Notem que aquest mecanisme calcula una convolució molt ràpida i més que suficient per al nostre objectiu de distorsionar la imatge per dificultar l'enginyeria social (veure Decisions de Disseny, Ap. 2).

5.5.4.- *cripto*

Aquest paquet conté les classes encarregades d'implementar els dos algorismes criptogràfics que utilitzarà l'aplicació: l'algorisme d'encriptació simètric AES-128 i la funció resum SHA-512. Tots els mètodes seran estàtics i faran ús de les classes criptogràfiques de *java* (*javax.crypto*). La justificació de l'ús d'aquests algorismes es pot trobar a l'apartat 2, de Decisions de Disseny.

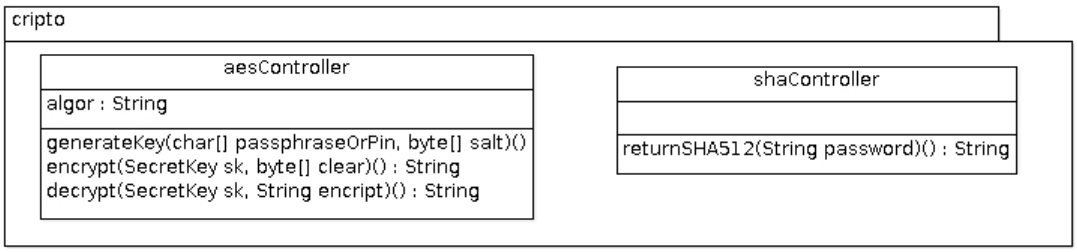


Fig. 19: **Diagrama de classes del paquet *cripto***

Veiem ara cadascuna de les classes que formaran aquest paquet.

8 http://en.wikipedia.org/wiki/Box_blur

aesController

Classe encarregada d'implementar l'algorisme AES. Per defecte el tipus de *cipher* que s'utilitzarà en l'algorisme serà la combinació `AES/CBC/PKCS5Padding` (veure l'*Annex-3* per a un major detall). L'algorisme AES utilitzarà codificació per intercanvi de blocs (CBC), així cada bloc dependrà del resultat de tots els anteriors. El *padding* utilitzat a la sortida de cada bloc serà de tipus PKCS5.

La classe disposarà dels següents mètodes públics:

- *generateKey*: Aquest mètode generarà una clau (*SecretKey*) a partir d'una *passphrase* i d'un *salt*. Recordem que a la nostra aplicació la *passphrase* serà la traça resultant de la fase 1 (DAS). El mecanisme implementat en aquesta funció serà `PBKDF2WithHmacSHA1`. Aquest mecanisme aplica a la *passphrase* una funció de tipus *HMAC* basada en un *hash* SHA-1. La clau obtinguda tindrà una longitud de 128 bits, per tant tindrem una clau de 32 caràcters hexadecimals (16 bytes). El nombre d'iteracions del procés per a derivar la clau serà de 500 passades (cercant el compromís entre velocitat i seguretat).

  La clau generada serà la que utilitzarem com a *SecretKey* per a les funcions d'encriptació i desencriptació mitjançant AES-128.

- *encrypt*: Funció que donada una clau i una cadena de bytes, encriptarà segons l'algorisme AES-128. Ja hem explicat anteriorment quin tipus de *cipher* farà servir.

- *decrypt*: Funció que donada una clau i una cadena de caràcters encriptada, desencriptarà segons l'algorisme AES-128. Ja hem explicat anteriorment quin tipus de *cipher* farà servir.

shaController

Aquesta classe implementarà, mitjançant el mètode *returnSHA512*, la funció resum SHA-512. Per fer-ho es farà ús de la classe *java MessageDigest*.

Aquest mètode s'utilitza per tal de generar una funció resum de la traça DAS i independitzar la longitud d'aquesta de la longitud de la *passphrase* que encriptarà la pròpia base de dades.

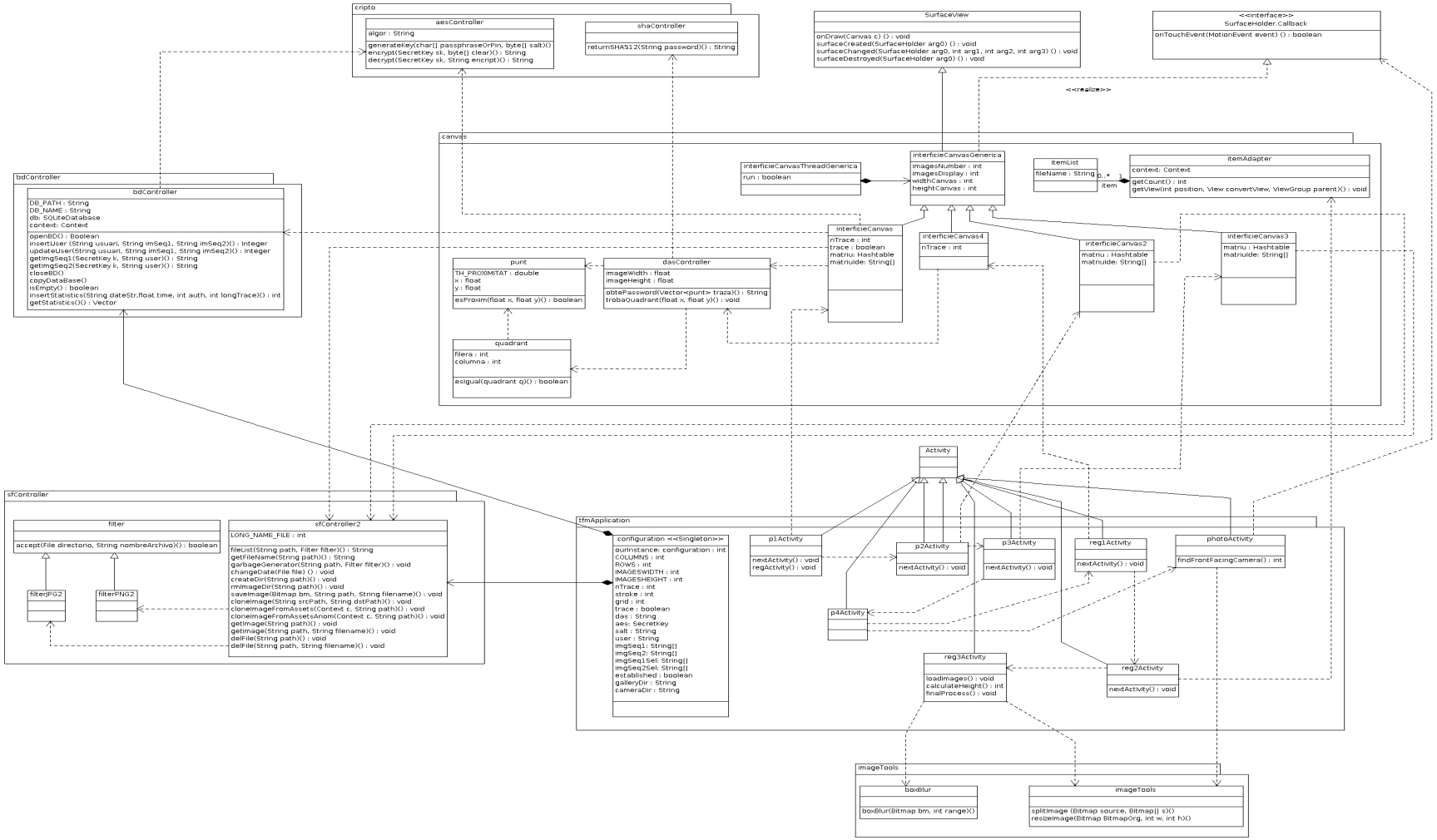
5.5.5.- *bdController*

Paquet encarregat de comunicar l'aplicació amb el sistema gestor SQLite3 (veure Disseny de la Base de Dades, Ap. 3.2). Sols contindrà la classe del mateix nom i que serà instanciada un sol cop i mantinguda en memòria mitjançant la classe *configuration* (sols si s'ha pogut obrir sense problemes la base de dades).

5.5.6.- *sfController*

Aquest paquet contindrà les úniques classes que podran interaccionar amb el sistema de fitxers del dispositiu. Totes les classes faran ús del paquet *java.io*. A destacar el mètode *garbageGenerator* que crea entropia a la galeria d'imatges per tal de confondre a un possible atacant (veure *Annex-3*).

5.5.7.- Diagrama complert de classes



6.- Implementació de l'aplicació

L'aplicació dissenyada ha estat implementada mitjançant l'eina *Android Studio 0.3.1* de Google sobre un sistema *Ubuntu Linux 12.04 x64*. Programada en el llenguatge *Java* versió 1.7.0, utilitzant el *SDK* d'*Android OS* versió 4.2.2. La compilació s'ha realitzat amb suport del *Gradle 1.8*.

L'aplicació ha estat dissenyada i implementada per a dispositius mòbils amb sistema operatiu *Android* 4.2.2 o superior, amb capacitat *multitouch*, càmera i pantalla de 4" o més. Per a la fase de proves s'ha utilitzat l'emulador d'*Android AVD* en diferents formats i els dispositius físics *Google Nexus 7* i *Samsung Galaxy Mini*.

L'aplicació ha estat signada i distribuïda per a fer proves en format *apk*. Al ser instal·lada requereix que l'usuari accepti donar-li permisos per a fer ús de la càmera i per a tenir accés a internet (per enviar les estadístiques).

L'aplicació requereix 3,71 MB d'espai per a ser instal·lada més l'espai necessari per augmentar la galeria d'imatges per part de l'usuari.

Trobarem perfectament detallat i documentada l'estructura de classes i el projecte *Android Studio* a l'*Annex-4*.

6.1.- Esquema de classes implementat

L'estructura del projecte dintre d'*Android Studio* és la següent,

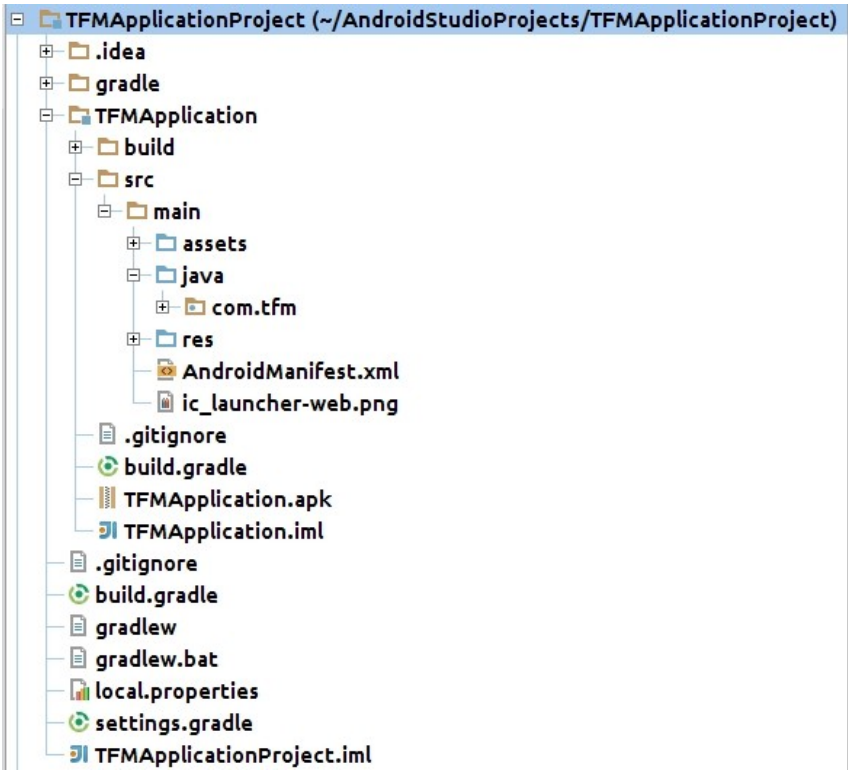


Fig. 20: Estructura del projecte

Totes les classes formen part del paquet *com.tfm*, seguint el següent esquema:

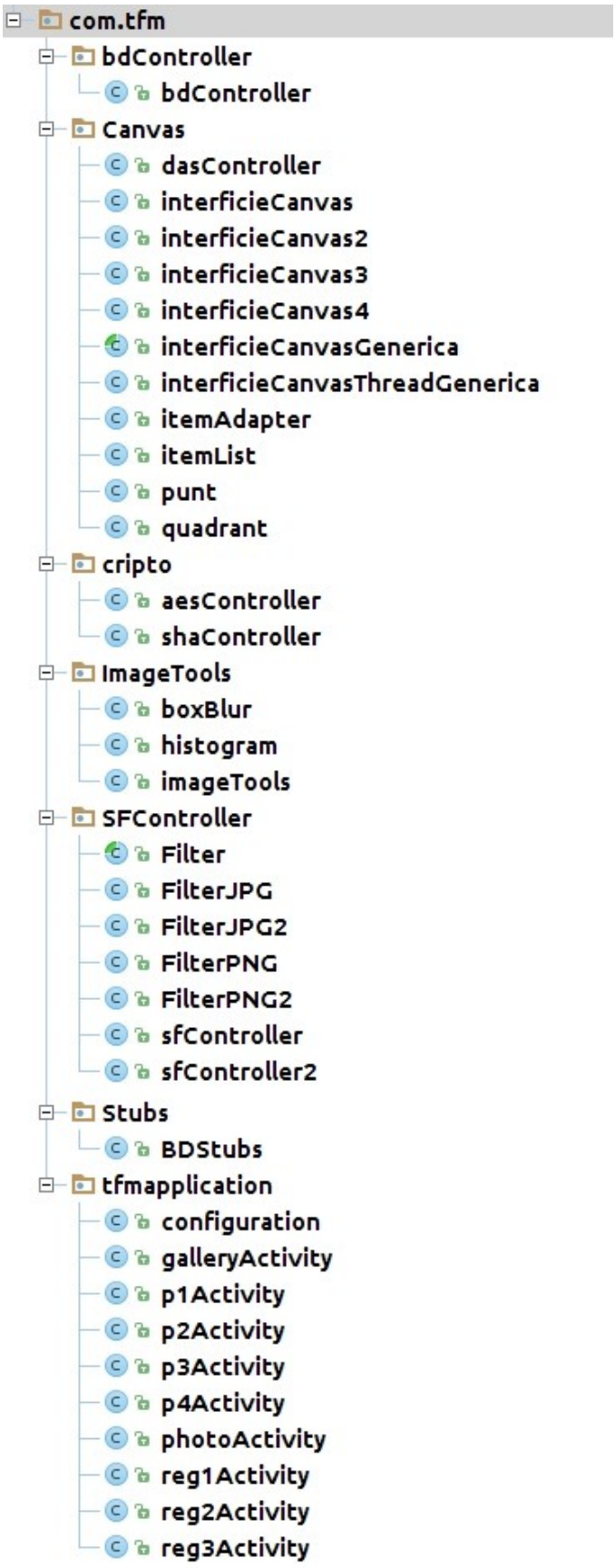


Fig. 21: Estructura de classes

Els *layouts* implementats, seguint les especificacions de disseny han estat els següents,

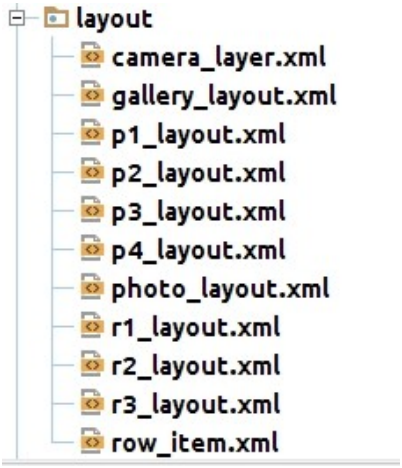


Fig. 22: *Layouts* implementats

6.2.- Classes que no formen part del disseny de l'aplicació

Si mirem la fig. 21 hi ha un conjunt de classes que no formen part del disseny de l'aplicatiu, però hi van formar part en algun moment del desenvolupament i per tant s'han mantingut en la solució final. Són les següents:

Stubs.BDStubs: Aquesta classe s'ha utilitzat a la fase de proves per tal d'inserir dades a la taula *up* de la base de dades i accelerar algun procediment, com per exemple, no haver de realitzar fotografies per canviar la contrasenya, o afegir directament un usuari amb contrasenya per a provar les fases d'autenticació.

imageTools.Histogram: En una branca paral·lela del desenvolupament de l'aplicació s'utilitzaven dades referents a l'histograma de les imatges per tal d'identificar les imatges que formaven la contrasenya i no haver d'encriptar la base de dades.

SFController.SfController: Primera versió del controlador del sistema de fitxers que permet treballar amb imatges esteganografiades, un solució que finalment ha estat descartada per fràgil.

SFController.FilterPNG i SFController.FilterJPG: Filtres de fitxers que permeten escollir fitxers sols per l'extensió, sense un format concret en el nom. Finalment l'aplicació utilitza un format estàndard per a nombrar els fitxers i s'han utilitzat les classes SFController.FilterJPG2 i SFController.FilterPNG2, molt més restrictives.

6.3.- La base de dades al sistema de fitxers

A l'aplicació implementada com a prototipus, si es detecta que la base de dades no existeix o és buida, es copia des del directori d'instal·lació *assets* i automàticament s'inicia el procediment de registre d'una nova contrasenya.

Això s'ha implementat així per facilitar la fase de proves, on diferents usuaris han de testejar

l'aplicació per poder extraure les conseqüents estadístiques. En un entorn real la base de dades sols hauria de ser creada per un procés independent i executat amb garanties de seguretat.

Si quan la base de dades és buida o inexistent (primera execució) l'aplicació passa directament al procediment de registre d'una nova contrasenya, un usuari amb privilegis sobre el dispositiu (*rootejat*) fàcilment pot eliminar la base de dades existent, forçant a l'aplicació a que li permeti generar una nova contrasenya i evitant la protecció que ofereix l'esquema implementat. Per aquesta raó, en un entorn real l'aplicació no haurà de permetre l'autenticació si la base de dades és buida o inexistent. Es podria pensar en dotar l'aplicació d'una base de dades amb una contrasenya per defecte, aplicable sols a la primera execució, però com que les claus per encriptar / desencriptar depenen del dispositiu, això no és factible.

Una altra solució per generar les dades de la base de dades passaria per implementar un protocol de tal forma que sols pugui ser creada en el moment d'instal·lació de l'aplicació, amb una contrasenya per defecte, però el sistema de distribució APK d'*Android OS* no contempla aquest protocol, igual que no contempla tampoc que un dispositiu pugui estar *rootejat*... Una altra possible solució és una arquitectura client-servidor on l'usuari pugui descarregar la base de dades mitjançant una contrasenya.

Per altra banda, si un usuari té privilegis d'administrador, pot realitzar un atac substituint la base de dades del propietari del dispositiu per una de seva, amb les dades que l'atacant ha generat amb la mateixa aplicació (una contrasenya creada i coneguda per ell).

Si aquesta nova base de dades l'ha creat mitjançant l'execució de l'aplicació en el seu propi dispositiu, l'atac no funcionarà, ja que la clau que encripta la base de dades depèn de la traça generada, però també d'un *salt* independent per a cada dispositiu, anomenat *secure.android_id*. Per tant a l'executar l'aplicació amb la base de dades canviada al dispositiu atacat no es podrà tornar a generar la mateixa clau d'encriptació.

Si la nova base de dades es vol generar al mateix dispositiu (per tal d'usar el mateix *salt*), s'haurà de desinstal·lar prèviament l'aplicació (no poden estar duplicades) i tampoc funcionaria l'atac. Així que l'única opció disponible és modificar el codi de l'aplicació per tal d'utilitzar el *secure.android_id* del dispositiu atacat o bé fer *spoofing*[9] d'aquest identificador al dispositiu de l'atacant.

És fa difícil lluitar contra aquesta situació ja que l'atacant és un usuari privilegiat i té control total sobre el dispositiu, per analogia és com deixar un servidor amb accés d'administrador en mans d'un usuari malintencionat. L'atacant no tindrà cap problema per obtenir el *secure.android_id* del dispositiu atacat i canviar-lo al seu dispositiu. Una possible solució passa per afegir un *pin* al *salt*, de tal forma que l'usuari l'hagi d'escriure a l'hora de fer servir l'aplicació, complicant encara més l'esquema presentat. Si l'atacant no coneix aquest *pin* no podrà generar una clau d'encriptació vàlida, encara que utilitzi el mateix *secure.android_id*.

9 http://www.strazzere.com/blog/tag/spoofing-android_id/

7.- Conclusions

Un cop l'aplicació ha estat implementada, és funcional i un grup d'usuaris hi ha estat realitzant proves, és el moment de fer balanç de quins són els trets més destacables de l'esquema de contrasenyes proposat. En aquest últim apartat de la documentació hi trobarem el resultat de les proves realitzades pels usuaris, que ens permetran extraure una primera idea de quin és el patró d'ús de l'esquema. Finalment es farà un resum de les conclusions i es proposaran un conjunt de millores o nous objectius per a un treball futur.

7.1.- Estadístiques de l'ús de l'aplicació

Durant el període d'un mes s'ha realitzat un petit estudi estadístic on han participat 5 persones[10] amb edats compreses entre els 17 i 45 anys. Tots els participants tenen coneixements tecnològics bàsics, a excepció de dos, que són enginyers informàtics. Donat que l'esquema presentat neix amb la idea de protegir una aplicació que requereixi un nivell mitjà de seguretat, és adequat escollir usuaris amb certs coneixements sobre l'ús de dispositius mòbils. L'aplicació s'ha instal·lat en els seus propis dispositius.

Tots els participants han rebut una explicació de menys de 5 minuts sobre l'ús de l'esquema presentat, però s'ha tutoritzat la primera elecció de contrasenya *in situ*.

Al finalitzar el període de prova cada participant ha enviat un conjunt de dades estadístiques (via email automàtic) que l'aplicació ha anat recollint a cada execució. Cada trama enviada té el següent format, pensat per a ser tractat mitjançant XML:

```
<id>1</id> <date>23/11/2013</date> <time>39.25</time> <auth>0</auth> <long>1</long>
```

Cada camp conté la següent informació:

- **id**: Identificador de l'execució, autonumèric.
- **date**: Data en que es va realitzar l'execució.
- **time**: Temps, en segons, que s'ha necessitat per a introduir la contrasenya complerta.
- **auth**: Indica si la contrasenya era correcta (1) o no (0).
- **long**: Longitud de la traça realitzada a la primera fase.

Amb aquestes dades, més un qüestionari sobre la utilització de l'esquema al final del període, es pretén conèixer:

- Comportament de la tassa d'encerts i la tassa d'errors al llarg del temps
- Evolució del temps dedicat al procés d'autenticació al llarg del temps
- Dades de la fase d'aprenentatge del procés d'autenticació

10 Nombre molt petit de participants però és el màxim de persones que he pogut motivar en tan poc temps i amb nul·la compensació. Serveixi aquest petit estudi per fer-nos una idea molt general de l'ús de l'esquema presentat.

Tassa d'errors al llarg del temps

S'ha demanat als diferents participants que durant un període de temps llarg no canviïn la contrasenya escollida i cada dia facin dos o tres intents d'autenticació.

A la fig. 23 podem observar l'evolució de les tasses d'error i d'encerts al llarg d'una mica més d'una setmana de proves. El període de temps és molt curt per a traure grans conclusions, però podem extraure les següents conclusions de la gràfica presentada:

- Al començar a fer ús del nou esquema de contrasenyes es produeixen més errors d'autenticació, possiblement deguts a la falta de pràctica en el procediment d'autenticació. A la gràfica es pot veure com fins al 19/11 no s'estabilitzen aquests paràmetres.

- A partir del quart / cinquè dia d'ús es produeix la màxima tassa d'encerts. Aquesta tassa baixa una mica i s'estabilitza al voltant del 85%. Segurament aquesta disminució és deguda a que amb el pas del temps, l'usuari pot anar oblidant la contrasenya.

- Caldria veure en un període més llarg de temps si la tassa d'encerts segueix baixant, per a poder valorar el paràmetre de *memorability* de l'esquema presentat.

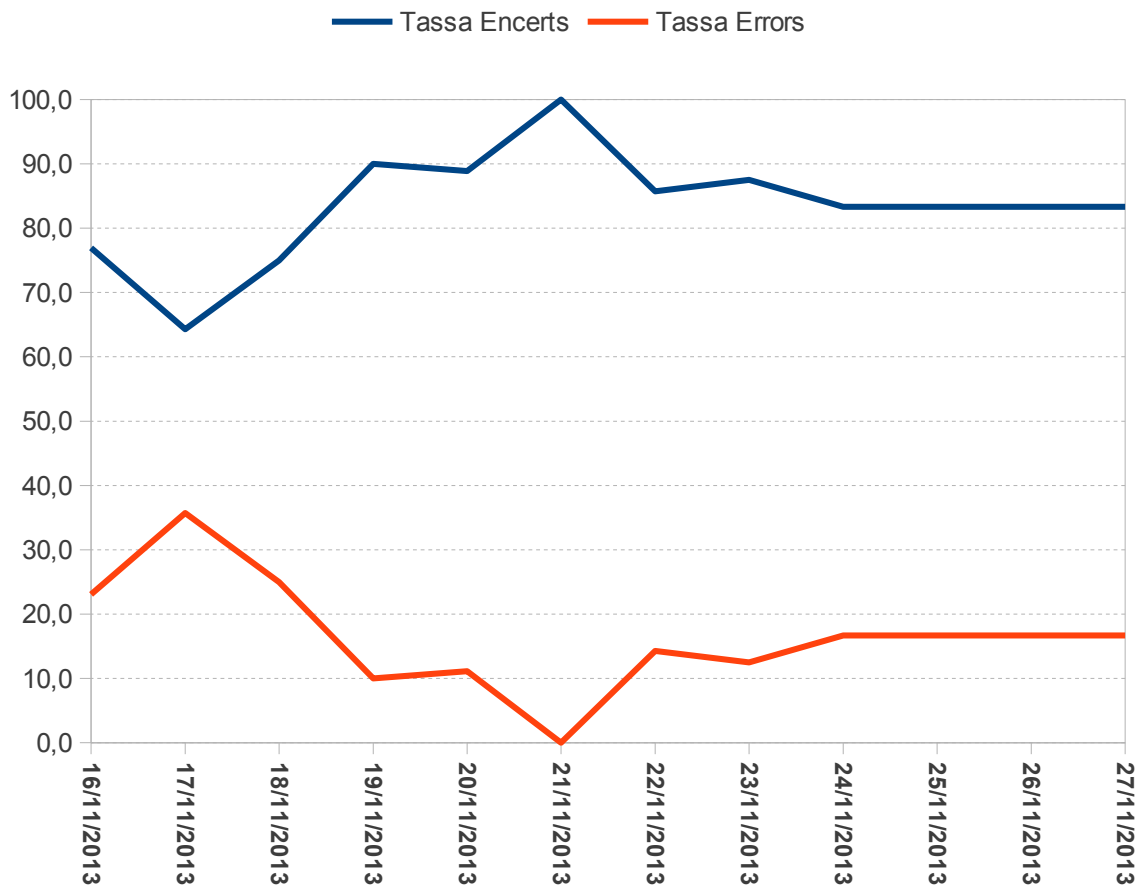


Fig. 23: Comportament dels errors i encerts al llarg del temps

Evolució del temps d'autenticació al llarg del temps

S'ha demanat als diferents participants que durant un període de temps llarg no canviïn la contrasenya escollida i cada dia facin dos o tres intents d'autenticació.

