Charles University in Prague

Faculty of Mathematics and Physics

MASTER THESIS

Juraj Hámorník

Signature-based User Authentication

Katedra distribuovaných a spolehlivých systémů

Supervisor of the master thesis:	Mgr. Pavel Jančík
Study programme:	Informatics
Specialization:	Software Systems

Prague 2015

I would like to thank my supervisor Mgr. Pavel Jančík for his time and his very helpful, constructive advice on this thesis.

I would like to thank the coffee machine for working long hours, always providing great espresso that keep the caffeine level high.

Last but not least, I would like to thank my family for their kind support during my whole bachelor and master studies and my girlfriend Zuzana for being my companion and for making me happy (besides that, she also read this whole thesis)

Název práce: Overováni osob podpisem

Autor: Juraj Hámorník

Katedra: Katedra distribuovaných a spolehlivých systémů

Vedoucí diplomové práce: Mgr. Pavel Jančík, Katedra distribuovaných a spolehlivých systémů

Abstrakt:
Tato práce zpracovává chybějící možnost autentifikace uživatele podpisem v systé -mu Windows. Výsledkem této práce je software, který umožňuje přihlášení uživatele do systému Windows podpisem. Zaměřili jsme se zejména na bezpečnost ověřování podpisu a na co největší uživatelskou přívětivost. Implementovali jsme autentifikační službu, která přijímá podpis a vrací přístupový token v případě jeho pravosti. Ověřování je realizované porovnáváním podobnosti zkoumaného podpisu se vzory. Podobnost je počítána metodou dynamic time wrap na základě dynamických vlastností podpisu jako jsou rychlost, zrychlení a tlak pera při psaní. Přístupový token využívá náš plugin přihlášení, nazvaný signature credential provider, na dekódování přístupových údajů a vykonání přihlášení. Výsledkem této práce je řešení, které dovoluje přihlášení uživatele do systému Windows ručně psaným podpisem s rizikem 5.27 procent na nepřihlášení nebo přihlášení nevhodné osoby.

Klíčová slova: overovávní ručne psaného podpisu, windows login, credential provider

Title: Signature-based User authenticaion

Author: Juraj Hámorník
Department: Department of Distributed and Dependable Systems

Supervisor: Mgr. Pavel Jančík, Department of Distributed and Dependable Systems

Abstract:
This work aims on missing handwritten signature authentication in Windows. Result of this work is standalone software that allow users to log into Windows by writing signature. We focus on security of signature authentification and best overall user experience. We implemented signature authentification service that accept signature and return user access token if signature is genuine. Signature authentification is done by comparing given signature to signature patterns by their similarity. Signatures similarity is calculated by dynamic time warp on dynamic signature features such as speed, acceleration and pressure. User access token is used by our Windows login plugin called signature credential provider to decrypt user credentials and perform log in. Result of this work is solution that allow user log to windows by handwritten signatures, with equal error rate of 4.17%.

Keywords:
handwriiten signature authentication,credential provider,windows login

Contents

Introduction

People, especially in these days when information about passwords leaks regular fills the media, are aware of what security threads in form of standard passwords. They seek other authentication methods. Most popular and secure are biometric authentication systems. Biometric attributes used for authentication include iris, hand geometry, face and fingerprints, but they require special hardware to capture. Behavioral biometric attributes have similar characteristics such as universality and uniqueness but are easier to obtain.Walk characteristic, keystroke dynamics, signatures and more belong among behavioral biometric attributes. Signatures are most widely and traditionally used in authentication documents and enforcing binding contracts in paper-based documents and hence are socially accepted. We will combine advantages of signatures with tablet pc capability to proceed handwritten input.

To successfully use handwritten signature to perform log in to Windows we must solve two problems: authenticate user by given signature and to be able to process information about user authentication to perform log in to the Windows.

User authentification is done by comparing similarity between given user signature and stored signature patterns. Similarity is calculated from dynamic-based signature features such as speed, acceleration and pressure. These features are compared by dynamic time warp method. To better reflect uniqueness of each person signature, we weight these features. Feature weight is based on feature entropy and deviation. Result of user authentification is access token, which is used in custom Windows login plugin. This log in plugin must transform access token to user credentials and submit it to Windows. Windows have limited capability to capture handwritten input data from login screen, so our Windows login plugin muse solve this problem as well.

Main motivation for writing this thesis was that in time of writing this thesis there are no solutions that use handwritten signature to log user to the Windows. This thesis has goal to develop such solution.

Structure of the Thesis

The thesis is dived into two main parts. Signature authentication and Windows logon plugin that is called Credential provider.

Signature part (Chapters 1. and 5 to 6). Chapter 1 overviews signatures. In this chapter we discuss history of signature, signature usage for authentification between other bionic traits, capturing signature to computer and signatures features. We also discuss some method used in signature verification. At the end of the chapter we briefly discuss signature forgery. Chapter 5 discuss design decision about signature authentication such as choosing signatures features for authentication and choosing method for signature authentication. Chapter 5 describes implementation of signature authentication mechanism based on this decisions.

Credential part (Chapter 2 to 4). Chapter 2 overview credential provider architecture and differences between login architectures in Windows XP, Windows Vista/7 and Windows 8. In Chapter 3 we discuss design decision that have to be made for implementation our signature credential provider. Implementation of signature credential provider is described in chapter 6.

Additionally Chapter 7 evaluate signature authentication with genuine and forged signatures from SVC2004 [30] (First international signature verification competition.

1. Signature overview

Signature, according to the American Heritage dictionary, can be define as “the name of a person written with his hand or “the act of signing one’s name”. Signatures are with us during whole lifetime. Birth certificates, papers that establish who we are and what right and privileges we have are signed by parents or doctors. First thing that a child usually learn is to write their own name, so basically a signature. At the opposite end one of last paper that person sign is a testament, paper that provide distribution of person property at death.

In next section, we look at signature history, usage, features, capturing and at least forging.

1.1 History of handwriting signature

Handwritten signature is based on written language so we start at the creators of handwriting. The Sumerians (5th to 3rd millennia BC) are considered as the creators of writing and also developers of ways to authentication writer. The Sumerians created intricate works of art carved in clay tables or seals to identify ownership. In many civilizations people used symbols and other marks to authenticate writings, acknowledge writings and even to accept the content of document [1].

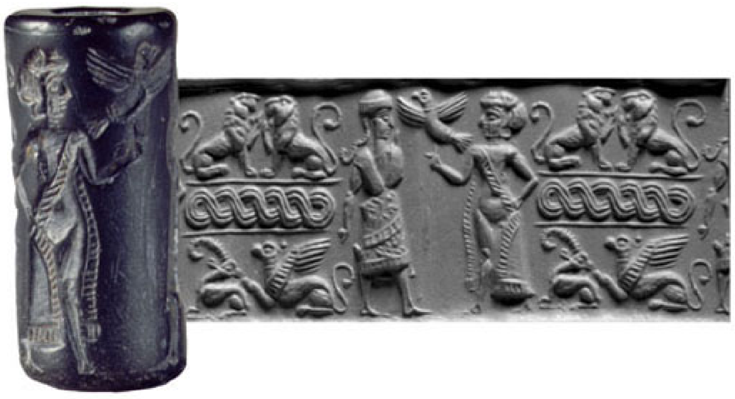


Figure 1.1: Sumerian cylinder seal and its imprint to clay. This imprint served as signature

Use of signatures is recorded in the Talmud [2] around 4th century, complete with security procedures to prevent the alteration of documents after they are signed. The Talmud describes use of a form of signature cart by witnesses to deeds. The practice of authentication of documents by affixing handwritten signatures began to be used with the Roman Empire during the rule of Valentian III in year 439 AD. A short handwritten sentence at the end of a document, named the subscipto stating for that the signer subscribed to the document. This subscipto was first used to authenticate wills. The practice of affixing signatures to documents spread rapidly from initial usage at wills and form of signatures (as we describe already it a hand written representation of one's own name) remain unchanged for over 1400 years [3].

Every authentication mechanism, however great its idea, is sooner or later bypassed. There is evidence that signature forgery was practiced shortly after the invention of writing. Problem with signature forgery remained ever since these days. Not surprisingly, signature forgery, was a very lucrative business for individuals in high office.

For example Titus was a skillful forger in his time. Another case is that Cicero berated Anthony for making profits by counterfeiting handwriting. Based on this two and more cases in the year 539 AD,only 100 years after Romans started using signatures, Romans generated legislation that establisher requirements for forensic document examination. This requirements specifying under what circumstances their testimony may be given in the cases of forgery.

"Comparison of handwriting shall only be made in the case of public documents, and in the case of private instruments where the adverse party can use them to his own advantage. For we entertain hatred for the crime of forgery. We order that experts charged with the comparison of the handwriting of public documents shall be sworn before any private instruments are placed in their hands for this purpose. Wherefore this law, as well the present modification of the same, shall remain in full force, and experts aforesaid shall by all means be sworn"[4]

Not all civilization held a negative attitude to signature forgery. Albert S. Osborn in book "Questioned Documents" [5] suggested that I was not as result of design, but as an outgrowth of circumstances and condition that the English common law for many years favored signature forgery. The great period of time in English history an atmosphere of mystiques was associated with the written word. This mystique was so big, that for many generation the "comparisons of hands" was not only illegal but it was even highly improper. In other word handwriting and handwritten signature could not be identified or verified. Thinks starting changing slowly in 1854 when British system of justice allow handwriting comparisons in civil cases. By 1865 all restriction were lifted to allow handwringing comparisons in criminal cases.

1.2 Signature as method of authentication

Today's security requirements place biometrics authentication to the center of interest. Term biometric refers to individual recognition based on a person's distinguishing characteristics. While other non-biometric technique use possession of item such as tokens (NFC badges, ID cards etc.) or the knowledge of something (password, key phrase) to perform authentication, biometric techniques use inherent characteristics of the person to perform this task. Biometric technique have advantage that it can't suffer from lost or theft as tokens or can't be forgotten as password.

Biometric trait should have follow characteristic

- Universality (each person should possess the trait)
- Uniqueness (no two person should share the same trait)
- Permanency (the trait should not change during lifetime)
- Collectability (the trait can be obtained easily)
- Reducibility and comparability (the trait should be capable of being reduced to format that is easy to handle and digitally comparable to others)
- Cost (how expensive is to perform authentication by the trait)

There are wide sets of biometric trait that are used for authentication (e.g. fingerprint, face, iris, palm veins). None of these traits satisfy all the desired characteristic. Selection of right biometric trait depend on the specific application, because it involve both technical issues and culture and social aspect [6].

Handwritten signature have special place in set of biometric trait. Handwritten signature is most widespread method of personal verification. Signatures are generally recognized as legal way of verifying person identity by administrative, government and financial institution. Person verification does not require invasive measurements and peoples are familiar with the use of signatures in their daily life [16].

Handwritten signature, alongside with keystroke dynamic and walk characteristic, are considered as behavioral biometric trait. Behavioral biometric trait is trait based on an individual behavior. This cause that signature can vary by conditions under which the signing occurs and depends on the psychophysical state of signer. Some complex theories have been made for psychophysical mechanism underlying handwriting [8].Because of the variability over time of most behavioral characteristic, an authentication system needs to be designed to be more dynamic and accept some degree of variability. On the other hand behavioral biometrics are associated with less intrusive systems, so they are better accepted by the users.

1.3 Signatures retrival

There are two signatures retrieval method:static and dynamic.

Static signatures retrievals are used when signing was completed. For digitalization of already written signatures scanning and photographing are used. Scanned signatures are represented as 2D image. Scanned signatures may contain spurious noise which has to be removed to avoid errors in the further processing steps. Processing is used to improve quality of retrieved information about signature. Processing stages include : image resizing, thinning (thinning make extracted features invariant to quality of paper and pen, thinning means reducing binary object or shapes to strokes a single pixel wide) and creating of the bounding box. Example of pre and post processed signature can be seen in 1.2. Mostly only parameters feature of signature can be obtained by this method.

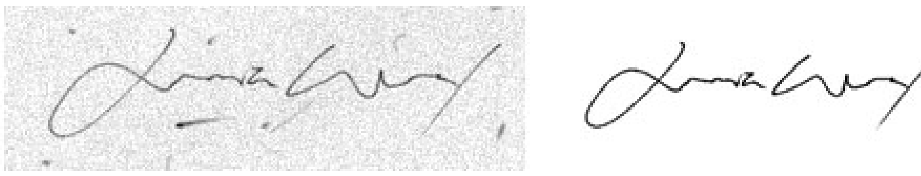


Figure 1.2: Scaned signature, pre and post processed

Dynamic method involving usage of special hardware(such as Wacom Intuis 1.3) to capture whole dynamic process of writing signature. Mostly used hardware for capturing signing are digitizing tablets. Person write with a specific electronic pen called stylus on tablet surface. Whole movement of stylus on (and usually at the small distance off) table surface is captured. Alongside exact position at the time additional information about pen pressure, writing force, pen inclination (angle between tablet suffice and pen) are stored.

