CENTER FOR
MACHINE PERCEPTION

CZECH TECHNICAL
UNIVERSITY IN PRAGUE

BACHELOR THESIS

Gesture Recognition for Mobile Phone Unlocking

Tomáš Sixta

sixta.tomas@gmail.com

May 23, 2014

Thesis Advisor: Mgr. Jan Šochman, Ph.D.

Center for Machine Perception, Department of Cybernetics
Faculty of Electrical Engineering, Czech Technical University
Technická 2, 166 27 Prague 6, Czech Republic
fax +420 2 2435 7385, phone +420 2 2435 7637, www: http://cmp.felk.cvut.cz

Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Cybernetics

BACHELOR PROJECT ASSIGNMENT

Student:	Tomáš S i x t a
Study programme:	Open Informatics
Specialisation:	Computer and Information Science
Title of Bachelor Project:	Gesture Recognition for Mobile Phone Unlocking

Guidelines:

Propose and implement a gesture based screen locking algorithm for mobile phones which is both user friendly and safe from peek-over-the-shoulder problem and smudge attack. Similar systems were proposed for automatic signature verification on tablet computers. The thesis goal will be to adapt one of these methods for the specific scenario of mobile phone unlocking.

Instructions:

1. Study the state-of-the-art in the online signature verification,
2. Choose and adapt one of the methods for the gesture-based mobile phone unlocking,
3. Evaluate the method based on false positives, false negatives, but also based on its safety and user friendliness and easiness of training.

Bibliography/Sources:
[1] Zhaoxiang Zhang, Kaiyue Wang, Yunhong Wang: A Survey of On-line Signature Verification. Biometric Recognition Lecture Notes in Computer Science Volume 7098, 2011.
[2] Stan Z. Li: Markov Random Field Modeling in Image Analysis. 2009.

Bachelor Project Supervisor: Mgr. Jan Šochman, Ph.D.

Valid until: the end of the summer semester of academic year 2014/2015

L.S.

doc. Dr. Ing. Jan Kybic	prof. Ing. Pavel Ripka, CSc.
Head of Department	**Dean**

Prague, January 10, 2014

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra kybernetiky

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student:	Tomáš S i x t a
Studijní program:	Otevřená informatika (bakalářský)
Obor:	Informatika a počítačové vědy
Název tématu:	Rozpoznávání gest pro odemykání mobilního telefonu

Pokyny pro vypracování:

Navrhněte a implementujte metodu odemykání mobilního telefonu založenou na gestech, která bude uživatelsky jednoduchá a zároveň bezpečná, tj. uživatelské gesto nebude snadno odkoukatelné druhou osobou (ať už přímo přes rameno nebo ze stopy po tahu prstem). Podobné systémy fungují pro verifikaci podpisů na základě podpisových vzorů na tabletu. Cílem práce bude adaptovat některou z těchto metod s přihlédnutím ke specifikům aplikace odemykání mobilního telefonu.

Postup:

1. Seznamte se s literaturou zabývající se verifikací podpisů na tabletu.
2. Implementujte vhodnou metodu pro rozpoznávání gest na mobilním telefonu.
3. Vyhodnoťte úspěšnost metody (falešné poplachy, přehlédnutá nebezpečí), ale také bezpečnost (obtížnost „odkoukání“) a snadnost naučení a ovládání.

Seznam odborné literatury:

[1] Zhaoxiang Zhang, Kaiyue Wang, Yunhong Wang: A Survey of On-line Signature Verification. Biometric Recognition Lecture Notes in Computer Science Volume 7098, 2011.
[2] Stan Z. Li: Markov Random Field Modeling in Image Analysis. 2009.

Vedoucí bakalářské práce: Mgr. Jan Šochman, Ph.D.

Platnost zadání: do konce letního semestru 2014/2015

L.S.

doc. Dr. Ing. Jan Kybic	prof. Ing. Pavel Ripka, CSc.
vedoucí katedry	**děkan**

V Praze dne 10. 1. 2014

Author's declaration

I hereby declare that I have completed this thesis independently and that I have listed all used information sources in accordance with the Methodical guidelines on maintaining ethical principles during the preparation of university theses.

Prohlášení autora práce

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickými pokyny o dodržování etických zásad při přípravě vysokoškolských závěrečných prací.

V Praze dne ....

Podpis autora práce

Acknowledgements

I would like to express my thanks to Jan Šochman for guidance and mentoring, without which this thesis could not be completed.

I would also like to thank the members of CMP and avast! Android team for taking the time to take part in the experiments and for their critique and comments.

Abstract

People use lockscreens to secure their mobile devices. However, many users aren't satisfied with currently available lockscreens - they either provide too little security or require too much effort to be unlocked. In this thesis, we analyze available lockscreens and summarize their weaknesses. Based on that analysis, a list of requirements for a "perfect" lockscreen is presented.

Following the requirements, we design Tap lock - lockscreen that is unlocked by tapping a unique sequence, which is based on a melody in user's head. We describe a relation between Tap lock and other gesture recognition problems and how it can be resolved using hidden Markov models.

Extensive testing is then performed that shows that Tap lock can be used in everyday use if simple recommendations on how to choose a proper sequence are followed. Tap lock is easy to be unlocked for the user, but difficult to hack for a potential thief. It is also extremely resistant to brute-force attacks.

During the summer 2014, we plan to publish Tap lock on Google Play in cooperation with avast! antivirus company.

Abstrakt

Lidé zabezpečují svá mobilní zařízení používáním lockscreenů (zámků obrazovky). Nicméně mnoho uživatelů není spokojeno s lockscreeny, které jsou v současné době k dispozici - buďto nejsou dostatečně bezpečné nebo vyžadují příliš mnoho úsilí k odemknutí. V této práci zanalyzujeme dostupné lockscreeny a shrneme jejich slabé stránky. Na základě této analýzy poté představíme seznam požadavků na "ideální" lockscreen.

S ohledem na tyto požadavky poté navrhneme Tap lock - zámek, který se odemyká vyťukáním sekvence na základě melodie, kterou má uživatel v hlavě. Popíšeme vztah mezi Tap lockem a rozpoznáváním gest a jak lze problém řešit pomocí skrytých markovských modelů.

Poté provedeme rozsáhlé testování které ukazuje, že Tap lock lze běžně používat pokud se sekvence vybírají s ohledem na jednoduchá doporučení. Uživatel zvládne Tap lock odemknout snadno, zatímco pro hackera je to složité. Tap lock je také velmi odolný proti útokům hrubou silou.

Během léta 2014 plánujeme zveřejnit Tap lock na Google Play ve spolupráci s antivirovou společností avast!.

Contents

1 Motivation

With the increasing use of mobile devices, smartphones became easy targets for theives. According to [10], there are about 10 000 reported thefts in the Czech Republic every month (with many more not reported). Because of the spread of the Internet to mobile devices, losing one's smartphone often gives the thief unlimited control over the victim's accounts. Exposing user's private Dropbox files or Gmail conversations can have serious consequences. Mobile banking apps or apps like PayPal or Google Wallet contribute to the possible threat even more.

According to androidcentral.com poll from 2012 [6], over 55 % people do not use any kind of secure lockscreen, 25 % use Pattern lock and 10 % use PIN lock. But according to the associated comments, neither of these solutions is satisfactory. People mostly complain about Pattern lock providing little protection and PIN Lock being too annoying to unlock regularly.

Therefore we decided to come up with a completely new approach that provides both high security and easy unlocking. This approach will be based on gesture recognition. Rather than forcing users to remember another unlock code, we will analyze their natural finger movements and use them for authorization.

2 Contributions

This thesis contributes to the mobile security research area by introducing a new way of device locking. All secure lockscreens based on the gesture recognition share the same weakness - if the data is collected from a wet surface or greasy fingers, the lockscreen becomes unusable. Our approach avoids this problem entirely while being more comfortable for the user at the same time.

The main contributions of this thesis can be summarized as following:

- Current lockscreen approaches and their weaknesses were analyzed. We found out that there is a demand for a new solution that is both safe and user-friendly.
- We proposed a new lockscreen, Tap lock. Tap lock relies on a unique concept of unlocking the device by tapping a custom sequence that is based on a melody in user's head.
- Tap lock was implemented on Android operating system.
- Experiments were conducted in order to measure the efficiency of the proposed solution. We found that Tap lock can be used in real life if simple recommendations are followed.

During the summer 2014, Tap lock will be finalized and with cooperation of avast! antivirus company, it will be then published on Google Play.

3 Lockscreen Solutions

Today, number of different operating systems is used on mobile devices. However, only two operating systems have remarkable market share - Android and iOS [23]. Because Android is more popular than iOS, we focused only on Android lockscreens in this chapter. Nevertheless, touchscreen device unlocking works on the same principles regardles of the operating systems, so very similar lockscreens are used on iOS.

Android lockscreens can be divided into two main groups. People either use one of the Android's default lockscreens, or they download a 3rd party lockscreen app or widget that is usually highly customizeable.

Numbers and conclusions presented in the following subsection are based on internet poll [6] and the associated user comments.

3.1 Default Android Lockscreens

3.1.1 None

About 25 % people do not use any kind of lockscreen. After waking up the phone, the Home screen is shown instantly.

3.1.2 Slide

Slide is the most common lockscreen. The phone is unlocked by simply dragging the finger across the screen. Even though it provides no security, about 30 % people use it.

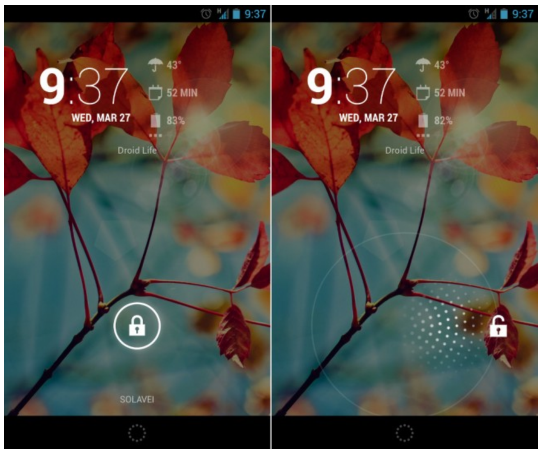

Figure 3.1 Slide lockscreen. Courtesy of droid-life.com [4]

3.1.3 PIN

User has to input a 4-digit PIN code in order to unlock his phone. While being quite secure, for users it is often annoying to input PIN code every time they want to check their phone. This solution is used by approximately 10 % of users.

3.1.4 Password

Password lock provides much better security compared to PIN lock, but is also extremely inconvinient, therefore remains virtually unused.

3.1.5 Pattern

This is the second most popular lockscreen. However, it is quite controversial by providing false sense of security. There are many possible combinations, but most people use very predictable patterns (without even realizing how predictable their pattern is). The issue of Patter lock security is very well analyzed in [26]. For example, according to that research, over 50 % people start their sequence in the top left corner, which greatly reduces the number of possible combinations. The unlock pattern is also easy to be spotted or discovered by focusing on smudge trails on the screen.

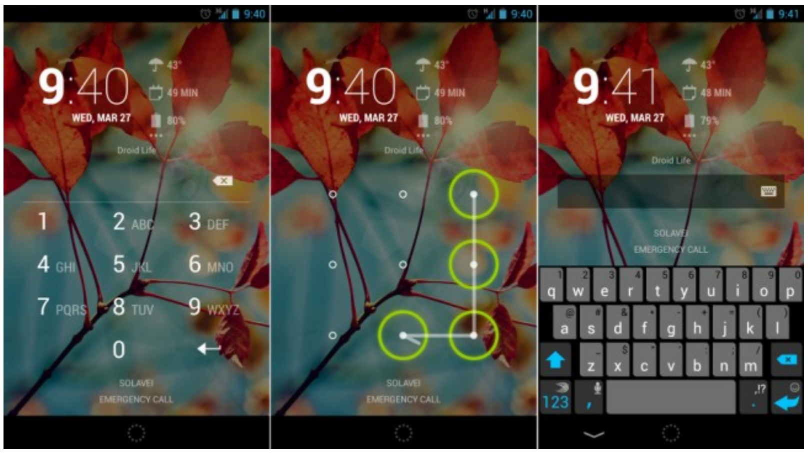

Figure 3.2 PIN, Pattern and Password lockscreens. Courtesy of droid-life.com [4]

3.1.6 Face Unlock

This innovative feature is available only on those devices that posses front camera. That itself limits the possible use to only a fraction of devices, but also the light conditions must be good enough for the lockscreen to work. This might be the problem in the night, in pubs, clubs etc. Therefore this solution is only used by about 3 % of users.

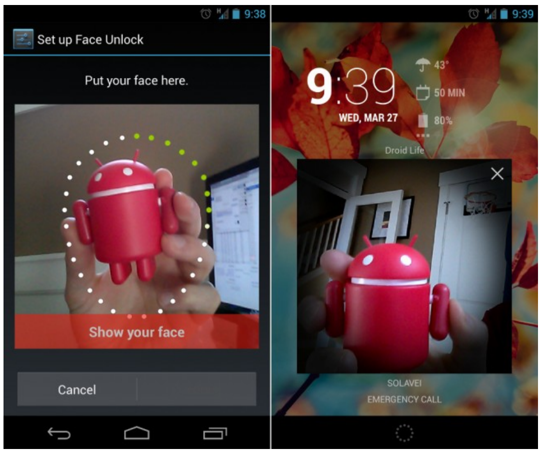

Figure 3.3 Face Unlock. Courtesy of droid-life.com [4]

3.2 3rd Party Apps / Widgets

3.2.1 Unsecure Locksreen Apps

Unsecure lockscreens are mostly variations of the Slide lock, however they provide high customizeability. People can choose their own shortcuts, background images, positioning of the elements on the screen and so on. Even though in Android 4.3 Lockscreen Widgets have been introduced, these apps still remain a popular choice with tens of millions of downloads on Google Play [11].

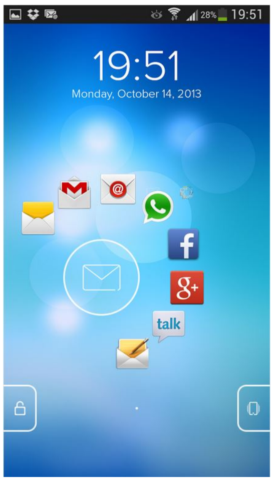

Figure 3.4 Unsecure 3rd party lockscreen - Start by Celltick. Courtesy of Google Play

3.2.2 Secure Lockscreen Apps

Most of the 3rd party secure lockscreens do not introduce any innovative way of unlocking one's device. They rather focus on providing the high customizeability or try to deal with the after-theft scenarios (controlling phone over SMS or internet, deleting any personal data, device tracking) - e.g. [9]. However there are a few applications that present something new, although it could often be considered rather "experimental" than practical for everyday use.

Smart App Lock [19]

This is a very popular lockscreen on Google Play with more than 10 millions downloads and high rating. Apart from providing custom variations of default Android lockscreens, it also introduces Gesture lock (see Figure 3.6). Their Gesture lock however only uses Android GestureLibrary, which is not precise enough.

Android GestureLibrary is designed to simplify gesture recognition for developers and its intended use is phone control (e.g. draw circle to open menu, swipe left to go back etc.). Therefore when recognizing a gesture, it has high tolerance and low refusal rate. Moreover, it only uses touchscreen coordinates for gesture evaluation and disregards time or pressure information. That could mean discarding important features of the gesture.

Nevertheless, the introduced concept of Gesture lock is not very common approach and might be worth further research.