L'aplicació mesura la mitja del temps que es dedica a aquest procés cada dia, i es pot veure el resultat a la fig. 24, en mil·lisegons. Notem que tant es comptabilitza el procés correcte com l'erroni, i aquest últim cas fa que la mitja pugi.

En aquesta gràfica s'aprecien algunes discontinuïtats ja que no tots els usuaris han seguit les instruccions donades al peu de la lletra i alguns dies no han provat l'aplicació. Tot i això, amb els pocs participants que s'han aconseguit s'ha cregut convenient no descartar aquestes dades.

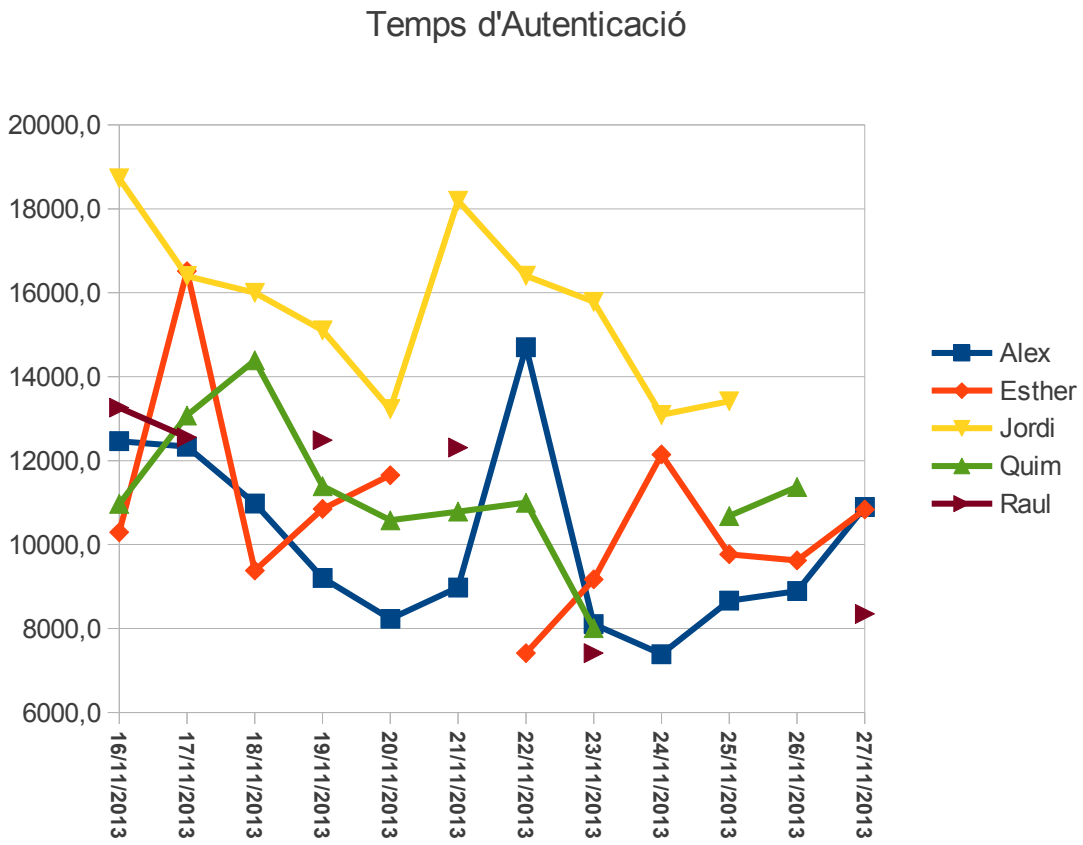


Fig. 24: Evolució del temps d'autenticació al llarg del temps

La veritat és que veient la gràfica es fa difícil extraure un patró de comportament concret. El més indicatiu és que gairebé tots els usuaris acaben convergint cap a un temps situat entre els 8,5 i els 11,5 segons. Caldrà valorar al següent apartat si això és molt o poc.

També es pot observar que 4 dels 5 usuaris han aconseguit realitzar tot el procés per sota dels 8 segons entre el setè i el novè dia de proves.

A part de l'oscil·lant fase d'aprenentatge, a la gràfica es pot veure que hi apareixen pics de temps, que es poden correlar amb l'augment de la tassa d'errors a partir del sisè dia. Una autorització errònia fa que pugi el temps dedicat al procés, ja que l'usuari no troba les imatges si la traça no es realitza correctament (veure apartat 7.1.1).

Dades de la fase d'aprenentatge del procés d'autenticació

De la fase inicial, on l'usuari ha d'aprendre com funciona l'esquema de contrasenyes presentat, es van recollir dades indicant a cada participant que anés provant diferents contrasenyes. Això es va realitzar durant un dia. Les dades més rellevants van ser les següents:

- Longitud mitjana de la traça en fase d'aprenentatge: 6,3 cel·les
- Tassa d'encerts en fase d'aprenentatge: 80%
- Tassa d'errades en fase d'aprenentatge: 20%
- Temps mitjà amb encert a la fase d'aprenentatge : 9500 ms
- Temps mitjà amb errada a la fase d'aprenentatge: 13457 ms

Aquestes dades són molt similars a les mostrades a les anteriors gràfiques per al primer dia (16/11). Es desprèn el que comentarem al següent apartat sobre l'augment de temps en cas d'errada de la traça, ja que l'usuari no troba les imatges escollides com a contrasenya, i també es pot inferir que les dades de la fase d'aprenentatge no són gaire distants de les dades finals, obtingudes al finalitzar el període de proves.

Donat el poc període de prova i els pocs participants és arriscat extraure conclusions, però podríem dir que el comportament general de l'esquema presentat té una tassa d'encert estable al llarg del temps propera al 80%, amb una duració mitjana d'entre 8 i 12 segons. La duració de la fase d'aprenentatge es pot considerar d'uns tres dies, segons el que es pot veure a la gràfica de la fig. 23.

7.1.1.- Problema de les traces diagonals

Un dels problemes detectats, gràcies al qüestionari final, és que els usuaris solen fallar gairebé sempre en la primera fase de l'autorització, és a dir, a l'hora de realitzar la traça. Això demostra que aquesta primera fase entranya més dificultat en quant a facilitat de memorització. Tal com ja s'havia comentat a l'estat de l'art, per defecte les contrasenyes de tipus recordatori són més difícils de recordar.

En termes generals, l'usuari que no falla al realitzar la traça no sol tenir problemes a l'hora de reconèixer les imatges. A més a la primera fase, la de la traçada, cal afegir-hi un problema extra, que és l'elecció d'una traça diagonal. Si la traça realitzada conté trams en diagonal, cal ser molt curós a l'hora de creuar les cel·les correctes, i en el cas d'una línia diagonal això no és trivial. Fer passar la línia uns píxels més amunt o més avall pot fer que la traça passi per una cel·la o per una altra, però visualment poden semblar traces idèntiques.

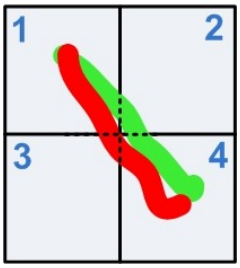


Fig. 25: La traça correcta (en verd) passa per les cel·les 1, 2, 4. La traça errònia (en vermell) passa per les cel·les 1, 3, 4. Visualment les dues traçades semblen correctes...

7.1.2.- Problema de l'error en la traçada

Quan la traçada feta per l'usuari és errònia l'esquema de contrasenya segueix amb la segona fase, tot i que no mostra obligatòriament les imatges que l'usuari ha escollit com a contrasenya. Si que ho pot fer escollint de forma aleatòria alguna d'aquestes imatges de la galeria. Si recordem que la traça forma part de la clau que encripta la base de dades és fàcil entendre el perquè no es poden mostrar les imatges de la contrasenya, ja que no es coneixen.

Si l'usuari no detecta el seu error en la traça queda "espectant" a la següent pantalla ja que no és capaç de reconèixer les imatges que formen la seva contrasenya. En aquests casos el més pràctic és escollir imatges qualsevol fins que l'esquema avisi de l'error en la contrasenya.

És fa difícil fer-li entendre a l'usuari que si en el moment en que es genera la traça errònia li indiquem, deixem reduït l'espai de contrasenyes a sols 76863 possibilitats (graella 4x4), donant-li moltes més facilitats a un possible atacant.

Una possible solució a aquest problema és dotar a la fase 2 d'un *time-out*.

7.2.- Conclusions finals i treball futur

Partíem amb l'objectiu inicial d'implementar un esquema de contrasenyes gràfiques robust al robatori de dades, ja sigui per captura o per robatori del propi dispositiu, sense sacrificar la facilitat d'ús i la facilitat de memorització (apartat 1.2). Arribats ara a la finalització del projecte es fa palès que mitjançant una aproximació successiva s'ha aconseguit arribar a la solució que equilibra tots aquests requeriments.

L'espai de contrasenyes aconseguit en la nostra proposta se situa entre els 2^{32} i els 2^{39}, amb longituds de traça entre 8 i 12 cel·les, força per sobre d'un *pin-level*, tal com es proposava als objectius inicials. Comparat amb un esquema gràfic de tipus recordatori, com pot ser DAS, amb traces amb més d'un cop (*stroke*) de longitud 12, s'obté un espai de 2^{50}, força més gran que el de la nostra solució, però cal fer notar que això implica que l'usuari del DAS esculli traces amb més d'un cop (obligatòriament), cosa que les fa rebuscades i de molt difícil memorització. Si comparem la nostra solució amb altres esquemes de tipus reconeixement (per exemple, *PassFace,* amb un espai de 2^{13}) obtenim un espai de contrasenyes força més gran, ja que molts d'ells són merament *pin-levels*.

Pel que fa a les estadístiques d'ús presentades a l'anterior apartat demostren que la tassa d'encerts al llarg del temps ronda el 85%, i que els usuaris, sense necessitat de cap sessió d'aprenentatge són capaços de dominar la metodologia proposada en 2 – 3 dies, depenen del seu domini tecnològic inicial. Val a dir que la convergència cap a la tassa d'error estable és més ràpida en els dos usuaris amb coneixements tecnològics superiors, i que en mitja la diferència entre aquests i la resta és d'un dia.

Per les impressions que ens han fet arribar aquest conjunt d'usuaris, la idea de que la contrasenya la formin imatges que ells mateixos han captat amb la càmera fa que siguin més fàcils de recordar, i alhora (amb una galeria d'imatges variada) fa que l'atacant tingui dificultats per predir quines són les possibles imatges que formen la contrasenya.

Els mateixos usuaris no han tingut cap mena de problema amb la distorsió de les imatges per dificultar el *shoulder-surfing*, en part gràcies al que s'ha comentat en l'anterior paràgraf, ja que les imatges escollides a la contrasenya són molt pròximes a ells.

Tot i això no podem deixar de banda un dels aspectes negatius de la solució proposada, que és el temps necessari per a realitzar tot el procés d'autenticació. Una mitja de 8 – 10s per a autenticar-se és un interval de temps massa elevat per a un esquema que aporta un espai de contrasenyes tan sols 2^6 vegades més gran que el d'un *pin-level* de quatre caràcters *ascii* (veure apartat 4.3). A això cal sumar-hi el desconcert que pot provocar-li a l'usuari el fet de no notificar el possible error en la traçada abans d'escollir les imatges (veure apartat 7.1.2). En un treball futur i sacrificant part de la seguretat aportada per l'esquema proposat, seria interessant mostrar-li a l'usuari un missatge indicant aquesta situació i aturant el procés d'autenticació, o bé afegir un *time-out* a tota la fase 2.

Alhora un dels punts forts de l'esquema aportat és l'èmfasi que s'ha fet en intentar lluitar contra el robatori de la contrasenya a partir del robatori del dispositiu. Els propis creadors dels sistemes operatius, per aquests tipus de dispositius, hi afegeixen robustos mecanismes de seguretat mitjançant permisos sobre el sistema de fitxers, per tal de que no sigui fàcil accedir a les dades de les aplicacions (base de dades amb contingut crític, com pot ser la contrasenya). Però mai es té en compte que el possible lladre pot obtenir privilegis d'administrador (dispositius *rootejats*).

La nostra proposta encripta les dades de la base de dades mitjançant un algorisme simètric fiable a nivell d'aplicació, i la clau és un secret compartit amb l'usuari. Fem notar que això no és una tasca fàcil si es fa ús d'un esquema de contrasenyes gràfiques de tipus reconeixement. És fàcil deduir que si l'usuari ha de reconèixer una imatge per poder validar-se, aquesta ha d'estar emmagatzemada al propi dispositiu, si més no per mostrar-li (no entrem en el terreny de validacions via xarxa). Si un possible atacant té accés total al sistema de fitxers li pot ser molt fàcil trobar quina és aquesta imatge. Per aquesta raó cal encriptar qualsevol dada que li permeti deduir la imatge escollida com a contrasenya. La clau que encripta aquestes dades no pot estar emmagatzemada tampoc al dispositiu, per la mateixa raó ja esmentada, així que cal buscar un mecanisme també gràfic perquè l'usuari la pugui generar en cada validació.

La nostra solució fuig d'una primera fase on s'utilitzi una contrasenya textual per a obtenir aquesta clau criptogràfica, proposant un algorisme de tipus DAS simplificat amb un sol cop (*stroke*). Això, com ja hem comentat, allarga el temps del procés, però no el dificulta excessivament.

També s'ha parlat a l'apartat 6.3 dels possibles atacs mitjançant canvi o generació d'una nova base de dades, en aquest cas, la nostra proposta dificulta un ample ventall d'aquests atacs, sols essent sensible a possibles *spoofings* dels identificadors propis de cada dispositius. Al mateix apartat es proposa alguna solució per aquest tema.

Finalment, i com a conclusions personals sobre el treball realitzat, afegir que la implementació d'un prototipus amb eines gratuïtes sobre el sistema operatiu *Android* ha permès anar una mica més enllà dels coneixements merament teòrics, i poder comprovar en primera persona que portar una idea a la pràctica sempre és més complex del que hom pot pensar a priori. Em sap greu haver deixat de banda, per manca de temps, haver pogut integrar l'esquema proposat dintre del propi sistema operatiu, un camp que resta obert de cara a treballs futurs.

8.- Referències

[1] J. Bentley and C. Mallows. “How much assurance does a PIN provide?”. In Baird and Lopresti, editors, Human Interactive Proofs (HIP), LNCS 3517, Springer-Verlag, p.111–126, 2005.

[2] M. D. Hafiz, A. H. Abdullah, N. Ithnin, and H. K. Mammi. “Towards identifying usability and security features of graphical password in knowledge based authentication technique”. In Second Asia International Conf. on Modelling & Simulation, p. 396–403, 2008.

[3] R. Biddle, S. Chiasson, P.C. Van Oorschot. “Graphical Passwords: Leraning from the First Twelve Years”, ACM Comput. Surv., Vol. 44, No. 4, 2012.

[4] I. Jermyn, A. Mayer, F. Monrose, M. Reiter, and A. Rubin. “The design and analysis of graphical passwords”. In 8th USENIX Security Symposium, 1999.

[5] J. Thorpe and P. C. van Oorschot. “Graphical dictionaries and the memorable space of graphical passwords”. In 13th USENIX Security Symposium, 2004.

[6] P. Dunphy and J. Yan. “Do background images improve “Draw a Secret” graphical passwords?”. In 14th ACM Conference on Computer and Communications Security (CCS), 2007.

[7] H. Gao, X. Guo, X. Chen, L. Wang, and X. Liu. “Yagp: Yet another graphical password strategy”. In Annual Computer Security Applications Conference, 2008.

[8] J. Goldberg, J. Hagman, and V. Sazawal. “Doodling our way to better authentication (student poster)”. In ACM Conference on Human Factors in Computing Systems (CHI), 2002.

[9] R. Weiss and A. De Luca. “PassShapes – utilizing stroke based authentication to increase password memorability”. In NordiCHI, pages 383–392. ACM, 2008.

[10] H. Tao and C. Adams. “Pass-Go: A proposal to improve the usability of graphical passwords.” International Journal of Network Security, p.273–292, 2008.

[11] GrIDsure. GrIDsure corporate website. http://www.gridsure-security.co.uk, últim accés Setembre 2013.

[12] Passfaces Corporation. The science behind Passfaces. http://www.passfaces.com/enterprise/resources/white_papers.htm, últim accés setembre 2013.

[13] D. Davis, F. Monrose, and M. Reiter. “On user choice in graphical password schemes”. In 13th USENIX Security Symposium, 2004.

[14] R. Dhamija and A. Perrig. “Déjà Vu: A user study using images for authentication”. In 9th USENIX Security Symposium, 2000.

[15] D. Weinshall. “Cognitive authentication schemes safe against spyware (short paper)”. In IEEE Symposium on Security and Privacy, May 2006.

[16] S. Wiedenbeck, J. Waters, L. Sobrado, and J. Birget. “Design and evaluation of a shoulder-

surfing resistant graphical password scheme". In International Working Conference on Advanced Visual Interfaces (AVI), 2006.

[17] E. Hayashi, N. Christin, R. Dhamija, and A. Perrig. "Use Your Illusion: Secure authentication usable anywhere". In 4th ACM Symposium on Usable Privacy and Security (SOUPS), Pittsburgh, 2008.

[18] K. Bicakci, N. B. Atalay, M. Yuceel, H. Gurbaslar, and B. Erdeniz. "Towards usable solutions to graphical password hotspot problem". In 33rd Annual IEEE International Computer Software and Applications Conference, 2009.

[19] S. Wiedenbeck, J. Waters, J. Birget, A. Brodskiy and N. Memon. "Authentication using graphical passwords: Basic results". In 11th International Conference on Human-Computer Interaction (HCI International), 2005.

[20] S. Chiasson, A. Forget, R. Biddle, and P. C. Van Oorschot. "Influencing users towards better passwords: Persuasive Cued Click-Points". In Human Computer Interaction (HCI), The British Computer Society, 2008.

[21] A. Stubblefield and D. Simon. "Inkblot Authentication", MSR-TR-2004-85. Technical report, Microsoft Research, 2004.

[22] H. Gao, Z. Ren, X. Chang and U. Aickelin. "A New Graphical Password Scheme Resistant to Shoulder-Surfing". In International Conference on Cyberworlds, p.194-199, 2010.

[23] A. Biryukov, D. Khovratovich. "Related-key Cryptanalysis of the Full AES-192 and AES-256". In 15th International Conference on the Theory and Application of Cryptology and Information Security, Tokyo, Japan, p.1-18, 2009.

[24] A. Biryukov, O. Dunkelman, N. Keller, D. Khovratovich, A. Shamir "Key Recovery Attacks of Practical Complexity on AES Variants With Up To 10 Rounds". In 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, French Riviera, p.299-319, 2010.

[25] A. Paivio "Imagery and verbal processes". New York: Holt, Rinehart, and Winston, p.438, 1971.

[26] M. Anand Kumar, Dr.S.Karthikeyan , "Investigating the Efficiency of Blowfish and Rejindael (AES) Algorithms ". In I. J. Computer Network and Information Security, p.22-28 , 2012-2.

9.- Índex d'il·lustracions

Annex – 1

Càlcul de l'espai de contrasenyes de DAS

Disposem d'una graella de GxG fileres/columnes. Volem conèixer el nombre de contrasenyes possibles amb una única traçada de fins a L_{MAX} cel·les, $\prod(L_{MAX}, G)$

$$\Pi(L_{\max}, G) = \sum_{L=1}^{L_{\max}} P(L, G)$$

L'anterior fórmula indica que per a cada longitud de la traçada entre 1 i L_{MAX}, calculem el nombre de possibilitats existents P(L,G), sent el resultat, la suma de les possibles traces amb totes les longituds.

$$P(L, G) = \sum_{\forall (x,y) \in [1..G]\, x [1..G]} n(x, y, L, G)$$

El nombre de possibles traces amb longitud L es calcularà com a sumatori del nombre de possibles traçades, d'aquesta longitud, que acaben a cadascuna de les cel·les de la graella. Per conèixer quantes traces de longitud L acaben en cadascuna de les cel·les farem servir la següent funció recursiva:

$$n(x,y,l,G) = n(x-1,y,l-1,G) + n(x+1,y,l-1,G) + n(x,y-1,l-1,G) + n(x,y+1,l-1,G)$$

On $n(x,y,1,G)=1$ i $n(x,y,L,G)$ si x o y no pertanyen a GxG valdrà 0.

Notem que donat que la separació entre cel·les és de com a molt un píxel, no es tenen en compte canvis de cel·la diagonal, ja que el gruix de la traça impedeix fer-ho en la realitat.

Si s'accepta que la traça sigui discontínua amb més d'un cop (*stroke*), caldrà seguir les indicacions de l'enllaç mostrat[11] per a calcular la següent taula,

L_{MAX}	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\log_2$	*4*	*8,32*	*12,57*	*16,82*	*21,07*	*25,32*	*29,57*	*33,82*	*38,07*	*42,32*	*46,57*	*50,82*	*55,07*	*59,32*	63,57

Com és evident l'espai de contrasenyes creix molt.

11 https://www.usenix.org/legacy/events/sec99/full_papers/jermyn/jermyn_html/node8.html#SECTION00033100000000000000

Annex – 2

Ús de l'algorisme AES-128

L'algorisme simètric AES-128 (10 rondes) és el recomanat per la NSA[12] com algorisme d'encriptació per a documents classificats com a secrets. Les versions AES amb més longitud de clau estan recomanats per a alts secrets, però en el nostre cas cercarem el compromís entre seguretat i eficiència, utilitzant la versió de 128 bits.

L'algorisme AES va ser trencat al 2005, entenent "trencar" com trobar una metodologia més ràpida que la força bruta. En aquest cas es va aconseguir reduir l'espai de cerca de 2^{128} a 2^{126}, cosa que no és una gran millora, però ja compleix la definició de "trencat". Al 2009, A. Biryukov i D. Khovratovich [23] van trobar una vulnerabilitat en la que utilitzant certes claus, s'aconseguia reduir l'espai de cerca de forma dràstica, per exemple en el cas de AES-256 (amb 14 rondes), l'espai de cerca es reduïa a 2^{96}. Tot i ser encara un espai molt gran, convertia l'algorisme en força ineficient, ja que de 256 bits de la clau obtenies les mateixes prestacions que amb 96 bits. Però la incidència d'aquest tipus de claus "perilloses" és tan baixa que l'algorisme es va seguir considerant segur.

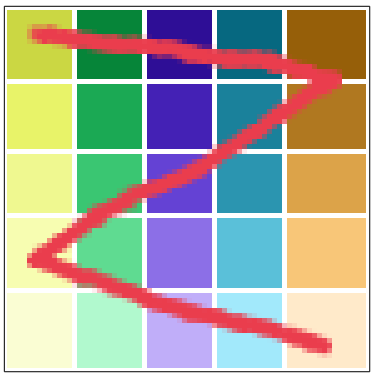
Però els mateixos autors van presentar un segon atac on es demostrava que es podia reduir l'espai de cerca de l'AES-128 (10 rondes) a 2^{123}, cosa que no sembla gaire problemàtica, però el mateix atac reduïa l'AES-256 (14 rondes) a 2^{119}, és a dir, per sota de l'AES-128 amb qualsevol tipus de clau. Això va ser confirmat en un segon article [24], on encara es reduïen més els espais de cerca[13].

Així que tot sembla confirmar que és més segur utilitzar l'AES-128 que alguna versió amb més longitud de clau, cosa que reforça la nostra elecció. Algoritmes millors, com per exemple *Blowfish*, requereixen massa recursos a nivell de memòria o processament, i com que l'aplicació està pensada per a dispositius mòbils no serien els més indicats. Tot i que es demostra a [26] que l'algorisme de *Blowfish* és més ràpid que AES-256, no es realitza cap comparativa respecte AES-128, cosa que fa pensar que aquest últim segueix sent més ràpid.

12 National Security Agency - USA
13 http://www.kriptopolis.org/wikileaks-insurance-aes-256

Treball Final de Màster

Contrasenyes Gràfiques
Annex - 3



Màster Interuniversitari de Seguretat de les Tecnologies de la Informació i de les Comunicacions
- *MISTIC* -

Alumne: Josep Escoda Herrando
Directora: Cristina Pérez Solà
Universitat: UOC

Curs 2013- 2014, Sem. 1

Índex de continguts

1.- Disseny de l'aplicació

1.1.- Casos d'ús d'alt nivell

La nostra aplicació interactuarà amb dos actors. El primer d'ells serà l'usuari encarregat d'autenticar-se, i el segon, l'aplicació que requerirà l'autenticació de l'usuari.