Figure 1.3: Wacom intuos tablet digitizer

Other methods of capturing writing of signatures included touch screens with specific or traditional pen. This methods provide visual feedback of writing signature. 1.4

Other Methods are using traceable digital pen. Traceable digital pens are often able to capture time additional information about writing like digitalize tablet. Tracking of pen is realized by: strain gauges, magneto elastic sensors, shifting of resonance frequency, laser diodes, tracking small patterns on table with camera build in pen and tracking pen movement by filming writing . Some

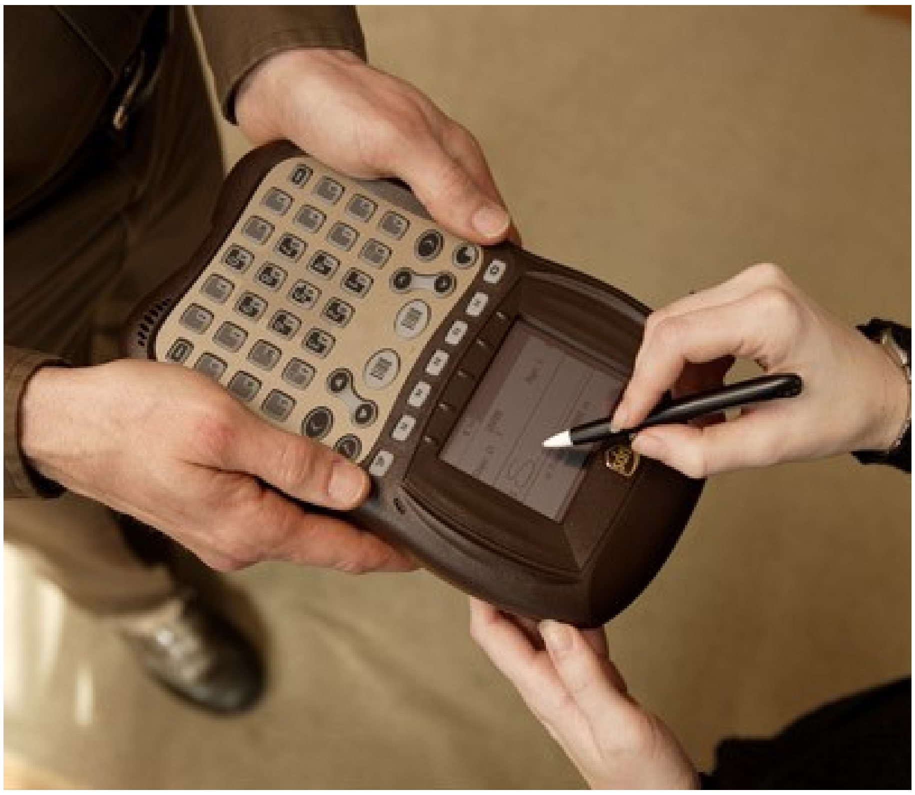


Figure 1.4: UPS sign pad

traceable digital pen (as shown in 1.5) use traditional ink pen and allow to write on paper. This provide important haptic feedback from paper.

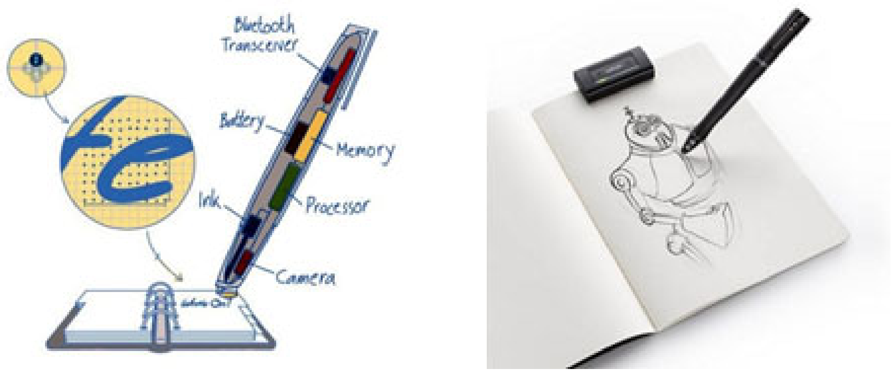


Figure 1.5: Example of digital pens

1.4 Signature features

There are two types of signature features that are used for signature verification: 1) Functions, 2) Parameters.

1. Function features are used in terms of time functions whose values constitute the feature set. Parameter features are used in terms of vector of elements, each one representative of the value of a feature. Function features allow

better performance than parameters, but require higher system cost for matching [21].

2. Parameter features are classified to two categories: 1) Global, 2) Local. Global parameters concern features that are related to whole signature i.e.: total writing time, numbers of pen strokes in signature and total written distance. Local features are like global features but retrieved only from specific part.

Depending on the level of details, local parameters can be divided into two categories: 1) component-oriented 2) point (pixel) oriented. Component oriented parameters are extracted at the level of component i.e. stroke height: width ratio and relative position of signature parts. Point oriented are extracted at levels of point (pixel) resolution, i.e. Pixel density and total count of the stylus points. This feature classification isn't strict. Some features can be used globally and locally. Feature classification is shown on 1.6

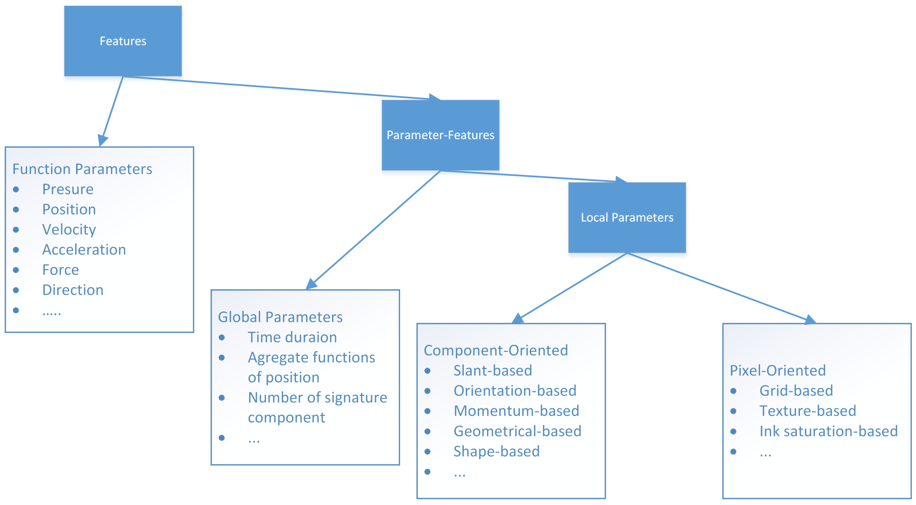


Figure 1.6: Features clasifications

We describe in short some of the most common function and parameter features found in literature.

- Position (X and Y position of pen tip)
- Speed Overall speed of writing or in X and Y direction)
- Acceleration (Overall acceleration or in X and Y direction)
- Pen inclination (inclination of pen with signing area)
- Pressure (pressure on which pen leaves to the signing area)
- Direction of pen movement (direction of pen speed vector)
- ...

Position, Speed and acceleration are widely used for online signature verification. Velocity and acceleration function can be obtained from specific device or numerically derived from position. Pen inclination is considered as most consistent feature, however, it can be captured only by specific device. Vizualization of theese pattens are shown on 1.6.

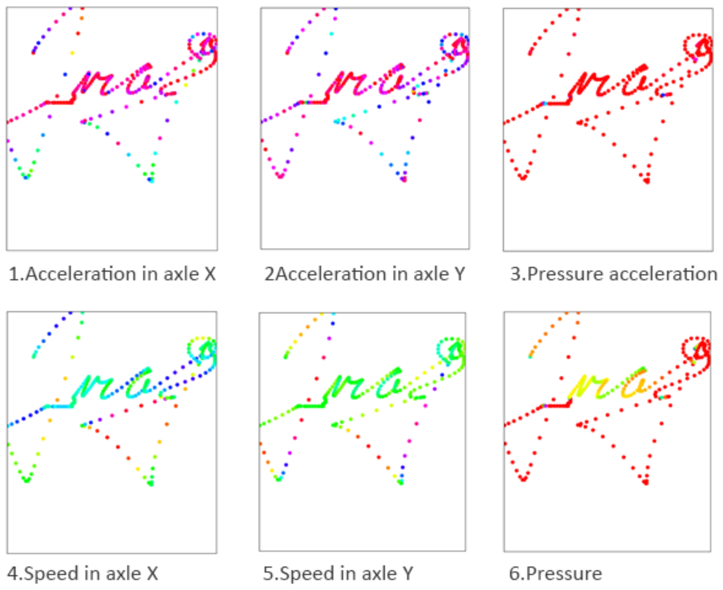


Figure 1.7: Dynamic features vizualization

- Total signing time duration
- Total writing time (time in which pen is in contact with signing area)
- Total lifting time (time in which pen is not in contact with signing area)
- Count of pen ups
- Count of pen downs
- Aggregate functions (max, min, average, median) of function features
- Fourier transformation of signature [12]
- Directory based features (in which direction was signature written (left $\rightarrow$ right, right $\rightarrow$ left)
- Geometrical-based features (occurrence of geometric shapes in signature)
- Curvature based features (count, shape, radius of curves in signature)
- Intensity of ink
- Pattern spectrum (spectrum of patterns found in signature)
- ...

Some of parameter features are related to signature dynamic such as total signing time duration, numbers of pen ups and down. Some parameter are obtained by aggregating feature function i.e. maximum speed, minimum acceleration in x direction. Some parameters are determinate as coefficient obtained from mathematical tools as Fourier, Hadamard, Cosine, wavelet, fractal transformation.

1.5 Signature verification method

Signature verification can be done manually by person or (semi)automatically by computer. Manual signature verification is called forensic signature examination. Forensic signature examination is used in cases of tax fraud, suicide notes, stalking letters, validating authenticity of autograph and so on. Forensic scientist consider: traces of practicing, guild lines, order of movement, line quantity, complexity, ease of simulation and more. Computer examination is also widely used method of forensic signature examination.

Computer signature verification can be dived to two categories: Off-line (static) signature verification, On-line (dynamic) signature verification.

Off-line signature verification methods are used on static signature. This methods used mostly parameter feature of static signature. This method is used by examination physical (written on paper) signatures.

On-line signatures verifications are used on dynamic signatures. This methods used mostly function features but can also use with some parametric features. On-line signatures required signer to have specific hardware or signer muse be physically present during signature extraction (by specific hardware)

Signature verification sustain of matching examined signature features to signatures features stored in knowledge base. As result of signature verification is statement: signature is or is not valid. Signature verification is much investigated problem, so there is many solution to it. We describe most common one. Additionally we provide list of other found solution with reference literature.1.9

1.5.1 Hidden Markov model

In the field of pattern recognition, model–based similarities were shown to be powerful tools to measure similarity of vector sets: computing distance consist of two steps: mapping each vector set to a probability distribution and computing probabilistic similarity. For signature features approach is very similar. In classic article authors introduce HMM with definition.

"An HMM is a doubly stochastic process with an underlying stochastic process that is not observable (it is hidden), but can only be observed though another set of stochastic processes that produce a sequence of observed symbols." [10]

In this paper are described principles behind HMM, training and scoring. In HMM consider each signature pattern in knowledge base as independent entity and base on this entity generate a lure set. Verified signature features are classified against stored signature patterns using those rules. HMM have advantage in signature verification because of its ability to model an unknown sample as one of the existing samples.

HMMs are naturally suited for modelling “flowing” entities such as speech and handwriting. They are made up of a series of states with transitions between these states. Signatures can be split up into strokes, with a similar number of sections to the number of states in the HMM. It is then possible to progress through the states of the HMM in a corresponding fashion to processing through the strokes of the signatures. At each state, local features of signature stroke are examined, therefore local features can be naturally modeled by HMMs. This approach with reasonably large training data set provide good results [9].

1.5.2 Support vector machines

Support vector machine with a set of examples from two classes (in signature authentication case valid and invalid class) finds the hyper plane, which maximizes the distance from either class to the hyper plane and separates the largest possible number of points belonging to the same class. By this approach the misclassification is minimalized. SVMs in their basic form use linear threshold function. SVMs measure the complexity of hypotheses according to the merging, which separates the signatures. SVMs can be apply on signatures with many features [11].