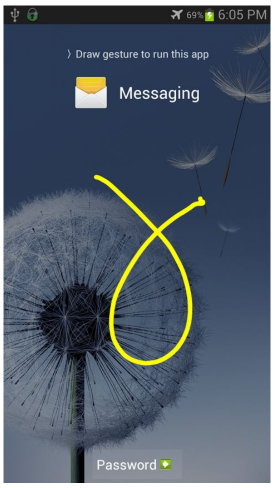

Figure 3.5 Gesture Lock as implemented by Smart App Lock. Courtesy of Google Play

Picture Password Lockscreen [21]

User chooses his own image, then he draws a gesture connecting several places of interest. For the user, this is extremely easy to accomplish, but the potential thief has no idea what the points of interest might be. But, as mentioned before, once the thief spots the user drawing the gesture, it is easy for him to repeat it.

For example, the user could choose city panorama as a background image. Then the user would choose some buildings as his points of interest and draw a gesture connecting them. User can easily remember those buildings and can repeat the gesture, but it would be difficult for thief to guess the gesture. But if the thief saw the user draw it, he would be probably able to repeat it easily.

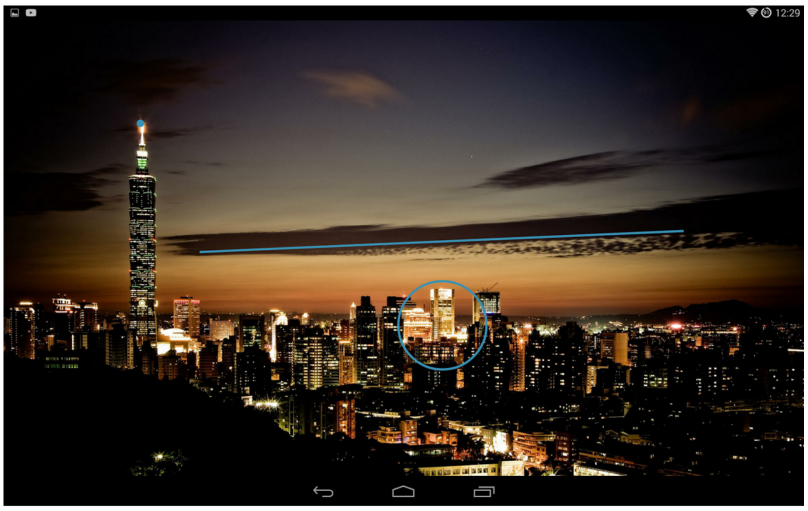

Figure 3.6 Picture lock. The unlock gesture consists of moving the finger across the distinctive skyline and touching the building in the middle, as marked by the blue lines. Courtesy of Google Play

Sensor Lock [8]

Most devices contain built-in gyroscope and accelerometer. This lock relies on a characteristic orientation and acceleration of the device for unlocking. Preliminary testing showed, however, that these sensors are not precise enough to be used for lockscreen purposes, which makes this lock rather "experimental". Moreover, there would be problems when trying to unlock this lockscreen e.g. in public transport.

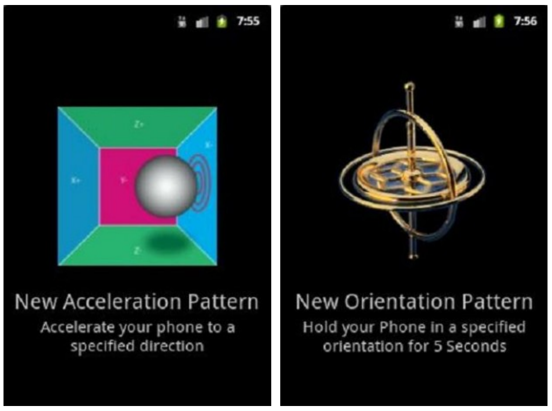
7:55
New Acceleration Pattern
Accelerate your phone to a specified direction
7:56
New Orientation Pattern
Hold your Phone in a specified orientation for 5 Seconds

Figure 3.7 Sensor Lock - acceleration and orientation patterns. Courtetsy of appszoom.com [8]

3.3 Drawbacks of Current Solutions

When evaluating lockscreens, it is neccessary to focus on the ratio between security they provide and the effort they require for unlocking. While the Slide lock is extremly easy to unlock, it still leaves the devices completely vulnerable. On the other hand, the effort to type a password to unlock the phone is too high for users to actually use Password lock, even though it is very secure.

The most popular secure lockscreen, the Pattern lock, has two major weaknesses apart from tempting users to unknowingly use very predictable patterns [26].

Peek-over-the-shoulder issue - Let us consider a user traveling in a bus. He checks his phone and decides to read a new message, thus unlocking the phone (drawing the pattern). The thief is standing behind the user and easily spots the pattern. He then grabs the phone, gets off the bus and runs away. Because he now knows the pattern, his access to the phone and related accounts is unlimited.

Smudge attack - Even if the thief doesn't see the sequence or gets the phone another way (finds it, steals it from handbag), it is often enough to position the screen against the source of light. Then there are usually visible smudges on the display, which can be used to guess the pattern or at least reduce the number of possible combinations.

Gesture lock is a better solution compared to Pattern lock, but it still shares the same weaknesses - at least in its current implementation. This implementation uses Android GestureLibrary, which isn't suitable for security purposes, as explained in section 3.2.2.

4 Related Work and Proposed Solution

4.1 Related Work

Suprisingly, the lockscreen research is widely avoided. There are only a two papers focusing on this issue [22] [7]. Both these papers share the same motivation and are quite similar to each other, because rather than designing a new lockscreen, the authors try to add a security layer to the Pattern lock. The main idea is that even if the thief can see the user drawing the pattern (thus knows the exact unlock sequence), he cannot repeat it because he cannot repeat certain characteristic features that cannot be easily observed - acceleration, pressure, curvature of the gesture etc.

In [22], the authors use the Random Forest technique (multi-class classification method based on combining results of random decision trees). They achieve error rate[1] 10.39 %, however the user has to provide 50 samples of their chosen pattern to set up the lock.

The lock proposed in [7] uses dynamic time warping (algorithm for measuring similarity between two temporal sequences). It requires even more sequences for setup - user has to repeat his pattern 80 times. Even still, the error rate of this lock is apx. 23 %.

Authors of these papers state that it is possible to create a unique pattern if enough training samples are provided. However, the amount of sample patterns required from the user in their presented solutions is too high to be used in real life.

4.1.1 Online Signature Verification

A domain very much related to the lockscreen problem is the online signature verification [12] [17] [18] .

The main objective of online signature verification is to authorize the user by analyzing the gesture he draws with special pen (his signature) on a tablet (online signature verifier - see Figure 4.1). The difference between signature verifier and a mobile device is, that while on mobile device we obtain features like x and y coordinates and pressure, the verifier provides additional features (inclination, azimuth) thanks to the presence of the pen. It also requires fewer samples for model training. Despite the lack of some features, the algorithms used in online signature verification could be used for touchscreen gesture recognition.

Independently of the used algorithm, all approaches share the first few steps. The acquired data is preprocessed. This usually means normalization of size, position and orientation, some smoothing or resampling, so different sequences can actually be compared with each other.

Then additional features that are not obtained directly from the device are computed. These typically include speed, acceleration or curvature and their derivatives [5].

Some authors use all available data points. However, in order to reduce this rather large amount of points (most of the online signature verifiers create new datapoint every 10 miliseconds), some authors only extract points of interest that are later compared.

[1] Error rate is the total ratio of wrong results, both false positive and false negative.

Figure 4.1 Online Signature Verifier. Courtesy of xyzmo.com

For example in [5], the points are clustered into several clusters and points of interest are centres of these clusters. In [3], points of interest are such points, where the gesture sharply changes direction (in terms of x and y coordinates).

After these initial steps, algorithm either uses provided sequences of points to create a new model or evaluates the sequence of measured points and decides, whether the signature is accurate enough. According to [1], there is a number of possible approaches, but there is no common agreement in the research community as to which of these approaches is the best. [12] gives the overview of the current state of the art and the most popular algorithms. Hidden Markov models (HMM) and dynamic time warping (DTW) seem to cover most of the research work and to be equally popular.

Also for a long time, there has been no estabilished benchmark that would independently measure efficiency of different algorithms [1]. But in 2004, the first international online signature verification competition took place [16]. We took a look at the methods that ranked on the highest positions - the first place was won by a team that used DTW [25]. On the other hand, team that placed second used HMM [5]. The team that placed second later claimed that when given more training sequences, their method (HMM) outperforms the first team's DTW approach.

Motivated by the success of HMMs in online signature verification, we decided to use hidden Markov models rather than DTW. Moreover, authors of [22] state that DTW didn't yield expected results and suggest that another method should be tried for touchscreen pattern recognition.

4.2 Requirements

When designing a new lockscreen, there are some basic principles that need to be followed. It is not possible to fullfill all these flawlessly and compromises have to be made (speed vs accuracy, false positive vs false negative ratio etc.). These compromises and particular settings will later be subject to experiments to maximize user experience and lockscreen usability.

- Low false negative rate (user is almost never denied access to his own device)
- Low false positive rate (stranger can almost never gain access to the device)
- Easy to unlock - preferrably using only one hand, one finger
- Fast to unlock - dynamic, with minimal evaluation time, max. couple of seconds
- Small training set - during the training phase, only 5-10 samples will be available for learning

Apart from these, the problems described in section 3.3 need to be dealt with, i.e. peek-over-the-shoulder issue and smudge trail issue.

4.3 Proposed Solution

After studying related work, it became appearent that designing a secure lockscreen based on gesture recognition is quite challenging with uncertain results.

The implemented solutions [22] [7] do not meet the above-mentioned requirements (specifically "small training set"). The authors have shown that the secure Pattern lock/Gesture lock can work, but requires huge training set and complex patterns. In real life environment, people tend to use simple shapes and do not want to draw it more than 5-10 times. But after considering various aspects of the touchscreen gesture recognition, we noticed that the biggest problem is the data collection itself. Even seemingly small problems, such as wet or shaking hands, would cause user to draw the pattern in a quite different way.

It became clear that in order to keep up with the presented requirements, the idea of Gesture lock had to be reformulated. Gestures provided by the users would probably be too simple and not characteristic enough, which would make the lockscreen prone to be hacked easily.

There are four basic features provided by the touchscreen - x and y coordinates, pressure and timestamp. All these features are used in online signature verification. But there are some differences which prevent them from being used for touchscreen gesture recognition for security purposes. x & y coordinates cannot be used, because they are easily influenced by common situations like greasy or shaking hands. The pressure info provided by the touchscreen is generally very "basic" and cannot be used for security purposes.

However, we noticed that users can repeat the gesture - in terms of time and speed - very well (even though the coordinates of the touch events differ a lot). Therefore, only the time information itself could be used for the device unlocking. While it may seem too simple at first look, the preliminary experiments have proved that users are very consistent in repeating the gesture in terms of time feature - that means keeping the rhytm. It is a simple and elegant solution, eliminating the mentioned problems (smudge attack, wet hands etc.) while following the requirements. Therefore we propose Tap lock.

4.3.1 Tap Lock

User has his favourite song or melody. It is very easy for the user to tap to the rhythm of that melody - people often tap to the rhytm when they hear the song or when they "sing it in their head". So instead of tapping on the table or parts of their body, people could actually tap on the touchscreen to unlock the device. As mentoned before, users are very consistent in keeping the rhythm - they can repeat the tap sequence easily (see Figure 4.2). On the other hand, we noticed that the strangers have difficulties repeating the sequence - they have no idea what song is the user thinking about. Even if the stranger could spot the sequence, which might be quite difficult, he would not be able to repeat it with the same precision as the user and would be denied access.

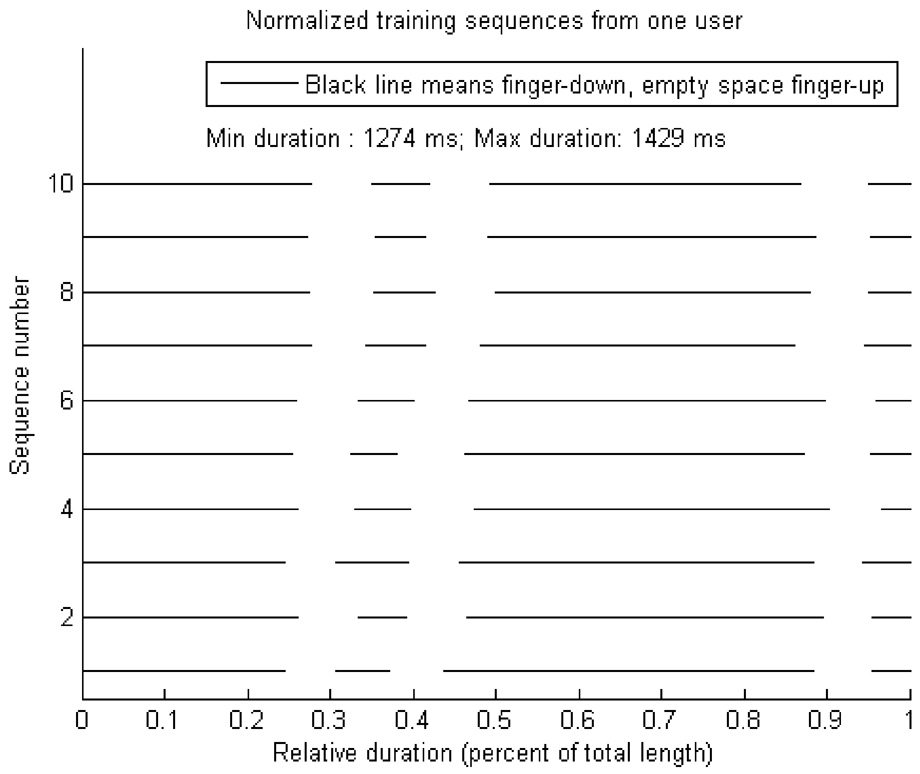

Figure 4.2 Rhythm consistency - Tap lock training sequences

The algorithms used in online signature verification can still be applied to this problem. Even though the feature set is reduced, the tapped sequence can still be considered a gesture.

5 Theoretical Background

Training and evaluation of Tap lock relies on hidden Markov models (HMMs), as reasoned in chapter 4.1.1. In the following sections, we give a short intoduction into basic concepts of HMMs and algorithms used for HMM learning and sequence evaluation.

5.1 Hidden Markov Models

Hidden Markov model (HMM) is a statistical model used for modelling Markov processes with hidden states. Compared to Markov model, where we know in which state we are at given time, in HMM we can only observe measurements that have certain relation to the real states (that are hidden from us).

To fully describe HMM, we use the notation of [15]:

- $S = \{s_1, ..., s_N\}$ - set of N hidden states
- $V = \{v_1, ..., v_M\}$ - set of M possible observations
- $Q = q_1 q_2 ... q_T$ - considered state sequence
- $O = o_1 o_2 ... o_T$ - observed sequence
- $t = 1, 2, ..., T$ - time indexes associated with state and observation sequences
- $a_{ij} \in A^{NxN}$ - $p(q_t = s_j | q_{t-1} = s_i)$, transition probability from state s_i to s_j
- $b_i(v_j) \in B^{NxM}$ - $p(o_t = v_j | q_t = s_i)$, probability of observing symbol v_j, given we are in state s_i
- π_i - probability of being in state s_i in time $t = 1$

To show how these parts interact with each other, consider the following Urn problem [15]. Observer knows that there are N urns, $s_1, ..., s_N$, and each of these urns contains M balls $v_1, ..., v_M$. Observer cannot see the urns, but there is a genie in the room. According to some random process, a genie chooses the initial urn (probability of starting with urn s_i is π_i). Then he randomly draws a ball from it (being at urn s_i, the probability of choosing ball v_k is $b_i(v_k)$) . He shows the ball to the observer, who records it as an observation o_1, then the ball is placed back into the urn it was drawn from.