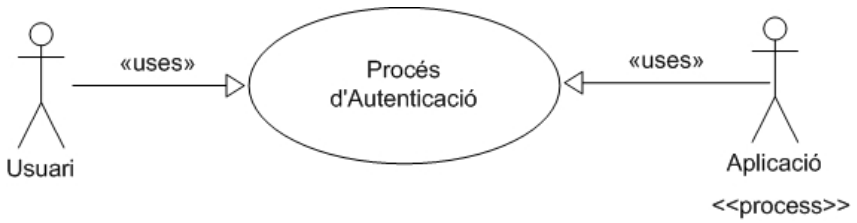


Fig. 1: **Diagrama de casos d'ús de nivell 1**

Si baixem de nivell en l'esquema anterior, podem veure quines són exactament les accions i processos amb els que interacciona cadascun dels actors.

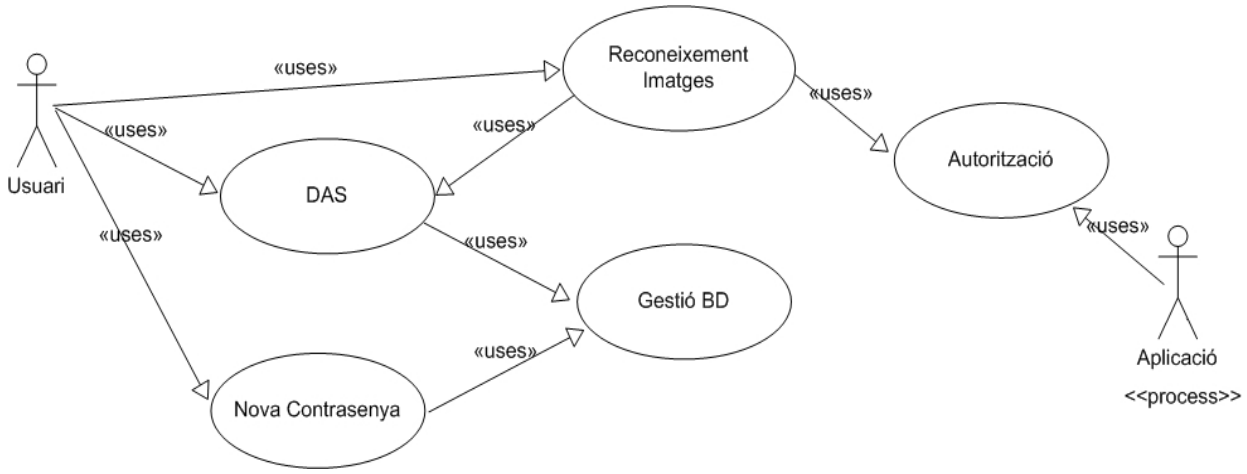


Fig. 2: **Diagrama de casos d'ús nivell 2**

En aquest segon nivell es pot apreciar que l'usuari realitzarà tres accions amb l'aplicació. La primera d'elles serà definir una nova contrasenya, on l'usuari realitzarà el procediment per a definir la nova contrasenya en les dues fases proposades a les decisions de disseny. La nova contrasenya serà emmagatzemada a la base de dades (*Gestió BD*).

Òbviament l'altra acció que podrà realitzar l'usuari és l'autenticació. Com ja s'ha comentat en l'anterior apartat, aquesta autenticació consta de dues fases: en una primera s'utilitza un esquema de tipus DAS, i en una segona fase es realitza un reconeixement d'imatges. Notem que el procés anomenat DAS és l'únic que interacciona amb la base de dades, ja que el secret compartit en forma de traça és el que permet desencriptar la base de dades.

Si tot el procés d'autenticació és correcte cal disposar d'una interfície que permeti comunicar-se amb l'aplicació interessada en aquest esquema d'autenticació (*Autenticació*). Aquesta part no serà contemplada en aquest treball.

1.2.- Disseny de la Base de Dades

En aquesta aplicació la base de dades serà molt simple, ja que sols cal mantenir les dades referents a l'autenticació de l'usuari. En el nostre cas tindrem dues seqüències d'imatges, per a les dues parts que formen el reconeixement de les imatges de la segona fase.

Notem que les imatges no s'emmagatzemen a la base de dades, ja que l'aplicació està pensada per a dispositius mòbils i les bases de dades típiques per aquests entorns no fan una gestió òptima de les dades multimèdia (*blob*). A més de cara a l'esquema de contrasenyes presentat és indiferent si les imatges estan emmagatzemades a la base de dades o bé al sistema de fitxers. Es pot pensar que emmagatzemant a la base de dades les imatges que formen la contrasenya, prèviament encriptades, l'esquema seria més segur, però llavors l'atacant que té accés al sistema de fitxers del dispositiu sols tindria que comparar les imatges que formen part del sistema de fitxers (galeria) i les que apareguin en la fase de reconeixement i no estiguin en la llista. Aquestes últimes serien les de la contrasenya (emmagatzemades a la base de dades), generant així un atac per intercepció.

Per aquesta raó s'ha optat per codificar a la base de dades simplement l'identificador del fitxer de les imatges escollides, i aquestes restaran al sistema de fitxers juntament amb la resta de les imatges que formen la galeria d'imatges.

També tindrem una taula on s'emmagatzemaran diferents estadístics per a la fase de proves de l'aplicació.

La nostra aplicació utilitzarà un sistema gestor de bases de dades relacional típic dels dispositius mòbils, com pot ser SQLite3. Aquests sistema gestor no proporciona cap mesura de seguretat, per això és la pròpia aplicació l'encarregada d'encriptar les dades amb AES-128, com ja s'ha comentat a l'apartat de Decisions de Disseny (Ap. 2). Existeixen extensions del propi sistema gestor que permeten treballar amb dades encriptades de forma transparent, però o bé són de pagament o bé compliquen la instal·lació dels *drivers* per a controlar la base de dades des de l'aplicació (a més de deixar la seguretat en mans de tercers).

Veiem el disseny de la nostra base de dades:

Fig. 3: **Disseny de la base de dades**

La taula *up*[1] s'encarregarà de mantenir les imatges i l'identificador de cada usuari. Estarà composta pels següents camps:

- **_id**: Camp autonumèric utilitzat com a clau primària. Aquest camp és necessari perquè el *driver* encarregat de comunicar el sistema gestor amb l'aplicació funcioni correctament.

- **up**: Camp de tipus *varchar* amb l'identificador de l'usuari, aquest camp tindrà la restricció *unique*, és a dir, que no es pot repetir el seu valor a la taula. Notem que aquest camp és realment el que hauria d'actuar com a clau primària.

1 *User-Password*

- **imSeq**: Camp de tipus *varchar* on emmagatzemarem el nom dels dos fitxers que formen la primera parella d'imatges de la fase 2 (reconeixement). El format de la cadena serà [*nomFitxer1A* | *nomFitxer2A*].

- **imSeq2**: Camp de tipus *varchar* on emmagatzemarem el nom dels dos fitxers que formen la segona parella d'imatges de la fase 2 (reconeixement). El format de la cadena serà [*nomFitxer1B* | *nomFitxer2B*].

L'aplicació estarà pensada per a gestionar un nombre indeterminat d'usuaris, però sembla evident que en un dispositiu mòbil és difícil que existeixi més d'un usuari real. L'aplicació presentada no realitzarà la identificació de l'usuari, ja que això es podria fer amb algun tipus de dada personal resident al propi dispositiu (targeta de comunicació, dades del sistema *Android OS*, ...)

La taula *statistic* manté dades estadístiques referents a cada autenticació amb l'esquema presentat:

- **_id**: Camp autonumèric utilitzat com a clau primària. Aquest camp és necessari perquè el *driver* encarregat de comunicar el sistema gestor amb l'aplicació funcioni correctament.

- **date**: Camp de tipus *varchar* amb la data en que s'ha capturat la dada estadística.

- **time**: Camp de tipus *float* amb la duració en segons (precisió de ms) de tot el procés d'autenticació.

- **auth**: Camp de tipus *smallint* que indica si l'usuari s'ha pogut autenticar satisfactòriament (1) o no (0).

- **longTrace**: Camp de tipus *smallint* que indica la longitud de la traça realitzada.

1.3.- Esquema SQL

L'esquema SQL de la base de dades serà molt simple, ja que com hem vist, sols estarà composta per dues petites taules.

```
CREATE TABLE up ("_id" INTEGER PRIMARY KEY, "up" VARCHAR, "imSeq" VARCHAR,
                                    "imSeq2" VARCHAR, UNIQUE("up") ON CONFLICT FAIL);

CREATE TABLE statistic ("_id" INTEGER PRIMARY KEY, "date" VARCHAR, "time" FLOAT,
                                              "auth" SMALLINT, "longTrace" SMALLINT);
```

A destacar la restricció *unique* al camp *up* com ja s'ha comentat anteriorment. En cas de no complir-se el sistema gestor emetrà una excepció que haurà de ser tractada per l'aplicació.

Afegir que el propi *driver* que comunica l'aplicació amb el sistema gestor és l'encarregat de convertir el camp "*_id*" en autonumèric. Això és un requeriment de l'accés a SQLite3 des del SDK *java* per *Android OS*.

1.4.- Casos d'ús a baix nivell

Si ens centrem en la fig.2 queda clar que l'usuari interacciona amb l'aplicació amb dues funcions: autenticació i registre de la contrasenya. Acte seguit baixem un nivell més en el disseny dels casos d'ús. Veiem els casos d'ús del procés de registre:

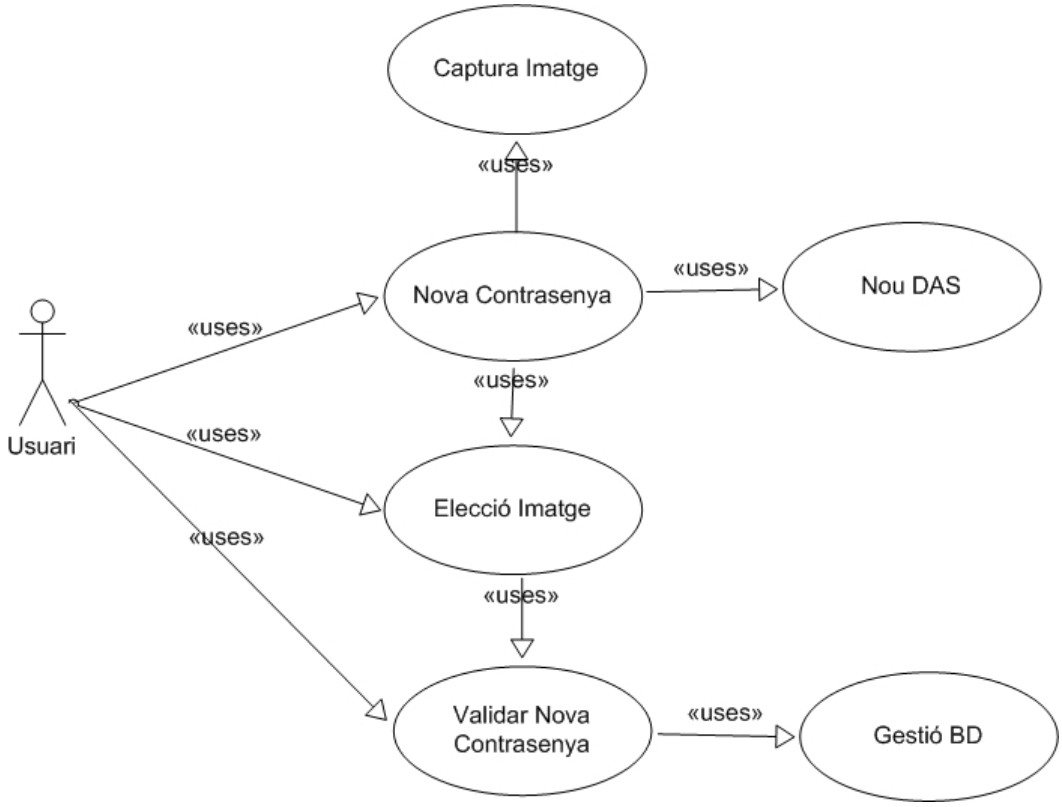


Fig. 4: **Casos d'ús del procediment de registre**

L'usuari iniciarà el procés de registre amb l'opció de "*Nova Contrasenya*". Aquest procés li farà reescriure una nova traça, que serà el nou secret compartit DAS ("*Nou DAS*"), o bé li permetrà capturar fotografies per incloure-les a la galeria d'imatges i poder-les seleccionar com a imatges de la seva contrasenya.

Un cop s'hagi dibuixat el nou DAS, l'aplicació cal que mostri un llistat amb totes les imatges que formen part de la galeria d'imatges capturades per l'usuari, que n'ha d'escollir dues ("*Escollir Imatges*"). Aquestes imatges seran distorsionades i separades en dues parelles cadascuna, com s'explica a l'apartat 2 de la memòria. Sols caldrà que l'usuari accepti el resultat ("*Validar Nova Contrasenya*") perquè totes aquestes dades siguin escrites a la base de dades.

Notem que el procediment de registre no es pot iniciar si l'usuari no s'ha autenticat correctament. Per facilitar les proves del prototipus a crear, si es detecta que la taula "*up*" resta buida, directament iniciarà el procés de registre. En un entorn real caldria dissenyar una aplicació que sols realitzés el procediment de registre de forma separada de l'autenticació[2].

2 Si un atacant amb permisos al sistema de fitxers elimina la base de dades, directament s'iniciarà el procés de registre, és obvi que això no ha de poder-se realitzar en un entorn real.

Si veiem ara el procediment d'autenticació, tenim el següent diagrama de casos d'ús:

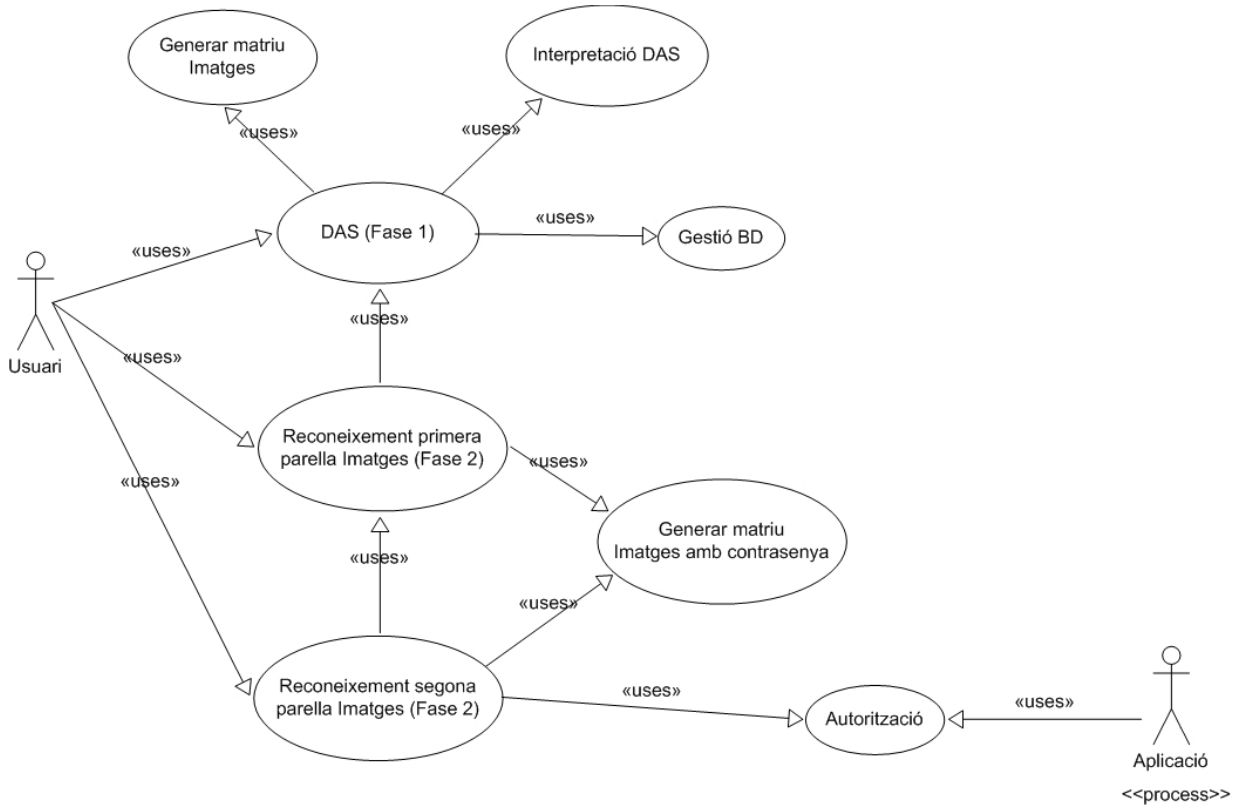


Fig. 5: **Casos d'ús del procediment d'autenticació**

En aquest diagrama es pot veure que l'usuari iniciarà el procediment realitzant una traça amb el seu DAS, que és el que defineix la fase 1. Per a realitzar la traçada s'utilitzarà una pantalla amb imatges aleatòries de fons per dificultar que un atacant pugui veure fàcilment la traça i a més aquestes serviran per delimitar cadascuna de les cel·les del DAS ("*Genera Matriu d'Imatges*").

Un cop s'hagi realitzat la traça i s'aixequi el punter/dit de la pantalla es procedirà a extraure els punts que formaran la seqüència del DAS ("*Interpreta DAS*") i a l'hora la *passphrase* per a generar la clau que encripta la base de dades ("*Gestió BD*"), tal com s'ha explicat a l'apartat de Decisions de Disseny (Ap. 2 de la memòria).

Després es passarà a la fase 2, composta per dos procediments iguals, on s'escolliran les parelles que formen les dues imatges. Aquestes parelles són la partició de les dues imatges generades a l'acceptar un canvi de contrasenya en el procediment de registre ("*Validar nova Contrasenya*", vist a l'anterior diagrama). Per a poder escollir les parelles, primer cal generar les matrius d'imatges que es mostraran per pantalla, tenint en compte que si el DAS és correcte, s'hauran d'incloure les parelles d'imatges que formen la contrasenya ("*Generar Matriu d'Imatges amb Contrasenya*"). Recordem que l'elecció de les imatges s'ha de fer alhora (*multitouch*).

Si l'usuari realitza correctament la traça, s'haurà pogut desencriptar la base de dades. Si a més ha escollit les dues imatges partides correctes, es farà una crida a la interfície de comunicació amb l'aplicació a protegir, indicant-li que l'autenticació ha estat satisfactòria.

1.5.- Disseny de les pantalles

En aquest apartat es descriurà el funcionament de les pantalles de que consta l'aplicació. Al següent apartat es podrà veure un diagrama de seqüència per comprovar l'ordre d'ús de cadascuna. Cal esmentar que totes les pantalles implementades sobre *Android OS* estan formades per dos components: una classe *java* que hereta d'*Activity* i fa de controlador, i una vista definida sobre XML, que es coneix com a *layout*. El *layout* és mostrat per l'*Activity* amb una acció pròpia que s'anomena *inflate*, i conté la definició i distribució de tots els components que apareixeran a la pantalla del dispositiu.

En aquest *layout* es poden utilitzar classes estàndard com a controladors de components d'interfície gràfica o bé controladors i components propis, com poden ser les definicions basades en la classe *surfaceView*. Tot això s'explicitarà amb més detall en la secció del disseny de les classes (Ap. 3.7).

1.5.1.- *p1Activity*

Aquesta primera pantalla ens mostrarà una matriu d'imatges que no tenen cap altra funcionalitat que distraure a un possible atacant per *shoulder-surfing*. Sobre aquesta matriu, l'usuari haurà de traçar, mitjançant el desplaçament del dit sobre la pantalla, un recorregut per les diferents cel·les de la matriu mostrada. Aquest recorregut serà la contrasenya *DAS* que es farà servir per desencriptar la base de dades, com ja s'ha comentat anteriorment. Notem que el recorregut sols pot contenir una traça, és a dir, en el moment en que s'aixequi el dit de la pantalla es donarà per acabat el traç i es passarà a executar *p2Activity*.

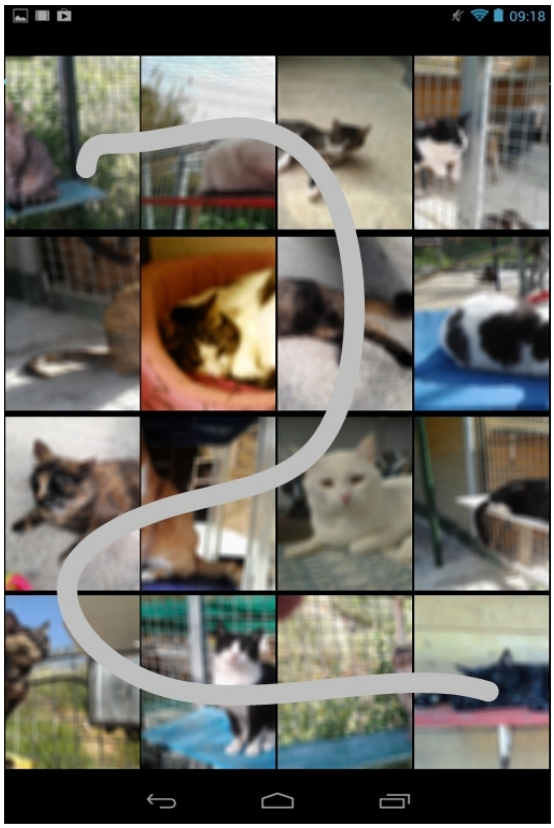


Fig. 6: Pantalla *p1Activity*

Per a dificultar el possible atac per *shoulder-surfing* la pantalla farà la traçada amb un color i una intensitat baixa, a més de fer desaparèixer la part inicial de la traça conforme s'avança en la realització d'aquesta.

Aquesta pantalla estarà definida al *p1_layout.xml* i farà un ús intensiu de les classes de creació pròpia *interficieCanvas* i *interficieCanvasThread*.

1.5.2.- *p2Activity*

A la pantalla de *p2Activity* apareixerà de nou una matriu amb imatges, de les quals s'hauran de seleccionar dues a l'hora, fent ús de la propietat *multitouch* del dispositiu. A aquesta pantalla hi arribarem encara que el *DAS* no sigui el correcte, ja que així un possible atacant no sabrà en quina part de l'esquema de contrasenya falla.

Les imatges són aleatòries, igual que la seva posició a la pantalla. No es mostrarà cap indicació de quines imatges han estat escollides per evitar l'atac de *shoulder-surfing*. També cal esmentar que les imatges es mostren amb cert nivell de distorsió per tal d'evitar que es puguin descriure fàcilment i així evitar l'enginyeria social (veure [17]).

Sota es mostren les pantalles i quines han estat les dues imatges seleccionades:

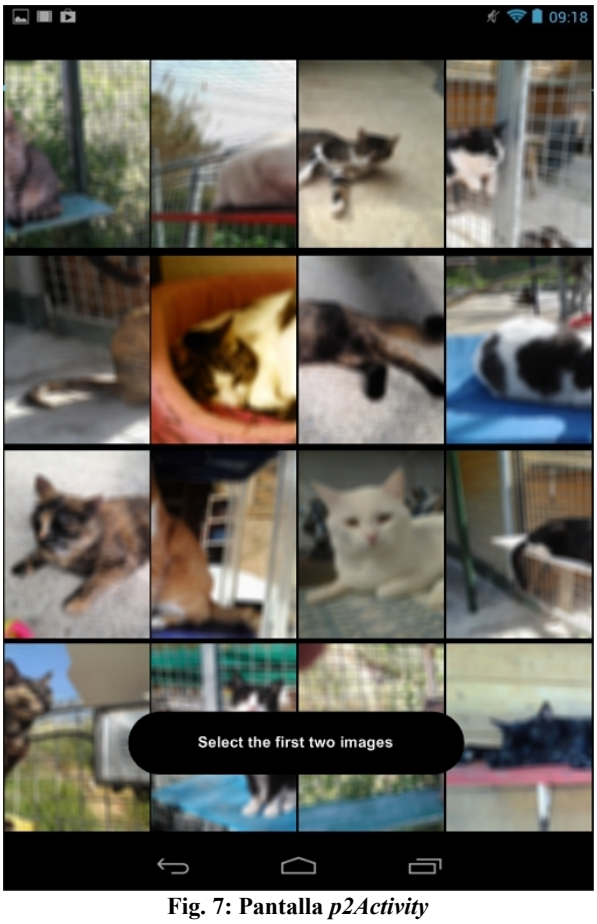


Fig. 7: Pantalla *p2Activity*

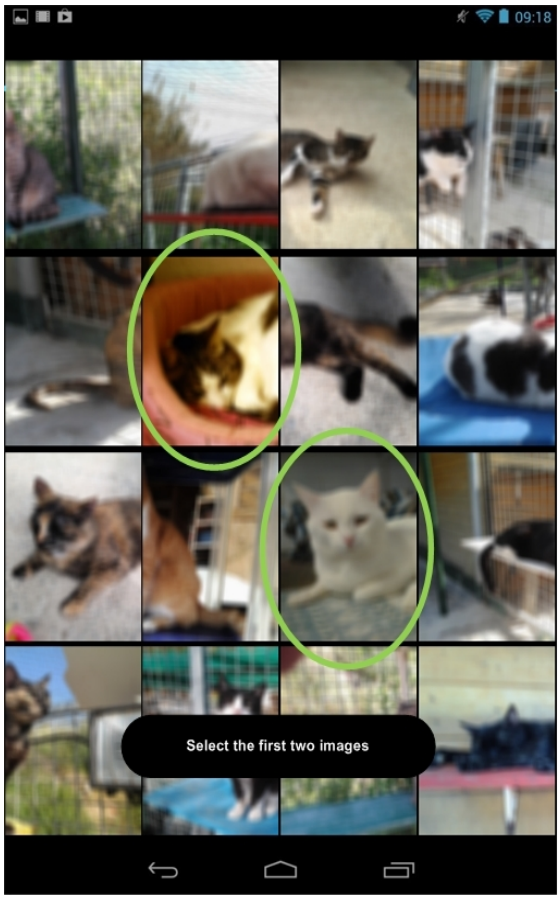


Fig. 8: Pantalla *p2Activity* amb imatges seleccionades

Aquesta pantalla estarà definida al *p2_layout.xml* i farà un ús intensiu de les classes de creació pròpia *interficieCanvas2* i *interficieCanvasThread2*.