1.5.3 Neural networks

Neural networks are widely used in signature verification. All kinds on neural networks are used: recurrent neural network, kohonen self-organizing network, feedforward neural network and more. Neural networks must be trained (with or without supervisor) to verify signatures. [13]

1.5.4 Dynamic Time Warp

Dynamic Time Warping is used to compute a distance between two time series. Naïve approach to calculating a matching distance between two series could be to resample one of them and then compare it sample by sample. DTW provide recovering of optimal alignment between sample points in two time series. The name Time Warping is derived from warping time axes of the two time series in way that corresponding samples appear at the same location as shown in 1.8 [14].

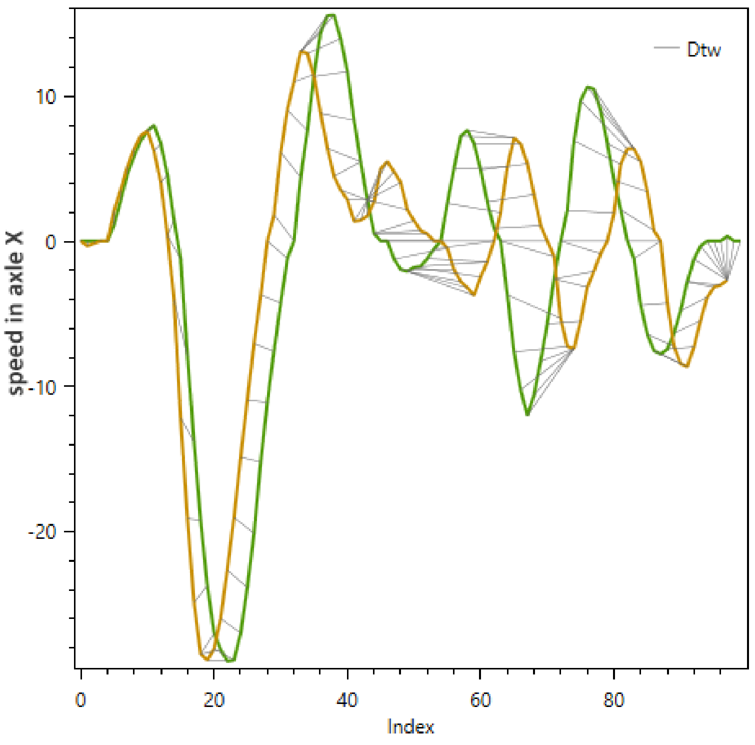
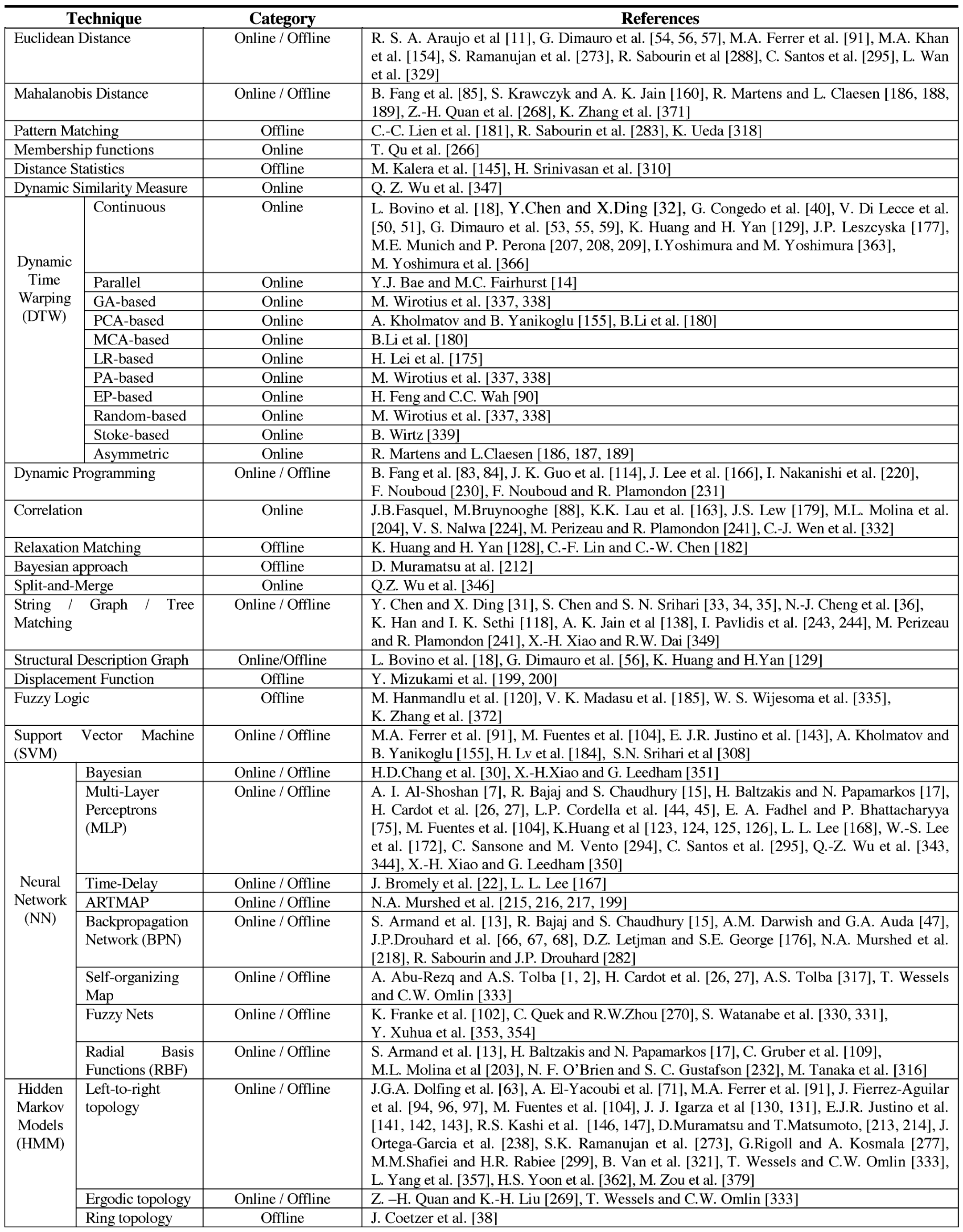


Figure 1.8: Visualization of DTW alignment. Green and orange lines are plotted two time series of signature speed in axle X. With grey lines are connected matching points.



Technique		Category	References
Euclidean Distance		Online / Offline	R. S. A. Araujo et al [11], G. Dimauro et al. [54, 56, 57], M.A. Ferrer et al. [91], M.A. Khan et al. [154], S. Ramanujan et al. [273], R. Sabourin et al [288], C. Santos et al. [295], L. Wan et al. [329]
Mahalanobis Distance		Online / Offline	B. Fang et al. [85], S. Krawczyk and A. K. Jain [160], R. Martens and L. Claesen [186, 188, 189], Z.-H. Quan et al. [268], K. Zhang et al. [371]
Pattern Matching		Offline	C.-C. Lien et al. [181], R. Sabourin et al. [283], K. Ueda [318]
Membership functions		Online	T. Qu et al. [266]
Distance Statistics		Offline	M. Kalera et al. [145], H. Srinivasan et al. [310]
Dynamic Similarity Measure		Online	Q. Z. Wu et al. [347]
Dynamic Time Warping (DTW)	Continuous	Online	L. Bovino et al. [18], Y.Chen and X.Ding [32], G. Congedo et al. [40], V. Di Lecce et al. [50, 51], G. Dimauro et al. [53, 55, 59], K. Huang and H. Yan [129], J.P. Leszcyska [177], M.E. Munich and P. Perona [207, 208, 209], I.Yoshimura and M. Yoshimura [363], M. Yoshimura et al. [366]
	Parallel	Online	Y.J. Bae and M.C. Fairhurst [14]
	GA-based	Online	M. Wirotius et al. [337, 338]
	PCA-based	Online	A. Kholmatov and B. Yanikoglu [155], B.Li et al. [180]
	MCA-based	Online	B.Li et al. [180]
	LR-based	Online	H. Lei et al. [175]
	PA-based	Online	M. Wirotius et al. [337, 338]
	EP-based	Online	H. Feng and C.C. Wah [90]
	Random-based	Online	M. Wirotius et al. [337, 338]
	Stoke-based	Online	B. Wirtz [339]
	Asymmetric	Online	R. Martens and L.Claesen [186, 187, 189]
Dynamic Programming		Online / Offline	B. Fang et al. [83, 84], J. K. Guo et al. [114], J. Lee et al. [166], I. Nakanishi et al. [220], F. Nouboud [230], F. Nouboud and R. Plamondon [231]
Correlation		Online	J.B.Fasquel, M.Bruynooghe [88], K.K. Lau et al. [163], J.S. Lew [179], M.L. Molina et al. [204], V. S. Nalwa [224], M. Perizeau and R. Plamondon [241], C.-J. Wen et al. [332]
Relaxation Matching		Offline	K. Huang and H. Yan [128], C.-F. Lin and C.-W. Chen [182]
Bayesian approach		Offline	D. Muramatsu at al. [212]
Split-and-Merge		Online	Q.Z. Wu et al. [346]
String / Graph / Tree Matching		Online / Offline	Y. Chen and X. Ding [31], S. Chen and S. N. Srihari [33, 34, 35], N.-J. Cheng et al. [36], K. Han and I. K. Sethi [118], A. K. Jain et al [138], I. Pavlidis et al. [243, 244], M. Perizeau and R. Plamondon [241], X.-H. Xiao and R.W. Dai [349]
Structural Description Graph		Online/Offline	L. Bovino et al. [18], G. Dimauro et al. [56], K. Huang and H.Yan [129]
Displacement Function		Offline	Y. Mizukami et al. [199, 200]
Fuzzy Logic		Offline	M. Hanmandlu et al. [120], V. K. Madasu et al. [185], W. S. Wijesoma et al. [335], K. Zhang et al. [372]
Support Vector Machine (SVM)		Online / Offline	M.A. Ferrer et al. [91], M. Fuentes et al. [104], E. J.R. Justino et al. [143], A. Kholmatov and B. Yanikoglu [155], H. Lv et al. [184], S.N. Srihari et al [308]
Neural Network (NN)	Bayesian	Online / Offline	H.D.Chang et al. [30], X.-H.Xiao and G. Leedham [351]
	Multi-Layer Perceptrons (MLP)	Online / Offline	A. I. Al-Shoshan [7], R. Bajaj and S. Chaudhury [15], H. Baltzakis and N. Papamarkos [17], H. Cardot et al. [26, 27], L.P. Cordella et al. [44, 45], E. A. Fadhel and P. Bhattacharyya [75], M. Fuentes et al. [104], K.Huang et al [123, 124, 125, 126], L. L. Lee [168], W.-S. Lee et al. [172], C. Sansone and M. Vento [294], C. Santos et al. [295], Q.-Z. Wu et al. [343, 344], X.-H. Xiao and G. Leedham [350]
	Time-Delay	Online / Offline	J. Bromely et al. [22], L. L. Lee [167]
	ARTMAP	Online / Offline	N.A. Murshed et al. [215, 216, 217, 199]
	Backpropagation Network (BPN)	Online / Offline	S. Armand et al. [13], R. Bajaj and S. Chaudhury [15], A.M. Darwish and G.A. Auda [47], J.P.Drouhard et al. [66, 67, 68], D.Z. Letjman and S.E. George [176], N.A. Murshed et al. [218], R. Sabourin and J.P. Drouhard [282]
	Self-organizing Map	Online / Offline	A. Abu-Rezq and A.S. Tolba [1, 2], H. Cardot et al. [26, 27], A.S. Tolba [317], T. Wessels and C.W. Omlin [333]
	Fuzzy Nets	Online / Offline	K. Franke et al. [102], C. Quek and R.W.Zhou [270], S. Watanabe et al. [330, 331], Y. Xuhua et al. [353, 354]
	Radial Basis Functions (RBF)	Online / Offline	S. Armand et al. [13], H. Baltzakis and N. Papamarkos [17], C. Gruber et al. [109], M.L. Molina et al [203], N. F. O'Brien and S. C. Gustafson [232], M. Tanaka et al. [316]
Hidden Markov Models (HMM)	Left-to-right topology	Online / Offline	J.G.A. Dolfing et al. [63], A. El-Yacoubi et al. [71], M.A. Ferrer et al. [91], J. Fierrez-Aguilar et al. [94, 96, 97], M. Fuentes et al. [104], J. J. Igarza et al [130, 131], E.J.R. Justino et al. [141, 142, 143], R.S. Kashi et al. [146, 147], D.Muramatsu and T.Matsumoto, [213, 214], J. Ortega-Garcia et al. [238], S.K. Ramanujan et al. [273], G.Rigoll and A. Kosmala [277], M.M.Shafiei and H.R. Rabiee [299], B. Van et al. [321], T. Wessels and C.W. Omlin [333], L. Yang et al. [357], H.S. Yoon et al. [362], M. Zou et al. [379]
	Ergodic topology	Online / Offline	Z. –H. Quan and K.-H. Liu [269], T. Wessels and C.W. Omlin [333]
	Ring topology	Offline	J. Coetzer et al. [38]

Figure 1.9: Most common signature authentication methods with technique used for signature authentication , category of method and reference literature [15]

1.6 Signature forgery

We can see evidence in history, that signature forgery has been formed shortly after signature acceptance as verification. Only small percentage of signatures in existence are forgery. Unsurprisingly signature forgery are generally part of important documents or of paper carrying a monetary value. This is reason why signature verification and forgery detection are so important.