The genie then proceeds and based on some random process associated with the current urn, he chooses another urn (probability that the genie is at urn s_i and chooses urn s_j as the next urn is a_{ij}). From the newly chosen urn, he randomly draws a ball. Again, the ball is shown to the observer, who records it as observation o_2, then the ball is placed back. Genie keeps choosing new urns, drawing balls from them and showing them to the observer until T observations are recorded.

In the end, the observer records a sequence of T observations $o_1 ... o_T$. He doesn't know from which urns they were drawn from, but there are ways how the most likely sequence of urns or the most probable urn at time t can be computed.

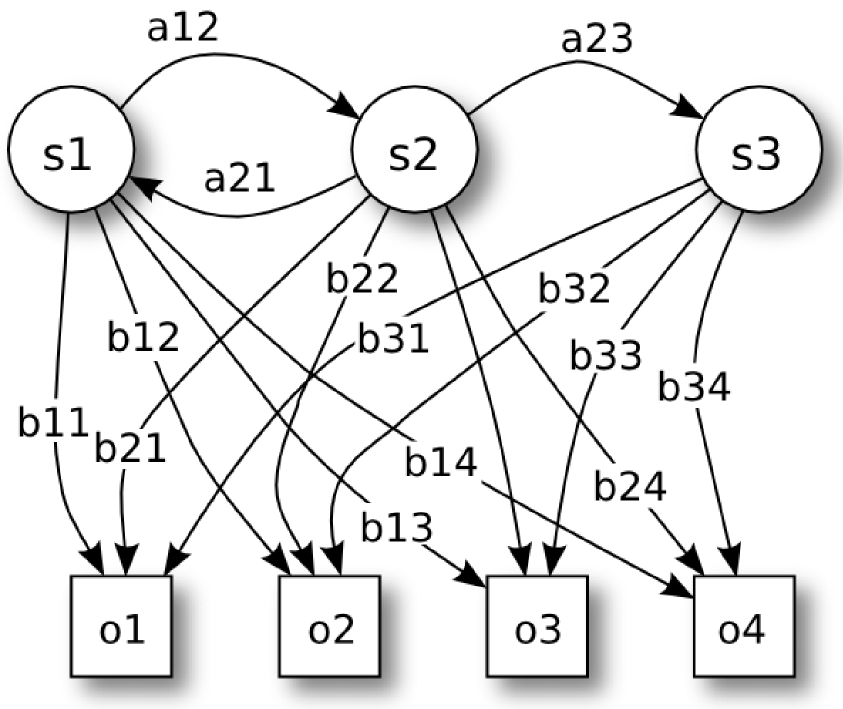

Figure 5.1 HMM schema. Courtesy of Wikipedia.org

In case of Tap lock, a hidden state / observation is a duration between two consecutive touch events relative to the total length of the sequence (details in chapter 6.1.2). Basically, melody in the user's head can be considered a sequence of hidden states and nobody except the user knows it. Touchscreen taps based on that melody can be considered observations

There are three main problems that need to be solved when using hidden Markov models:

1. Determine appropriate state representation and the number of states
2. Given observed sequence $O = o_1 o_2 ... o_T$ and a model defined by parameters $\lambda = (A, B, \pi)$, determine the probability $p(O|\lambda)$ of that sequence
3. Learn sub-optimal set of of parameters $\lambda = (A, B, \pi)$

While the state-related settings are problem-specific and usually subject to experiments, the other two problems - λ optimization and computing $p(O|\lambda)$ - can be solved using Expectationg-Maximization and Forward-Backward algorithms.

5.2 Forward-Backward Algorithm

Consider we observe a sequence $O = o_1 o_2 ... o_T$ of length T and, given $\lambda = (A, B, \pi)$, we want to find out $p(O|\lambda)$, probability of observing that sequence. The most simple, brute-force solution would be to compute it for all possible underlying state sequences Q (of length T) and then sum the results.

Consider a fixed sequence of states $Q = q_1 q_2 ... q_T$ of length T. Then the probability of observing O given Q and λ is

$$p(O|Q, \lambda) = b_{q_1}(o_1) \cdot b_{q_2}(o_2) ... b_{q_T}(o_T) \tag{5.1}$$

Also, we need to compute probability of state sequence Q actually happening:

$$p(Q|\lambda) = \pi_{q_1} \cdot a_{q_1 q_2} \cdot a_{q_2 q_3} ... a_{q_{T-1} q_T} \tag{5.2}$$

The joint probability of O and Q - probability that Q occurs and we observe O - is simply the product of the equations (5.1) and (5.2). To compute the probability $p(O|\lambda)$, we need to sum over all possible sequences Q, yielding equation:

$$p(O|\lambda) = \sum_{Q} p(O|Q, \lambda) \cdot p(Q|\lambda) \tag{5.3}$$

It is obvious that this solution with time complexity $2T \cdot N^T$ is unfeasible and we need to use more sophisticated method.

5.2.1 Forward process

Let us define a variable

$$\alpha_t(i) = p(o_1 o_2 ... o_t, q_t = s_i | \lambda) \tag{5.4}$$

It represents the probability of observing subsequence $o_1 o_2 ... o_t$ and being in the state s_i at time t. It can be computed recursively, as illustrated by Figure 5.2. First we initialize

$$\alpha_1(i) = \pi_i \cdot b_i(o_1) \qquad 1 \leq i \leq N \tag{5.5}$$

Then we repeat the following recursion until we compute α for all times and states:

$$\alpha_{t+1}(j) = [\sum_{i=1}^{N} \alpha_t(i) \cdot a_{ij}] \cdot b_j(o_{t+1}) \qquad 1 \leq t \leq T-1, 1 \leq j \leq N \tag{5.6}$$

We can get to state s_j at time $t+1$ through N possible states $s_i, 1 \leq i \leq N$. Since $\alpha_t(i)$ is the probability observing subsequence $o_1 o_2 ... o_t$ and being in the state s_i at time t, then product $\alpha_t(i) a_{ij}$ is the probability of observing subsequence $o_1 o_2 ... o_t$ and being in state s_j at time $t+1$, passing through state s_i at time t. When we sum this product over all possible states $s_i, 1 \leq i \leq N$, we get the probability of being in the state s_j at time $t+1$ considering observations up till time t. Then, by simply multiplying it by observation probability $b_j(o_{t+1})$, we get the $\alpha_{t+1}(j)$.

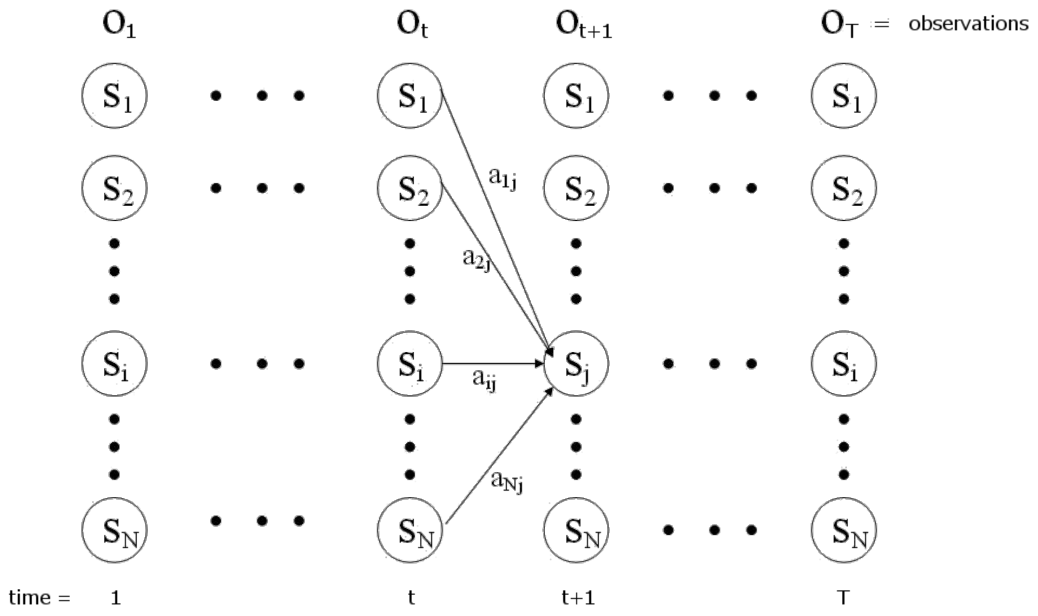

Figure 5.2 Forward algorithm. Courtesy of bioinfopakistan.ucoz.com

5.2.2 Backward process

The backward process is similar. Let us define

$$\beta_t(i) = p(o_{t+1}o_{t+2}...o_T|q_t = s_i, \lambda) \tag{5.7}$$

This represents the probability of observing sequence $o_{t+1}o_{t+2}...o_T$ when we know that we are in the state s_i at time t.

It naturally comes from the definition that we need to set the following values before starting the recursion (probability of observing empty sequence is always 1):

$$\beta_T(i) = 1 \qquad 1 \le i \le N \tag{5.8}$$

Then we recursively compute the values for β for all times and states:

$$\beta_t(i) = \sum_{j=1}^{N}(a_{ij} \cdot b_j(o_{t+1}) \cdot \beta_{t+1}(j)) \qquad 1 \le t \le T-1, 1 \le i \le N \tag{5.9}$$

5.2.3 Getting the Observed Sequence Probability

Now that we have computed α and β, we can use them to compute the probability $p(O|\lambda)$. From the definitions it is obvious that

$$p(O|q_t = s_i, \lambda) = \alpha_t(i) \cdot \beta_t(i) \tag{5.10}$$

meaning that the probability of seeing sequence $o_1o_2...o_T$ given we are at state s_i at time t is simply product of $\alpha_t(i)$ and $\beta_t(i)$. Getting the overall sequence probability is as simple as summing that equation over all possible states $s_i, 1 \le i \le N$ for arbitrary time t.

$$p(O|\lambda) = \sum_{i=1}^{N} \alpha_t(i) \cdot \beta_t(i) \tag{5.11}$$

Since we can choose the time t, we can utilize this to further reduce time complexity by half. Insted of running the whole Forward-Backward algorithm, we can make use of the fact that $\beta_T(i) = 1$ for every i. Therefore, if we choose $t = T$, we can completely disregard the backward process and simply compute the sequence probability by running only the forward part of the algorithm:

$$p(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i) \tag{5.12}$$

The time complexity of Forward-Backward algorithm is TN^2, which is far better than the brute-force solution. Moreover, we acquired the variables α and β, which will be usefull in the Expectation-Maximization algorithm.

5.3 Expectation-Maximization

The EM algorithm (known as Baum-Welch algorithm when referring to HMMs) is used for $\lambda = (A, B, \pi)$ optimization. We are given training sequence $O = o_1 o_2 ... o_T$, from which we want to infer the best possible set of parameters λ, maximizing $p(O|\lambda)$.

The algorithm has two stages - E-stage (expectation) and M-stage (maximization). In the E-stage, auxiliary variables are computed using the results of Forward-Backward algorithm. These auxiliary variables are then used in the M-stage for updating the set of parameters λ. This whole process is repeated until the local maxima is reached or some other stopping criterion fulfilled.

EM algorithm only reaches local maxima. One way to deal with this issue is to perform k-fold crossvalidation with random initial values. But estimating the inintial values based on performed experiments or some reasoning tends to achieve much better results (as shown in chapter 6.1.3).

5.3.1 Expectation Stage

Here we introduce two auxiliary variables that will be later used for updating λ.

State variable $\gamma_t(i) = p(q_t = s_i|O, \lambda)$ is the probability of being in state s_i at time t given observed sequence O. This can be computed easily using the results from Forward-Backward algorithm.

$$\gamma_t(i) = \frac{\alpha_t(i) \cdot \beta_t(i)}{p(O|\lambda)} = \frac{\alpha_t(i) \cdot \beta_t(i)}{\sum_{i=1}^{N} \alpha_T(i)} \qquad 1 \leq t \leq T, 1 \leq i \leq N \tag{5.13}$$

The other variable we introduce is joint variable $\xi_t(i, j)$. This denotes the probability of being in state s_i at time t and in state s_j at time $t + 1$.

$$\xi_t(i, j) = p(q_t = s_i, q_{t+1} = s_j|O, \lambda) \tag{5.14}$$

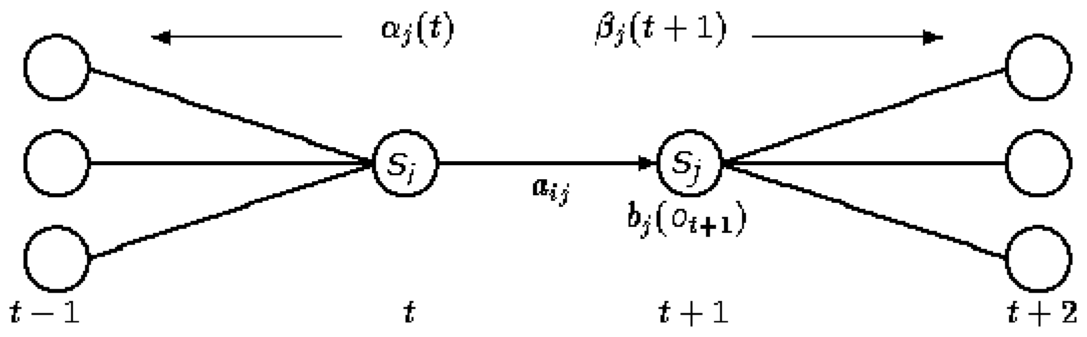

Figure 5.3 Joint variable$\xi_t(i,j)$. Courtesy of winnie.kuis.kyoto-u.ac.jp

This can be actually computed quite easily, again using α and β from Forward-Backward algorithm. Figure 5.3 can be usefull when trying to understand this variable.

Probability of being in state s_i at time t, given observations $o_1 o_2 ... o_t$ is stored in $\alpha_t(i)$. Probability of observing $o_{t+1} o_{t+1} ... o_T$ given we are are state s_j at time $t+1$ is stored in $\beta_{t+1}(j)$. So we only need to add the transition probability from state s_i to s_j, a_{ij}, and the observation probability of symbol o_{t+1} given state s_j, $b_j(o_{t+1})$.

$$\xi_t(i,j) = \frac{\alpha_t(i) \cdot a_{ij} \cdot b_j(o_{t+1}) \cdot \beta_{t+1}(j)}{p(O|\lambda)} \qquad 1 \le t \le T-1, 1 \le i \le N, 1 \le j \le N \tag{5.15}$$

5.3.2 Maximization Stage

Now that we computed the auxiliary variables, we can use them to compute a new $\bar{\lambda}$ from λ. It is guaranteed that the new sequence likelihood will be at least as high as the current sequence likelihood [15]:

$$p(O|\bar{\lambda}) \ge p(O|\lambda) \tag{5.16}$$

The update equations can be derived by solving optimization problem of maximizing $p(O|\lambda)$ given following constraints:

$$\sum_{j=1}^{N} a_{ij} = 1 \qquad 1 \le i \le N \tag{5.17}$$

$$\sum_{k=1}^{M} b_j(k) = 1 \qquad 1 \le j \le N \tag{5.18}$$

$$\sum_{i=1}^{N} \pi_i = 1 \tag{5.19}$$

That is, rows of A and B must sum up to one and initial probabilities π must sum up to one, so they actually represent valid probability distributions.