1.5.3.- *p3Activity*

A la pantalla de *p3Activity* tornaran a aparèixer noves imatges, de les quals s'hauran de seleccionar les dues parelles de les seleccionades a *p2Activity*. S'haurà de tornar a fer alhora, fent ús de la propietat *multitouch* del dispositiu. Ja s'ha comentat en el procediment d'autenticació que el fet de dividir les imatges sent pròximes i conegudes per l'usuari no afegeix dificultat per recordar-les. També a aquesta pantalla hi arribarem encara que el *DAS* no sigui el correcte, ja que així un possible atacant no sabrà en quina part de l'esquema de contrasenya falla.

No es mostrarà cap indicació de quines imatges han estat escollides per evitar l'atac de *shoulder-surfing*.

Acte seguit es mostra la pantalla i quines han estat les dues imatges seleccionades:

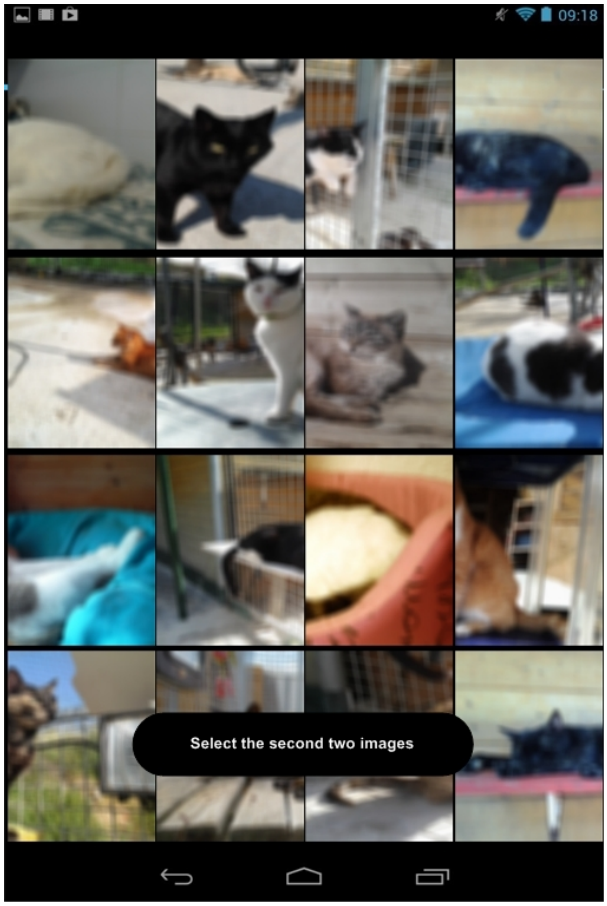


Fig. 9: Pantalla *p3Activity*

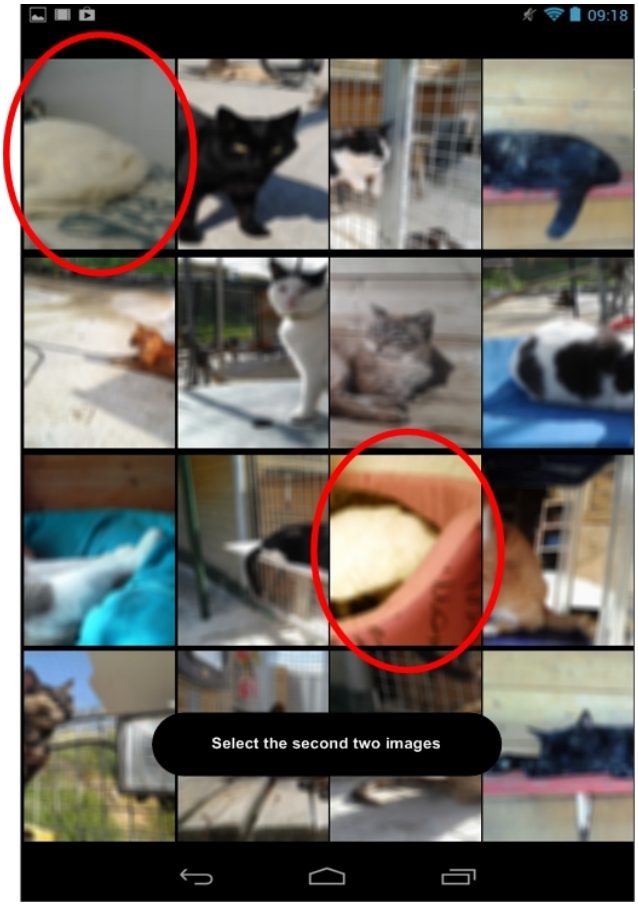


Fig. 10: Pantalla *p3Activity* amb imatges seleccionades

Aquesta pantalla estarà definida al *p3_layout.xml* i farà un ús intensiu de les classes de creació pròpia *interficieCanvas3* i *interficieCanvasThread3*.

1.5.4.- *p4Activity*

Si arribem a la pantalla de p4Activity voldrà dir que el procés d'autenticació s'ha realitzat amb èxit. En aquest moment s'hauria de presentar la primera pantalla de l'aplicació a la que s'hi aplica

l'esquema de contrasenya dissenyat. En el nostre cas hi apareixerà *p4Activity* que és la pantalla encarregada de gestionar l'assignació d'una nova contrasenya a l'usuari (tema tractat en el disseny dels casos d'ús, Ap. 3.4).

A part del botó per a sortir de l'aplicació, també hi trobarem un botó que ens permetrà afegir fotos nostres per a ser usades com a contrasenya (*Add Photo*) i el botó que ens permetrà canviar la contrasenya (*New Password*). El primer d'ells ens portarà a la *photoActivity* mentre que el segon ho farà a *reg1Activity*.

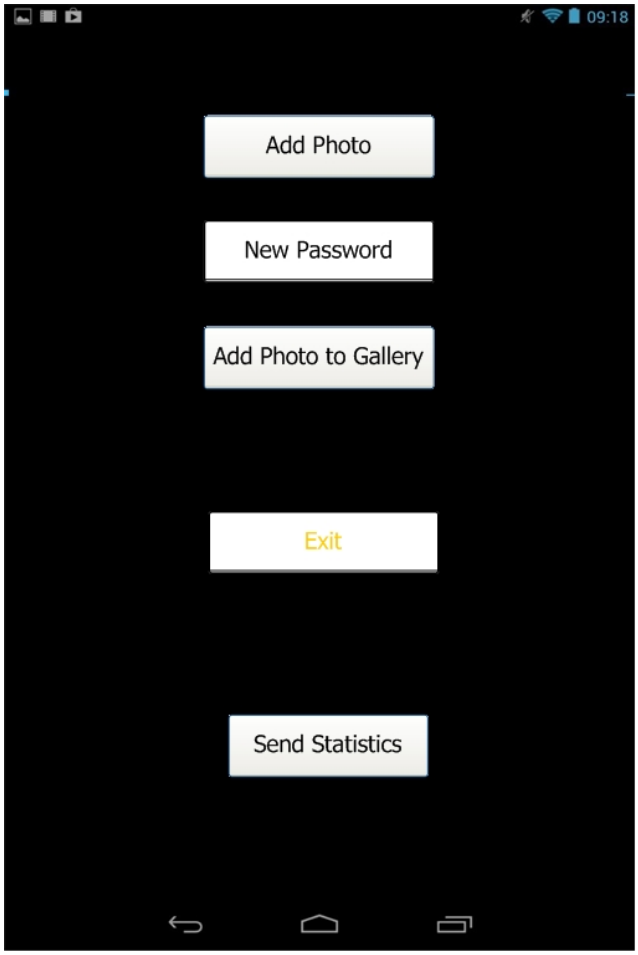


Fig. 11: Pantalla de *p4Activity*

S'han afegit, respecte la fase de disseny, dos nous botons: *Add photo to Gallery*, que ens permetrà afegir fotos nostres a la galeria d'imatges, mitjançant *galleryActivity*, i un botó per a enviar dades estadístiques, *Send Statistics*.

Aquesta pantalla estarà definida al *p4_layout.xml*.

1.5.5.- *photoActivity*

Aquesta pantalla estarà pensada per a mostrar la imatge captada per alguna de les càmeres que té el dispositiu (si no en té mostrarà un missatge indicatiu). Sobre aquesta imatge es definirà un nou *layout* que contindrà sols un botó (*camera_layer.xml*). Aquest apareixerà sobre la imatge captada per la càmera i servirà per indicar quan capturar la fotografia.

Fig. 12: **Pantalla de *p4Activity***

Aquesta pantalla estarà definida al *photo_layout.xml*. Un cop captada la fotografia es tornarà automàticament a *p4Activity*.

1.5.6.- *reg1Activity*

La pantalla reg1Activity permet que l'usuari registri o torni a definir la traça que farà de *DAS*. Molt similar en quant a característiques a la *p1Activity* però en aquest cas sense la necessitat de mostrar imatges de fons per a distreure un possible atacant. S'entén que l'usuari ja posarà mitjans que evitin la captura de la seva contrasenya, en el moment de la definició. Un cop acabada la traça, aquesta pantalla ens portarà fins a *reg2Activity*.

La pantalla *reg1Activity* restarà definida al *r1_layout.xml* i farà un ús intensiu de les classes pròpies *interficieCanvas4 i interficieCanvasThread4*.

1.5.7.- *reg2Activity*

La pantalla reg2Activity permet escollir les imatges que formaran part de la contrasenya. En ella simplement hi apareixerà una llista amb totes les imatges que es troben al directori d'imatges de càmera. Notem que aquesta llista no inclourà les imatges que formen part de la galeria d'imatges de la pròpia aplicació.

Això es fa així perquè l'usuari sols pugui utilitzar imatges captades per ell amb la càmera com a part

de la contrasenya. El fet d'utilitzar una imatge generada i coneguda per ell augmenta la facilitat de memorització d'aquesta (veure Decisions de Disseny, Ap.2).

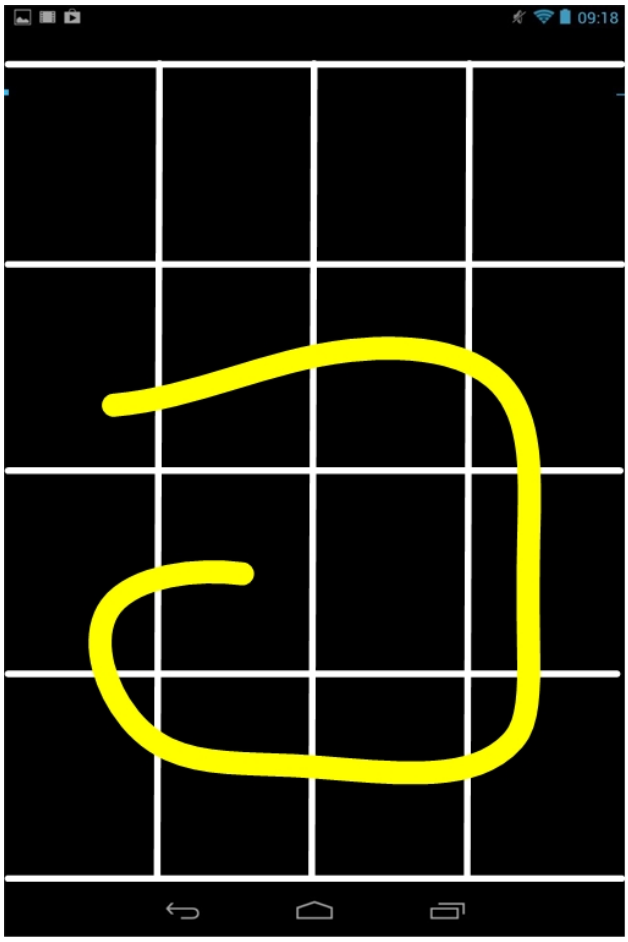


Fig. 13: Pantalla *reg1Activity*

En aquesta pantalla caldrà escollir dues imatges, ja que recordem que la contrasenya està composta per dues parelles d'imatges dividides. Un cop seleccionades, a la següent pantalla (*reg3Activity*), l'usuari podrà decidir si realment accepta l'elecció com a nova contrasenya.

La pantalla *reg2Activity* estarà definida al *r2_layout* i farà ús de les llistes amb *scroll* pròpies del SDK d'*Android OS*. Per a poder utilitzar aquest tipus de control caldrà definir un nou *layout* per a cada filera. En el nostre cas, aquest *layout*, anomenat *row_item*, estarà compost per una imatge i pel nom del fitxer que la conté.

Fig. 14: Pantalla *reg2Activity*

1.5.8.- *reg3Activity*

La pantalla reg3Activity mostrarà el resultat de difuminar (*blur*) i dividir en dues parts les imatges escollides a l'anterior pantalla. Si l'usuari està conforme amb l'elecció podrà acceptar el canvi de contrasenya o bé cancel·lar-lo. Tant en un cas com en un altre tornarem a la pantalla d'inici del registre (*p4Activity*).

Aquesta pantalla estarà definida a *r3_layout.xml*.

1.5.9.- gallery*Activity*

La pantalla galleryActivity és idèntica a la mostrada a *reg2Activity*, i ens permetrà escollir quina fotografia feta amb la càmera volem que passi a formar part de la galeria d'imatges. Sols caldrà escollir una en aquest cas.

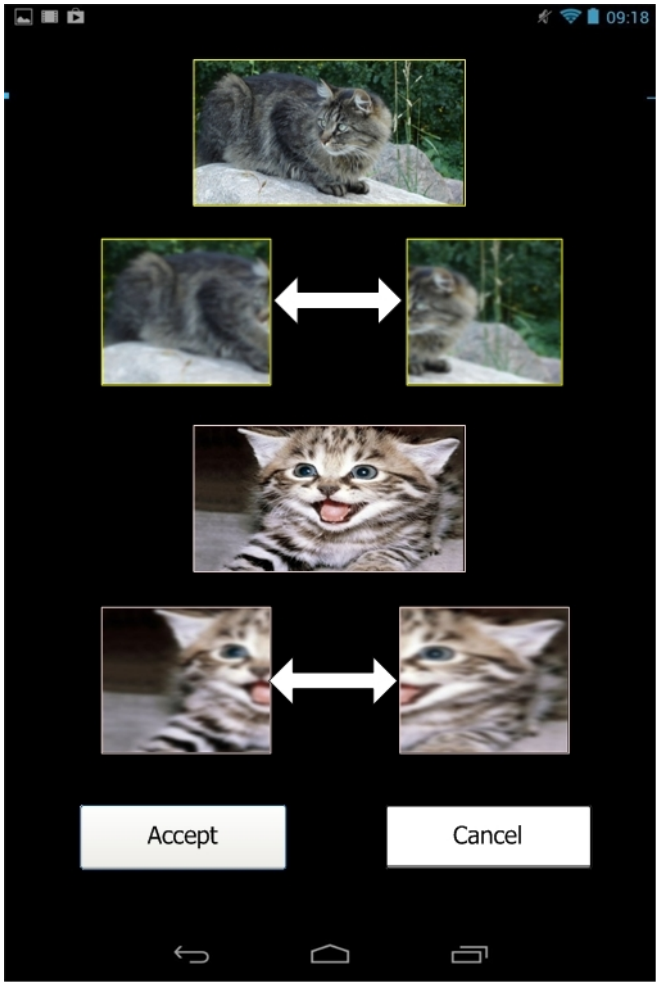


Fig. 15: Pantalla *r3Activity*

1.6.- Seqüència de pantalles

En la següent imatge (fig. 16), es mostra la seqüència en la que apareixeran les diferents pantalles de l'aplicació, ja comentades en l'apartat anterior.

Al gràfic es poden diferenciar dues fileres: a la superior hi apareix la seqüència d'autenticació, on l'usuari introdueix la seva traça (primera fase) i després selecciona els dos parells d'imatges que formen part de la seva contrasenya (*p2Activity* i *p3Activity,* formen la segona fase).

A la filera inferior hi apareix el procés de registre o de canvi de contrasenya, on es poden captar amb la càmera les noves imatges candidates a ser contrasenya o bé es pot passar a generar una nova contrasenya, definint un nou *DAS* i escollint les dues parelles d'imatges noves.

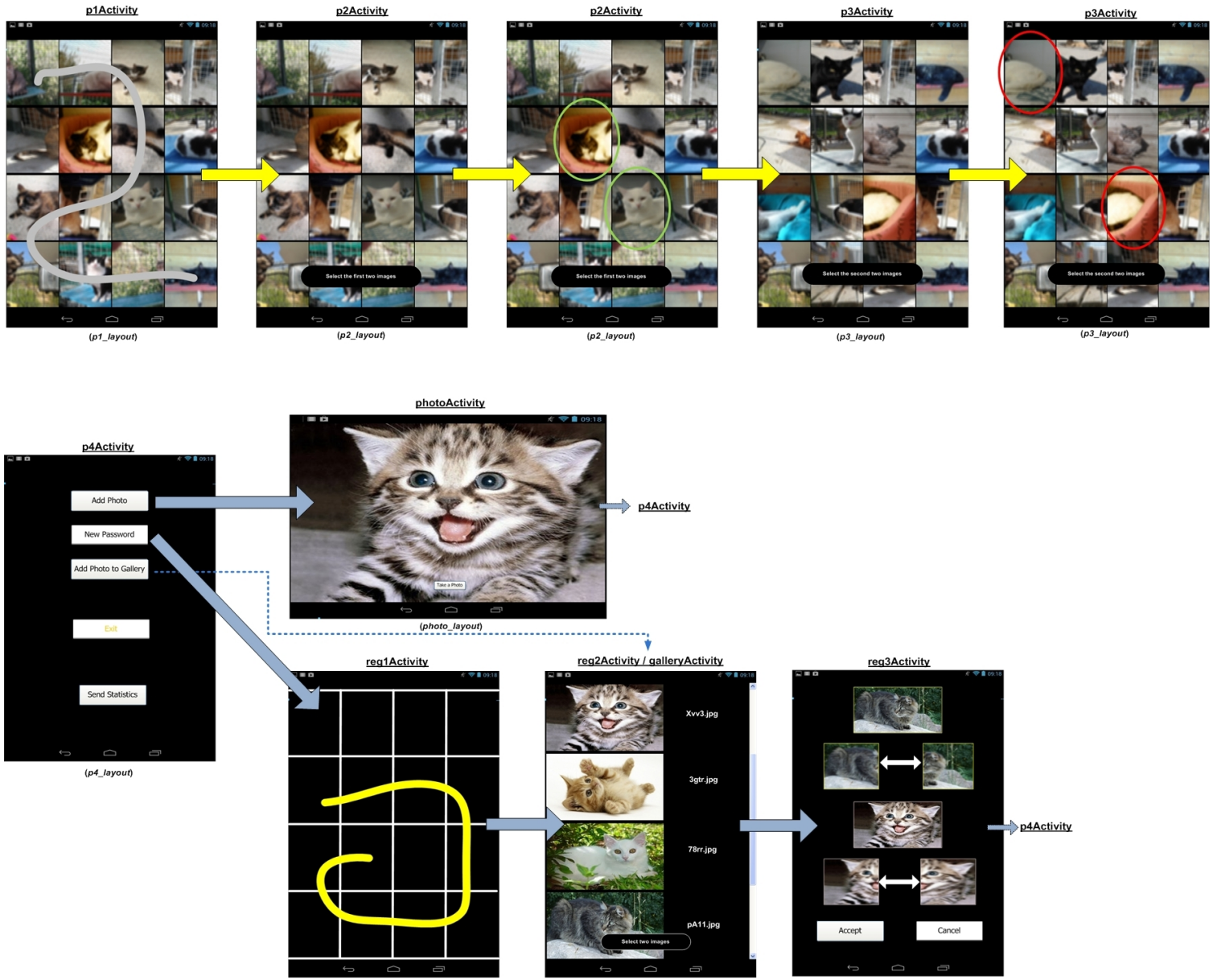


Fig. 16: Seqüència de pantalles

1.7.- Disseny de les classes

L'aplicació estarà dividida en 6 paquets, cadascun amb una funcionalitat concreta. Abans de passar a descriure'ls per separat veiem quins són:

tfmApplication: Paquet on hi trobarem les classes que formen el flux de l'aplicació i que actuen majoritàriament com a controladors.
bdController: Classes encarregades de comunicar l'aplicació amb la base de dades.
sfController: Classes encarregades de gestionar i interactuar amb el sistema de fitxers.
cripto: Classes encarregades d'implementar els algoritmes criptogràfics utilitzats per l'aplicació.
imageTools: Classes encarregades del tractament de les imatges.
canvas: Classes que implementen interfícies gràfiques no estàndards i pròpies de l'aplicació.

1.7.1.- *tfmApplication*

Dintre d'aquest paquet hi trobarem les classes que representen el flux de l'aplicació. El SDK de *java* per a *Android OS* està basat en *Intents*. Els *Intents* no són més que la voluntat de realitzar una acció. Una d'aquestes accions és l'inici d'una nova *Activity*. Les *Activity*'s són els controladors de cadascuna de les vistes (pantalles) d'una aplicació *Android OS*, com bé s'ha comentat a l'apartat 1.5 Recordem que una *Activity* està associada a una vista que resta definida en un fitxer XML anomenat *layout*.

Gran part de les classes del paquet *tfmApplication* seran herència de la classe *Activity* del SDK d'*Android OS,* i representaran les diferents pantalles de l'aplicació. Moltes d'aquestes pantalles tenen components gràfics que es crearan especialment per a aquesta aplicació (paquet *canvas*).

Dintre d'aquest paquet cal destacar la classe *configuration*. Aquesta classe serà l'encarregada de mantenir en memòria els paràmetres globals necessaris per a l'aplicació i seguirà un patró de tipus *singleton*. Aquest patró permet crear una instància de la classe en el seu primer ús i mantenir-se en memòria mentre s'estigui executant l'aplicació. La clau és la definició d'un dels seus atributs com a,

```
private static configuration ourInstance = new configuration();
```

Això permet que una crida al mètode *getInstance* ens retorni el valor de la variable *ourInstance* i sempre accedim al mateix objecte en memòria. Aquest funcionament la fa ideal per a emmagatzemar paràmetres globals de l'aplicació. També s'utilitzarà aquesta classe per a passar dades entre diferents *Activity*'s. Això es pot fer mitjançant un mecanisme propi d'aquesta classe anomenat *Bundle*, però com que el pas de paràmetres serà mínim en el nostre cas i ja tenim definit aquest objecte global l'aprofitarem.

Dintre de la classe *configuration* tindrem accés (mètodes set/get) als següents paràmetres:

Paràmetre	Tipus	Valor Inicial	Descripció
COLUMNS	int	4	Nombre de columnes de la matriu per mostrar les imatges o generar el DAS
ROWS	int	4	Nombre de fileres de la matriu per mostrar les imatges o generar el DAS
IMAGEWIDTH	int	300	Ample per defecte en píxels de les imatges utilitzades
IMAGEHEIGHT	int	300	Alçada per defecte en píxels de les imatges utilitzades
TH_PROXIMITAT	int	60	Distància en píxels per considerar un punt diferent a l'anterior
ntrace	int	100	Punts de la traça a mostrar
trace	boolean	false	S'ha acabat la traça?
stroke	int	8	Gruix de la traça a mostrar
grid	int	3	Gruix del grid a mostrar en la fase de registre
das	String	-	Seqüència de posicions que formen el DAS generat
aes	SecretKey	-	Clau secreta per desencriptar la base de dades
salt	String	*SaltValue*	Salt utilitzat a l'hora de generar la clau *aes*
user	String	*IdUser*	Usuari que utilitza l'aplicació (veure Disseny de la Base de Dades)
imgSeq1	String[]	-	Seqüència d'imatges de la primera part de la fase 2
imgseq1sel	String[]	-	Seqüència d'imatges escollides per l'usuari a la primera part de la fase 2
imgSeq2	String[]	-	Seqüència d'imatges de la segona part de la fase 2
imgseq2sel	String[]	-	Seqüència d'imatges escollides per l'usuari a la segona part de la fase 2
extension	String	.JPG	Tipus d'imatge gràfica utilitzada
established	boolean	false	Està la configuració establerta?
gBD	bdController	-	Gestor de la base de dades. És el mateix per a tota l'aplicació
isEmpty	boolean	false	Està la base de dades buida? (veure Disseny de la Base de Dades)
sf	sfController2	-	Gestor de fitxers. És el mateix per a tota l'aplicació
filter	Filter	-	Filtre a aplicar a l'hora de seleccionar fitxers d'imatges
galleryDir	String	-	Directori on es troba la galeria d'imatges
cameraDir	String	-	Directori on s'emmagatzemen les fotos fetes per l'usuari
STATISTICS	boolean	true	Indica si esta activat el modul estadístic
timeAuth	long	-	Temps inicial per a calcular la durada del procés d'autenticació
longTrace	int	-	Longitud de la traça realitzada

Taula 1: Paràmetres de configuració de l'aplicació

La resta de classes que formaran el paquet heretaran de la ja comentada classe *Activity* i per tant heretaran tots els mètodes encarregats de gestionar el flux d'execució així com les vistes associades. A la fig. 17 podem veure el cicle d'execució d'una *Activity.* A nosaltres ens interessarà sobreescriure el mètode *onCreate* on definirem com volem que es crei la nostra activitat.

A la fig. 18 podem veure el diagrama de classes complert del paquet *tfmApplication* i la seva interacció. La nostra aplicació tindrà 8 activitats, les quatre primeres destinades al procés d'autenticació, una altra destinada a la captura d'imatges amb la càmera i les tres restants del procés de registre, tal com s'ha explicat en el Disseny de les Pantalles (Ap. 5.6).