Handwritten signatures can be forged in many ways, there are three most common:

- Forging by copying
- Forging by mimicking
- Free hand forgery

Forging by copying can be archive through many methods. One method involving usage of windows to trans illuminates paper having valid signature. On top of this paper, we place destination paper of forgery signature. We adjust the position and outline genuine signature. Other method use grooves on writing surface to obtain valid signature. These methods of forging is reason why cheque-leafes are made of thick paper. These methods can be considered as "static forging "because it use similar way as static signature retrieval [17].

To perform forging by mimicking, forger must have sample of signature writing as an act i.e. forger must be whiteness of signature writing or have video material capturing it. With this knowledge, forger try to mimic signature writing with all dynamic aspect of it. This methods can be consider as "dynamic forging" for its similarity of dynamic signature retrieval.

Forger, who is familiar whit a another signature can simulate a free hand forgery. This type of forgery represents a careful and painstaking drawing of the signature of another person without examples of any kind. This method have interesting drawback. If the forger is familiar with the language of the signature, there is bigger chance that forged signature have more legible manner that the genuine signature.

There is false common awareness that more complex signature is, then is more difficult to be forged. Signature of people that often sign are simplified and signing is done almost in unconscious way. They dont bother about the spelling, later designs and beauty of signature. By this, signatures are becoming more personal and gain behavioral biometric trait [17].

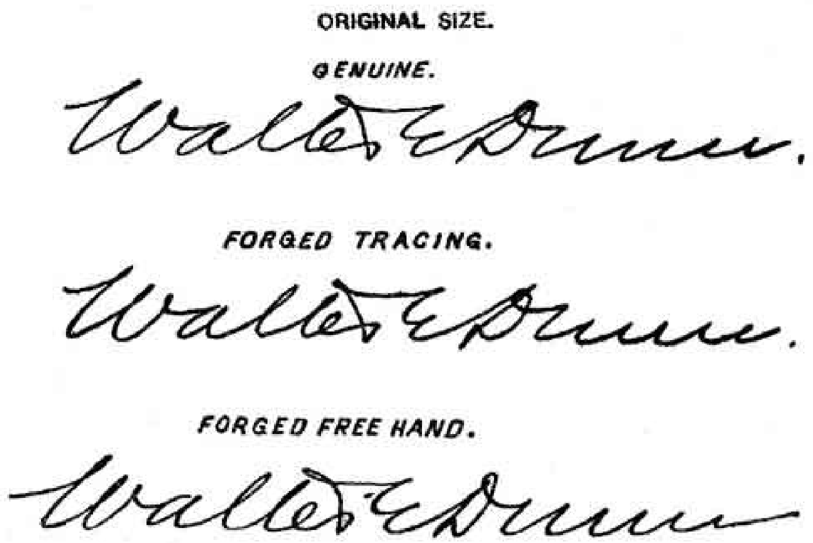


Figure 1.10: Example of forged signatures

2. Windows credential provider overview

We have talked about signatures and signatures verification but it's equally important that we are able to perform log in to windows by user signature. . Our target version of operating system will be Windows 8.1. Supported operating system will be Windows Vista up to Windows 10. We are not taking into account Windows XP. While Windows XP was the first Microsoft OS to support handwritten input, it used different logon architecture. In the next chapters, we will discuss Windows XP logon architecture to see how much it is different from modern Windows logon architecture and then talk about modern logon architecture.

2.1 Windows XP

Windows XP uses credentials provider called GINA. GINA is abbreviation for Graphical Identification and Authentication dynamic-link library (DLL). GINA is replaceable DLL component that is loaded by WinLogon executable. GINA implements the authentication policy and performs all identification and authentication user interaction.

What was the reason to abandon this architecture? In pre-Vista environment, every session had an instance of WinLogon, which is responsible for driving the interactive logon sequence for that session. On newly booted systems, an interactive logon at the console level is always performed in session zero. Session zero hosted system services and other crucial processes, including local security authority process.

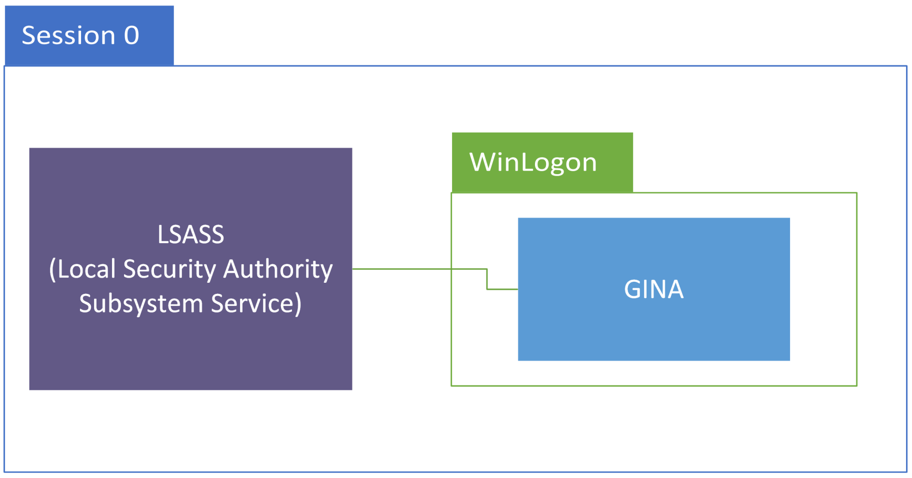


Figure 2.1: Windowx XP login architecture

After interactive logon is loaded, registered GINA starts loading in the WinLogon process space. (This is not GINA chaining, this is regular Multiple GINA loading routine). After GINA is loaded, it makes calls to Logon user and related authentication APIs. GINA is also responsible for rendering whole login GUI. This is the reason, why we can install 3rd part visual enhancement of login screen in Windows XP and not in more recent systems. [18].

2.1.1 Reason to give up GINA

As an answer for a high-demanded multi-factor authentication and complex GUI GINA drop support. Demand for multi-factor authentication places greater burden on the abstraction layer between core windows credentialing and engine, and more complex GUI GINA drop support. In addition, windows logon process (winlogon.exe) has been completely re-architected in Windows Vista.

The core requirement was to move software out of the WinLogon process space. That requirement was motivated by reliability. If, for instance, a defected or poorly written GINA is loaded into the WinLogon in session zero, a software failure could kill some critical process, even cause BSOD (Blue screen of death). Even if GINA could have been adopted to run out of process, there would still be the issue in design that hasn't provided a consistent controlled experience across arbitrary complex or interactive credentials gathering scenarios.

2.2 Windows Vista, Windows 7

Difficult GINA chaining and one point of failure was limitation that Credential Providers were designed to address. Design point was oblivious:

- Make it easier for multiple logon providers to co-exist on the same OS installation without conflict
- Provide robustness for Credential provider's failure. So if one credential provider fails that doesn't drag the whole system.

Credential providers pay for its consistency. Credential providers are only supposed to be used for gathering credentials and passing then on. Credential providers also are not fully responsible for login GUI. Credential provider architecture requires each provider to enumerate its UI elements. Logon render given UI elements as controls on behalf of the credential provide. This helps to achieve consistent look and login experience.

Another design chart for credential providers was to move to COM based plug-in model. However in early build of Windows Vista the first internal design for the new interface was based (Like GINA) purely on Load Library and function pointers. Luckily this was redesigned to COM-based and resulted to cleaner interface that is easier to use.

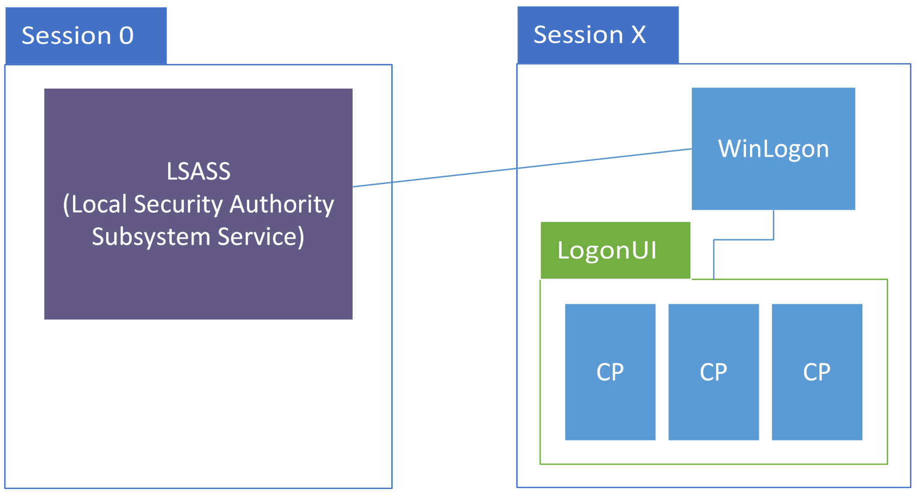


Figure 2.2: Windowx Vista and above login architecture

2.3 New Logon Architecture

In Windows Vista, session zero is never used for interactive logon. This is good for security reasons, because there is security boundary that separates all per–machine processes from pre- user processes. Also the kernel Global namespace is now more tightly controlled, since objects are created by user application are kept out of it by design. There's still an instance of WinLogon in every session other than session zero, but all credential providers are loaded by new Logon UI process.

2.4 Credential provider design

First we briefly look at the design of basic credential provider. For this design overview we will use one full lifecycle of credential provider: started with its loading (shown in 2.4), user interaction and finally user log in (shown in 2.5). With basic knowledge how credential providers work, we will look on our custom credential provider. On our custom credential we will focus on modification that allow user log in to the system by his signature.

Credential provider must exposes the two COM interface: **ICredential-ProviderCredential**. And **ICredentialProvide**

- **ICredentialProviderCredential** (for short Credential, in calls prefix Credential::) defines the behavior of a credential tile. It's responsible for the way the credentials is responding to user input.
- **ICredentialProvider** (for short Provider, in calls prefix Provider::) defines the behavior of the credential provider, which typically manages one or more credentials. It's responsible for the way credentials are enumerated and handle background calls and credential layout.