The technique that is used for this optimization problem is know as Lagrange multipliers. Details of derivation can be found at [20]. In the end, we end up with following update formulas:

$$\bar{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)} \tag{5.20}$$

$$\bar{b}_j(v_k) = \frac{\sum_{t=1}^{T} \gamma_t(j) \cdot d(o_t, v_k)}{\sum_{t=1}^{T} \gamma_t(j)} \tag{5.21}$$

$$\bar{\pi}_i = \gamma_1(i) \tag{5.22}$$

where d is so called *indicator function* defined as $d(o_t, v_k) = 1$ iff $o_t = v_k$ and $d(o_t, v_k) = 0$ iff $o_t \neq v_k$

To easily understand why these formulas work, we can also derive them by counting event occurences:

$$\begin{aligned} \bar{a}_{ij} &= \frac{\text{expected number of transitions from state } s_i \text{ to state } s_j}{\text{expected number of transitions from state } s_i} \\ &= \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)} \end{aligned} \tag{5.23}$$

$$\begin{aligned} \bar{b}_j(v_k) &= \frac{\text{expected number of times in state } s_j \text{ and observing sysmbok } v_k}{\text{expected number of times in state } s_j} \\ &= \frac{\sum_{t=1}^{T} \gamma_t(j) \cdot d(o_t, v_k)}{\sum_{t=1}^{T} \gamma_t(j)} \end{aligned} \tag{5.24}$$

$$\bar{\pi}_i = \text{expected number of times in state } s_i \text{ at time } t = 1 = \gamma_1(i) \tag{5.25}$$

Generally the algorithm will converge at increasingly slower rate as it approaches the local maxima (as described in chapter 6.5). Therefore, some stopping criterion should be used to break the iterations when we are satisfied with the result. Simple example of such stopping criterion might be a condition that if two consecutive likelihoods $p(O|\lambda_{iter})$, $p(O|\lambda_{iter+1})$ differ by less than some chosen tolerance ϵ, the algorithm is terminated.

5.4 Additional Modifications

The algorithms introduced so far have two major disadvantages.

First, the symbols we can observe are discrete values. However, in reality we observe continuous measurements (e.g. tap does not last 1 or 2 seconds, but rather 1.452s). The most simple solution would be to round the measurements and possibly increase the state count N. But we are using EM algorithm, we could also model $b_j(v_k)$ as a Gaussian distribution rather than a NxM matrix and then optimize its parameters with the EM algorithm.

Second, the number of training sequences is limited to only one. That is certainly undesirable, because it causes overfitting. We need to use more training sequence for λ learning.

5.4.1 Continuous Observation Probabilities

Up to this moment, we have considered $b_j(v_k) = p(o_t = v_k|q_t = s_j)$. Instead, imagine that v_k is a continuous measurement rather than a discrete value.

So let us reconsider the observation probability. We can redefine it as following:

$$b_j(v_k) = \mathcal{N}(v_k, \mu_j, \sigma_j^2) \tag{5.26}$$

where $\mathcal{N}()$ is the probability density function of the normal distribution and μ_j and σ_j^2 are mean and variance related to state s_j.

If we kept using discrete observations, the measurements would have to be rounded, which would impair the results.

The problem that arises with this change of b is how to learn it properly. We obviously cannot use the formula (5.24). Fortunatelly the learning process is just as easy as in the discrete version. The variable μ is given by:

$$\mu_j = \frac{\sum_{t=1}^{T} \gamma_t(j) \cdot o_t}{\sum_{t=1}^{T} \gamma_t(j)} \qquad 1 \leq j \leq N \tag{5.27}$$

The mean μ_j for state s_j is thus just a mean of observations weighted by thier probabilities of being in state s_j at time t.

The formula for updating σ^2 is given by:

$$\sigma_j^2 = \frac{\sum_{t=1}^{T} \gamma_t(j) \cdot (o_t - \mu_j) \cdot (o_t - \mu_j)'}{\sum_{t=1}^{T} \gamma_t(j)} \qquad 1 \leq j \leq N \tag{5.28}$$

This is very similar to stadard formula for variance computation, however here the additions have different weights, again based on the probability $\gamma_t(j)$.

5.4.2 Multiple Training Sequences

So far we have been working with only one training sequence. While that may be fine in some cases, to resolve our problem we need to make use of all the provided training sequences to prevent overfitting. Actually this is very easy and probably best explained by equations (5.23 - 5.25).

As we can see, the set of parameters λ is just a result of counting the observation and transition frequencies. The only thing we have to do when modifying the algorithm to work with multiple sequences is to not only count the frequencies for one sequence, but over all available sequences. The formal derivation of these equations can be found in [13].

Let us define $\mathbf{O} = \{O^{(1)}, O^{(2)}, ..., O^{(K)}\}$ as a set of K training sequences, where each sequence $O^{(k)} = o_1^{(k)} o_2^{(k)} ... o_T^{(k)}$ consists of T observations. Similarly, the superscripts will be used for $\gamma_t(i)^{(k)}$, $\xi_t(i,j)^{(k)}$ to refer to the variables computed over the particular sequence.

Then the transition probabilities become

$$\bar{a}_{ij} = \frac{\sum_{k=1}^{K} \sum_{t=1}^{T-1} \xi_t(i,j)^{(k)}}{\sum_{k=1}^{K} \sum_{t=1}^{T-1} \gamma_t(i)^{(k)}} \qquad 1 \leq i \leq N, 1 \leq j \leq N \tag{5.29}$$

and the initial probabilities π are just as easy to compute:

$$\bar{\pi}_i = \frac{\sum_{k=1}^{K} \gamma_1(i)^{(k)}}{\sum_{k=1}^{K} \sum_{j=1}^{N} \gamma_1(j)^{(k)}} \qquad 1 \leq i \leq N \tag{5.30}$$

These are basically the same equations as (5.23, 5.25), only this time they are summed over all the training sequences **O**. The denominator in (5.30) is there only to ensure that $\bar{\pi}$ sums up to one.

Finally we need to adjust the observation probability. Even though continuous observation probabilites are used, as stated in section 5.4.1, the update step is all about summing over K training sequences:

$$\mu_j = \frac{\sum_{k=1}^{K} \sum_{t=1}^{T} \gamma_t(j)^{(k)} \cdot o_t^{(k)}}{\sum_{k=1}^{K} \sum_{t=1}^{T} \gamma_t(j)^{(k)}} \qquad 1 \leq j \leq N \qquad (5.31)$$

$$\sigma_j^2 = \frac{\sum_{k=1}^{K} \sum_{t=1}^{T} \gamma_t(j)^{(k)} \cdot (o_t^{(k)} - \mu_j) \cdot (o_t^{(k)} - \mu_j)'}{\sum_{k=1}^{K} \sum_{t=1}^{T} \gamma_t(j)^{(k)}} \qquad 1 \leq j \leq N \qquad (5.32)$$

6 Implementation

In this chapter, we give a short overview how the described methods can be combined to create a functioning lockscreen.

The whole process can be divided into two stages - a learning stage and an authorization stage. In the learning stage, data is collected from the user and measurements are acquired from them. These measurements are then used for HMM construction and HMM parameter learning using the EM algorithm. The cross-validation is performed to determine a likelihood threshold.

In the autorization stage, after the user/hacker taps the sequence, the measurements are acquired from the data. Using Forward algorithm, likelihood of the observed sequence given the learned HMM is found. Finally, the sequence likelihood is compared to the selected threshold and it is determined, whether to unlock the device or not.

6.1 Learning Stage

6.1.1 Data Collection

User needs to think of a melody and create a sequence based on that melody, preferably following recommendations presented in chapter 7.2.3. Then the user is asked to tap his chosen sequence ten times.

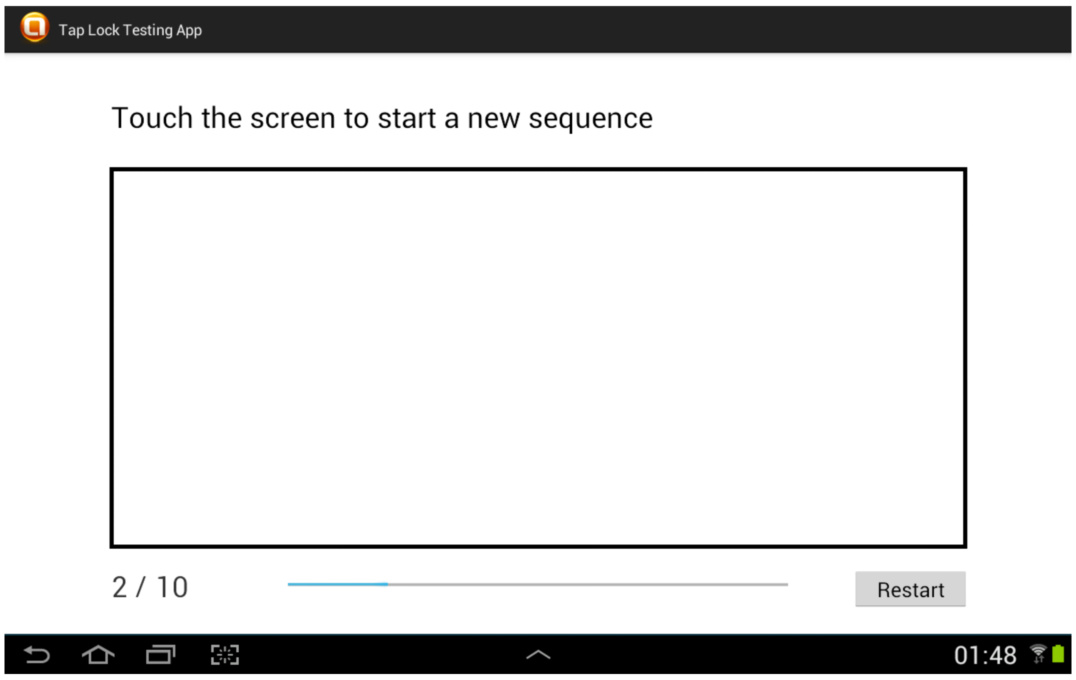

Figure 6.1 Tap lock implementation - data collection

6.1.2 Data Measurements

Consider a data sequence $D = d_1 d_2 d_{T+1}$ is collected. New data point d_i is added, whenever a touch event is registered (finger is lifted up or touches the screen). d_i is the number of miliseconds that elapsed from the first touch event. Then value d_t does not only depend on d_{t-1}, but also on all previous values $d_1 ... d_{t-1}$. That contradicts the Markov property, which says that the value d_t only depends on the previous value, d_{t-1}. Therefore the sequence D cannot be used as the observation sequence O.

Instead, the first derivative of the acquired data sequence D is used. The derivative is defined as follows:

$$o_t = d_{t+1} - d_t \tag{6.1}$$

which is basically means that the measurements are the durations between two consecutive touch events. Sequence $o_1 ... o_T$ is then used as the observation sequence O.

Then these durations are normalized, so they sum up to 1. This means that the total length of the sequence doesn't matter, only the rhythm is considered. Then the measurement can be interpreted as a duration between two consecutive touch events relative to the length of the whole tap sequence. As can bee seen in Figure 6.2, total lengths of the sequences representing the same melody differ by up to 1 second. It depends on the situation - in stressed situations, the user taps the sequence faster than when he is relaxed. However, he can still keep the rhythm.

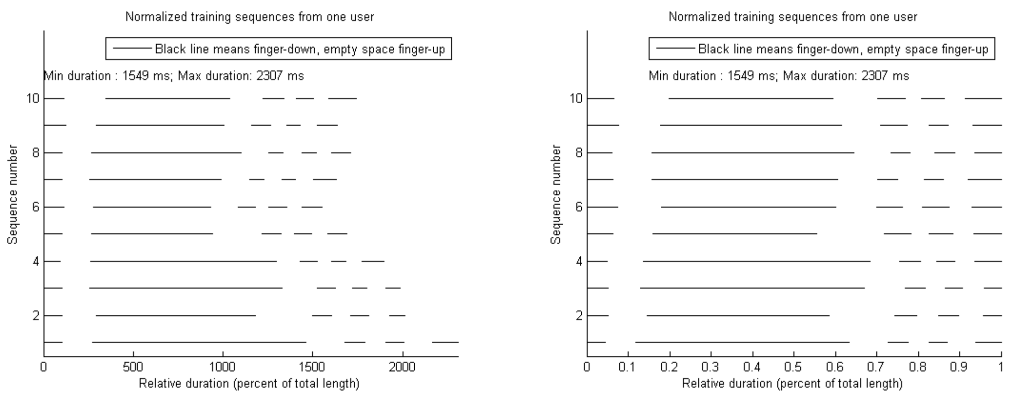

Figure 6.2 The same sequences before and after normalization. User is quite consistent in keeping the rhythm, but the total lengths differ by up to 1 second.

6.1.3 HMM Initialization

State Representation

The hidden states, just like observations, represent the duration between two consecutive tap events relative to the total length of the sequence. State $s_i, 1 \leq i \leq N$ represents the duration of length $\frac{i}{N}$.

In our implementation, we use $N = 30$ states. The lower number of states means lower number of false negatives, but also higher number of false positives. Obviously it would be better to increase the number of states, but with the algorithm's time complexity of TKN^2, where T is the length of the tap sequence, K number of sequences used for learning and N number of states, it is impractical to use too many states. We found that $N = 30$ states provides low error rate while maintaining reasonable execution speed.

Initial Parameters Estimates

As mentioned in section 5.3, the EM algorithm requires good initialization in order to give satisfying results. In our case, the k-fold crossvalidation with random initial values gave very poor results. But observed sequences can be used to set up the transition matrix A, observation vectors μ and σ^2 and the initial probabilities π.

The transition matrix A is initialized as an empty matrix of size $N \times N$. Then the algorithm iterates through all pairs of observations in all sequences $o_t^{(k)} o_{t+1}^{(k)}$ for $1 \leq t \leq T-1$ and $1 \leq k \leq K$. The closest state representations of these observations, s_i, s_j respectively, are found: $s_i = \texttt{round}(N * o_t^{(k)})$ and similarly for s_j. The state transition from s_i to s_j is then entered into the matrix A: $A_{ij} += 1$.

Because these are just estimates, some tolerance ϵ needs to be introduced. This tolerance says that they underlying state $q_t^{(k)}$ of the observation $o_t^{(k)}$ isn't necessarily $q_t^{(k)} = s_i$ (the closest state). The underlying state $q_t^{(k)}$ can rather be $q_t^{(k)} = s_l$, where $i - \epsilon \leq l \leq i + \epsilon$ and ϵ is the chosen tolerance. In our implementation, we decided to use $\epsilon = 3$ given we use $N = 30$ states. This seems like a reasonable value. Lower tolerance would mean that the learning would be too limited by the initial estimates, higher tolerance would make the initial estimates unnecessary.

To implement this tolerance into the transition probabilities estimates, the matrix A is smoothed by Gaussian filter. The width of this filter is given by the chosen tolerance. Variance used for this filter is set as $\sigma^2 = 1$. Preliminary experiments showed that if the variance of the Gaussian filter is too high, the algorithm converges slowly, but if it is too low, the learning is limited by the initial estimates just like if the tolerance ϵ was too low. In MATLAB notation, the Gaussian filter would be created as `filt = fspecial('gaussian',` $2 * \epsilon + 1, 1$`);`

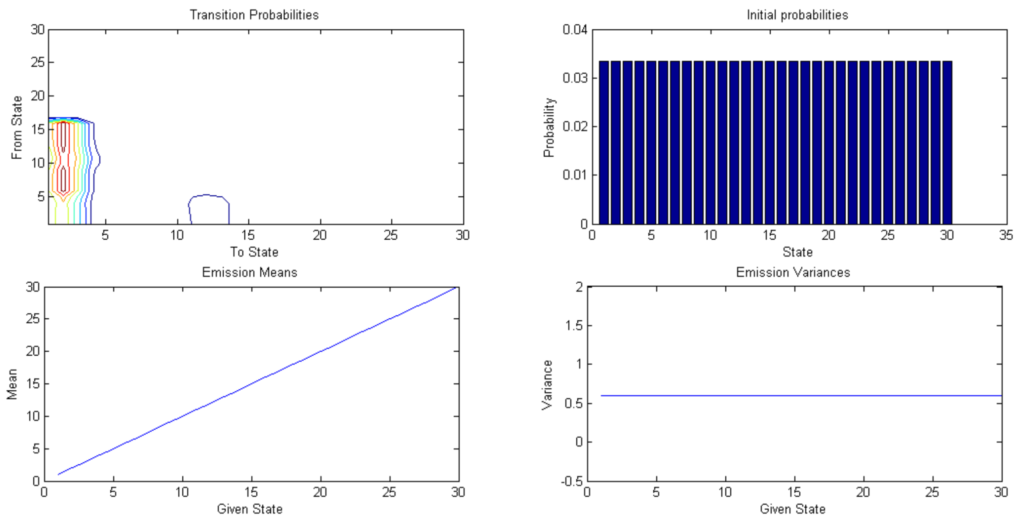

Figure 6.3 Initial estimates of transition probabilities, initial probabilities, observation means and observation variances

The state-related observation means $\mu = (\mu_1, ..., \mu_N)$ and observation variances $\sigma^2 = (\sigma_1^2, ..., \sigma_N^2)$ are then initialized following the above-mentioned concept of tolerance. Given we are in state s_i, we will most likely observe value $\frac{i}{N}$. Therefore, $\mu_i = i$ for $1 \leq i \leq N$. Tolerance is then implemented into the observation probabilities using the observation variances $(\sigma_1^2, ..., \sigma_N^2)$.