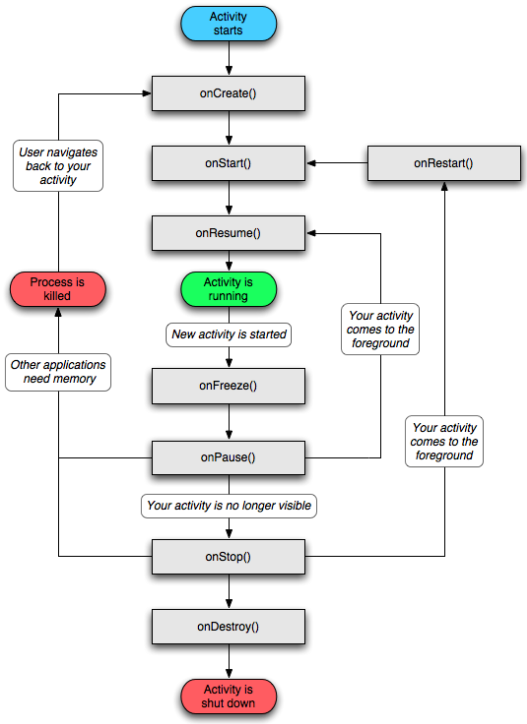


Fig. 17: **Cicle de vida d'una *Activity Android OS***

El flux d'execució també s'ha explicat en el disseny de les pantalles i estarà gestionat per les crides *nextActivity,* a excepció de les *Activity*'s on la pròpia vista gestiona el pas a una altra pantalla (*photoActivity* i *p4Activity*). De nou a la fig.18 es pot comprovar aquest flux d'execució amb les crides d'ús entre les diferents classes.

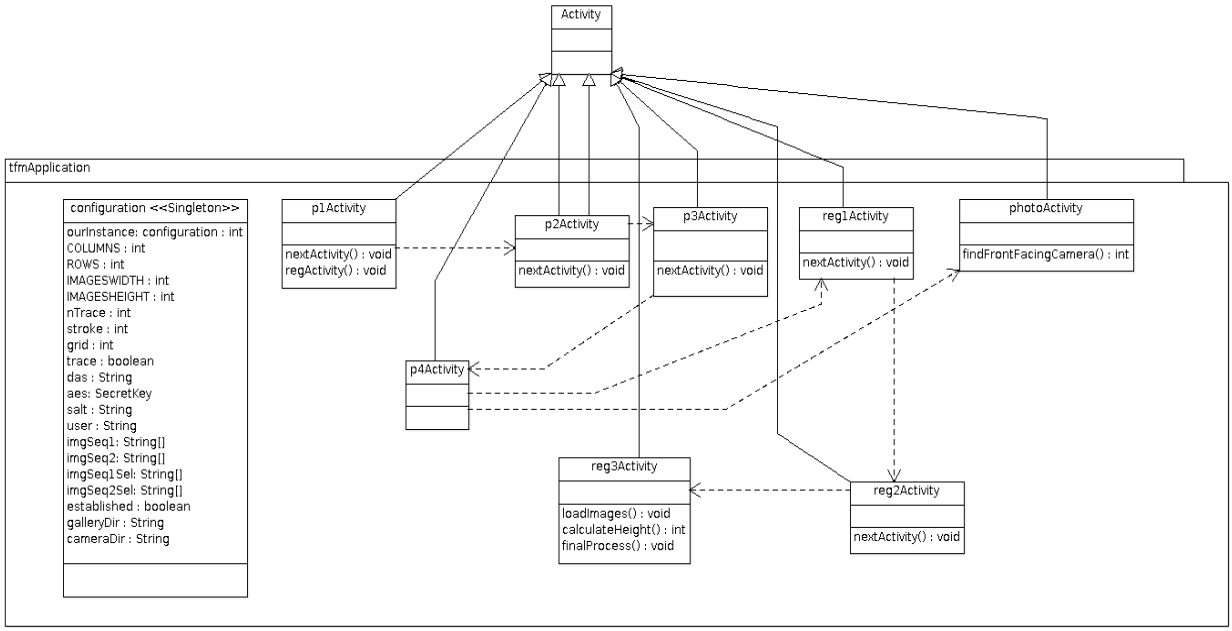


Fig. 18:Diagrama de classes del paquet *tfmApplication*

Acte seguit comentarem cadascuna de les classes.

p1Activity

Classe encarregada d'iniciar l'aplicació. El primer que farà és inicialitzar tots els paràmetres de la configuració que no tinguin valor per defecte. Instanciarà una nova connexió amb la base de dades i crearà un nou controlador del sistema de fitxers.

Acte seguit es comprovarà si la base de dades és buida i en aquest cas, amb el mètode *regActivity* es passarà directament al procés de registre (*p4Activity*). Si tot és correcte es mostrarà la seva vista associada (*p1_layout*).

La vista *p1_layout* està formada per un component gràfic de tipus *interficieCanvas* que permet registrar una traça que actuarà com a DAS. Aquesta classe serà comentada dintre del paquet *canvas*.

Un cop realitzada la traça, s'iniciarà *p2Activity*.

p2Activity - p3Activity

Aquestes classes simplement carreguen les respectives vistes (*p2_layout* i *p3_layout*). Les vistes utilitzen les classes *interficiecanvas2* i *interficiecanvas3* que seran explicades més endavant.

p4Activity

Igual que les altres activitats, mostrarà la seva vista associada (*p4_layout*). Aquesta vista disposa de tres botons que permetran realitzar les següents accions:

- *Afegir foto*: Permet accedir a *photoActivity* per a capturar una fotografia
- *Nova contrasenya*: Permet iniciar el procés de registre amb una crida a *reg1Activity*
- *Afegir foto a la galeria*: Permet afegir una fotografia a la galeria d'imatges. Crida a *galleryActivity*.
- *Sortir*: Elimina les possibles imatges capturades per l'usuari (*cameraDir*) i tanca l'aplicació

photoActivity

Aquesta activitat inicia dues vistes sobreposades (*photo_layout* i *camera_layout*). A la primera d'elles es fa un volcat de la imatge que està capturant en directe la càmera fotogràfica. La segona vista presenta un botó que al ser clicat capturarà la imatge i l'emmagatzemarà a *cameraDir*.

Notem que sempre s'intentarà utilitzar la càmera frontal, si no es disposa d'ella s'utilitzarà la càmera posterior i si el dispositiu no té càmera, s'avisarà de l'error a l'usuari. De tot això s'encarregarà el mètode *findFrontFacingCamera*.

Un cop capturada la imatge de la càmera, es destruirà aquesta activitat i es tornarà a *p4Activity*.

reg1activity

Classe que inicia el procés de registre. Actuarà de forma similar a *p1Activity* ja que l'usuari haurà de

definir una nova traça com a DAS. En aquest cas s'utilitzarà la *interfícieCanvas4* mitjançant la vista *r1_layout*.

reg2activity

Aquesta classe ens mostrarà en format de llista les imatges que l'usuari ha capturat al *cameraDir*. La vista estarà definida a *r2_layout* i consistirà en una *ListView* on l'activitat serà l'encarregada d'afegir-hi les dades.

Les dades que composen la llista constaran d'un *layout* per a cada entrada (*row_layout*). Cada entrada serà la pròpia imatge i el nom del fitxer on s'emmagatzema, per tant l'activitat farà una cerca d'imatges pel *cameraDir* i crearà un *row_layout* per cadascuna. Després sols caldrà esperar que l'usuari esculli dues imatges per activar *reg3Activity*.

reg3activity

Classe encarregada de mostrar la vista *r3_layout* on apareixeran les dues imatges escollides per l'usuari a *reg2activity*. Com es pot veure a l'apartat del Disseny de Pantalles (Ap 1.5), aquesta vista mostrarà com quedaran les imatges un cop dividides en parelles i distorsionades. Si l'usuari accepta el resultat aquesta activitat s'encarregarà de modificar la base de dades, introduint la nova seqüència d'imatges, prèviament encriptada, i eliminant de la galeria les imatges que formaven l'anterior contrasenya. A més es canvien totes les dates dels fitxers d'imatges de la galeria per no deixar cap traça d'aquest canvi de contrasenya.

Si tot ha funcionat correctament ho indicarà a l'aplicació, tornarà a *p4Activity* i es destruirà (*finalProcess*).

Com que l'aplicació ha de funcionar en pantalles petites de fins a 4" i les imatges que captura l'usuari són tractades a nivell de píxels (cosa que *Android OS* evita[3]) es necessari un mètode que distribueixi les imatges de forma coherent a la pantalla, aquest és el mètode *loadImages*.

galleryActivity

Aquesta classe ens mostrarà en format de llista les imatges que l'usuari ha capturat al *cameraDir*. La vista estarà definida a *gallery_layout* i consistirà en una *ListView* on l'activitat serà l'encarregada d'afegir-hi les dades. La imatge escollida per l'usuari a la llista passarà a formar part de la galeria d'imatges de l'aplicació.

3 *Android OS* fa servir com a unitat de mesura gràfica virtual els *density pixels,* que estan relacionats amb la densitat de la pantalla. 1 dp =1 pixel en pantalles de 160 dpi. Això permet gestionar fàcilment el disseny d'interfícies gràfiques multidispositiu.

1.7.2.- *canvas*

Dintre d'aquest paquet hi trobarem les classes que implementaran components gràfics específics de la nostra aplicació.

La classe principal del paquet és la *interficieCanvasGenerica*, aquesta classe heretarà de la classe *SurfaceView* del SDK, que permet crear components gràfics dibuixables (*drawable*) des de la pròpia aplicació (anomenats tècnicament *canvas*). També implementarà la interfície *SurfaceHolder.Callback* que li permet detectar i capturar events generats per la pantalla tàctil.

Els atributs d'aquesta classe són referents al nombre d'imatges que cal mostrar per pantalla (*imagesDisplay*), el nombre d'imatges que forma la galeria (*imagesNumber*) o bé el tamany de la pantalla (*widthCanvas* i *heightCanvas*).

Les classes que implementen els components utilitzats en cada vista (veure apartat anterior) heretaran d'*interficieCanvasGenerica* i hauran d'implementar els mètodes següents:

- *surfaceCreated*: Encarregat de crear els gràfics que apareixeran al *canvas*
- *onDraw*: Encarregat de dibuixar el *canvas* i mostrar els components gràfics necessaris
- *onTouchEvent*: Captura i tractment dels events generats per la pantalla tàctil

Notem que cada classe de tipus *interficieCanvasX* té un *thread* (*interficieCanvasThreadGenerica*) associat. Aquest procés de baix pes serà activat dintre del mètode *surfaceCreated* un cop s'hagin creat tots els components del *canvas*. La seva funció és cridar de forma cíclica al mètode *onDraw* per tal de refrescar la pantalla. Al ser un *thread* independent s'evita que en cas d'error es col·lapsi tot el sistema operatiu.

Veiem ara de forma independent algunes de les classes més importants.

interficieCanvas

Aquesta classe implementarà el *canvas* encarregat de mostrar una matriu aleatòria d'imatges d'entre totes les que es troben al *galleryDir*. Com ja s'ha comentat aquestes imatges sols serviran per marcar les cel·les per les que l'usuari farà la traça del DAS i a l'hora per dificultar que un atacant pugui veure fàcilment el recorregut d'aquesta traça.

Per obtenir les imatges (*surfaceCreated*) simplement ens caldrà fer una cerca aleatòria sense repetició pel llistat d'imatges del *galleryDir* que ens proporcionarà el *sfController2*, i mostrar-les mitjançant el mètode *onDraw*.

A l'hora es programarà l'event de detecció de pulsació a la pantalla (*action_down*) i el de moviment (*action_move*) per anar capturant les coordenades on s'ha produït l'event. Aquestes coordenades seran convertides en objectes de tipus *punt* i aquests emmagatzemats en un vector si està prou lluny de l'anterior (*punt.esProxim*). Això últim es fa per no emmagatzemar punts innecessaris, més sabent que pel propi mètode DAS acabaran convertits tots els punts d'un *quadrant* o cel·la en un de sol.

Quan es detecti l'event *action_up*, és a dir, es deixi de fer la traça, el vector de punts serà enviat al *dasController*, que és la classe encarregada d'obtenir l'ordre dels *quadrants* per on ha passat la traça de l'usuari. El mètode *dasController.obtePassword* ens retornarà directament una funció resum de

tipus *sha-512* amb la llista cronològica dels quadrants que formen la traça.

Aquí un altre cop el mètode *onDraw* serà l'encarregat de pintar a la pantalla els últims N punts que formen la traça a cada execució. Recordem que el nombre de punts a pintar és un paràmetre que resta a la classe *configuration* (*ntrace*).

Quan la traça hagi estat feta, es farà la consulta a la base de dades, ja que amb la traça ja podem obtenir la clau AES-128 que l'encripta (veure Decisions de Disseny, Ap. 2). Si l'autenticació és correcta obtindrem la seqüència d'imatges que formen la contrasenya i que hauran de ser mostrades en la segona fase.

Finalment aquesta classe li notificarà a la seva *activity* associada que el procés ha acabat per tal que aquesta carregui la següent *activity*.

interficieCanvas2 i *interficieCanvas3*

Aquests dos components gràfics ens permetran mostrar la matriu d'imatges per tal que l'usuari pugui escollir les que formen la contrasenya. Si la fase 1 de l'autenticació ha estat correcta, carregarem en posicions aleatòries les parelles d'imatges que formen la contrasenya, acompanyades d'altres imatges de la galeria, sense repetició. Tot això es farà als mètodes *surfaceCreated* i *onDraw*.

En cas de que la traça de la fase 1 no fos correcta es carregarien només imatges de la galeria d'imatges.

Sols mancarà programar la captura d'events de la pantalla per a detectar l'event *multitouch*. En aquest cas es verificarà que les imatges seleccionades siguin les que formen la contrasenya. Si el procés d'autenticació és correcte caldria avisar a l'aplicació que utilitza l'esquema de contrasenyes perquè actués. Com ja hem comentat, en el nostre cas s'avisarà a l'activitat associada perquè passi al procés de registre (*p4Activity*). Notem que això no es realitzaria fins al final de la segona fase d'autenticació, per tal que un possible atacant no pugui tenir informació fins al final del procés.

Si l'autenticació no és correcta s'avisarà a l'usuari i es finalitzarà l'aplicació. Aquesta classe s'encarregarà d'emmagatzemar les dades estadístiques a la base de dades, al finalitzar el procés d'autenticació.

interficieCanvas4

Aquest component gràfic s'encarregarà de gestionar la traça de l'usuari en el procés de registre. És absolutament igual a *interficieCanvas* però en aquest cas es mostra un grid i no es mostren imatges ja que s'entén que l'usuari protegirà la pantalla de les mirades en el procés de registre d'una nova contrasenya. Per si de cas tampoc es mostrarà totalment la traça, sols una vista parcial.

itemList

Cadascun dels objectes que formen la llista que es mostra a *reg2Activity* (veure la definició d'aquesta classe per a més detalls).

itemAdapter

Aquesta classe estén la classe del SDK *BaseAdapter* i implementa l'adaptador que permet generar la llista (*ListView*) de *reg2Activity*. Cal implementar el mètode *getView* que li indica a la vista com ha de mostrar cadascun dels components de la llista (*itemList*). Veure la definició de la classe *reg2Activity* per a més detall.

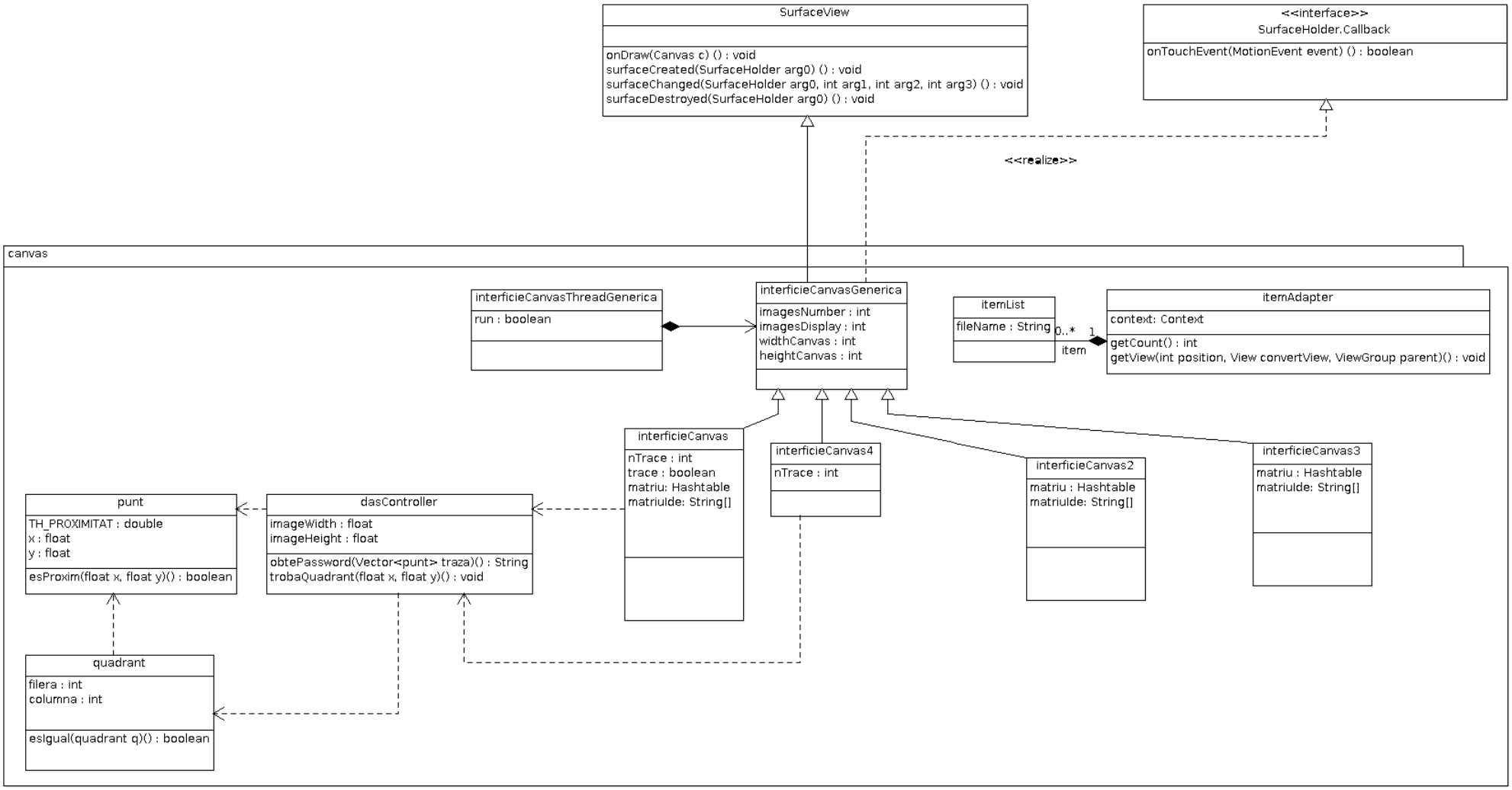


Fig. 19: **Diagrama de classes del paquet *canvas***

Per tal d'entendre millor el funcionament intern de les classes *interficiaCanvas* s'adjunten els diagrames d'estat de *interficieCanvas* (idèntic a *interficieCanvas4*) i de *interficieCanvas2* (idèntic a *interficieCanvas3*).

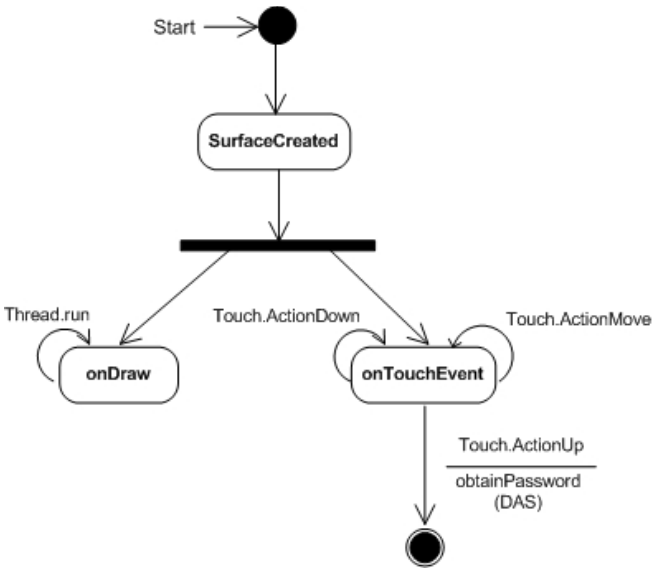


Fig. 20: Diagrama d'estat de *interficieCanvas*

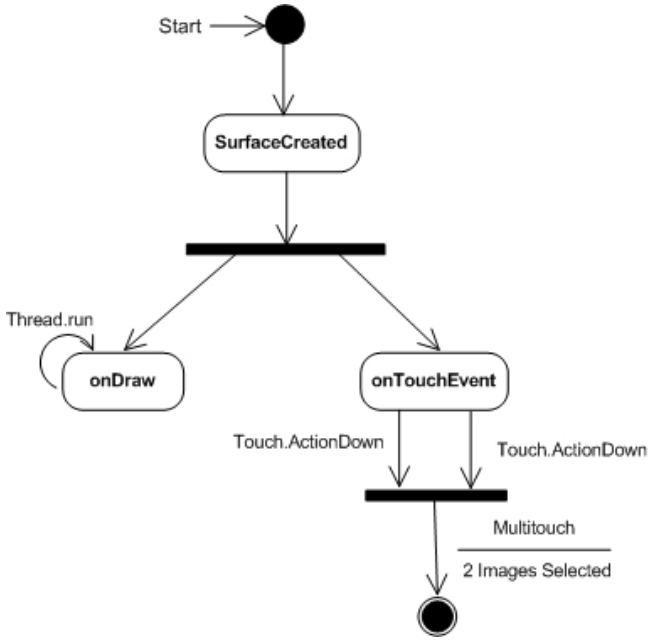


Fig. 21: Diagrama d'estat de *interficieCanvas2*

1.7.3.- *imageTools*

Dintre d'aquest paquet trobarem les classes encarregades d'implementar els tractaments gràfics que rebran les imatges. Normalment els mètodes seran estàtics. Rebran com a paràmetre un objecte de tipus *Bitmap* que és capaç d'emmagatzemar informació gràfica i recodificar-la en diferents formats. Aquesta classe forma part del SDK d'*Android OS*.

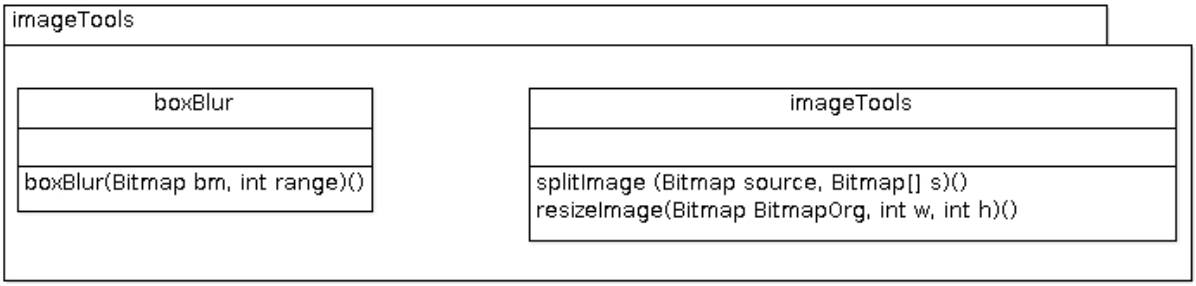


Fig. 22: **Diagrama de classes del paquet *imageTools***

boxBlur

Aquesta classe implementarà la distorsió de la imatge mitjançant el mecanisme de *box blur*[4]. Notem que aquest mecanisme implementa una convolució molt ràpida i més que suficient per al nostre objectiu de distorsionar la imatge per dificultar l'enginyeria social (veure Decisions de Disseny, Ap. 2).

imageTools

Aquesta classe implementarà la funció per dividir la imatge en dues parts (*splitImage*) i el mètode per a fer un reescalat de la imatge *(resizeImage*). Aquest últim mètode s'utilitzarà al capturar mitjançant la càmera una imatge per part de l'usuari. Un cop capturada cal redimensionar-la perquè sigui apte i lleugera per al nostre propòsit (veure *configuration*).

El reescalat d'imatges és un procediment ràpid implementat per el SDK d'*Android OS* mitjançant la classe *graphics.Matrix*.

4 http://en.wikipedia.org/wiki/Box_blur

1.7.4.- *cripto*

Aquest paquet conté les classes encarregades d'implementar els dos algorismes criptogràfics que utilitzarà l'aplicació: l'algorisme d'encriptació simètric AES-128 i la funció resum SHA-512. Tots els mètodes seran estàtics i faran ús de les classes criptogràfiques de *java* (*javax.crypto*).

La justificació de l'ús d'aquests algorismes es pot trobar a l'apartat 2 de la memòria.

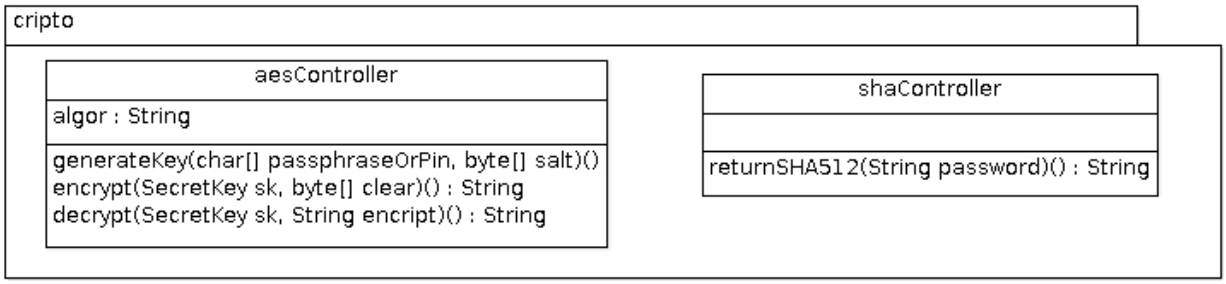


Fig. 23: **Diagrama de classes del paquet *cripto***

Veiem ara cadascuna de les classes que formaran aquest paquet.

aesController

Classe encarregada d'implementar l'algorisme AES. L'atribut *algor* permet escollir el tipus de *cipher* que s'utilitzarà en l'algorisme, per defecte serà la combinació `AES/CBC/PKCS5Padding`, però això no funciona en algunes versions velles d'*Android OS* i per aquesta raó es pot canviar al *cipher* AES que el sistema implementi per defecte.