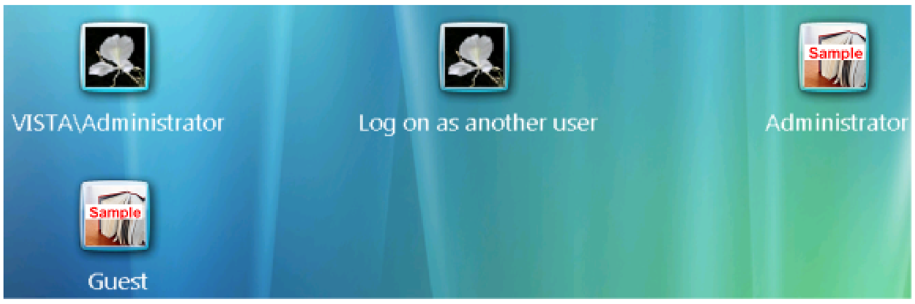


Figure 2.3: Example of login screen with one provider have multiple credentials

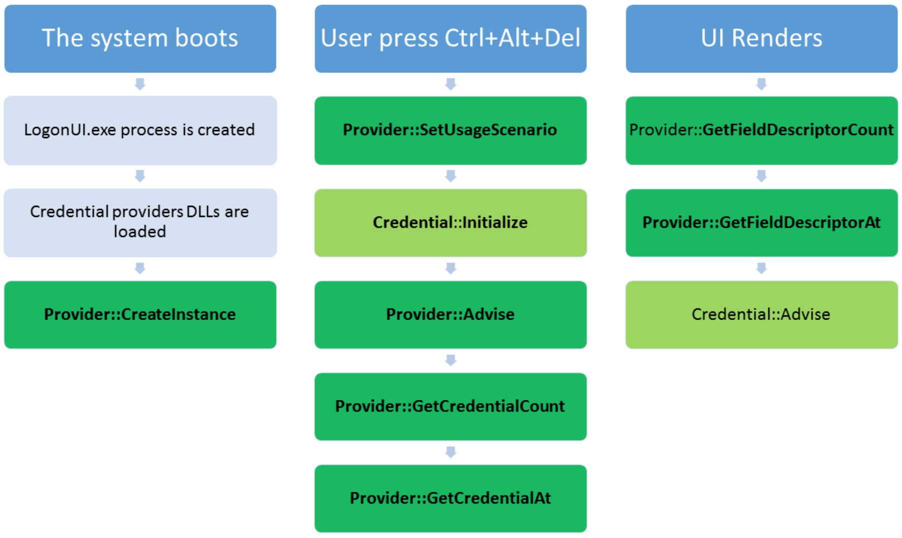

The system boots
LogonUI.exe process is created
Credential providers DLLs are loaded
Provider::CreateInstance
User press Ctrl+Alt+Del
Provider::SetUsageScenario
Credential::Initialize
Provider::Advise
Provider::GetCredentialCount
Provider::GetCredentialAt
UI Renders
Provider::GetFieldDescriptorCount
Provider::GetFieldDescriptorAt
Credential::Advise

Figure 2.4: Schema of credential loading

2.4.1 Loading

The system boots

After WinLogon is loaded, process LogonUI is initialized in console session. During initialization, LogonUI enumerates all registered credential provider that are registered. Credential provider registration are stored in windows registry under HKLM\Software\Microsoft\Windows\CurrentVersion\Authentication\Credential Providers by its GUID.

Each provider DLL is loaded and it receives Provider::CreateInstance call. At this moment, user sees the logon screen.

Ctrl+Alt+Del screen

Then **Provider::SetUsageScenario** notification is received. If machine is part of domain, CTRL+ALT+DELETE screen is shown and this notification is send after well-known combo is pressed. This notification tell credential that user want to perform logon. Only now is called **Credential::Initialize** for credential initialization.

Logon UI then calls **Provider::Advise** on each loaded provider. Purpose of advise call is to give the providers mechanism to notify Logon UI asynchronously for UI changes.

Next Logon UI call **Provider::GetCredentialCount**.By output parameters of this function provider tell LogonUI, how much credentials it contains, what is the index of default credential or it contain no default credential and if default credential have auto login capability.

UI Renders

Next Logon UI calls **Provider::GetFieldDescriptorCount**. In return parameter provider returns number of UI elements that will be rendered.

Logon UI calls then **Provider::GetFieldDescriptorAt** for each UI element to determinate its type. For example password box return `CPFT_PASSWORD_TEX` and for this element then Logon UI call **Credential::GetStringValue** to get its value.

After all element tell its type, Logon UI calls **Credential::Advise**. That's other advise call, target of first one was Provider, but purpose is similar, to have opportunity to asynchronously notify LogonUI to change state of UI element. This is used for example after long time of inactivity with password box filled, then **ICredentialProviderCredentialEvents SetFieldString** is called on password box and text is cleared.

2.4.2 Login

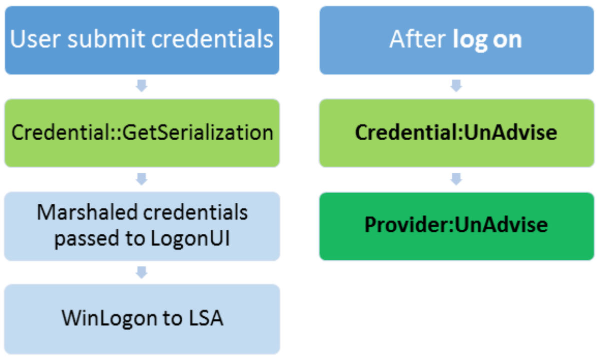


Figure 2.5: Schema of credential logi in

After user input credential (or credential is gathered different) the submit button is pressed and in this case, **Credential::GetSerialization** is called. Credential prepares the return value for that routine by marshaling "user name" "password" and "domain name" in format expected by Kerberos. Credential informs LogonUI that credentials is prepared by calling

`CREDENTIAL_PROVIDER_GET_SERIALIZATION_RESPONSE` output parameter in which compete credential is returned.

After GetSerialization, Logon UI passed credential info to winlogon. Winlogon passes it to the Local Security Authority (LSA) by calling Logon User. Based on this, Logon UI Calls **Credential:UnAdvise and Provider:UnAdvise** to notify both entities. After winlogon gets the result of LogonUser, result its passed back to Logon UI and then passed again to focused credential instances (focused credential instances is the same that have focus from GetSerialization). Before credential receive status code of login in, callback for UI element changes is passed.

Login result is returned to credential via **Credential::ReportResult** routine. Another calls are made for UI changes based on ReportResult. One case of this scenario is if user password expires and new one is entered.

2.4.3 Default credential

It's possible, and it's common, that multiple providers will enumerate its default credential. How does LogonUI allow user to select from multiple defaults and then non-default credentials? In general for each credential a tile is shown, with focus set to the default one. In case of multiple defaults, the true default is selected based on rules including auto-logon and LLO (last logged-on provider)

2.5 Windows 8

First of all, windows 7 credential provider, unlike windows XP GINA, is forward compatible with windows 8/8.1. Unfortunately not all new features that bring windows 8 can be achieve via old credential provider, but we will discuss that later in this paper. Changes with Microsoft Windows 8 vs Windows 7 are mostly in UI based on changes in user experience while login. Windows 8 is designed to be extremely personalized to each user of PC. New user-centric philosophy is present in Windows 8 Logon UI. Instead of displaying more abstract concept of authentication method, actual users are at the center of Logon UI in windows 8 [19].

2.5.1 Old model

In windows 7 or Windows vista, first thing that user must do in login process is choose authentication method first and foremost. For example if user is part of domain, have fingerprint reader on his/her LogonUI will shows many entities:

- One that showed user tile and user name (basic password provider)
- One that showed Finger print reader tile and user name (finger print provider)

For user, this can be confusing, user is one person, why there are two thing with user name on it?

2.5.2 New model

In windows 8. User tiles are shown instead of multiple methods to authenticate single user in Deselected screen (screen where credential tiles are shown). This user tiles are generated by Windows and is rendered for any user, which have associated credential provider with his account. Credential provider that tells windows which users are associated with it are referred as v2 credential provider. Credential provider which don't provide this information is referred as v1. If there is any v1 credential provider in system, it will show on deselect screen alongside user tiles

When user click on v1 credential provider, nothing new happen, it's shown

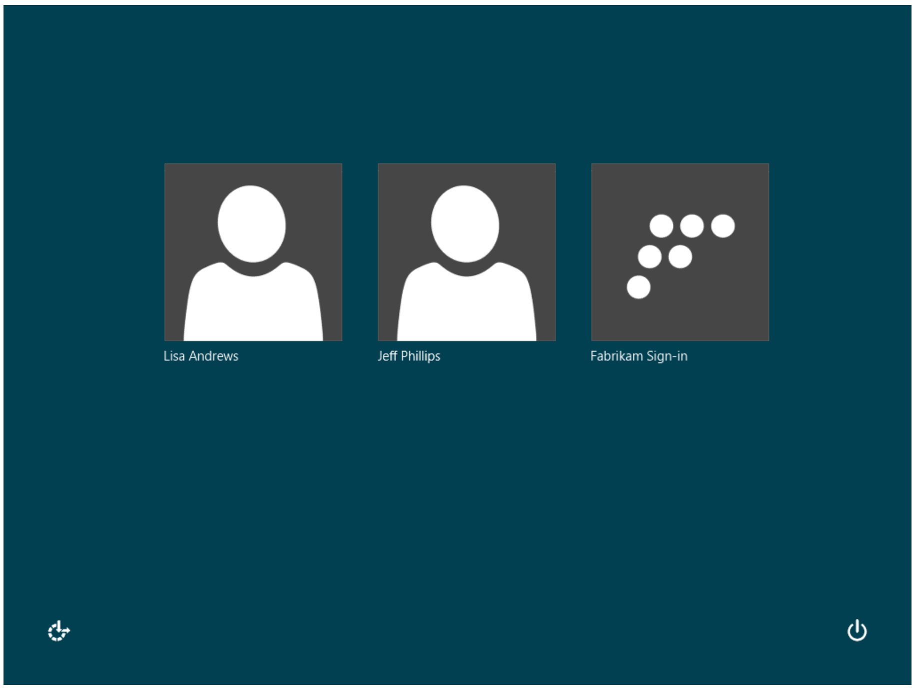


Figure 2.6: V1 Credential in Windows 8

the same interaction as we describe above, just title size changed

When user click on user tile, new UI is showed. Then along basic information things as back button (1), User image (2) and name, logon state and addition info (3). These elements are fully controlled by LogonUI. Credential provider selection is done by its small images under sign in options label (4) and its render place is above that's label. Fields that credential provider requests to drawn will appear between user name area and this label. This UI element are the same one that are used in v1 credential provider.

2.5.3 Inside changes

As it looks like, differences between v2 credential providers and v1 are mostly in per-user philosophy and from that divided graphical modification. Small changes have been made inside. V2 credential provider must implement new interface **ICredentialProviderCredential2** and return valid SID from function **GetUserSID**. GetUserSID tells Windows which user are oblique for specific credential provider. SID (Security identifier) is unique, immutable identifier of user,

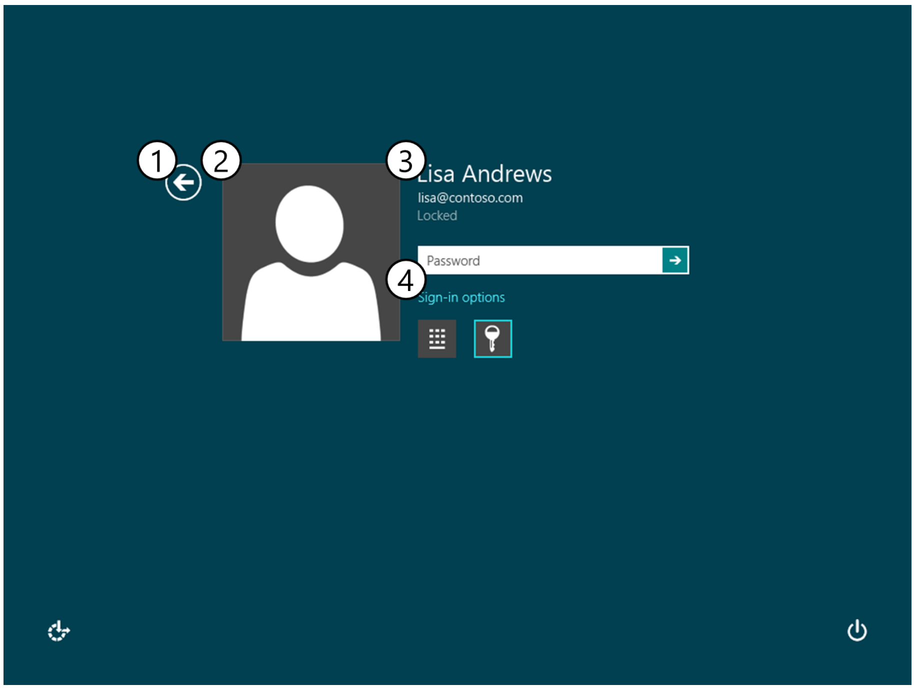


Figure 2.7: V2 Credential in Windows 8

user group or other security principal in windows. SIDs are gathered by enumerating all users by function **GetUserSID**. When user is oblique, it return its SID, when not it returns false. This allow credential provider walk through all of users and determine if user is associate with it.

Modifications in calls responsible for GUI changes was also made. Call **Provider::CredentialsChanged** was used for UI changes in windows 7, but it has high cost in re-loading all credentials. Now **Provider::CredentialsChanged** should be used only for auto login feature or to change numbers of credentials. For UI changes instead there is **Provider::CredentialProviderCredentialEvents2**.

Another small change is associated with default provider. In V2 there is no thing as default provider, because that might confuse users, so there is now only last used provider which are shown by default [20].