Observation variances $(\sigma_1^2, ..., \sigma_N^2)$ could be initially set to a constant value just like the variance of the Gaussian filter. However, preliminary experiments showed that if $(\sigma_1^2, ..., \sigma_N^2)$ are set with respect to the given observations, it leads to faster convergence of the algorithm.

Consider $\rho = \underset{k \in K, t \in T}{\mathtt{max}}(\mathtt{var}(o_t^{(k)}))$. Then the initial observation variance is set as the maximal variance among the observed sequences for all states: $\sigma_i^2 = \rho, 1 \leq i \leq N$.

Initial probabilities $\pi_i = \frac{1}{N}, 1 \leq i \leq N$ are simply set to a constant value. Preliminary experiments proved that this initial setting has no effect on the results nor convergence speed, algorithm will adjust these accordingly after the first iteration.

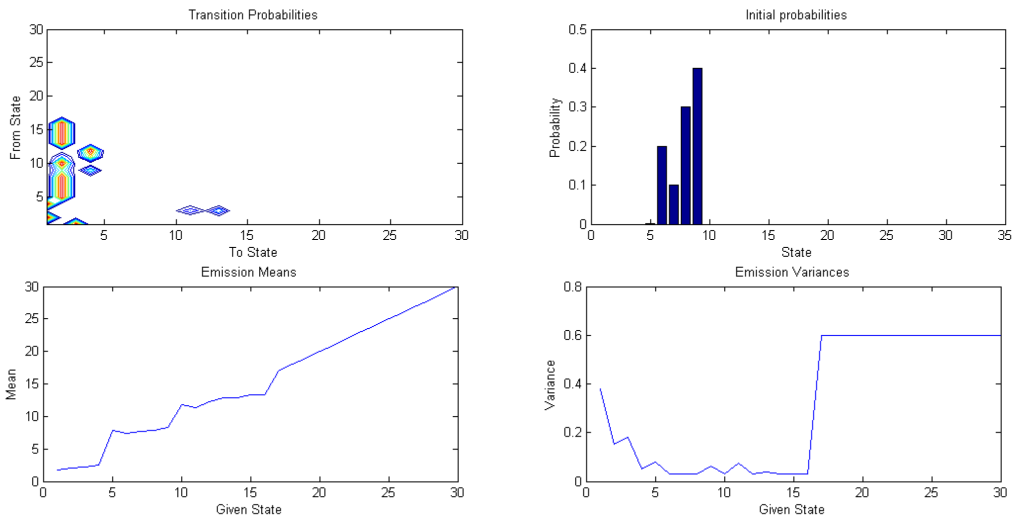

Figure 6.4 Final values afther the EM algorithm is terminated

6.1.4 HMM Parameters Optimization

After obtaining the initial estimates, the EM algorithm is iteratively executed to tune the parameters and to maximize $p(\mathbf{O}|\lambda)$.

The algorithm takes - especially on mobile devices - quite some time for each iteration to complete. Therefore it is advisable to limit the number of iterations to reduce the execution time, while still aiming for the best possible precision. We used a simple stopping criterion - as long as the algorithm converges steeply, we let it run. If the convergence becomes too slow, the algorithm is terminated (see Figure 6.5).

Consider $P_i = p(\mathbf{O}|\lambda_i)$ in iteration i. If $P_{i-4}/P_i > 0.1$, the algorithm is terminated. That is, if the likelihood $p(\mathbf{O}|\lambda)$ hasn't improved at least $10\times$ during the last 4 iterations, the convergence is considered to be too slow.

6.1.5 Threshold Selection

Because only sequences from the genuine user are availabe, and not from the hacker, the cross-validation is performed. 8 sequences are chosen at random and used for HMM training, then the remaining 2 sequences are evaluated. This process is repeated 10 times, resulting in 20 different likelihoods. The 2 lowest likelihoods are filtered out (presumed to be outliers), and the threshold is then selected as the value of the lowest remaining likelihood divided by 10. This approach is reasoned in chapter 7.2.4.

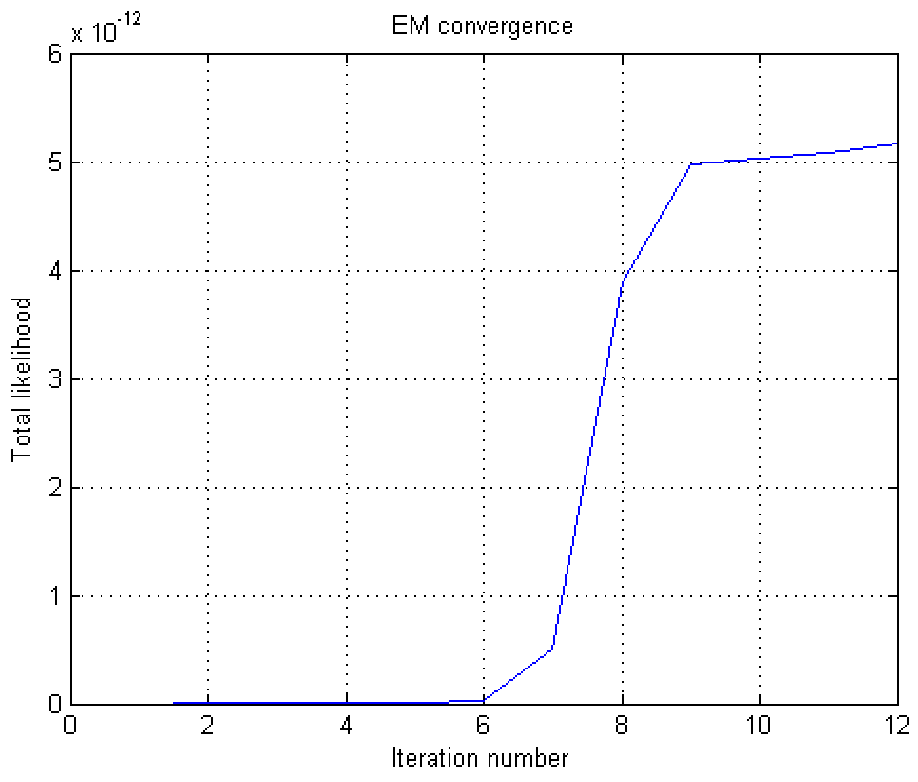

Figure 6.5 Stopping criterion based on algorithm's convergence. As long as the likelihood is improving fast enough, EM keeps iterating. After the 9th sequence, the likelihoods starts to improve very slowly, so the algorithm is terminated

6.2 Authorization Stage

When unlocking the device, the user is shown a simple, empty screen [figure]. He only needs to tap his sequence to unlock the device. The user doesn't even need to look at the device.

There are no indicators displayed that would help the potential attacker to spot the sequence, e.g. tap feedback.

6.2.1 Sequence Evaluation

When the user taps his unlock sequence, the measurements are acquired from the data (just like in Learning stage) and then the likelihood of the sequence is evaluated according to the chapter 5.2.3. If the likelihood is higher than the previously selected threshold, the device is unlocked, otherwise the access is denied.

7 Experiments

Extensive testing was conducted in order to determine Tap lock efficiency and usability. The main goal of this testing was to determine, whether the Tap lock is usable in real-life scenarios and to find out people's reactions when confronted with this new lockscreen. Apart from that, we also analyzed the sequences and their strengths and weaknesses, so we could give recommendations on how to choose a proper sequence.

7.1 Experimental Protocol

Simple testing app was developed for testing purposes. This app was designed for Samsung Galaxy Tab 10.1, but can run on most of the Android tablets.

The experiment had three parts:

- Data Collection, where the users provided training sequences,
- Testing, where the users tried to unlock their own lock
- Hacking, where the users tried to hack someone else's lock

7.1.1 Data Collection

Users were asked to choose a favourite melody, remember it and based on that chosen melody, tap a sequence 10 times (referred to as a set of observations $\mathbf{O}$[2]). Users could contribute with up to three different melodies. HMM was created and its parameters optimized right away and the result was saved into device's SD card.

Users were told that this sequence would be used for phone unlocking and all of them agreed, that they would use their selected sequence (i.e. that it's not too long or too complex for them).

The whole process was recorded on a video camera. The recording was later cut into ten parts - each part covered one sequence. These parts were then used in the Hacking part of the experiment. When choosing a video recording for hacking purposes, one of these ten parts was chosen at random and shown to the subject.

7.1.2 Testing

In the context of this experiment, *testing* refers to the process of the user unlocking his own lock (while *hacking* is the process of the user unlocking someone else's lock). Testing was divided into two stages.

The first stage took place right after the data collection. The user was asked to tap his chosen sequence 5 times. The purpose of this stage was to see whether the user could unlock his own device.

The second stage took place several days after data collection. The purpose of this stage was to find out, wheter users could remember their sequence(s) with high enough accuracy to unlock the device.

[2]Set of observations $\mathbf{O}$ = 10 sequences representing the same melody from one user.

Every user that contributed with at least one set had to go through stage 1, most of the users went through stage 2 as well[3].

7.1.3 Hacking

In this part of the experiment, we measured how easy/difficult it is to spot a sequence, how many times the thief needs to see it to be able to reproduce it and whether it can actually be reproduced with high enough accuracy.

A subject (mostly those users who took part in Data Collection, but also some new people) was assigned an observation set **O** chosen at random (excluding his own sets). He then tried to hack lock based on that set.

Hacking part was divided into three stages. First, the subject was shown one part of the recording of the other user tapping a sequence. He was allowed to see it only once to imitate real-life situations like trying to spot the unlock sequence in the public transport or in a pub. The subject was then encouraged to repeat the sequence and to try to hack the lock. He tapped 5 sequences, then the next stage took part.

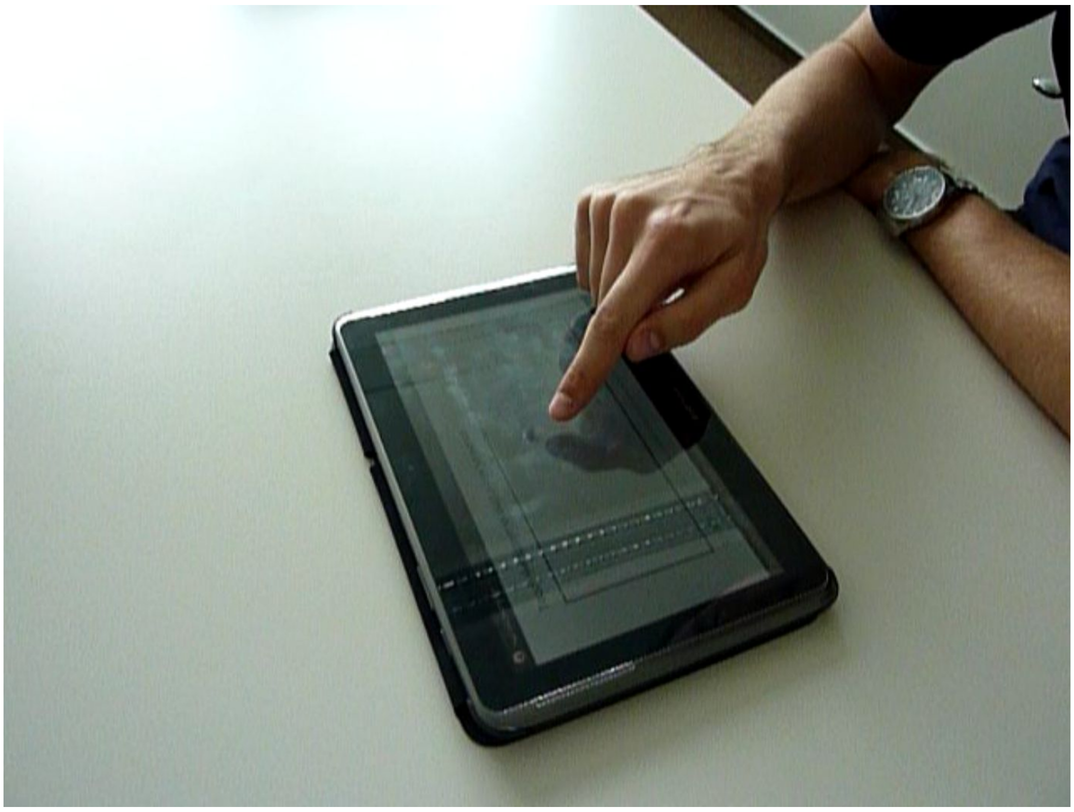

Figure 7.1 One frame of a video recording that illustrates what hackers could see

In stage 2, the subject was shown the video 3 more times (different parts of the same recording). He was then again asked to reproduce it 5 times.

In the final stage, the subject was given unlimited access to the video recording. After the subject felt confident enough, he would try to reproduce the sequence 5 more times.

[3]Some users could not be reached or were too busy to take part in the stage 2.

7.1.4 Data Statistics

15 people provided total of 29 training sets. All these people tested their locks in stage 1, 11 of those people (total of 21 sets) tested their locks in stage 2.

14 people (from which 9 provided training sets) then tried to hack someone else's lock. Total of 30 locks (chosen at random) were attempted to be hacked. Because of the random selection, some locks might have been chosen multiple times, while other locks would not be used at all.

Even though given opportunities to watch more parts of video recording in hacking stage 2 and stage 3, many users felt that it wasn't necessary and proceeded to tap the sequences right away. On the other hand, some users (usually those who failed to hack the lock in the previous stages) took the opportunity of watching unlimited video recording and studied the sequence carefully.

In 7 cases, the hackers couldn't even spot the correct number of taps during the hacking stage 1.

Users' Reactions

Users grasped the idea of Tap lock very quickly. However, for most of them it was difficult to think of more than one non-trivial melody. Therefore, most users contribuded with only one or two melodies, because they couldn't think of any other suitable melody.

In the hacking part of the experiment, hackers often became quickly frustraded when they couldn't hack a seemingly easy sequence. For some users, this was a motivation to try even harder, while other users just gave up.

The users' overall reaction was positive. They mostly appreciated the idea of Tap lock, even though some users expressed their concerns about the practical use. Approximately half of the users asked, when will Tap lock be available on Google Play.

7.2 Sequence Analysis

Evaluation of Tap lock was based on the requirements presented in chapter 4.2. Users agreed that the lock is fast and easy enough for them. Therefore the main goal of this experiment was to find false positive and false negative rate, how they affect the lock and what can be done about it. We also studied both easy-to-hack sequences and sequences that couln't be hacked, so the method could be improved and some recommendations could be made on how to choose the "perfect" melody. Finally, we analyzed the sequences in order to choose the best way of selecting threshold and to choose the minimum acceptable training set size.