En el cas de que tot funcioni l'algorisme AES utilitzarà codificació per intercanvi de blocs (CBC), així cada bloc dependrà del resultat de tots els anteriors. El *padding* utilitzat a la sortida de cada bloc serà de tipus PKCS5.

La classe disposarà dels següents mètodes públics:

- *generateKey*: Aquest mètode generarà una clau (*SecretKey*) a partir d'una *passphrase* i d'un *salt*. Recordem que a la nostra aplicació la *passphrase* serà la traça resultant de la fase 1 (DAS). El mecanisme implementat en aquesta funció serà `PBKDF2WithHmacSHA1`. Aquest mecanisme aplica a la *passphrase* una funció de tipus *HMAC* basada en un *hash* SHA-1. La clau obtinguda tindrà una longitud de 128 bits, per tant tindrem una clau de 32 caràcters hexadecimals (16 bytes). El nombre d'iteracions del procés per a derivar la clau serà de 500 passades (cercant el compromís entre velocitat i seguretat).

  La clau generada serà la que utilitzarem com a *SecretKey* per a les funcions d'encriptació i desencriptació mitjançant AES-128.

- *encrypt*: Funció que donada una clau i una cadena de bytes, encriptarà segons l'algorisme AES-128. Ja hem explicat anteriorment quin tipus de *cipher* farà servir.

- *decrypt*: Funció que donada una clau i una cadena de caràcters encriptada, desencriptarà segons l'algorisme AES-128. Ja hem explicat anteriorment quin tipus de *cipher* farà servir.

shaController

Aquesta classe implementarà, mitjançant el mètode *returnSHA512*, la funció resum SHA-512. Per fer-ho es farà ús de la classe *java MessageDigest*.

Aquest mètode s'utilitza per tal de generar una funció resum de la traça DAS i independitzar la longitud d'aquesta de la longitud de la *passphrase* que encriptarà la pròpia base de dades.

1.7.5.- *bdController*

Paquet encarregat de comunicar l'aplicació amb el sistema gestor SQLite3 (veure Disseny de la Base de Dades, Ap. 3.4). Sols contindrà la classe del mateix nom i que serà instanciada un sol cop i mantinguda en memòria mitjançant la classe *configuration* (sols si s'ha pogut obrir sense problemes la base de dades).

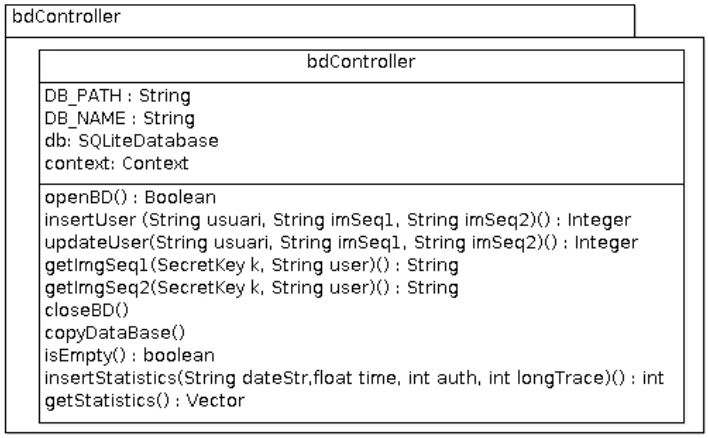


Fig. 24: Diagrama de classes del paquet *bdController*

Els atributs seran bàsicament el context de l'aplicació (objecte del SDK *Android OS* que permet accedir a paràmetres globals de l'aplicació), la ubicació i nom de la base de dades al sistema de fitxers (*DB_PATH* i *DB_NAME*), i finalment l'objecte que implementa el *driver* de comunicació amb els sistema gestor (*db*). Els *drivers* d'accés a la base de dades formen part del paquet del SDK `android.database.sqlite.SQLiteDatabase`.

Veiem els diferents mètodes que implementarà aquesta classe:

- *openBD*: Estableix la connexió amb el sistema gestor de la base de dades.
- *closeBD*: Tanca la connexió amb el sistema gestor de la base de dades.
- *isEmpty*: Indica si la taula *up* està buida.

- *copyDataBase*: Realitza una còpia de la base de dades des del directori *assets* de l'aplicació fins a *DB_PATH.*

- *insertUser*: Verifica que no existeix l'usuari *usuari* a la taula *up* i crea un nou registre a la base de dades. Els paràmetres d'entrada ja han d'estar encriptats. Retorna el nombre de registres actuals a la taula *up* .

- *updateUser*: Verifica que l'usuari *usuari* existeix a la taula *up* i modifica les seves dades. Els paràmetres d'entrada ja han d'estar encriptats. Retorna el nombre de registres modificats.

- *getImgSeq1*: Retorna la primera parella d'imatges que *usuari* té definides com a contrasenya a la fase 2 de l'autenticació. Els paràmetres d'entrada han d'estar en clar i les cadenes resultants seran el nom, en clar, dels dos fitxers que conten les imatges.

- *getImgSeq2*: Retorna la segona parella d'imatges que *usuari* té definides com a contrasenya a la fase 2 de l'autenticació. Els paràmetres d'entrada han d'estar en clar i les cadenes resultants seran el nom, en clar, dels dos fitxers que conten les imatges.

- *insertStatistics*: Insertarà a la taula *statistic* els valors estadístics del procés d'autenticació.

- *getStatistics*: Retorna totes les fileres de la taula *statistic* en format XML.

Quan l'aplicació s'instal·li per primera vegada en un dispositiu amb *Android OS* la base de dades no existirà al sistema d'arxius i l'aplicació no podrà utilitzar-la. Les aplicacions *Android* disposen d'un directori privat de recursos anomenat *assets*. Aquest directori està pensat sols per a lectura i per aquesta raó el primer que cal fer si no es detecta la base de dades al sistema de fitxers és realitzar una còpia byte a byte des del directori de recursos *assets*. Aquesta base de dades que es còpia restarà buida.

1.7.6.- *sfController*

Aquest paquet contindrà les úniques classes que podran interaccionar amb el sistema de fitxers del dispositiu. Totes les classes faran ús del paquet *java.io*.

filter

Aquesta classe implementarà la interfície `FilenameFilter` que amb el seu mètode *accept* permet decidir si un fitxer passa un filtre. Aquest filtre queda determinat en la implementació d'aquesta funció.

filterJPG2* i *filterPNG2

Aquestes classes heretaran de la classe *filter* i permetran implementar la funció *accept* de forma que sols passin els filtres de tipus JPG o de tipus PNG que tinguin un nom format per quatre caràcters alfanumèrics. Aquests seran els tipus de nom estàndard utilitzat en els fitxers de la galeria d'imatges.

Veiem com es definirà el filtre per a *filterPNG2*, a partir de gramàtiques regulars *java*,

```
if(nombreArchivo.matches("\\w\\w\\w\\w\\.PNG")||
                              nombreArchivo.matches("\\w\\w\\w\\w\\.png"))
```

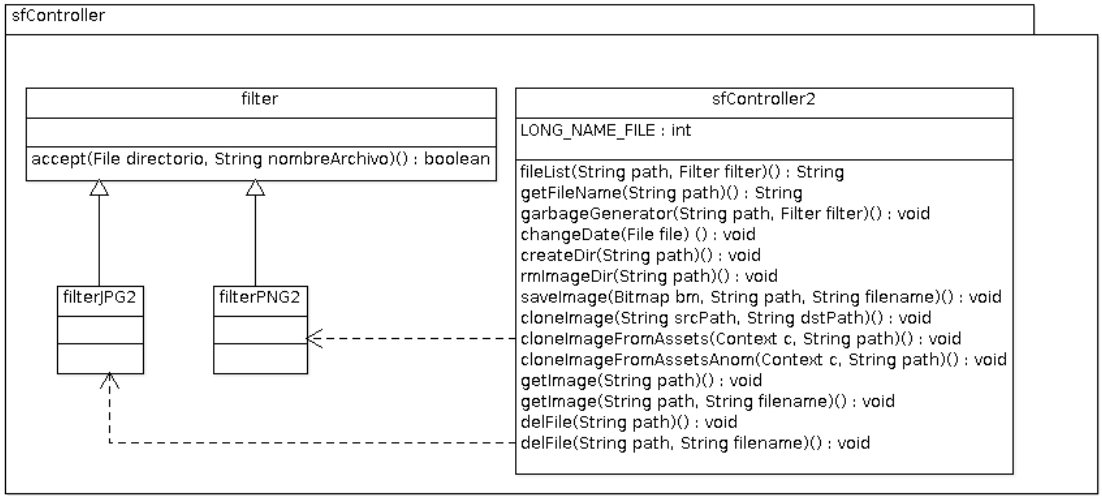


Fig. 25: Diagrama de classes del paquet *sfController*

sfController2

Classe encarregada d'interaccionar amb el sistema de fitxers del dispositiu. L'atribut *LONG_NAME_FILE* indica la longitud que farem servir al generar nous noms d'arxius (4 per defecte).

Veiem ara cadascun dels seus mètodes, alguns d'ells sobrecarregats:

- *fileList*: Donat una *path* i un filtre aquest mètode retorna una llista amb tots els noms de fitxers que passen el filtre, dintre del *path*.

- *getFileName*: Donat un *path*, retorna un nom aleatori no repetit, format per caràcters alfanumèrics de longitud *LONG_NAME_FILE.*

- *garbageGenerator*: Donat un *path* i un filtre, canvia la data a tots els arxius que passin el filtre dintre del *path*. La nova data serà la més vella que reconegui el sistema de fitxers. Aquest mètode es utilitzat per evitar que un possible atacant amb permisos d'accés al sistema de fitxers pugui saber quins són els fitxers modificats recentment.

- *changeDate*: Mètode privat utilitzat pel *garbageGenerator*.

- *createDir*: Donat un *path* absolut, crea un directori si no existeix.

- *rmImageDir*: Donat un *path* absolut elimina de forma recursiva el directori i el seu contingut.

- *saveImage:*Funció sobrecarregada que donat un *Bitmap*, un *path* absolut, o bé un *path* i un nom de fitxer, emmagatzema la imatge al sistema de fitxers. El format de l'arxiu gràfic ve

donat per *configuration.Extension*.

- *cloneImage*: Donat un *path* d'origen, còpia l'arxiu a un *path* destí. Els dos *paths* han de ser absoluts.

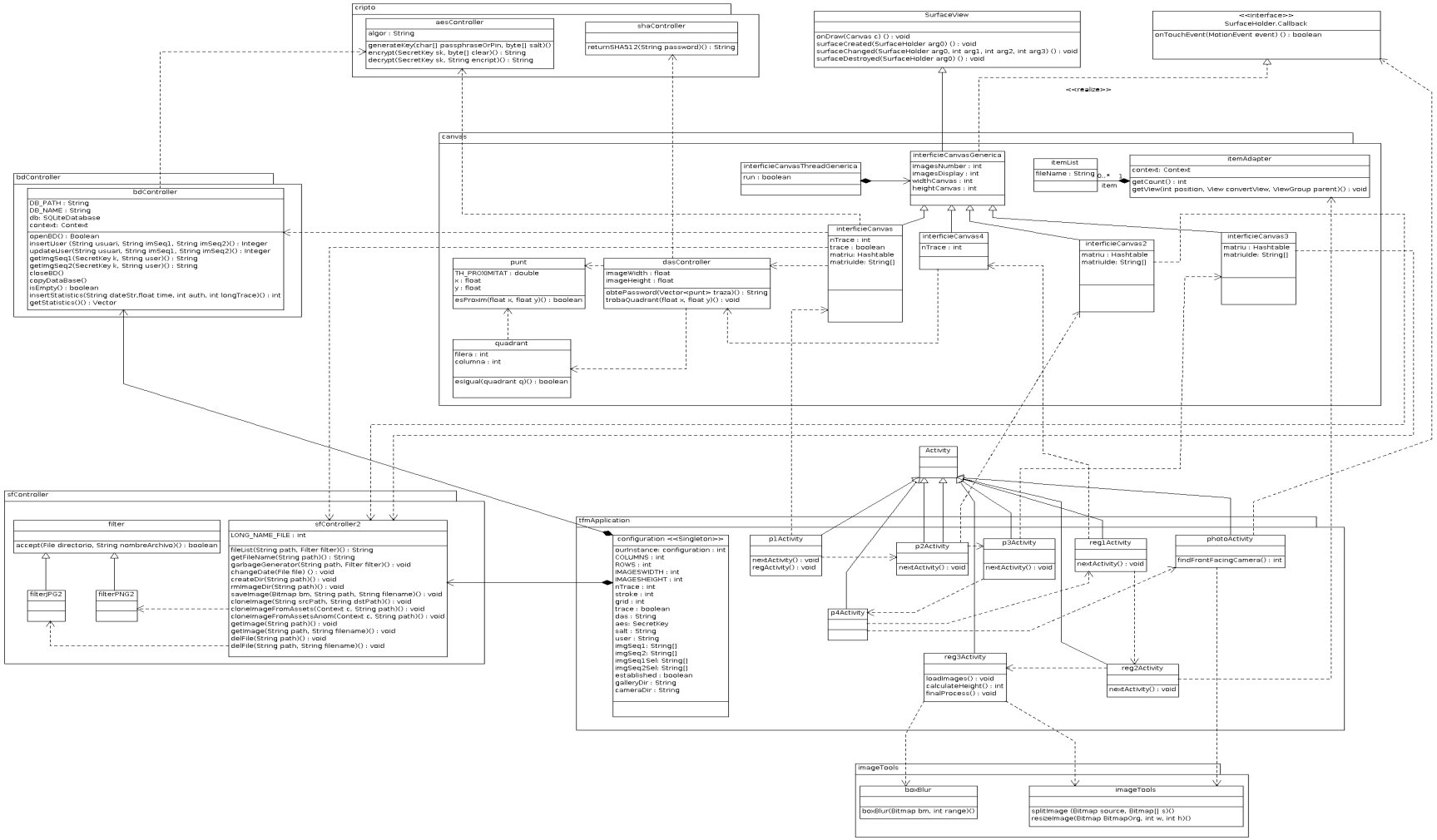
- *cloneImageFromAssets*: Donat un *path* de destí, agafa tots els fitxers d'imatges, segons *configuration.Extension*, del directori *assets* i els copia. Veure la definició de la classe *bdController* per entendre el significat del directori *assets*.

- *cloneImageFromAssetsAnom*: Igual que el mètode anterior, però la imatge copiada obté un nou nom generat per *getFileName*.

- *getImage*: Funció sobrecarregada que ens retorna un *Bitmap* a partir del fitxer gràfic indicat, ja sigui amb *path* absolut o amb *path* i nom de l'arxiu.

- *DelFile*: Mètode sobrecarregat que elimina l'arxiu indicat, ja sigui amb *path* absolut o amb *path* i nom de l'arxiu.

4.7.7.- Diagrama complert de classes



1.8.- Diagrames de seqüència

En aquest apartat es mostra com interactuaran les classes i els mètodes d'aquestes entre sí per tal de realitzar el procés d'autenticació i el procés de registre[5].

1.8.1.- *Procés d'autenticació*

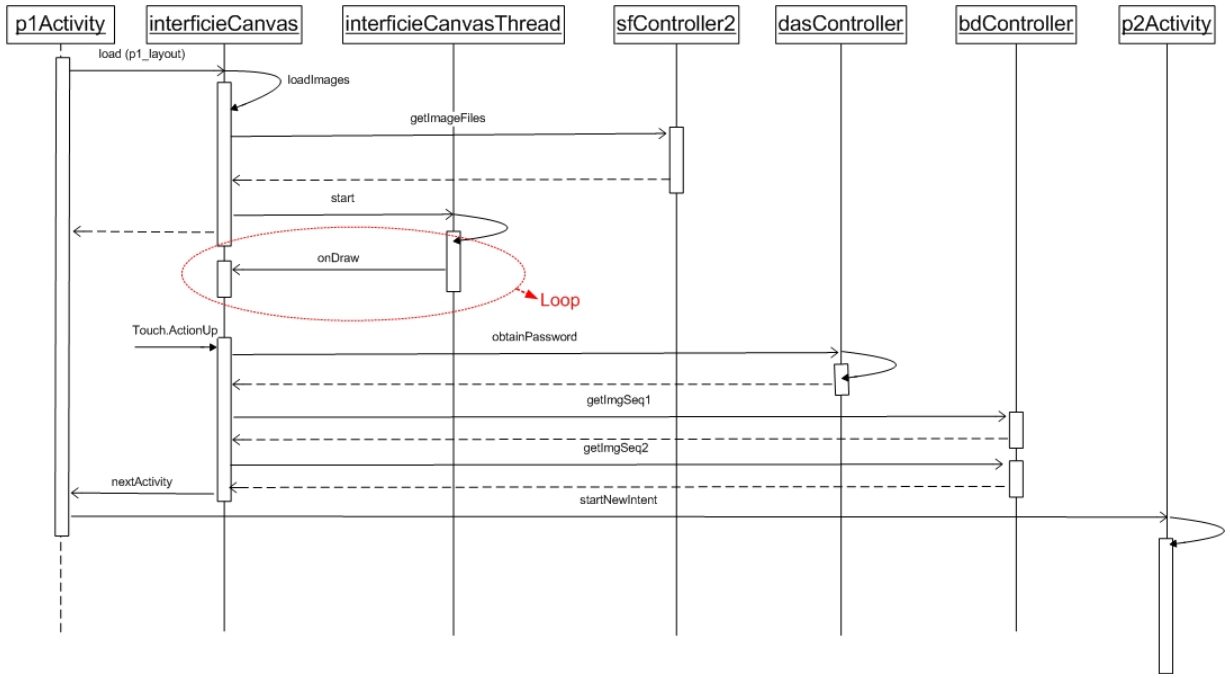


Fig. 26: Procés d'autenticació - 1

El procés d'autenticació s'iniciarà a *p1Activity* on l'usuari realitzarà la seva traça (DAS). Aquesta part és el que hem anomenat anteriorment fase 1. Com ja s'ha explicat en el disseny de les classes, l'*Activity* serà l'encarregada de mostrar una vista formada per un objecte propi anomenat *interficieCanvas*, que fent ús del controlador del sistema de fitxers (*sfController2*), mostrarà una matriu d'imatges aleatòries de la galeria d'imatges. Gràcies al *thread* associat es produirà el refresc de pantalla (*onDraw*).

Quan l'usuari hagi acabat la seva traça, a l'aixecar el dit o el punter, la pantalla generarà un event de tipus *Touch.ActionUp*, en aquest moment, gràcies al *dasController*, es recolliran tots els punts generats per l'usuari en la traçada i es convertiran en una llista en ordre cronològic de cel·les (*quadrants*) creuats per aquesta traça. Aquesta llista serà resumida mitjançant un *hash* SHA-512 (aquesta part i la referent a l'obtenció de la clau AES-128, on es fa ús del *shaController,* no apareix al diagrama per no complicar-lo més).

Amb la traça resumida i a mode de *passphrase* obtindrem la clau que ens permetrà desencriptar la base de dades i obtenir les imatges que formen la contrasenya de l'usuari a la fase 2 (*bdController*, crides *getImgSeq1* i *getImgSeq2*).

La vista avisarà a l'activitat que ja pot realitzar la crida a la següent activitat (*nextActivity*), en aquest cas *p2Activity* (*startNewIntent*).

5Algunes funcions no corresponen a les reals per tal de simplificar el diagrama.

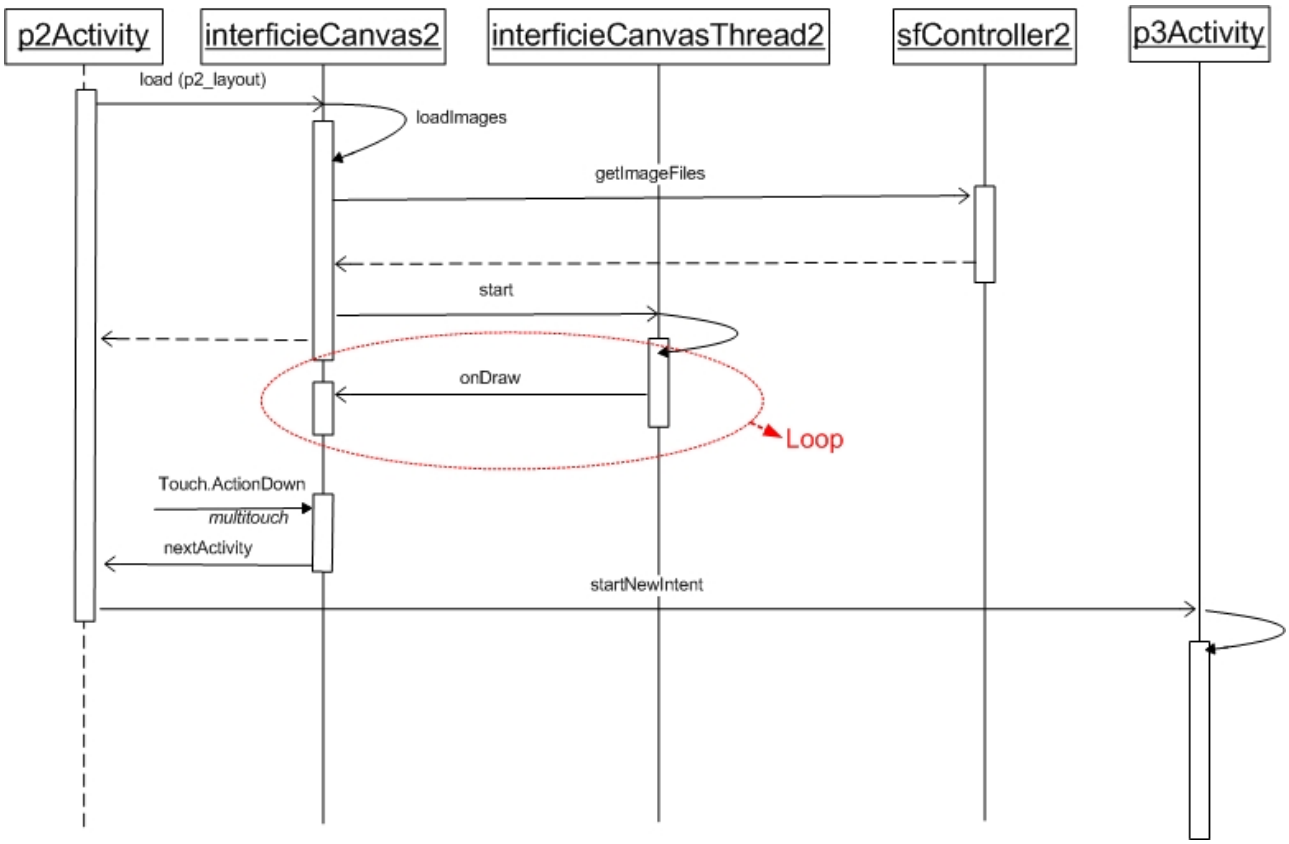


Fig. 27: Procés d'autenticació - 2

p2Activity serà l'encarregada d'activar la vista que, mitjançant *interficieCanvas2*, ens mostrarà una matriu amb imatges. El procediment és el mateix que a l'anterior activitat, però en aquest cas caldrà assegurar que dintre d'aquesta matriu hi apareix la parella d'imatges que formen la primera part de la contrasenya. Com que a l'anterior activitat hem pogut desencriptar la base de dades no tindrem problemes per conèixer aquesta dada. Si la traça (DAS) fos errònia i no haguéssim pogut desencriptar la base de dades, totes les imatges mostrades serien de la galeria d'imatges, però sense indicar que el procés d'autenticació no és vàlid.

En aquest cas s'esperarà, per partida doble (*multitouch*), un event de tipus *Touch.ActionDown*, generat a l'utilitzar el punter o el dit sobre la pantalla. Un cop detectats s'emmagatzemarà quines dues imatges han estat seleccionades i es procedirà a activar *p3Activity*.

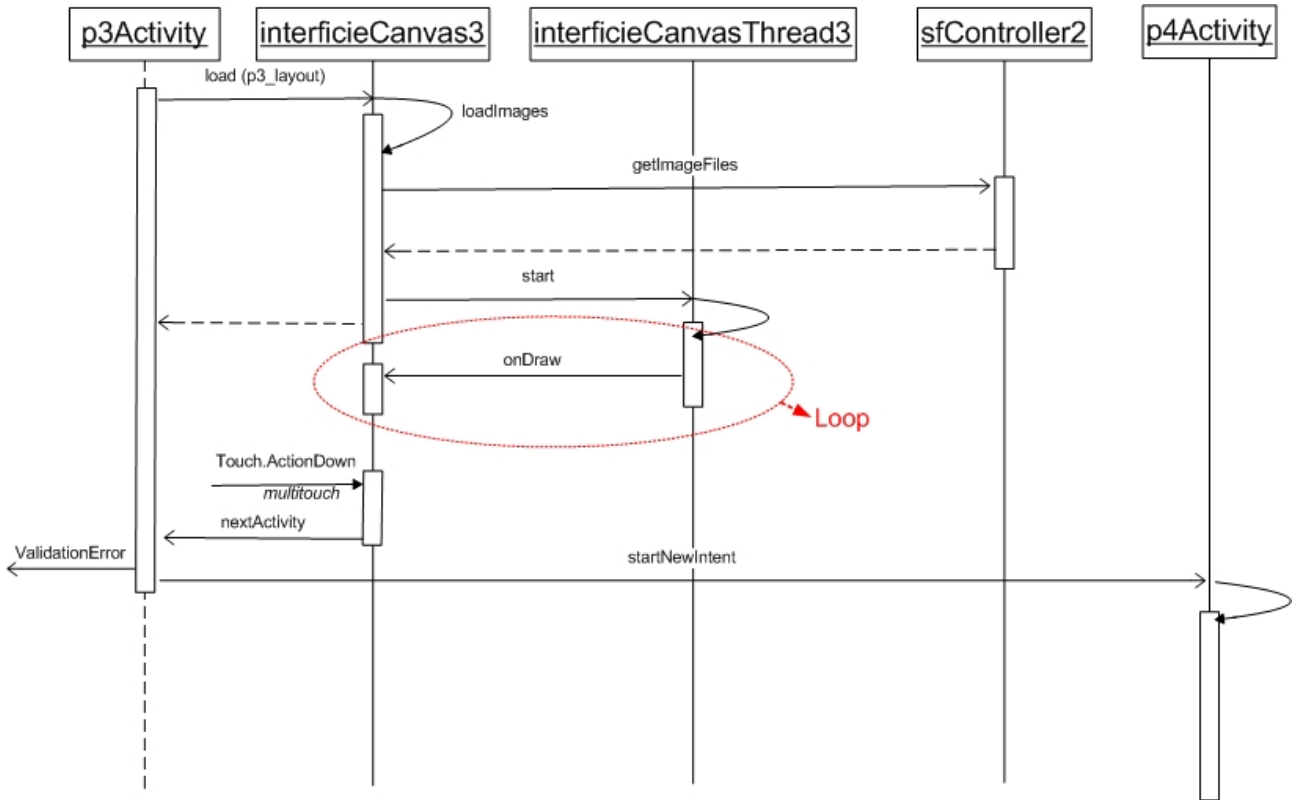


Fig. 28: Procés d'autenticació - 3

En aquest cas el procés és exactament igual a l'explicat per a *p2Activity*, però ara l'usuari seleccionarà la segona parella d'imatges que formen la contrasenya. Un cop fet això, es comprovarà que les quatre imatges seleccionades (*p2Activity+p3Activity*) siguin les correctes. Si és així caldria avisar a l'aplicació que utilitza l'esquema de contrasenyes.