3. Signature credential provider design

In this chapter, we discuss design decision that have been made during signature credential provider (signature provider in short) implementation. Signature credential provider design reflect in every aspect login scenario via this provider, so first of all we describe it. Logon screen is shown to user. User click to perform log in by handwritten signature. Some kind of signature form is displayed. User enter his or hers genuine signature and click to log in. After successful signature verification, user will be log in to the system. After unsuccessful signature verification visual feedback is provided.

Then we go step after step and we will ask a question after each step such as:

- How and where user enter signature?
- Will we use native credential provider implementation or use third party solution?
- Where signature verification happen, inside credential provider or somewhere else?
- How and where user credential will be stored and passed or can we log in without it?
- If we must pass user credential into our credential provider, will we pass it to our credential or provider and how we do it?
- Will we implement V1 or V2 credential provider?
- Can we use our credential provider for authentication different that just signature

3.1 Signature enter

How and where user enter signature? First of all we must ask, if we are able to capture user signature on logon screen by resources given by credential providers. As we told, for Login GUI is responsible LogonUI process. Logon UI renders elements that are given by provider. Set of possible UI element type is limited in many ways, it contains only basic types such as bitmap image, checkbox, textbox, text label, and clickable text label In UI set is no build in element type that can capture handwritten user input and set of UI element can't be extend by custom element. Due this limitation, we won't be able use only credential provider to achieve our goal. Solution of this problem is quite strait forward: From our custom credential provider launch external program, that will have capability to capture handwritten signature.

3.2 Native credential provider or third part solution

When deciding if we choose native credential provider, or third party solution, there are not many options. There is of course native credential provider and from third part solution there is pGina.

pGina is pluggable, open source credential provider. Plugins are written in managed code and allow for user authentication and authorization.

Pros of pGina is that plugins are written in .NET platform, so it's easy to debug.

It looks like, pGina is all we need, but there is a small problem. As we told, credential provider itself doesn't give us tools to capture handwritten signatures, so we must launch external program. There is problem, pGina restrict launching external programs from plugins for security reasons.

Gina is ideal for situation, where we are satisfied with given UI input types, and we will implement custom authentication background, which in our case, isn't enough.

Pros of Native credential provider are there is possibility to launch external program from itself, which is shown on logon screen and from its nature its can.

Cons on other hand are limited debugging possibilities.

Despite complicated debugging, our goal, let user sign in to system via handwritten signature, leaving us no other option that choose native credential provider.

3.3 Location of signature verification

When deciding about location of signature verification, there are two oblivious places. First two are oblivious:

- Inside signature credential provider
- Inside external program that capture handwritten signatures

Less oblivious solution, will be placing signature verification to addition program (service)

First option is, that verification will be inside credential provider. This approach have advantages that only fewer application are required (signature credential provider and sign form). One advantages is outweighed by many disadvantage: loading user pattern every time credential provider is initialized, very limited debugging capability and difficult signature transmission. Last disadvantage is in code duplication, since verification of signatures is using for preprocessing of signatures during saving user pattern. More of that in chapter 6.3.3 about credential manager.

Another option is, that verification will be inside sign form. This solution have same advantages as location inside provider: fewer application are required. This option lack disadvantages in problematic debugging, since sign form will be written in `C#`. Anyway loading user patter every time sign form is displayed and code duplication is still big disadvantage.

Last option trade advantages of few program for multiple pattern loading and code duplication. We will add authentication services. Since there must be communication between sign form and credential provider in both previous option, we just place authentication between them.

By this approach we achieve that user pattern will be loaded only once, during startup since this service will have auto start. This have cons in system usage, since authentication service will be running always, not only during authentication. Over and above that, we now can encapsulate whole signature verification and user pattern management in one place. This is important for signature management: new signature recording and signature re-recording. Debugging will be also a lot of easier, since we can debug each connection: Signature authentication Service ↔ Credential provider and Signature authentication service ↔ Sign form independently. There will be significant saving in data transfer to credential provider and type conversions. Between authentication services and sign form we will transfer complex data type (digitalized handwritten signature) to authentication. Between authentication service and credential provider is transferred only result of signature authentication.

We choose ls solution by addition authentication service, mostly for localization of signature processes in one place.Overall signature provider architecture is shown on 3.1.

3.4 User credential storage and pass

From previous talk about credential providers design, we know that for successful login our credential provider must pass triplet of credentials (User name, Password and optional domain name) to LogonUI. There is no option to do login without user credentials. Problem is, how this triplet will be store and how we pass it to credential provider?

For question about storage, there are two sub questions: Format and locations of credentials. For format there are basically two options: plaintext format and encrypted format.

Pros for plaintext is its simplicity, but cons are absolute lack of security. We chose encrypted credentials. It had Cons in little difficult usage, and we must

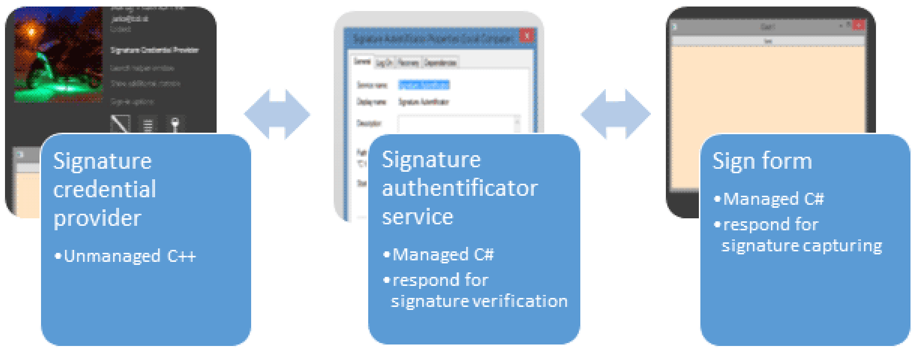

Signature credential provider
•Unmanaged C++
Signature authentificator service
•Managed C#
•respond for signature verification
Sign form
•Managed C#
•respond for signature capturing

Figure 3.1: Signature credential architecture

maintain key, but it have advantage in security.

For storage locations, we considered three options: store credentials in file, registry or by third part solution.

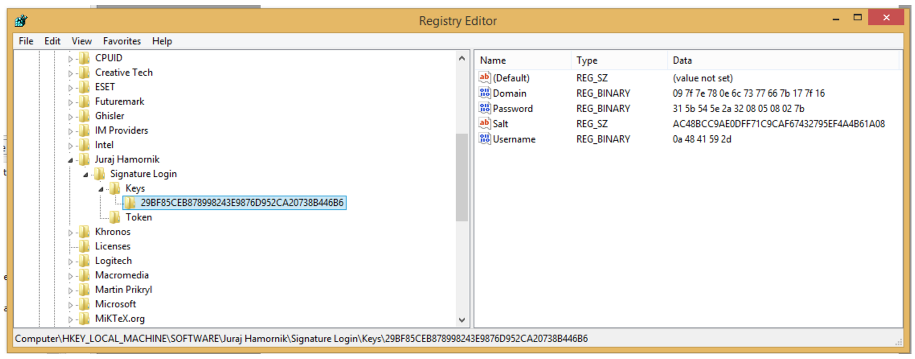
Third part solution have none pros while considering storing only few simple typed object, but have cons of increased number of programs needed to perform log in, system resources consumption, and potentially slow down boot time. Credentials must be availed during first log in after boot, so third part program must run before user login.

File storage share registry storage are basically the same. Both have Pros with nearly zero system cost while I/O and we can manage access right. Difference it that in file storage, we can manage access right only per file. In registry storage we can set access right per key.

We chose approach in storing credential in registry and in encrypted format. Now let's discuss how we pass them into the credential and how we store it in details.

After successful user verification, Signature authenticator service send the access token to credential. This access token is key to encrypted credentials that are stored in windows registry. User's credentials are stored under LocalMachine\SOFTWARE\Juraj Hamornik\Signature Login\Keys In this registry directory, all user credentials are stored. Name of folder that belongs to the user isn't his user name, but sha1 hash of combination of access token and "token salt". Token salt is in root of Keys directory.

Inside user folder, there are another salt, we called it user salt. User salt and access token hashed by sha1 is key for decrypting user name, password and domain name.



```
▷ CPUID
▷ Creative Tech
▷ ESET
▷ Futuremark
▷ Ghisler
▷ IM Providers
▷ Intel
◢ Juraj Hamornik
  ◢ Signature Login
    ◢ Keys
        29BF85CEB878998243E9876D952CA20738B446B6
      Token
▷ Khronos
  Licenses
▷ Logitech
▷ Macromedia
▷ Martin Prikryl
▷ Microsoft
▷ MiKTeX.org
```

Name	Type	Data
(Default)	REG_SZ	(value not set)
Domain	REG_BINARY	09 7f 7e 78 0e 6c 73 77 66 7b 17 7f 16
Password	REG_BINARY	31 5b 54 5e 2a 32 08 05 08 02 7b
Salt	REG_SZ	AC48BCC9AE0DFF71C9CAF67432795EF4A4B61A08
Username	REG_BINARY	0a 48 41 59 2d

Computer\HKEY_LOCAL_MACHINE\SOFTWARE\Juraj Hamornik\Signature Login\Keys\29BF85CEB878998243E9876D952CA20738B446B6

Figure 3.2: User credentials registry structure

Access right for credential folder is set as follows:

- Token salt are accessible to: Users and system. Every user must have access to salt to retrieve its user folder for credential management.[1]
- User folder are accessible to system and for corresponding user. Only user (and admin and system) have right to manage his/hers credential.[1]

3.5 Localization and implementation of credentials injections

Now we have all components to build our signature credential provider, last question remain: Will Credential or Provider communicate with Signature authenticator service, and then will be responsible for passing user credential and to perform login?

3.5.1 Credential

Looking at credential provider architecture, it might seems better to communicate with Credential, because it is first place, from which user credential are passed to provider and further. This solution was successfully implemented and work. However there was one major problem. From credential only, we were unable to perform submit without pressing submit button. This led for double submit to log in, one for signature and then from credential itself. We abandoned this solution despite easier implementation in prior for better user experience.

3.5.2 Provider

Our entry point for communication with Signature authenticator service will be Provider. Credential will be responsible only for user interaction with title and for launching Sign form.

[1]Administrator can obtain access right to all registry.

After Provider initialization, we launch named pipe watcher that will look on named pipe for incoming token. When token is received. Token is then verify by hashing it with salt and from this salt, we obtain name of user folder in registry. If token isn't right, calculated hash doesn't match user folder in registry. From user registry folder we encrypt password, user name and domain.

Here comes first problem, how we can trigger login from provider? There is no function that will emulate click on submit button. Only way credential provider can enforce login, is through doAutoLogin parameter during credential initiation. We can't just simple set auto Login to true by default, because without proper set of user credential provider will cause infinite loop of logins. This is caused because when credential provider fail to log in (in this case for wrong username/password), first credential that have doAutoLogin set to true is launched.

To prevent this, we must set doAutoLogin to true after we obtain user credentials via token. After doAutoLogin is set, we call LogonUI to reload credential. After credential is reloaded, because it have set doAutoLogin to true, login procedure is triggered and user will log in without click on submit button.

First think that comes to mind, what if user credentials that are stored in registry correct? Correctness of stored user credentials might be caused for example if password expired, so it isn't rare case. Then infinite loop of login start just after provider receive token. We think on this possibility. On provider, we will remember if last login fail or no. If last login fail, credentials are incorrect or other problem occurs, provider don't set doAutoLogin to true second time.

3.6 Credential provider for Window Vista/7 or Windows 8

Since Windows 8 provide backward compatibility for V1 credential provider, we just need to implement V1 provider that will work both on Windows Vista/7 and Windows8, This solution have one deficit which make us implement both V1 and V2 Credential provider.

First of all, back to Windows 8 account philosophy. In Windows 8 Microsoft introduce its Live account as replacement for basic user accounts. This Live account provider number of benefits as setting backup and cross device synchronization, access to Store and OneDrive. Unlucky, by V1 credential provider, we didn't find way to log in by Live account. We wasn't able to find any valid combination of domain and user. For importance of Lice account in Windows 8+, we decide to implement our signature credential provider also in V2 versions.

4. Signature credential provider implementation

In this chapter, we will discuss implementation of our signature credential provider. We focus particularly on credential provider itself and leave most of signature authentication services and form windows to the signature part of the paper. Since signature authentication service and form are more signature related than provider, we only describe communication between credential provider and authentication service. First we describe implementation based on our design decision. At the end we will discuss some of alternative usage of our credential provider. We will first describe implementation of V1 credential provider and then we describe implementation of V2, because lot of function used in V1 credential provider was reused. Reason of this order is because requirement for V2 credential provider (we were unable to log in via live account) was established after V1 credential provider was fully implemented and deployment on physical computer, not virtual machine.