7.2.1 Lock Testing

The lock is considered successfully *tested* by the author, if he could unlock it at least 3 times during the testing stage 1. It turned out that the testing stage 2 couldn't be used for evaluation. Approximately half of the people forgot the melody, because they hadn't been using it for a couple of days, and they needed to see the video recording of themselves tapping their sequence. That put them in more or less the same situation as hackers. In real-life, according to [24], people in USA check their phone up to 150 times a day. That means that they would repeat the sequence so often that they would have no chance of forgetting it.

Of those 29 different locks, 25 were successfully tested. That gives the success rate of apx. 86 %, i.e. false negative rate is about 14 %. Taking look at the first melody that even the author couldn't repeat, we can see in Figure 7.2 that the user was so inconsistent, that he couldn't repeat the same sequence.

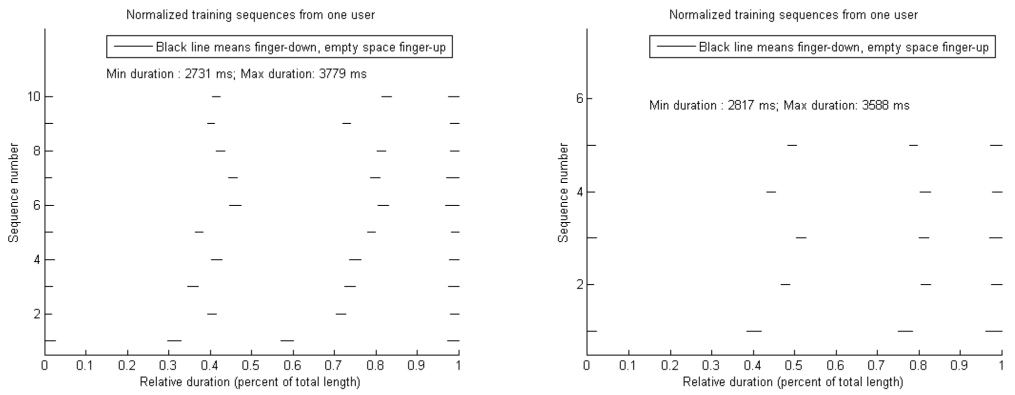

Figure 7.2 Lock that failed to be tested by the author - rhythm is too inconsistent. Left figure represents sequences used for HMM learning, figure on the right represents sequences used for lock testing.

The problem of the other melody represented by Figures 7.3 is exactly the opposite. These sequences are too consistent and similar to each other and that caused overfitting. Thus the threshold is set too high that the testing sequences are not authorized.

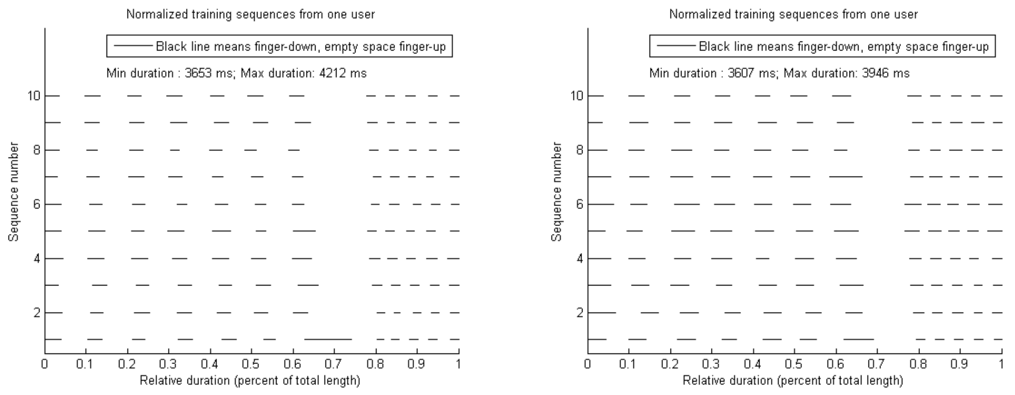

Figure 7.3 Lock that failed to be tested by the author - training sequences are so similar that it caused overfitting. Left figure represents sequences used for HMM learning, figure on the right represents sequences used for lock testing.

The remaining two locks, represented by Figures 7.4, 7.5, were created based on reasonably similar sequences. However, during the testing, the authors provided not-so-similar sequences, as if they forgot the melody. Therefore they failed to unlock the device.

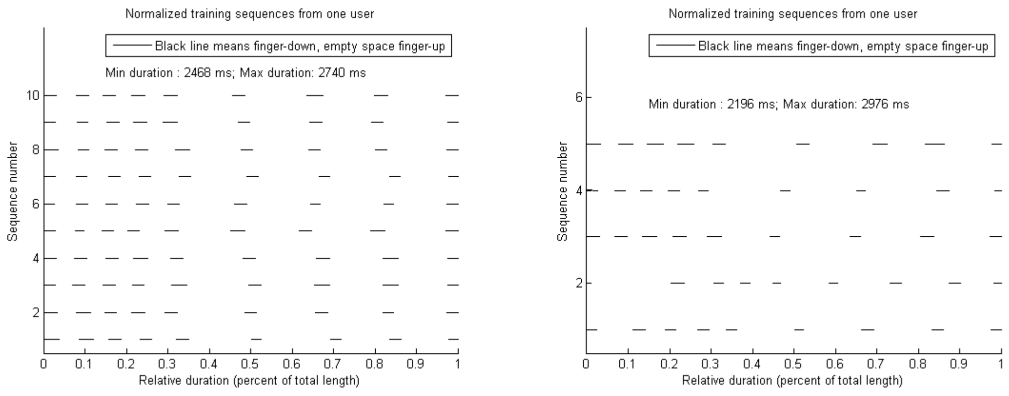

Figure 7.4 Lock that failed to be tested by the author - training sequences are reasonably similar to each other. Testing sequences, however, are quite different from the training sequences. Left figure represents sequences used for HMM learning, figure on the right represents sequences used for lock testing.

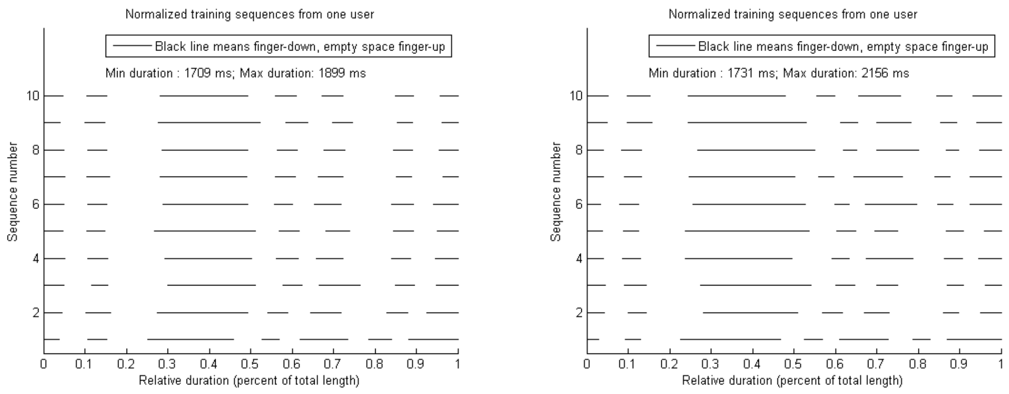

Figure 7.5 Lock that failed to be tested by the author - training sequences are reasonably similar to each other. Testing sequences, however, are quite different from the training sequences. Left figure represents sequences used for HMM learning, figure on the right represents sequences used for lock testing.

7.2.2 Lock Hacking

When defining what *hacked* lock is, the typical Tap lock use needs to be considered. Due to the nature of the Tap lock, the user doesn't need to look at the screen when unlocking his device. This is a great advantage compared to more conservative locks - the user can start tapping his sequence right after taking the device out of the pocket. This makes it even more difficult for the potential thief to spot the sequence.

In our experiments, each hacker knew what to expect and what to focus on. In real life, the hacker would probably need several opportunities before realizing he needs to repeat the tap sequence and then spotting the tap sequence clearly. In our experimental environment, we provided the hackers with perfect conditions - they were told what to expect and the sequence could be seen clearly.

Therefore, we consider the lock *hacked*, if the hacker managed to successfully unlock the device within the first 5 attempts - after watching the recording only once. In reality, the hacker wouldn't probably get more than one "perfect" opportunity anyway.

Figure 7.6 gives an overview about the hacking part of the experiment. Total of

30 locks were hacked. Some locks were hacked multiple times. Black lines above the columns connect the same locks hacked multiple times by different users. The result of yth attempt to hack lock x is marked at the position $[x, y]$ by the appropriate color. Green color means that the hacker gained access to the device, red color means that the access was denied. The missing hacking data (because hackers didn't provide more hacking sequences) are marked by the blue color.

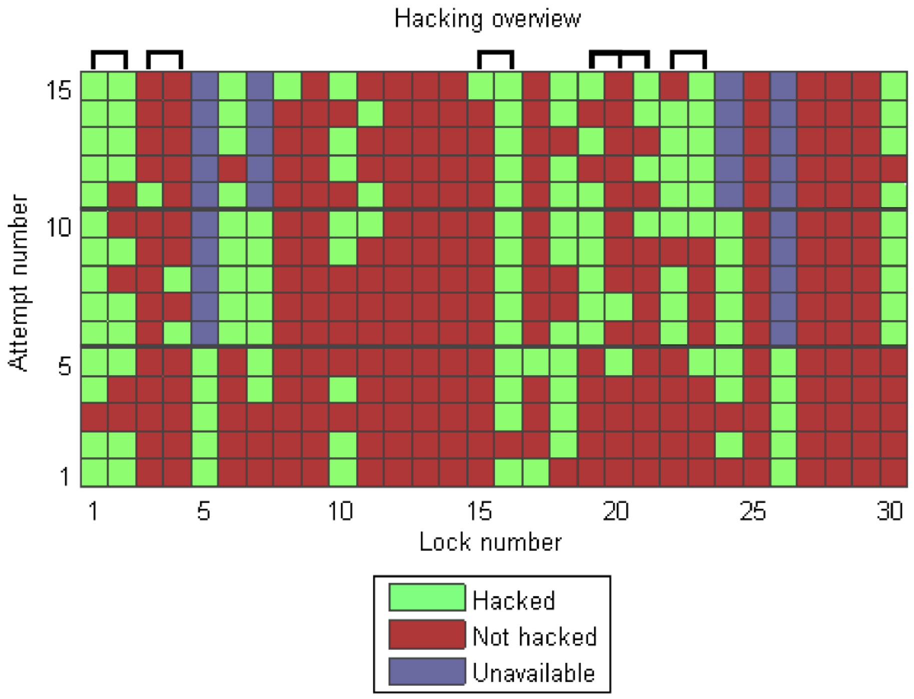

Figure 7.6 Overview of hack results recorded during the hacking part of the experiment.

As it can be seen in Figure 7.7, seeing the sequence being tapped only once was enough to hack it in 12 cases. Watching the recording 3 more times then resulted in hacking 7 more locks. Interestingly, unlimited video recording access improved the number of hacked locks by only 3. In 8 cases, the hacker didn't succeed at all.

Hacking the lock in 12 of 30 instances gives false positive rate 40 %. This number is quite high, but it can be easily reduced by following two basic recommendations on how to choose a proper sequence as discussed in the following section.

Low false negative and low false positive requirements go usually against each other. Therefore it is important to find a compromise. In order for users to use the Tap lock on a daily basis, it needs to authorize the users with as few mistakes as possible. As explained above, even if the number of false positives is high, the lock will still be difficult to hack, because the sequences are difficult to spot. In 7 cases, hackers didn't even spot the correct number of taps despite the perfect hacking conditions. Moreover, the lock is extremely resistant to random forgeries (brute-force hacking) as explained in section 7.2.6.

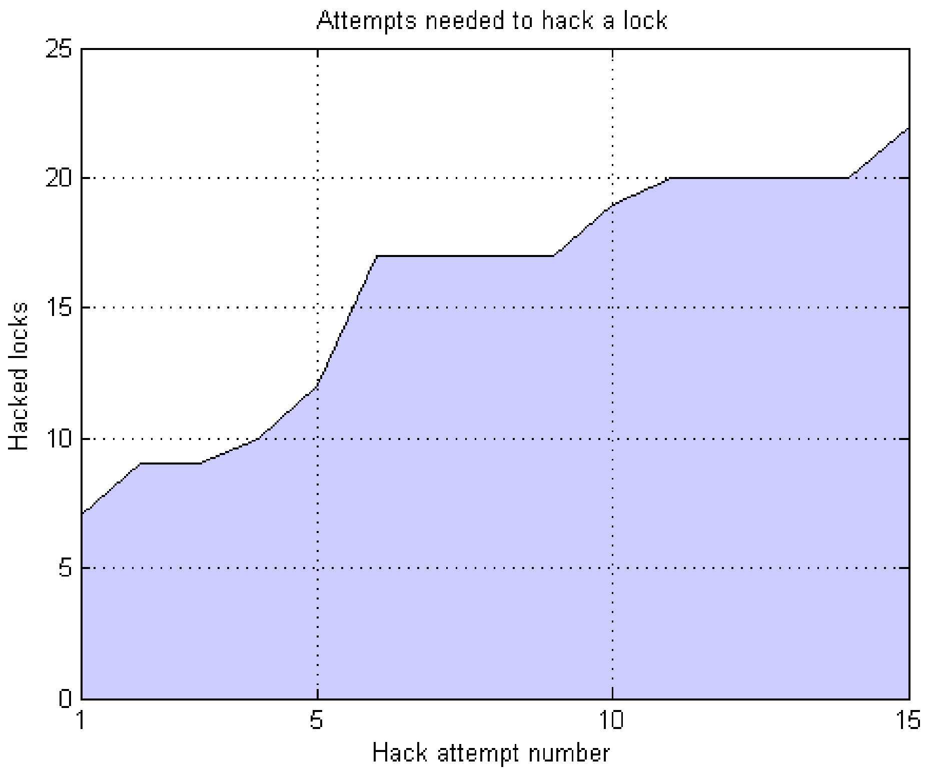

Figure 7.7 Number of locks that were hacked within the given number of attempts

7.2.3 Sequence Recommendations

Some sequence are prone to be hacked - by analyzing their weaknesses, and by analyzing the strengths of those unhacked sequences, recommendations can be made on how to create a proper sequence that is easy to repeat for the author, but difficult to reproduce for the hacker.

The two obvious things that matter the most are the tap count and the length of the taps.

As can be seen in in Figure 7.8, 7 of 12 locks hacked in the stage 1 only use sequences of 4 taps. Resistant locks usually use 6 or 7 taps. This can be explained easily - short sequences are easier to spot and remember, and thus repeat for both the user and the hacker. Longer sequences require more concentration, which is a problem for the hacker, but should not matter to the user. The user knows the sequence very well and should be able to repeat it without thinking too much about it (it's his favourite melody after all).

However, tap count itself doesn't guarantee a safe lock, as illustrated by Figure 7.9. Even though these sequences consist of a high number of taps, these taps are too monotonous, which makes those sequences easy to reproduce.