Si el procediment d'autenticació no és correcte, en aquest punt es mostrarà un avís a l'usuari i se sortirà de l'aplicació.

Recordem que en el nostre cas, a mode de prototipus, si l'autenticació ha estat correcta es cridarà a *p4Activity*, que és l'inici del procés de registre.

1.8.2.- Procés de registre

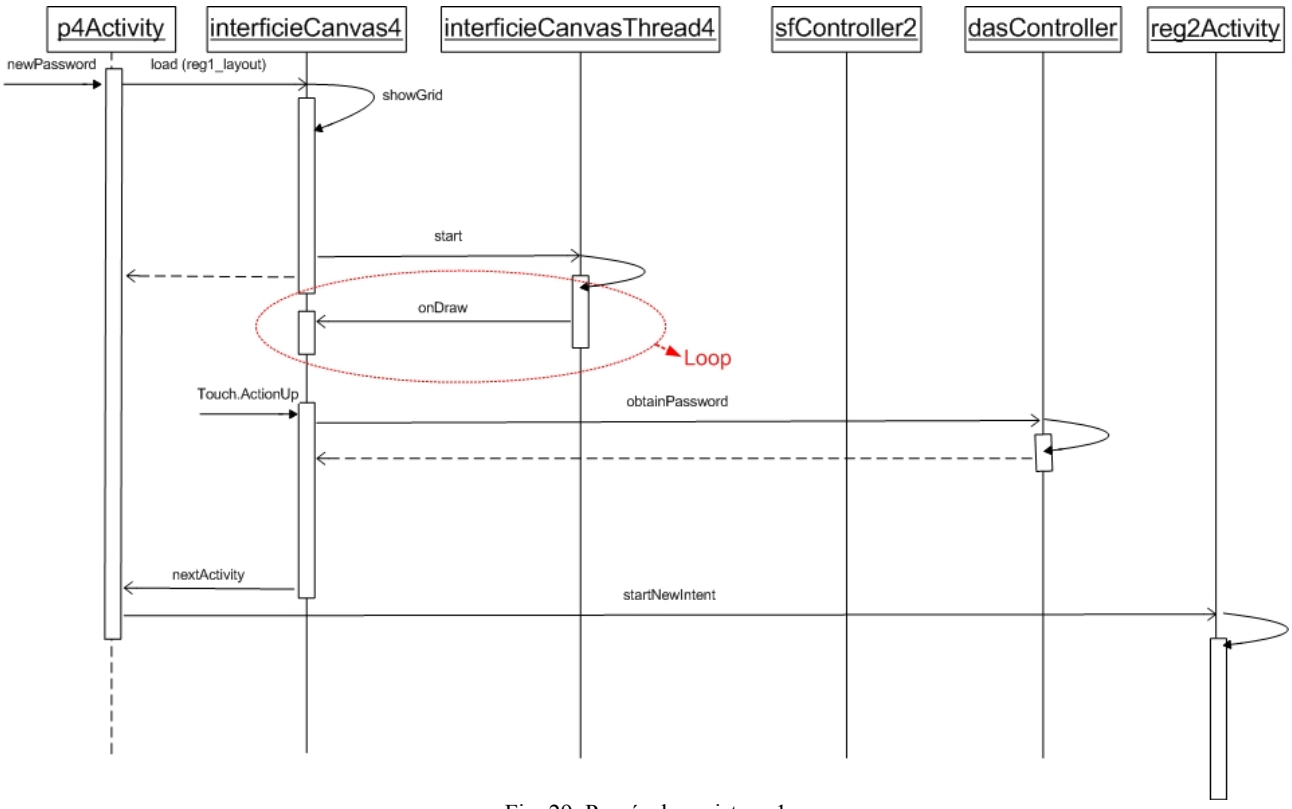


Fig. 29: Procés de registre - 1

El procés de registre s'iniciarà a *p4Activity* al fer ús del botó *New Password* (veure Disseny de Pantalles, Ap. 3.5). Mitjançant *reg1Activity* (no apareix al diagrama per simplificar) es carregarà l'objecte *interficieCanvas4*, que ens mostrarà un *canvas* on l'usuari podrà definir la nova traça. Tot el procés funciona de forma idèntica al que feia *p1Activity*, ja explicada al procés d'autenticació, sols que en aquest cas no caldrà mostrar una matriu d'imatges de fons, ja en tindrem prou amb un grid (*showGrid*).

Quan la traça hagi estat definida, es carregarà la següent activitat, *reg2Activity*.

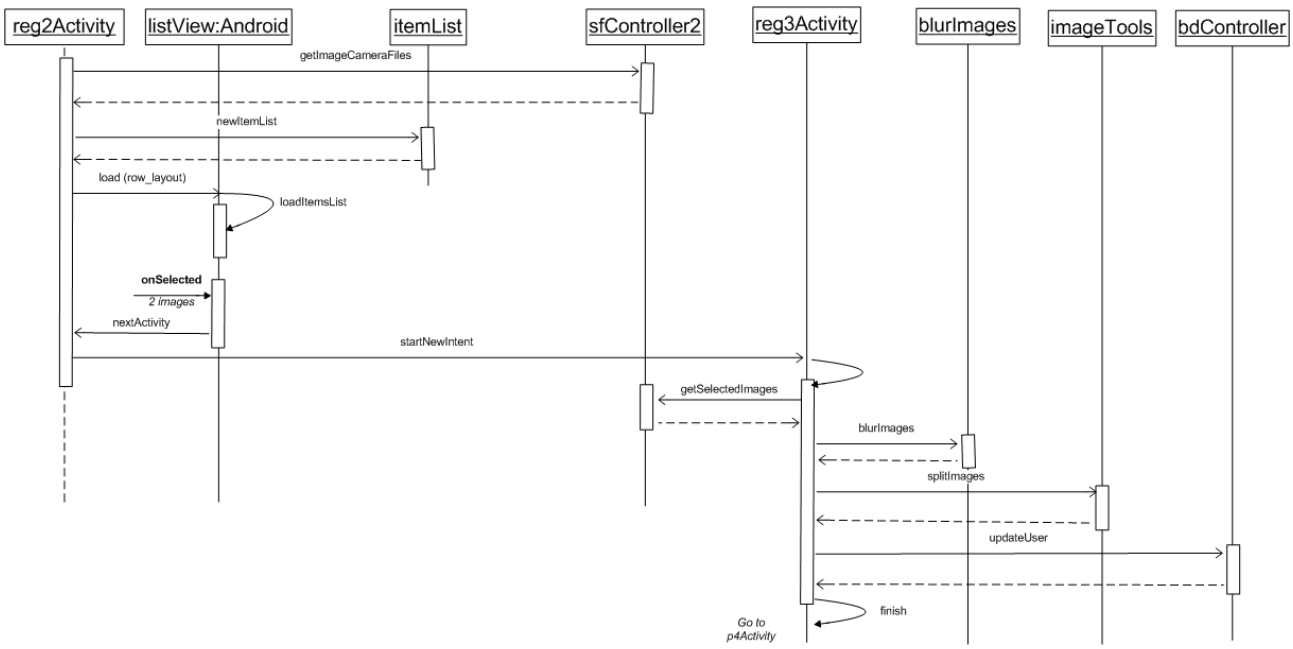


Fig. 30: Procés de registre - 2

En aquest cas, *reg2Activity* crearà una llista (*ListView*) amb totes les imatges del directori de la càmera, on van a parar les imatges captades per l'usuari. Cadascuna d'aquestes imatges es mostrarà amb el seu nom al costat (*row_layout*), formant una llista d'items (*itemList*).

Caldrà que l'usuari seleccioni dues imatges, que seran les que formaran la seva contrasenya en la fase 2. Seguidament es farà una crida a *reg3Activity*, que ens mostrarà a la seva vista les dues imatges triades, i el resultat després de distorsionar-les i dividir-les en dues parts. Si l'usuari accepta el resultat final, es modificaran les dades de la contrasenya (nova traça + noves imatges) a la base de dades.

Recordem (veure Decisions de Disseny, Ap. 2) que de les dues imatges triades i ara partides, a la primera pantalla de la segona fase d'autenticació hi apareixeran les primeres porcions de les dues imatges triades, mentre que a la segona pantalla hi apareixeran les dues porcions restants.

Per finalitzar aquest apartat veiem el diagrama de seqüència de la captura d'una imatge per part de l'usuari.

Sols destacar que un cop capturada la imatge serà reescalada segons els paràmetres de la classe *configuration*.

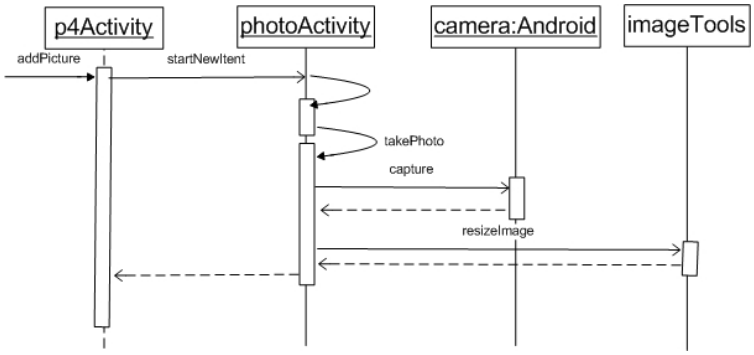
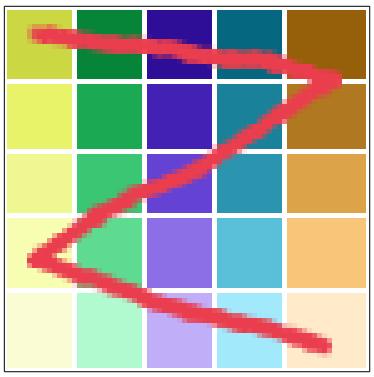


Fig. 31: Procés de captura d'una imatge amb la càmera

Treball Final de Màster

Contrasenyes Gràfiques
Annex - 4



Màster Interuniversitari de Seguretat de les Tecnologies de la Informació i de les Comunicacions
- *MISTIC* -

Alumne: Josep Escoda Herrando
Directora: Cristina Pérez Solà
Universitat: UOC

Curs 2013- 2014, Sem. 1

Índex de continguts

1.- Implementació de l'aplicació

L'aplicació dissenyada ha estat implementada mitjançant l'eina *Android Studio 0.3.1* de Google sobre un sistema *Ubuntu Linux 12.04 x64*. Programada en el llenguatge *Java* versió 1.7.0, utilitzant el *SDK* d'*Android OS* versió 4.2.2. La compilació s'ha realitzat amb suport del *Gradle 1.8*.

L'aplicació ha estat dissenyada i implementada per a dispositius mòbils amb sistema operatiu *Android* 4.2.2 o superior, amb capacitat *multitouch*, càmera i pantalla de 4" o més. Per a la fase de proves s'ha utilitzat l'emulador d'*Android AVD* en diferents formats i els dispositius físics *Google Nexus 7* i *Samsung Galaxy Mini*.

L'aplicació ha estat signada i distribuïda per a fer proves en format *apk*. Al ser instal·lada requereix que l'usuari accepti donar-li permisos per a fer ús de la càmera i per a tenir accés a internet (per enviar les estadístiques).

L'aplicació requereix 3,71 MB d'espai per a ser instal·lada més l'espai necessari per augmentar la galeria d'imatges per part de l'usuari.

1.1.- Esquema de classes implementat

L'estructura del projecte dintre d'*Android Studio* és la següent,

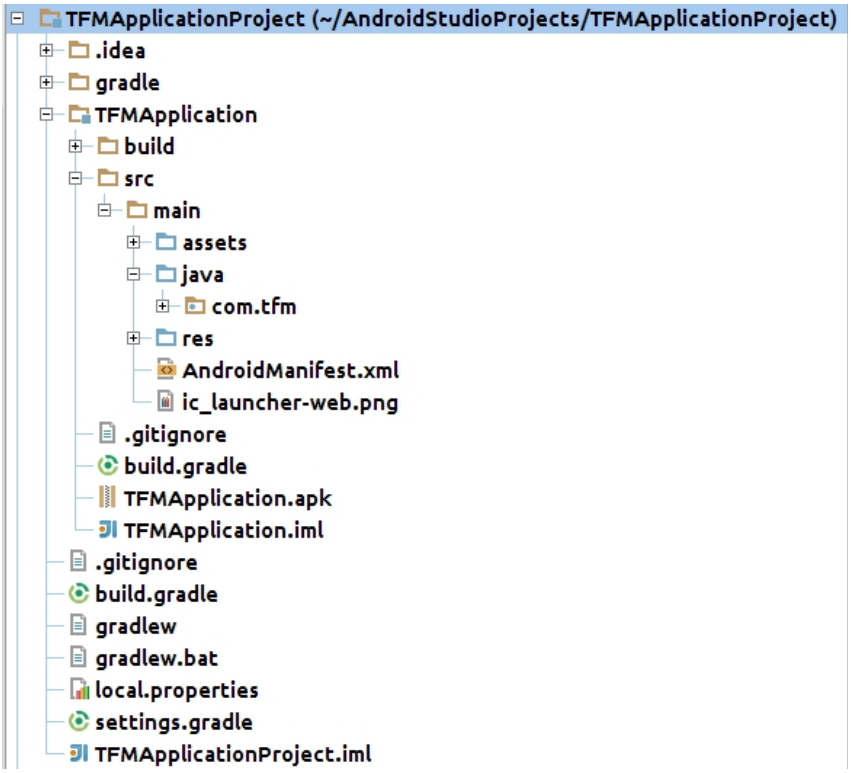


Fig. 1: Estructura del projecte

Totes les classes formen part del paquet *com.tfm*, seguint el següent esquema:

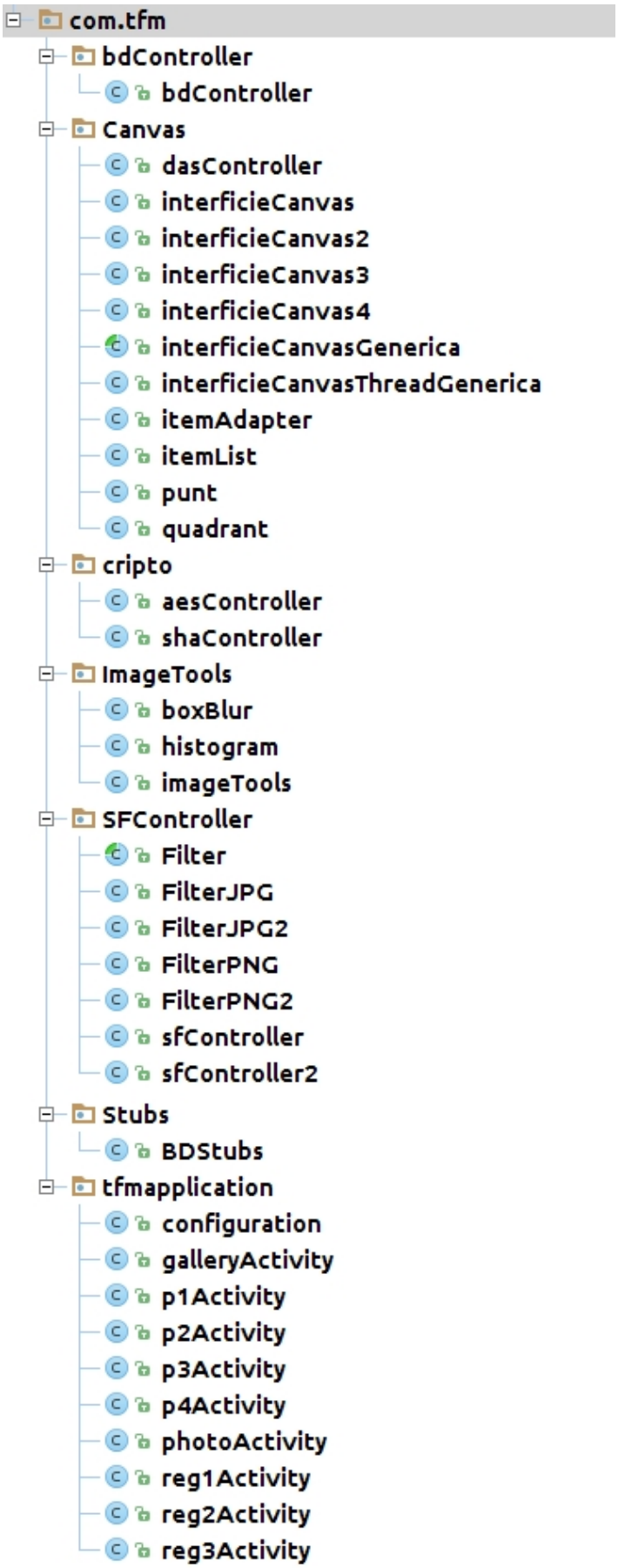


Fig. 2: Estructura de classes

Els *layouts* implementats, seguint les especificacions de disseny han estat els següents,

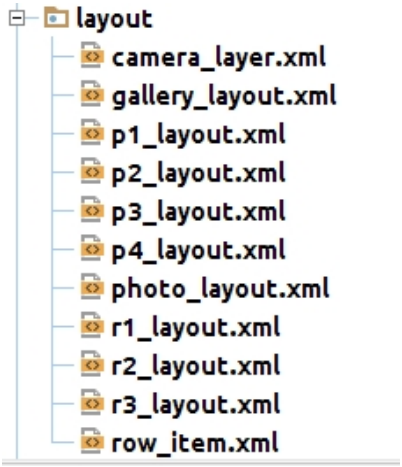


Fig. 3: *Layouts* implementats

1.2.- Classes que no formen part del disseny de l'aplicació

Si mirem la fig.2 hi ha un conjunt de classes que no formen part del disseny de l'aplicatiu, però hi van formar part en algun moment del desenvolupament i per tant s'han mantingut en la solució final. Són les següents:

Stubs.BDStubs: Aquesta classe s'ha utilitzat a la fase de proves per tal d'inserir dades a la taula *up* de la base de dades i accelerar algun procediment, com per exemple, no haver de realitzar fotografies per canviar la contrasenya, o afegir directament un usuari amb contrasenya per a provar les fases d'autenticació.

imageTools.Histogram: En una branca paral·lela del desenvolupament de l'aplicació s'utilitzaven dades referents a l'histograma de les imatges per tal d'identificar les imatges que formaven la contrasenya i no haver d'encriptar la base de dades.

SFController.SfController: Primera versió del controlador del sistema de fitxers que permet treballar amb imatges esteganografiades, un solució que finalment ha estat descartada per fràgil.

SFController.FilterPNG i *SFController.FilterJPG*: Filtres de fitxers que permeten escollir fitxers sols per l'extensió, sense un format concret en el nom. Finalment l'aplicació utilitza un format estàndard per a nombrar els fitxers i s'han utilitzat les classes *SFController.FilterJPG2* i *SFController.FilterPNG2*, molt més restrictives.

1.3.- El sistema de fitxers

Android OS disposa d'un sistema de fitxers propi, on a cada aplicació instal·lada se li assigna una carpeta dintre del directori */data/data*, on poder desar les dades que utilitzi. El contingut d'aquest directori, per defecte, sols és accessible per a un usuari privilegiat (*root*).

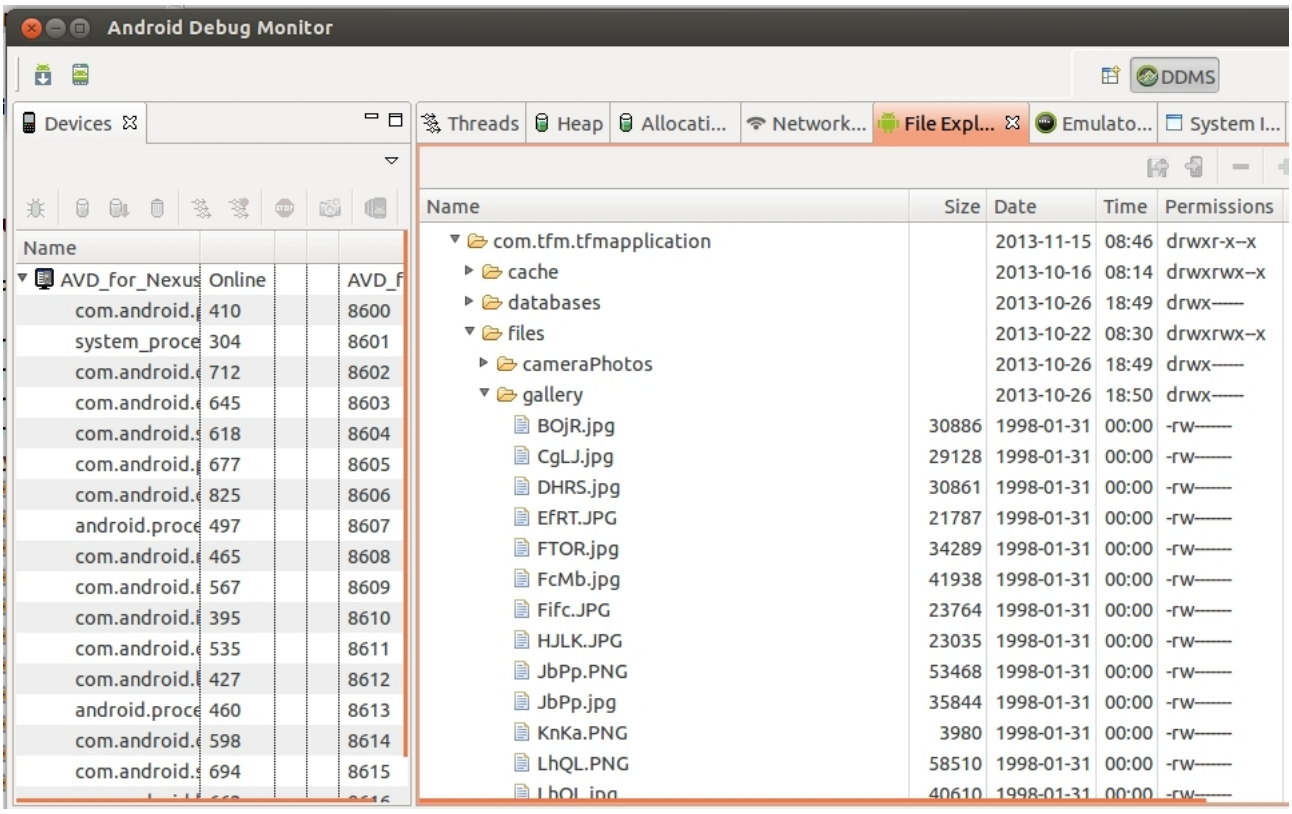


Fig. 4: Sistema de fitxers de l'aplicació

La nostra aplicació disposa d'una carpeta coneguda com *assets* en el paquet instal·lable, que és on s'emmagatzemen la base de dades i la galeria d'imatges. Un cop instal·lada al dispositiu, el primer cop que s'iniciï l'aplicació es crearà el sistema de fitxers al dispositiu (*/data/data/com.tfm.tfmapplication*) i es generaran les següents carpetes:

- *databases*: En aquesta carpeta serà copiada la base de dades buida si no existeix ja (veure subapartat següent).

- *files*: En aquesta carpeta s'emmagatzemaran els fitxers necessaris per a l'aplicació. En ell es crearan les següents carpetes:

    - *cameraPhotos*: Carpeta on s'emmagatzemaran temporalment les imatges captades amb la càmera. Un cop se surti de l'aplicació tot el directori serà esborrat. Això és així per no deixar traça sobre les possibles imatges escollides com a contrasenya.

    - *gallery*: Carpeta on s'emmagatzemen totes les imatges que formen la galeria. Aquestes imatges són les que es mostren a l'usuari en el procediment d'autenticació. Notem que les imatges utilitzades com a contrasenya també resten en aquesta galeria,

perquè no siguin identificades respecte la resta d'imatges que la formen.

Els fitxers inclosos a la galeria poden ser de tipus *jpg* o *png* segons la versió de l'aplicació utilitzada, i tenen com a nom quatre caràcters alfabètics (*case sensitive*). També és interessant observar (fig. 4) que totes les imatges tenen la mateixa data de modificació / creació per tal que un possible atacant no pugui saber quines han estat les imatges afegides recentment a la galeria (ja siguin part de la contrasenya o no). A aquest procediment l'hem anomenat *garbage* i es produeix cada vegada que entra una nova imatge a la galeria.

La gestió de la galeria és un tema obert i que no ha estat treballat en aquest projecte. Per a enfortir l'esquema seria bo que la galeria fos el més variada possible en quant a temàtica de les imatges que la formen. Una possible gestió d'aquesta seria fer ús de servidors de descàrrega de galeries, i anar-la canviant cada cert temps per part de l'usuari.

La resta de directoris són propis de l'arquitectura de les aplicacions *Android OS*.