For implementation of signature provider we used Windows Vista signature credential provider sample that can be obtained on .We use default credential provider as main frame of function name and parameters [22].

4.1 Language selection

First of all there is decision about programing language that will be used. Unmanaged C++ is used in Signature credential provider implementation, because it is required by Windows.

4.2 Implementation

Credential provider have simple task. Provide user to log in to system by user input. The simplified run of one login scenario will be:

- Signature credential provider load up
- Gathering of user input required to log in
- User input transformation to password
- Password pass
- In case of login error, show result

We will walk through on each step and discuss our modification on default credential provider. On function that are intact, we just describe its purpose. First of all, we provide some overview of class from witch are signature credential provider made of. Content of this class will be revealed during login scenario.

4.2.1 File overview

Registry.cpp

This file contain functions responsible for reading of windows registry. It contains function responsible for gathering registry values of type: DWORD, Bool, String and Char. Parameter for this function is always hkey: path to registry directory, strValueName name of registry inside hkey directory, output parameter in witch if returned value. For DWORD, BOOL and STRING is additional parameter (type) DefaultValue to check if registry value doesn't have default value.

Logger.cpp

This file was most important file during implementation and debugging. It contain function that provide at least basic debug capabilities in form of text log. It contain Log function that take pointer to const char as parameter. This const char is then alongside with timestamp append to the file specific inside class.

SHA1.cpp

This class implement NSA designed hashing algorithm. More information about this cryptographic hash function can be found in [23]

SHA1 hashing run

Sha1 run start with Sha1 initialization.Sha1 function must by first initialized by its parameters construction CSHA1 (). Then we need insert text and salt hash with function Update. Update have parameter unsigned char* t data and its length. Then we run hashing by calling parameters function Final. We gathered result of hashing by calling function ReportHashStl. ReportHashStl have two parameters, first is TCHAR* in which hash will be stored and then Enum of report type. For complete run of one sha1 we delete it at the end. We will substitute this run for abstract function later in this chapter for higher clarity.

Helpers.cpp

Since some exotic data types are used in credential provider this file contains function responsible for conversion from and to them.

Encryption.cpp

This file contain function responsible for decrypt: Decrypt. Decrypt take toDecrypt char* and char *password and decrypt it over toDecrypt parameter.

Guid.cpp

In this file is only reference to guid header file. In Guid.h is stored credential provider GUID which is used in registration.

Dll.cpp

In this file all function required for proper signaturecredential.dll loading to LogonUI are implemented. Signature Credential provider instance is created through factory pattern.

SignatureProvider.cpp

This is one of required part of credential provider com object. Function in this class is responsible for behavior of the credential provider, for the way credentials are enumerated, for receiving and handling token from signature authenticator service. It is responsible also for credential looks, by enumerating all its UI components.

SignatureCredential.cpp

This class is responsible of behaviors of user title. Since signature capturing is moved to external application, credential provider provide launching signature form and notification about login event.

4.2.2 Signature credential provider login scenario

When logon screen display is triggered, during system startup or when user trigger it, LogonUI load Signature credential provider.dll is loaded by GUI stored in guid.cpp.

Function that are called by LogonUI must have specific names strictly given to us by credential logon architecture. For better clarity, we will prefix functions called by LogonUI by **Log::**, functions called by LogonUI on the basis of user interaction **UI::**. Our functions will be without prefix.

4.2.3 Signature credential provider loading

First function that is executed after dll is loaded, is **Log::Provider::Create Instance**. This function as name describe create instance of Signature provider. Then **Log::Provider::SetUsageScenario** is called with paramether cpus (CredentialProviderUsageScenario). This parameter parameter contain information about logon usage. In this state, we should choose in which scenario our credential provider will be used. Possible scenario values are:

- `CPUS_LOGON` This usage scenario is for workstation logon.
- `CPUS_UNLOCK_WORKSTATION` This usage scenario is passed when user is when workstation is lock and logon screen is for unlocking it
- `CPUS_CREDUI` This usage scenario is passed when Signature Credential Provider is used for authentication on remote machines or over the shoulder prompting in User Account Control.

- `CPUS_CHANGE_PASSWORD` This usage scenario is passed in response to a user request to change their password (or other private information such as a PIN). We don't implement this scenario for security reasons. For password management we have dedicated app that must be run from user environments. In this case user will must use other credential to log in to the system.

After switching logon scenario, named pipe communication for token transport is initialized by function**InitSignatureReader**. **InitSignatureReader** first set signatureThreadRunning to true, then initialized hPipe2 by its name that is stored in lpszPipename2. Then new thread is created that contain function for reading that name Pipe **_NamedPipeReader**. We will return to description **NamedPipeReader** function later during token accepting.

Then initialization of credential begin via function **Log::Credential::Initialize**. During Credential initialization we set labels by function **SetFieldString** that will inform about credential name, logon status and logon use.

After credential initialization **Log::Provider::Advise(ICredentialProviderEvents* pcpe,UINT_PTR upAdviseContext)** is called. In this call we store Credential provide event reference and store upAdvise condex.

Then function **Log::GetCredentialCount (DWORD* pdwCount, DWORD* pdwDefault,BOOL* pbAutoLogonWithDefault)** is called.We have in credential flag named doAutoLogin that tell us if we are about to perform auto login. In Credential we have public flag lastLoginFailed that tell us if last login failed (this is for auto login infinite loop prevention). If we are about to doAutoLogin, last Login didn't failed and we have credential (_dwNumCreds is greater than zero) then we return count of credentials and set pbAutoLogonWithDefault to true.

Otherwise we pass pdwDefault with value `CREDENTIAL_PROVIDER_NO_DEFAULT` and pbAutoLogonWithDefault to false, since we are not going to perform auto Login now.

After then, user controls are rendered. Because we capture user signature not direct from credential, we don't modify function responsible for rendering logon UI element.

4.2.4 Gathering of user input required to log in

We will capture user signature by external program: sign form. But how we will launch it and when? It make sense launch it only after user click the signature Credential tile. Before that it doesn't make sense shown user sign form on shared credential screen. On the contrary after user click on user tittle there will muse be some additional user control for launching sign form. So right after user click title is the right time. Whole scenario is shown in 4.1

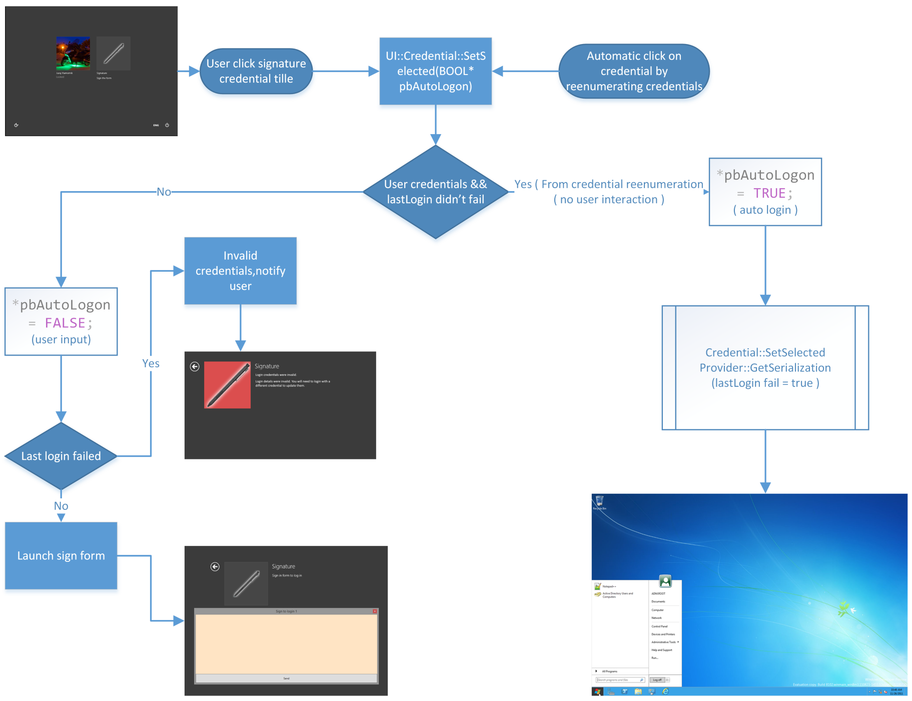

User click signature credential tille
UI::Credential::SetSelected(BOOL* pbAutoLogon)
Automatic click on credential by reenumerating credentials
User credentials && lastLogin didn't fail
No
Yes (From credential reenumeration (no user interaction)
*pbAutoLogon = TRUE; (auto login)
Invalid credentials,notify user
*pbAutoLogon = FALSE; (user input)
Yes
Credential::SetSelected Provider::GetSerialization (lastLogin fail = true)
Last login failed
No
Launch sign form
Signature

Figure 4.1: Part of login scenario

When user click live tile, function **UI::Credential::SetSelected(BOOL* pbAutoLogon)** is called. Note that in same function where auto Login is settled.

First we will try do auto Login, because this function might be called automatically during auto login. We check if there are defined (in other words correctly gathered and decrypted and stored) username, password and domain and if last auto login didn't fail .It so, we set pbAutoLogon to true and LogonUI perform auto login.

If we are not going to do auto login it's because two reasons: credentials are not ready or last login failed.

If last Login failed, we inform user by labels: "Login details were invalid. You will need to login with a different credential to update them."

If last login didn't failed, so this is the first time user interact with signature credential, we create sigh process by function CreateProcess. Sign form must be present in known direction.

4.2.5 User input transformation to password

When user enter handwritten signature to signForm, captured signatures in form of Stroke Collection is serialized to string and send through namedPipe to Signature Authorization service. When signature Authorization service authenticate signatures, it send token to Signature credential provider.Simplyfied scenario is shown on 4.2

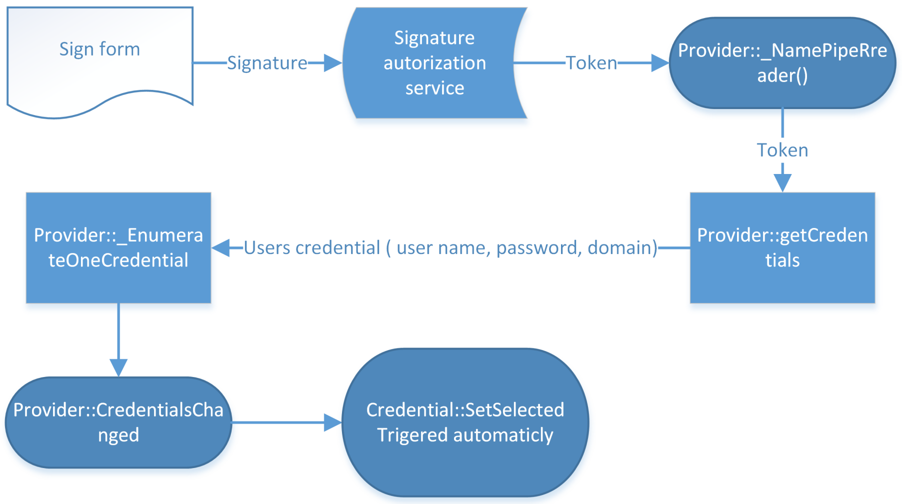

Sign form
Signature
Signature autorization service
Token
Provider::_NamePipeReader()
Token
Provider::getCredentials
Users credential (user name, password, domain)
Provider::_EnumerateOneCredential
Provider::CredentialsChanged
Credential::SetSelected Trigered automaticly

Figure 4.2: User input to login diagram

Now we are back in Sigture credntial proviver. When token is send, inside provider is received via function **_NamedPipeReader**(that's the function that we run in separate thread during provider setUsageScenario).

We receive token and we can move next to gathering credential information from this token. For this gathering we use function **getCredentials(token)**.**Get-**

Credentials first check if registry structure is correct by checking the Key directory. If Key directory is not found, we terminate this function, because none user have stored credential information or worst, registry was compromise.

If the registry structure is correct, we gather salt value and calculate name of user folder by given token. This calculation is done by sha1 hash of token and salt, more in chapter3.4.

If we can reach user folder by calculated name, we know token was right. From hashed token with user salt and this folder, we decrypt username, password and domain. On this place we can put easily pre-fill user credentials for test purpose. Now when we have user credential, we clean up support data structures that we used during hash calculation and decrypting.