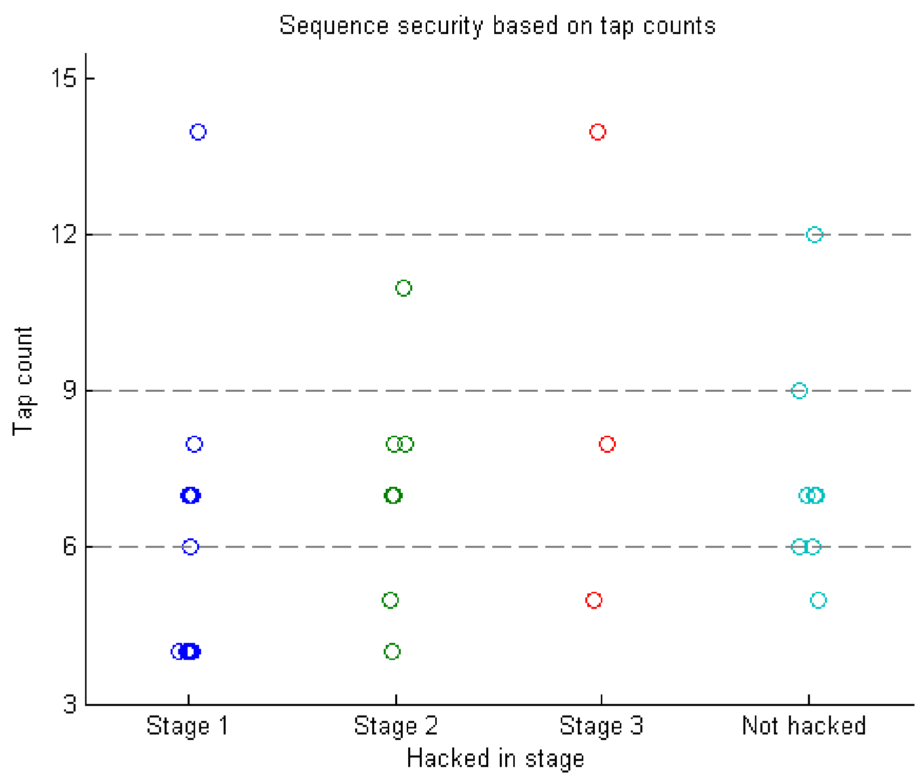

Figure 7.8 Tap counts of locks hacked in different stages. Easy-to-hack locks are characteristic by a very low tap count. Jittering (points are randomly moved to a side a bit) is applied, so overlapping values can be told apart.

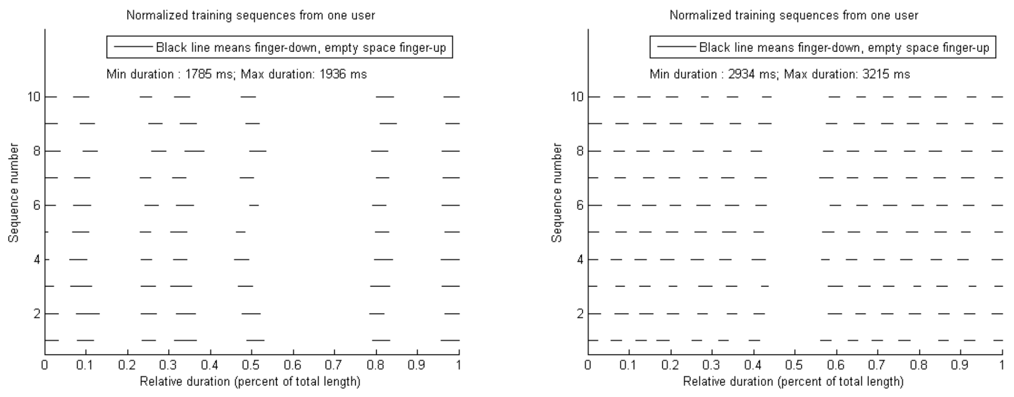

Figure 7.9 Examples of easy-to-hack sequences. Even though they consist of many taps, these taps are too monotonous.

Now consider for example the sequences in Figure 7.10. These sequences represent two very hack-resistant locks. Clearly, the authors combined short and long taps and together with sufficient tap count created safe sequences.

The best idea when choosing a melody seems to be to combine taps of varying lengths. Different tap durations make the sequences difficult to repeat for the hacker.

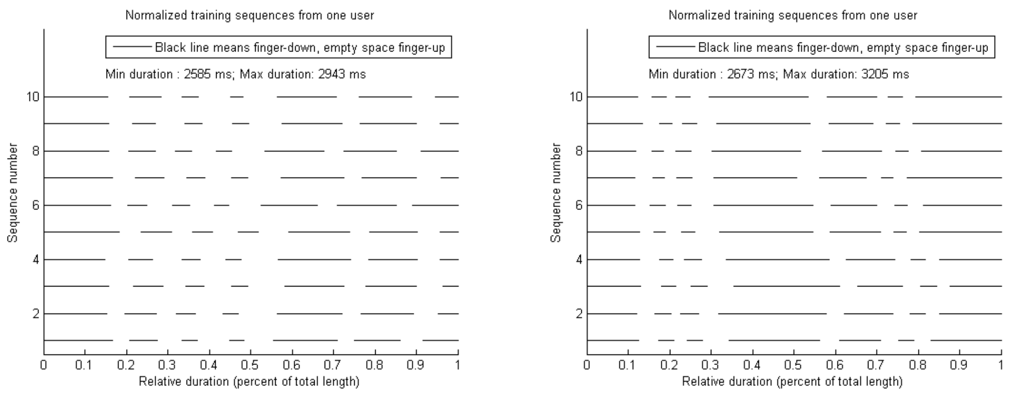

Figure 7.10 Examples of resistant sequences. They both utilize many taps of different lengths.

Recommendation

When creating a sequence, the user should utilize taps of varying length and combine them for maximal safety. The sequence should consist of at least 6 taps.

This might take a bit more time then simply tapping the first thing that comes to users's mind, but it's worth spending extra few seconds by creating a better sequence. In return, the sequence will be resistant to hacking.

As for the time required to tap the sequence - those resistant sequences in Figure 7.10 last approximately 3s. Easily hackable sequences presented in Figure 7.9 last about 2s and 3s respectively. It depends on the user, some users might want to pick an unsafe sequence that will open the device quickly, while other users prefer more security and don't mind spending an extra second unlocking the device. Still those 3 seconds is a reasonable time compared to e.g. Pattern lock or PIN lock (especially when user can tap the sequence without looking at the screen).

7.2.4 Threshold Selection

Because only the sequences from the author and not from the hacker are available when creating the HMM, the crossvalidation is performed. From the 10 available sequences, 8 are chosen at random and HMM is constructed, then the remaining 2 are evaluated. This process is repeated 10 times, resulting in a set $\mathbf{L}$ of 20 different likelihoods $L_1, ..., L_{20}$.

Preliminary experiments have shown that these likelihoods are approximately the same as the likelihoods of the testing sequences evaluated on fully learned HMM[4] and can be used for the threshold selection.

Initially, using the maximum likelihood estimation, we tried to model a Gaussian distribution $N(\mu, \sigma^2)$ based on $\mathbf{L}$. The idea was the threshold T would be selected as $T = \mu - 3 * \sigma$. So called 68–95–99.7 rule says that approximately 99.7 % values lie within 3 standard deviations from the mean. Thus we expected that 99.7 % of sequences from the author would be accepted. However, for nearly all sequences the standard deviation turned out to be so high that the threshold mostly ended up as a negative value, or such a low value that virtually any sequence was accepted, as illustrated by Figure 7.11.

By analyzing the likelihood sets $\mathbf{L}$ from various sequences, we found that most likelihoods $L \in \mathbf{L}$ lie relatively close to each other, but some likelihoods are far away and

[4]HMM learned using all 10 training sequences

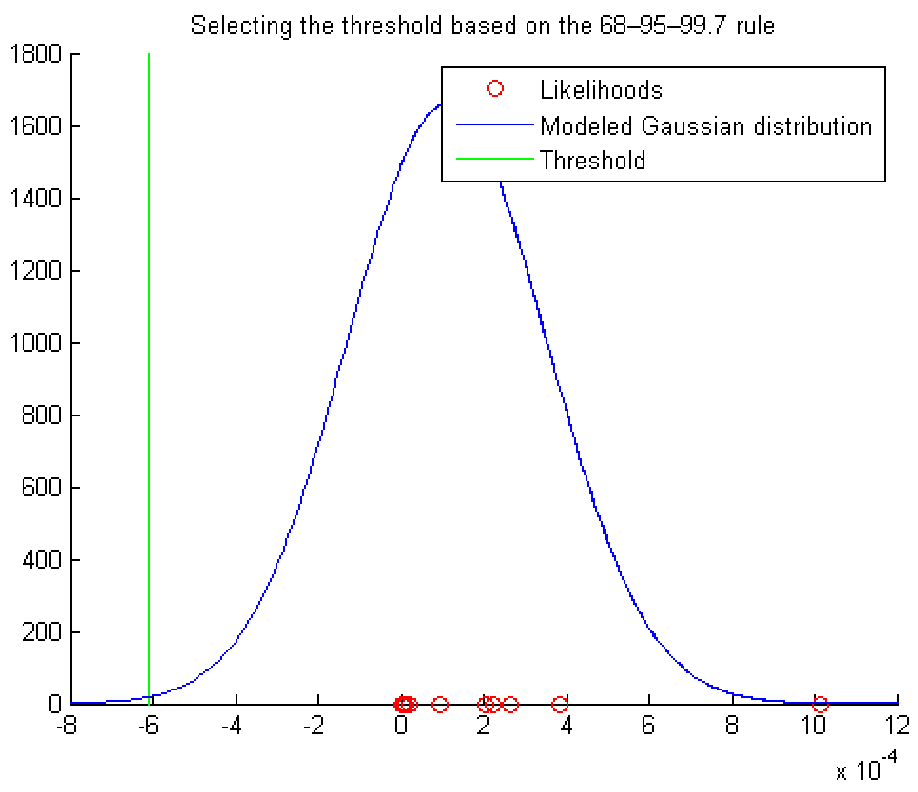

Figure 7.11 Threshold selected by using the 68-95-99.7 rule usually ends up as a negative value. Then every sequence, including hacker's sequences, is accepted.

can be considered outliers. In each sequence, there were about 5-10 % outliers.

So given there are 20 likelihoods available for threshold detection, 2 with the lowest values are filtered out preventively (presumed to be outliers). Threshold T is then set as the value of the lowest remaining likelihood divided by 10 (to add extra tolerance). This way, 90 % of the sequences from the author are expected to be above the threshold.

Figure 7.12 shows that most of the author's sequences are accepted, while hacker's sequences are refused. This lock could be considered very safe. For better visualization, log-likelihoods are used instead.

However, as can be seen in Figure 7.13, in some cases (especially in the later stages of hacking) the hacker can repeat the sequence with high engouh accuracy, that it cannot be told apart using only threshold. Therefore this lock is considered prone to being hacked.

As shown in Figure 7.14, likelihoods of the author and the hacker can overlap and cannot be told apart using threshold. In this case however, the threshold is set too high even for the author, who failed to *test* the lock. Therefore, this lock wouldn't probably be used anyway.

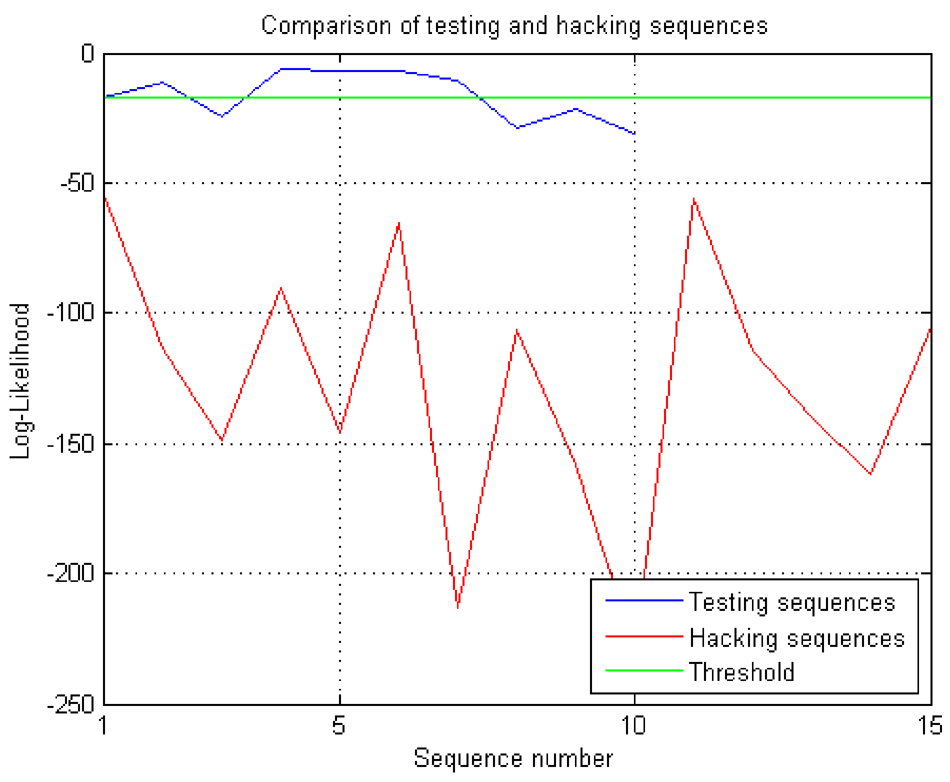

Figure 7.12 Threshold divides sequences acquired from user and hacker with a few errors. Better threshold selection would result in accepting all author's sequences while refusing all hacker's sequences.

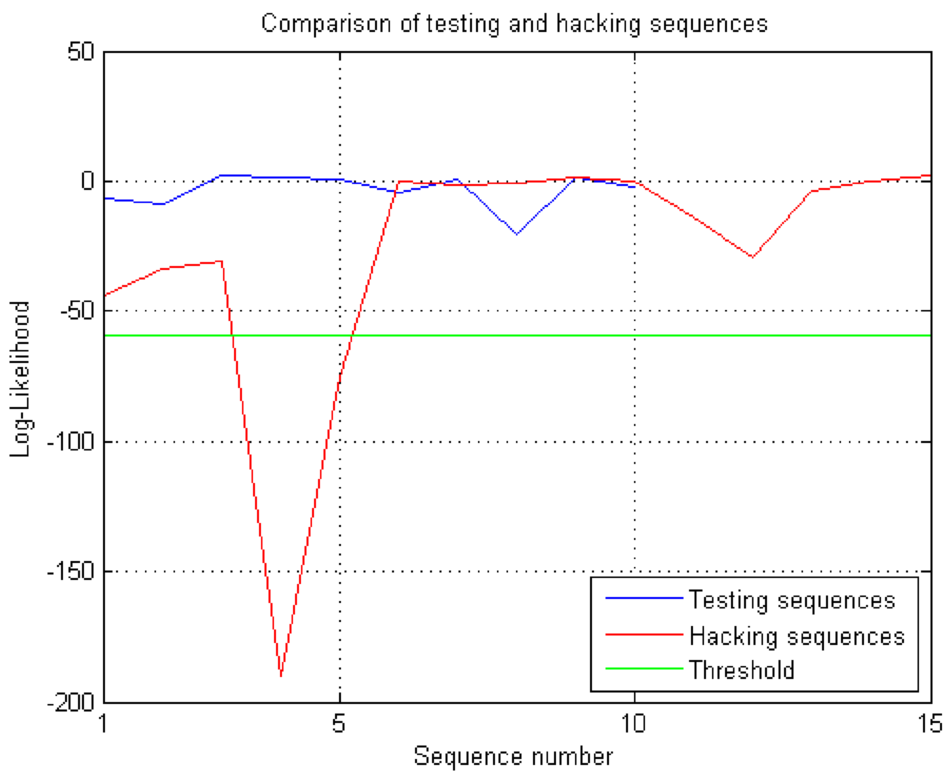

Figure 7.13 Hacker's and user's likelihoods can be told apart by the threshold, at least at the hacking stage 1. Threshold is, however, chosen to be too low, which results in easy hacking.