1.3.1.- La base de dades al sistema de fitxers

A l'aplicació implementada com a prototipus, si es detecta que la base de dades no existeix o és buida, es copia des del directori d'instal·lació *assets* i automàticament s'inicia el procediment de registre d'una nova contrasenya.

Això s'ha implementat així per facilitar la fase de proves, on diferents usuaris han de testejar l'aplicació per poder extraure les conseqüents estadístiques. En un entorn real la base de dades sols hauria de ser creada per un procés independent i executat amb garanties de seguretat.

Si quan la base de dades és buida o inexistent (primera execució) l'aplicació passa directament al procediment de registre d'una nova contrasenya. Un usuari amb privilegis sobre el dispositiu (*rootejat*) fàcilment pot eliminar la base de dades existent, forçant a l'aplicació a que li permeti generar una nova contrasenya i evitant la protecció que ofereix l'esquema implementat. Per aquesta raó, en un entorn real l'aplicació no haurà de permetre l'autenticació si la base de dades és buida o inexistent. Es podria pensar en dotar l'aplicació d'una base de dades amb una contrasenya per defecte, aplicable sols a la primera execució, però com que les claus per encriptar / desencriptar depenen del dispositiu, això no és factible.

Una altra solució per generar les dades de la base de dades passaria per implementar un protocol de tal forma que sols pugui ser creada en el moment d'instal·lació de l'aplicació, amb una contrasenya per defecte, però el sistema de distribució APK d'*Android OS* no contempla aquest protocol, igual que no contempla tampoc que un dispositiu pugui estar *rootejat*... Una altra possible solució és una arquitectura client-servidor on l'usuari pugui descarregar la base de dades mitjançant una contrasenya.

Per altra banda, si un usuari té privilegis d'administrador, pot realitzar un atac substituint la base de dades del propietari del dispositiu per una de seva, amb les dades que l'atacant ha generat amb la mateixa aplicació (una contrasenya creada i coneguda per ell).

Si aquesta nova base de dades l'ha creat mitjançant l'execució de l'aplicació en el seu propi

dispositiu, l'atac no funcionarà, ja que la clau que encripta la base de dades depèn de la traça generada, però també d'un *salt* independent per a cada dispositiu, anomenat *secure.android_id*. Per tant a l'executar l'aplicació amb la base de dades canviada al dispositiu atacat no es podrà tornar a generar la mateixa clau d'encriptació.

Si la nova base de dades es vol generar al mateix dispositiu (per tal d'usar el mateix *salt*), s'haurà de desinstal·lar prèviament l'aplicació (no poden estar duplicades) i tampoc funcionaria l'atac. Així que l'única opció disponible és modificar el codi de l'aplicació per tal d'utilitzar el *secure.android_id* del dispositiu atacat o bé fer *spoofing*[1] d'aquest identificador al dispositiu de l'atacant.

És fa difícil lluitar contra aquesta situació ja que l'atacant és un usuari privilegiat i té control total sobre el dispositiu, per analogia és com deixar un servidor amb accés d'administrador en mans d'un usuari malintencionat. L'atacant no tindrà cap problema per obtenir el *secure.android_id* del dispositiu atacat i canviar-lo al seu dispositiu. Una possible solució passa per afegir un *pin* al *salt*, de tal forma que l'usuari l'hagi d'escriure a l'hora de fer servir l'aplicació, complicant encara més l'esquema presentat. Si l'atacant no coneix aquest *pin* no podrà generar una clau d'encriptació vàlida, encara que utilitzi el mateix *secure.android_id*.

1 http://www.strazzere.com/blog/tag/spoofing-android_id/

Manual d'Usuari

L'esquema de contrasenyes que vostè té a les seves mans consta de dues fases en el procés d'autenticació. En la primera d'elles vostè haurà de recordar una traça seguint les cel·les que la pantalla defineix mitjançant imatges aleatòries de la galeria d'imatges. En una segona fase, vostè haurà d'escollir dues parelles d'imatges, repetint dues vegades aquest procediment. Notarà que les imatges que apareixen a la seva pantalla estan distorsionades, per dificultar possibles mirades indiscretes.

Si el dibuix de la traça i l'elecció de les imatges correspon amb el que vostè havia definit com a contrasenya personal, tindrà accés a l'aplicatiu protegit. Si la seqüència no és la correcta, se li mostrarà un missatge d'error a l'acabar les dues fase d'autenticació, no abans.

Al iniciar-se l'aplicació per primera vegada no hi haurà cap contrasenya definida a la base de dades, per aquesta raó aquest prototipus el portarà directament al procés de registre[1].

Registre d'una nova contrasenya

A la fase de registre vostè haurà de definir la seva contrasenya. Se li mostrarà el menú de gestió de l'aplicatiu, tal com es veu a la següent imatge:

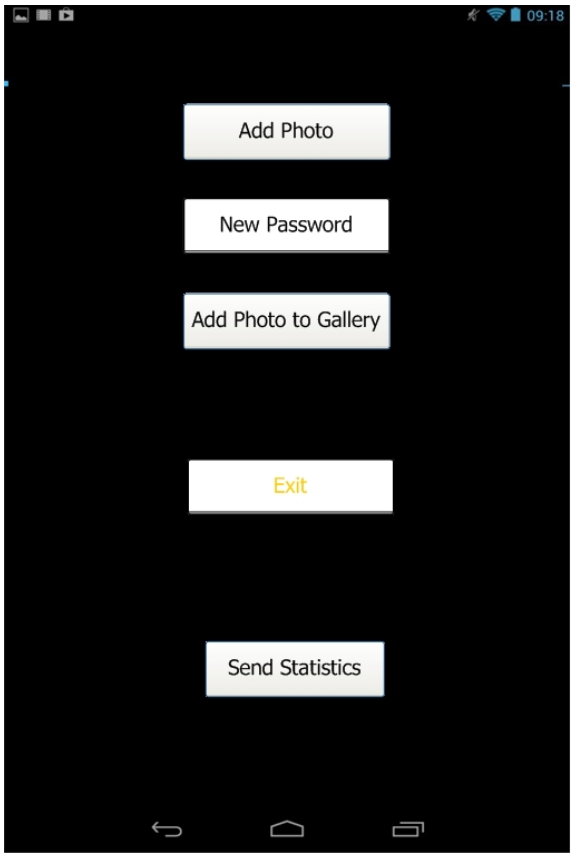


Fig. 1: Menú de l'aplicació

Prenent el botó **Exit** vostè podrà sortir de l'aplicació.

1 En una versió definitiva aquest registre inicial l'haurà de portar a terme un altre procés.

Pas 1

Vostè haurà d'escollir l'opció **Add Photo** per tal de fotografiar el seu entorn i crear les dues imatges que formaran la seva contrasenya. Recordi que cal crear com a **mínim dues fotografies** (fig. 2).

Fig. 2: Captura de dues fotografies per formar la contrasenya

Per capturar les fotografies sols caldrà que enfoqui la seva càmera a l'entorn que vol capturar i premi el botó **Take a Photo**. Aquesta s'emmagatzemarà automàticament a la galeria d'imatges de la càmera.

Totes les imatges registrades amb la càmera seran **eliminades al sortir de l'aplicació**.

Si el seu dispositiu disposa de dues càmeres, l'aplicació escollirà per defecte la càmera posterior (la de més resolució normalment). Si vostè no disposa de cap càmera, no podrà utilitzar aquest aplicatiu.

Pas 2

Un cop registrades les dues fotografies, com a mínim, haurà d'escollir l'opció **New Password** de la pantalla de gestió (fig. 1). Amb aquesta opció ser li presentarà una pantalla amb cel·les on vostè haurà de definir la nova traça que formarà la primera fase del procés d'autenticació.

Sols cal que amb el dit dibuixi la traça per la pantalla, tenint en compte les cel·les per on passa i en especial l'ordre en el que ho fa. Recordi que vostè haurà de ser capaç de reproduir aquesta traça, a nivell de cel·les, a l'hora d'autenticar-se.

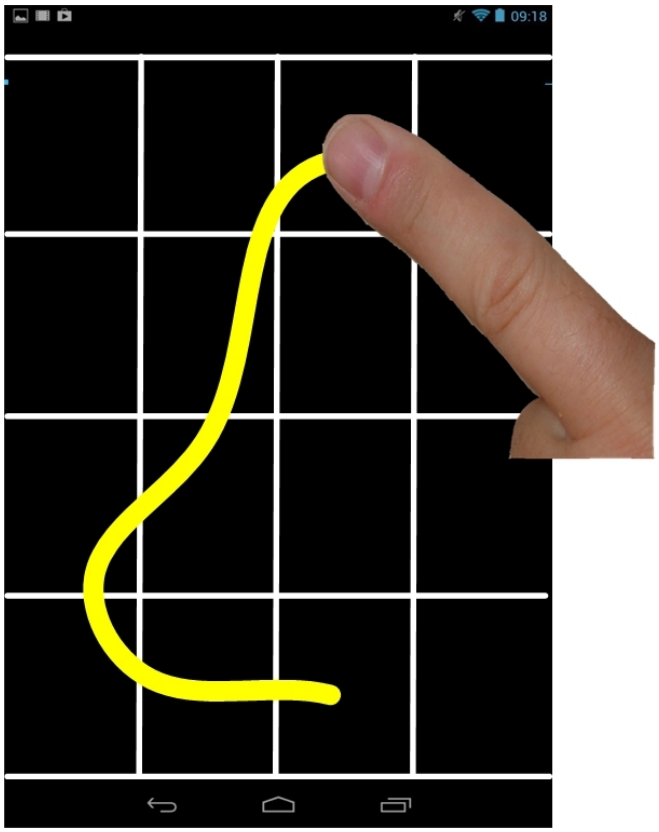


fig. 3:Realització de la traça

Si la traça va desapareixent a l'hora que la dibuixa no es preocupi, és un mecanisme per evitar mirades indiscretes. És important que tingui en compte que **una traça no podrà ser discontinua**, és a dir, un cop aixequi el dit de la pantalla es considerarà que és el final d'aquesta.

Li recomanem de cara a fer més segura la seva contrasenya que eviti simetries i que utilitzi una traça de 7 o més cel·les de longitud.

Pas 3

Un cop definida la nova traça li apareixerà una llista amb totes les imatges capturades amb la càmera. Vostè haurà d'**escollir les dues imatges** que formaran la seva contrasenya.

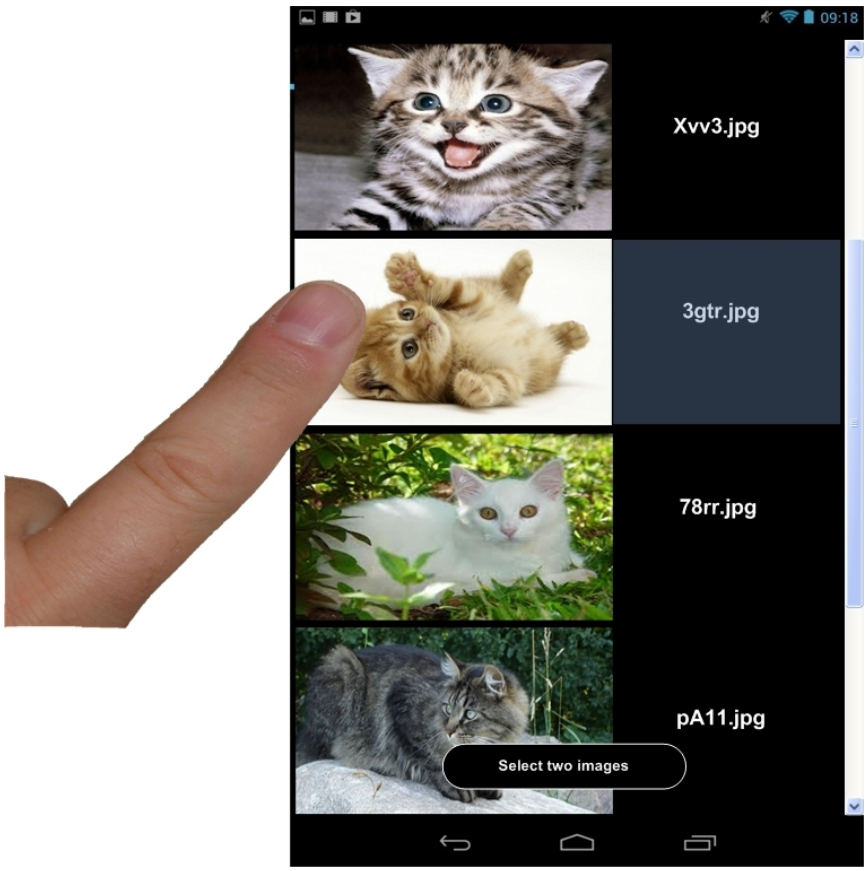


Fig. 4: Elecció d'imatges de la contrasenya

Pas 4

Les dues imatges escollides seran distorsionades i partides en dos trossos. El resultat d'aquest procés es mostrarà a l'última pantalla del procés de registre. Si vostè està d'acord amb el resulta cal que premi el botó **Accept** i la seva contrasenya s'actualitzarà, tornant a la pantalla de gestió (fig. 1).

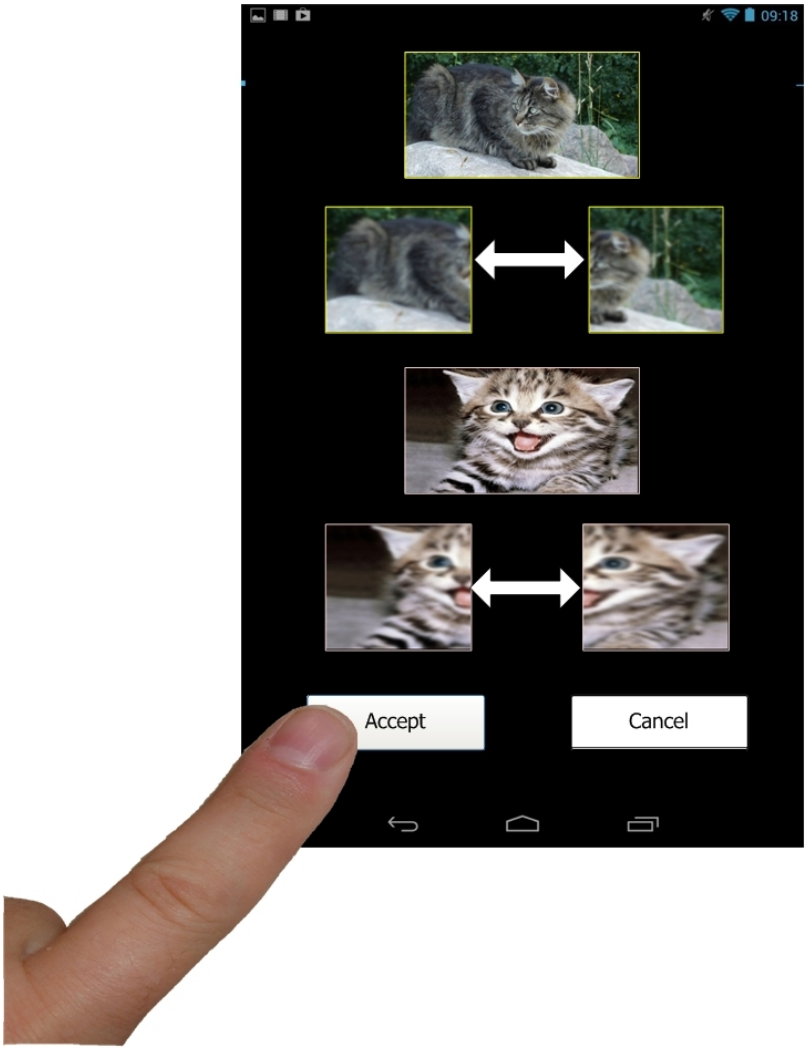


Fig. 5: Resultat final del registre

Recordi que aquestes quatre noves imatges (les dues originals dividides en dues parts) seran les que formaran les parelles a escollir a la segona fase del procés d'autenticació. Si ens fixem en la fig. 5, a la primera pantalla, després de dibuixar la traça en el procés d'autenticació, es mostraran les **dues imatges partides del costat esquerra** (aquestes seran la primera parella), mentre que a la següent pantalla es mostraran les **dues imatges partides de la dreta** (aquestes seran la segona parella).

Si existeix alguna contrasenya anterior, les imatges d'aquesta seran eliminades al crear la nova contrasenya.

Procediment d'autenticació

Com ja s'ha comentat aquest procés passarà per tres pantalles diferents, molt similars entre elles, però on s'espera una acció diferent per la seva part en cadascuna.

Pas 1

A l'iniciar l'aplicació, un cop ja definida la contrasenya, es trobarà amb una pantalla on apareixeran un conjunt d'imatges aleatòries de la seva galeria d'imatges. **La funció d'aquestes imatges sols és delimitar les diferents cel·les** per on vostè **haurà de dibuixar la seva traça**, definida prèviament com a contrasenya.

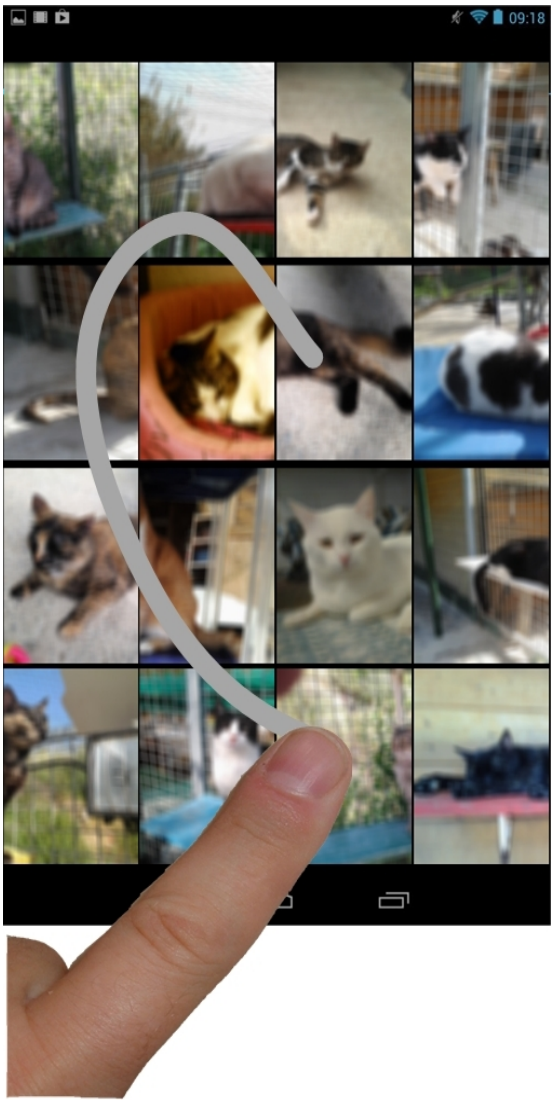


Fig. 6: Dibuix de la traça

Sols cal que segueixi les cel·les que formen la seva traça amb l'ordre correcte, mitjançant el dit. Pot veure més informació d'aquest procediment al *pas 2* del procediment de registre. Li recordem que la traça pot anar desapareixent a l'hora que dibuixa i que aquesta es considera acabada a l'aixecar el dit de la pantalla.

Si vostè no realitza bé la traça no rebrà cap mena d'indicació i passarà igualment a la següent pantalla.

Pas 2

A la següent pantalla vostè haurà d'**escollir la primera parella d'imatges que forma la seva contrasenya**, com s'ha explicat al *pas 4* del procediment de registre.

Fig. 7: Seleccio *multitouch* de les imatges

És molt important que vostè seleccioni les dues imatges alhora mitjançant dos dits, si no l'elecció no tindrà efecte (*multitouch*).

Tant si l'elecció és correcta com si no, vostè passarà a la següent pantalla. Si vostè no identifica les imatges que formen la seva contrasenya pot ser degut a un error al dibuixar la traça, en el pas anterior. Si aquest fos el cas, segueixi el procediment escollint dues imatges qualsevol.

Pas 3

Ara repetirem el procés del pas anterior (*pas 2*), però amb la parella d'imatges restants.

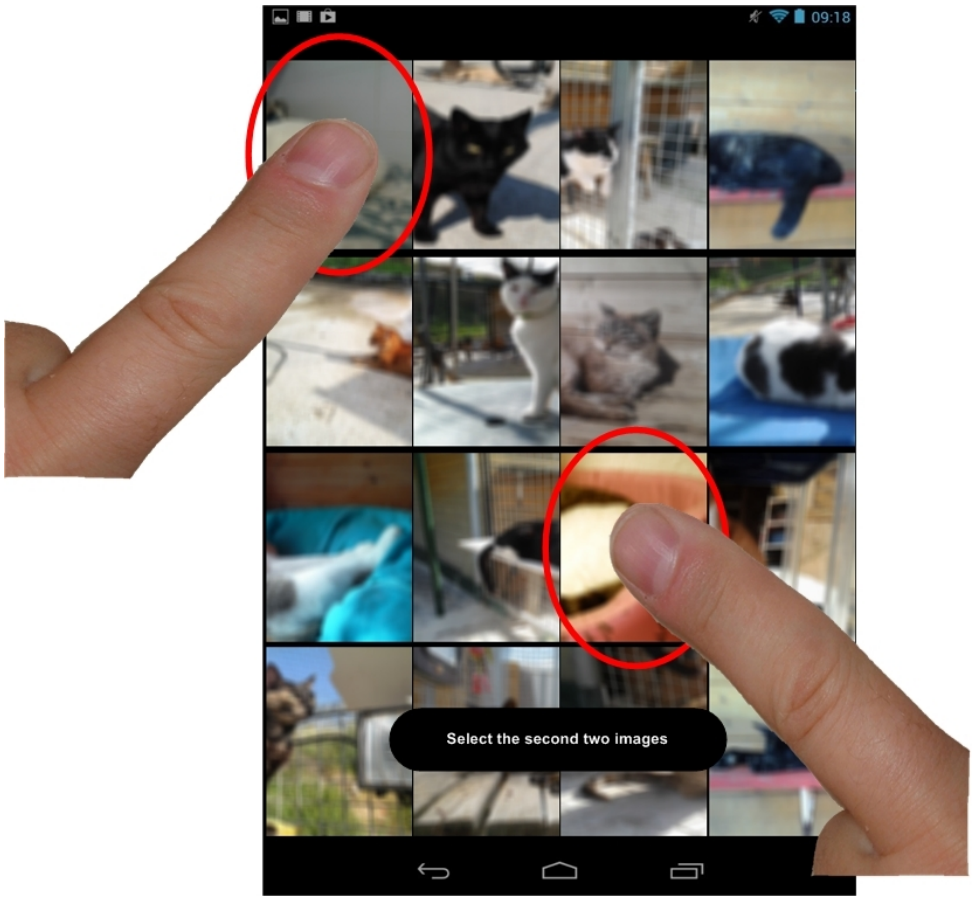


Fig. 8: Selecció de la segona parella d'imatges

És molt important que vostè seleccioni les dues imatges alhora mitjançant dos dits, si no l'elecció no tindrà efecte (multitouch).

Si l'elecció de totes les imatges és correcta, tornarà a la pantalla de gestió (fig. 1). Si no és correcta la contrasenya, en aquest punt rebrà un missatge de notificació i sortirà de l'aplicació.

Altres funcionalitats

Si ens fixem en la pantalla de gestió (fig. 1) podrem observar l'existència del botó **Add Photo To Gallery**. Amb aquesta opció podrem afegir a la galeria d'imatges de l'aplicació fotografies fetes amb la càmera del dispositiu, tal com s'explica al *pas 1* del procediment de registre.

Si aquest és el nostre objectiu, sols caldrà que realitzem les fotografies que ens interessin passar a la galeria.

Un cop prenem el botó **Add Photo To Gallery** podrem escollir una fotografia de la llista mostrada. Automàticament aquesta passarà a formar part de la nostra galeria d'imatges i s'esborrarà de la llista.

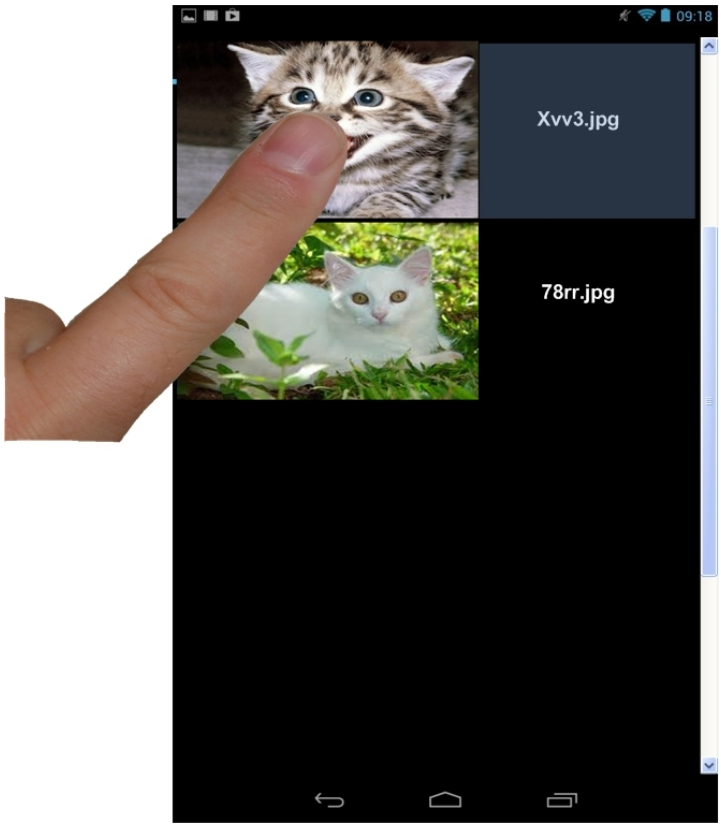


Fig. 9: Afegint una fotografia a la galeria d'imatges

A la pantalla de gestió (fig .1) també podrem veure el botó **Send Statistics** que simplement envia un mail al programador de l'aplicació amb dades estadístiques sobre el seu ús. Si prenem aquest botó es crearà un nou *e-mail* que haurem d'enviar amb el nostre client de correu per defecte.