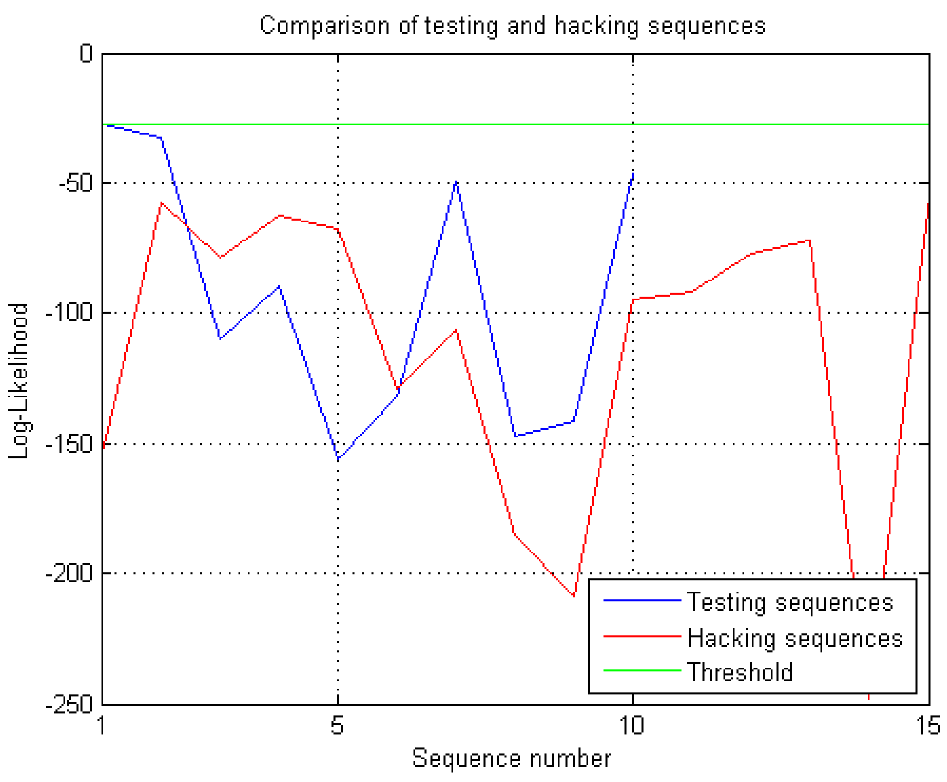

Figure 7.14 Likelihoods of sequences from the user and the hacker are nearly identical and cannot be told apart using threshold.

As demonstrated, likelihoods of user's testing sequences are usually higher than likelihoods of hacker's hacking sequences, at least in the hacking stage 1. But the threshold selection we have used should be improved. That is a challenging task, because at the time of threshold selection, neither testing or hacking sequences are available and the threshold can only be chosen using the training dataset.

7.2.5 Training Set Size

During the experiment, the HMM was trained using 10 sequences. However, in real use, it might be practical to require fewer number of training sequences.

To test how many samples it is really necessary to collect, simple experiment was performed:

Using only 3 of the sequences acquired in the Data collection stage (randomly chosen), HMM was trained. Then the sequences from Training and Hacking stages were evaluated. The number of sequences that author could *test* and hacker hack was counted. This was repeated using $4, ..., 10$ sequences.

Obviously the goal is to maximize the number of *tested* sequences (Figure 7.15), while minimizing the number of hacked ones (Figure 7.16).

Currently used way of choosing the threshold is through cross-validation. When only a small number of training sequences is provided, the data to be used for threshold selection will be very limited. The task will then become even more complicated and will give much worse results. When only 3 or 4 training sequences are used, the threshold is set very low. That results in most of the authors successfully unlocking their locks, but also means that the hackers will succeed just as easily.

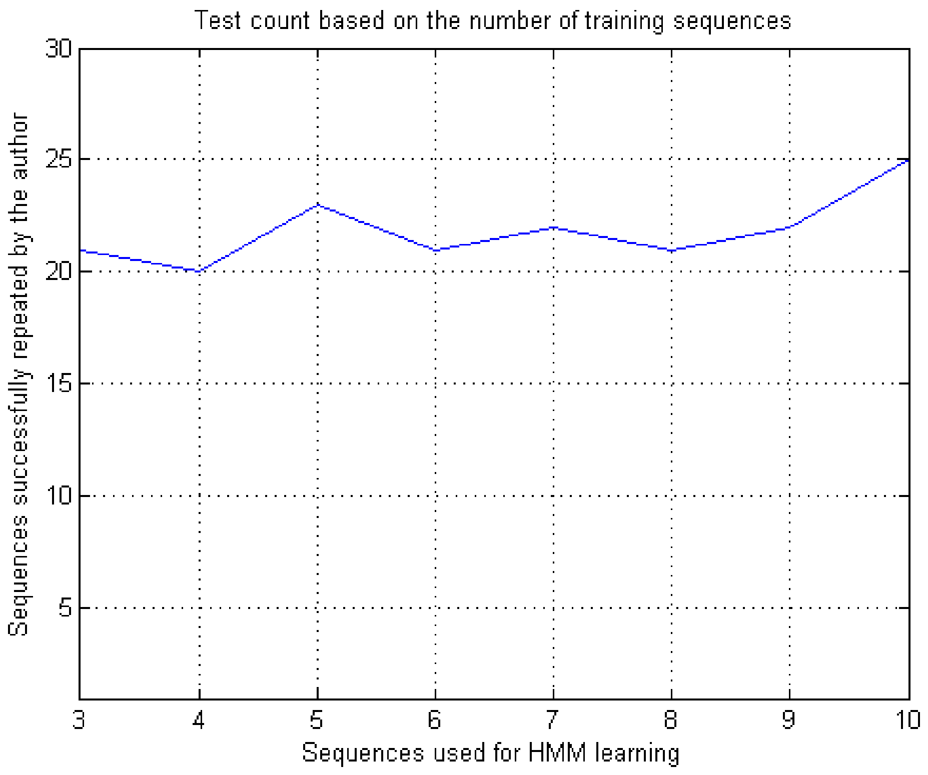

Figure 7.15 Most of the 29 available locks is successfully tested by their author even with a small number of training samples. With increasing number of available training sequences, the number of tested locks increases relatively slowly.

More training sequences mean more data that can be used for threshold selection. Better threshold selection leads to increased security and lowers the chances of the lock being hacked.

During the Data collection part of the experiment, users provided 10 sequences. It took about 1 minute to tap the sequences and apx. 20-30 seconds for HMM optimization and threshold selection, which is still acceptable. By providing only 5 sequences, the users would save about 30 seconds, but that would make the Tap lock useless.

7.2.6 Resistance To Random Forgeries

One of the main advantages of Tap lock, compared to common lockscreens, is its resistance to random forgeries.

Random forgery is a brute-force attack, sequence tapped without any knowledge of how the author's sequnces look like. The hacker doesn't know the tap count nor the tap lengths.

In Tap lock, it is not possible to systematically try all possible combinations like e.g. when guessing PIN or Pattern. There is a huge number of possible combinations and even miliseconds can make difference. The hacker has no idea how long his own taps are (cannot measure it with milisecond precision). The sequence normalization complicates the situation (in terms of random forgeries) even more.

There is also no way how to reduce the number of possibilite combinations, such as looking for smudge trail. So if the user doesn't pick an easy sequence (low number of taps or monotonous taps) or some extremely popular melody (such as Star Wars theme)

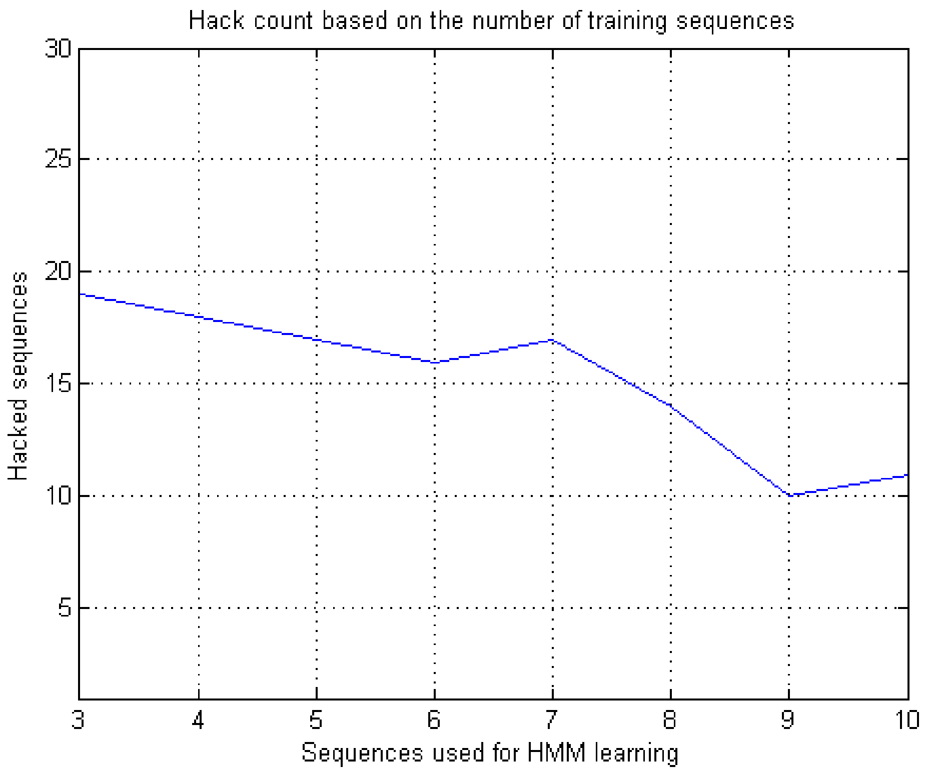

Figure 7.16 The locks provide very low security when only a small number of training samples is used.

and rather follows the recommendations (7.2.3), the Tap lock will be nearly impossible to hack without spotting the sequence.

7.3 Summary

The Tap lock is a functioning lockscreen. It provides fun and innovative way of unlocking one's device. The number of false positives is acceptable for common use and will probably decrease over time (user will get more accustomed to his own melody). If the recommendations for choosing a proper sequence are followed, it also provides high security. Unless the unlock sequence is spotted, which is quite difficult, it is nearly impossible to be hacked by a brute-force attack.

7.3.1 Future Work

Even though the Tap lock can be used as is, there are still some things that could be improved. This is mainly the process of threshold selection. As demonstrated in the section 7.2.4, likelihoods of user's and hacker's sequences can often be told apart, but the threshold is set too high or too low, which results in increased number of false positives/false negatives.

Also, a completely different approach might be used instead of hidden Markov models to compare the methods.

8 Conclusions

We have found that current lockscreens either do not provide enough security or require too much effort to be unlocked. There is a demand for a new lockscreen that is both secure and user-friendly. We have decided to come up with a new lockscreen that fulfills these requirements.

Rather than forcing users to remember another code, the natural movement of their fingers can be used for unlocking. Several papers were already written on the subject, but they all analyze the gesture or the pattern the user draws on the touchscreen. In our opinion this is a dead end, because simple things like raindrops on the touchscreen or greasy fingers can make the whole lockscreen unusable.

We have proposed Tap lock, a unique lockscreen that relies on tapping sequences based on the melody in the user's head. This approach fulfills the above-mentioned requirements and has a great marketing potential.

There are still some aspects of our approach that could be improved - namely the likelihood threshold selection. This might be a topic for a future work.

Finally, Tap lock has been implemented for Android devices. Experiments have shown that it could be used in real life if simple recommendations on how to choose a proper tap sequence are followed.

During the summer 2014, we plan to publish the Tap lock on Google Play in cooperation with avast! antivirus company.

Bibliography

[1] Signature verification, 2005. http://www.cse.msu.edu/~cse802/Papers/802_Signature_Verification.pdf. 13

[2] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[3] Anil K. Jain; Friederike D. Griess; Scott D. Connell. On-line signature verification. *Pattern Recognition*, 35, 2002. 13

[4] Kellex; droid life.com. An overview of android lock screen security options, 2013. http://www.droid-life.com/2013/03/27/an-overview-of-android-lock-screen-security-options-beginners-guide/. 6, 7

[5] Julian Fierrez; Javier Ortega-Garcia; Daniel Ramos; Joaquin Gonzalez-Rodriguez. Hmm-based on-line signature verification: Feature extraction and signature modeling. *Pattern Recognition Letters*, 28(16), 2007. 12, 13

[6] Jerry Hildenbrand. Late-night poll: Do you use lockscreen security?, 2012. http://www.androidcentral.com/late-night-poll-do-you-use-lockscreen-security. 4, 6

[7] Alexander De Luca; Alina Hang; Frederik Brudy; Christian Lindner; Heinrich Hussmann. Touch me once and i know it's you!: implicit authentication based on touch screen patterns. In *SIGCHI Conference on Human Factors in Computing Systems*, 2012. 12, 14

[8] Keyvan Kambakhsh. Sensor lock pattern demo. http://www.appszoom.com/android_applications/tools/sensor-lock-pattern-demo_deayw.html. 2, 10

[9] LSDroid. Cerberus anti theft, 2014. https://play.google.com/store/apps/details?id=com.lsdroid.cerberus. 8

[10] Police of the Czech Republic. Statistical criminality overview, 2014. http://www.policie.cz/clanek/archiv-statistiky-statisticke-prehledy-kriminality.aspx. 4

[11] Nick T.; phonearena.com. 10 lock screen replacement apps for your android smartphone, 2013. http://www.phonearena.com/news/10-lock-screen-replacement-apps-for-your-Android-smartphone_id46003. 8

[12] Donato Impedovo; Giuseppe Pirlo. Automatic signature verification: The state of the art. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 38(5), 2007. 12, 13

[13] Xiaolin Li; M. Parizeau; Rejean Plamondon. Training hidden markov models with multiple observations-a combinatorial method. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2000. 23

[14] Simon J. D. Prince. *Computer vision: models, learning and inference*. Cambridge University Press, 2012.

[15] Lawrence R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. In *Proceedings of the IEEE*, 1989. 16, 21

[16] Dit-Yan Yeung; Susan George; Ramanujan Kashi; Takashi Matsumoto; Gerhard Rigoll. Signature verification contest 2004, 2004. `http://www.cse.ust.hk/svc2004/index.html`. 13

[17] C. Gruber; T. Gruber; S. Krinninger; B. Sick. Online signature verification with support vector machines based on lcss kernel functions. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 40(4), 2009. 12

[18] Colin Sindle. Handwritten signature verification using hidden markov models. Master's thesis, Stellenbosch University, 2003. 12

[19] SpSoft. Smart app lock, 2014. `https://play.google.com/store/apps/details?id=com.sp.protector.free`. 2, 8

[20] Stephen Tu. Derivation of baum-welch algorithm for hidden markov models. `http://people.csail.mit.edu/stephentu/writeups/hmm-baum-welch-derivation.pdf`. 21

[21] TwinBlade. Picture password lockscreen, 2014. `https://play.google.com/store/apps/details?id=com.TwinBlade.PicturePassword`. 2, 9

[22] Julio Angulo; Erik Wastlund. Exploring touch-screen biometrics for user identification on smart phones. Master's thesis, Karlstad, Sweden, 2012. 12, 13, 14

[23] Lance Whitney. iphone market share shrinks as android, windows phone grow, 2014. `http://www.cnet.com/news/iphone-market-share-shrinks-as-android-windows-phone-grow/`. 6

[24] Mary Meeker; Liang Wu. 2013 internet trends, 2013. `http://www.kpcb.com/insights/2013-internet-trends`. 32

[25] Alisher Kholmatov; Berrin Yanikoglu. Identity authentication using improved online signature verification method. *Pattern Recognition Letters*, 26, 2005. 13

[26] Panagiotis Andriotis; Theo Tryfonas; George Oikonomou; Can Yildiz. A pilot study on the security of pattern screen-lock methods and soft side channel attacks. In *WiSec '13*, 2013. 7, 11