We can tell (and set lastLoginFailed to false), that last Login didn't failed, because we receive token and this is done by user-interaction, not because auto login caused by infinite loop. Then we set doAutoLogin true ,pass user credential to function **_EnumerateOneCredential** with user credentials as parameters alongside with LogonUI required Field state and field descriptor. Then we trigger re-loading of credential by **Provider::CredentialsChanged (UpAdvseContext)**.

_**EnumereOneCredential** initialize new credential provider with user credentials. If Initialization succeeded, we replace current credential with new credential and increase number of credentials. This increase of numbers is due to remunerating purpose.

When **Provider::CredentialsChanged** is called, **Log::Provider::GetCredentialCount** is again re-run but in this run, doAutoLogin is true, last Login didn't failed and number of credential is greater than one: pdwDefault will be last credential (that one we reinitialized with user credential) and it have auto login capability. By this circumstances, on **Log::Credential::SetSelected** is triggered automatically 4.1. In **SetSelected** (we have properly set user credential via initialization) and pbAutoLogon is set to true.

Then **Log::Provider::GetSerialization** is triggered and user credentials that we gathered from are passed. From now, logon process take LogonUI. In **GetSerialization** function we also set lastLoginFailed true. In case login failed credential will be re-enumerated. Default credential will be our signature credential, this means that Signature tile will be selected automatically: function **Log::Credential::SetSelected** will be called.

In first condition, we check if all credentials are given (this is true, without them we won't be able to perform login) and if last Login failed. There we set doAutoLogin to false. Also we notify user about unsuccessful last login.

4.2.6 Windows 8 Signature credential differences.

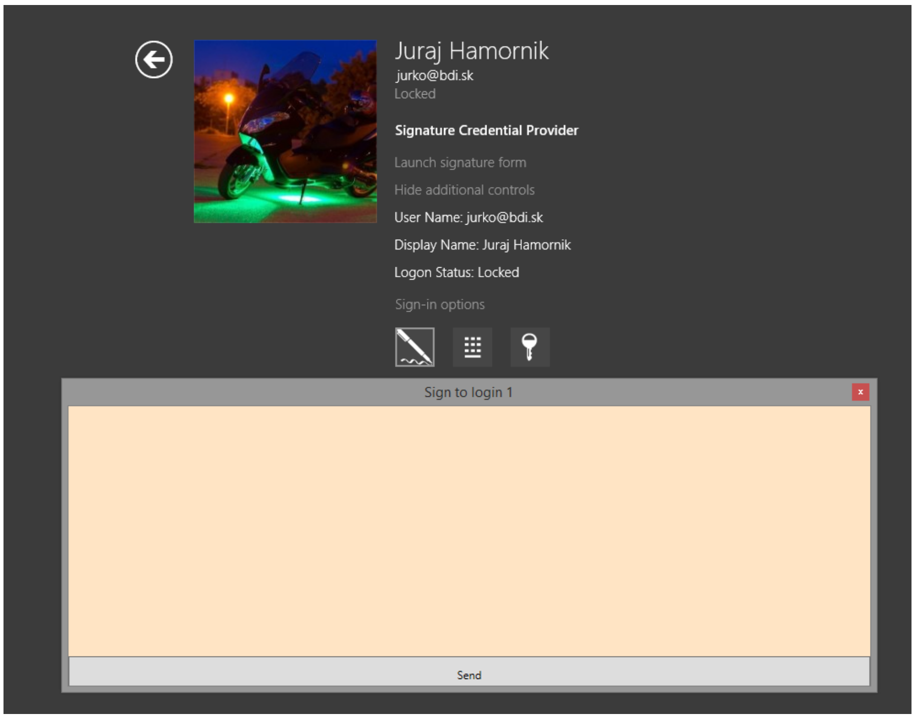

Juraj Hamornik
jurko@bdi.sk
Locked
Signature Credential Provider
Launch signature form
Hide additional controls
User Name: jurko@bdi.sk
Display Name: Juraj Hamornik
Logon Status: Locked
Sign-in options
Sign to login 1
Send

Figure 4.3: V2 Signature credential provider

V2 Signature credential provider have lot common with V1 Signature credential provider. As framework we use Windows 8 credential samples. Differences are caused by changed credential design to be more user oriented. Major differences, except UI 4.3 changes, are :

Implementation of required V2 function GetUserSid in Signature Credential. This function return UserSid if user are able to use signature credential provider. Without token, or list of users with registered signatures we can't tell if user is or is not able to perform login by signature, thus we allow all user.

We must change launching of sign form. Sign form are now launched when user click label "Launch signature form". This seems little contradictory: we spend nontrivial effort to achieve login by one button and now we added another user control. Sigh form can be launched automatically via Set Selected function (like in V1 credential provider), but by this approach, launch form will launch immediately after logon screen is triggered: it show after "pre login picture". Since this is very confusing, we choose launching sign form by user interaction.

Last modification was made in function _EnumerateOneCredentials(). For unknown reason (probably undocumented changes in credential provider V2 API) we were unable to recreate credential before we call CredentialsChaged function. This was bypass by temporarily storing token in registry. When credentials are remunerating, we create new signature credential with properly set auto login and password. Token stored in registry is immediately deleted.

4.2.7 Implementation notes

Debugging on the same machine is not possible because the credentials provider is running under LogonUI and our IDE (Visual studio) is running in user session which is inaccessible after login. Another problem with debugging on the development machine is that the credentials provider is very unstable during development and problems might arise such as Blue screen of death or inability to log in to the system, even after restart. Debugging over virtual machine is much safer, but we were unable to manage the connection between credentials provider running at virtual machine and debug program on host machine. This leaves us with only text debug output to file which, fortunately, might be stored on NAS and then accessed even if the virtual machine stops responding.

4.2.8 Alternative usage of signature credential provider

During implementation, there was only few place where we use signature itself. In fact only places was launching external application to capture user signature (sign form) and receiving token from signature authenticator services. So what we can achieve when we change sign form and authenticator services for different one? Here is some examples that was successfully tried:

USB authentification

First of all, what if user don't want to enter any information, but rather that use some kind of hardware token to authentication. There are many types of hardware that can serve as token: nfc tag, RFID card and more. For lack of this special hardware, we will for test purpose use basic usb stick. On that usb stick will be file that will have access token in in.

When user click on tile that belong to our credential provider. External program will launch. This external program check if there is any usb stick on computer, and if so check if any of them contain acess-token file. Then content of this file send to authenticator services that will only resend it to credential. Therefrom login procedure continue all the same as in signature credential provider.

There is lot of space to improve. Such as token encrypting, using usb id as token rather than file but this was only proof of concept of credential provider reusability.

Two factor authentification

Two factor authentication if based on the premise that an unauthorized action is unlikely to be able to supply both factors required for access. In this example, external program will capture and transfer to authentication services all user credentials (name, password, optional domain name) alongside with additional access token. We tried pre-generated and dynamically generated one-time tokens.

First we tried pre-generate one-time token. User received ten one-time 8 digit number. During each login, first unused number is entered.

Second we tried HOTP algorithm to generate one-time password. HOTP is an HMAC-based One Time Password algorithm. More information about HTOP can be found here [24]

User credentials are entered to external application for follow reasons: Password that we use in two factor authentication can and should be different from system password and for avoiding changes on credential provider.

After user enter credentials and access token, this information is passed to authentication service. If entered information is valid, send provider access token to credential provider. Therefrom login procedure continue all the same as in signature credential provider.

Mobile authentification

Last tried example is using device that everybody carry all the time: Mobile phone. We will use presence of mobile phone on network as kind of access token. Then user click on our credential tile, external application is launched. This external application only trigger authentication service to do start authentication process.

Authentication services have stored tuples of values: mac address of device; access token. During authentication service find all device on network. If exactly one device is on the network and have mac address stored in service, authentication send token that belongs to that device mac address. Therefrom login procedure continue all the same as in signature credential provider.

5. Signature authentication design

In this chapter, we will discuss design decisions that have been made during signature authentication implementation. In chapter Signature credential design some design decision about signature authentication have been already made. User signature will be captured by standalone program called Sign Form and signature authentication will happen in Signature authentication services. Since signature credential provider doesn't use handwritten signature as is, there are many design question about handwrite signatures that are left unanswered:

- How and what data we will capturing from handwritten signature?
- What signature feature(s) will we use?
- Can we get aces token from signature or we must store user-related data to perform signature authentication?
- How we use signature features to authentication?
- If we use many signature features, will it be equal or weighted?
- If we use weighted signature features, how will we set weight?
- How we determinate if signature is valid or is not?

5.1 Capturing of handwritten signature

Before we can do anything with signatures, first of all we must obtain signature from user. In chapter Signatures retrival 1.3 we describe some methods for retrieving user signatures. Since device of capturing signature is built in to the tablet itself, we will discuss method of capturing handwritten signature through it. We can use third party solution or solution build in Windows (.net)

First option is to use API solution of digitalize manufacturer. In case of testing device, as well as most devices on market, Wacom. Solution from Wacom is called Wintab [25]. Wintab API gave us access to all information that are available from stylus in form of strokes: time ordered array of stylus points. Wintab stylus point contain information of X and Y coordinate, strength which user pushed stylus to tablet surface (pressure) and index of stylus point in stroke. On supported devices it can provide additional information such as azimuth: clockwise rotation of stylus about the z-axis and altitude: angle upward toward the positive z-axis. Cons of this solution is that it provide all available information. On the other hand big disadvantage is, that Wintab driver must be installed for Wintab usage. Wintab drivers are available for both major digitalizer manufacturesr: Wacom and N-trig. Problem with Wintab driver is in stability and availability. On our test device (Microsoft Surface pro) WinTab drivers cause stylus to work only on 1cmx1cm area in top left corner (this is known issue, only solution is restart).

There are also problem with Wintab drivers availability, since drivers are often released months after device release [25].

Second option is universal and provided by Windows right after installation. .Net have UI element named InkCanvas that is capable to capture handwritten user input. Result of user input to InkCanvas is StrokeColection (collection of stokes). Each stroke is basically list of stylus points. Stylus points like in WinTab contain information about X and Y coordinates, pressure and index of stylus point in stroke. Pros of InkCanvas solution is that is available on Windows without installing custom drivers. On windows 8+ it's available by default, on Windows Vista and Windows 7 user must install .net framework in required version. Cons of this solution is that didn't provide all information from user input: we don't have azimuth and altitude information. Because this features are device specific and in time of writing this paper only one device capable capture this type of data was on market, we don't considered it as big disadvantage.

We will choose approach by InkCanvas, because no third party driver installation is required. We use all data that this approach provides.

5.2 Signature features used to authentication

First of all we decide not to use any parametric feature. This decision is based on fact, that supposed forger will have full visual copy of user signature. This assumption was supported by work Smudge Attacks on Smartphone Touch Screens [26]. In this work they manage to obtain significant large information about user pattern password from smudges on smartphone screens. Since tablets (as targeted platform) are device of daily usage and its screen can be little dirty they can suffer from same problem. Figure 5.1 shown example of this signature exposure on tablet surface.

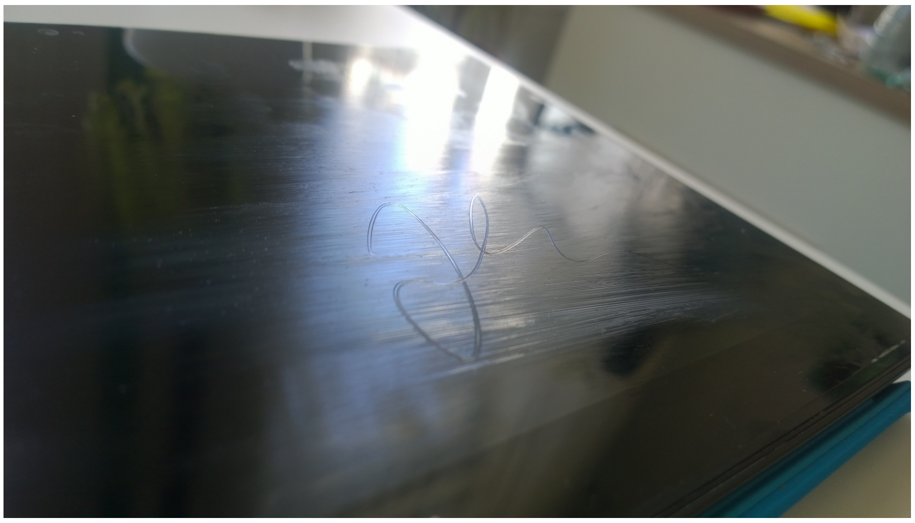


Figure 5.1: Picture taken by mobile phone without flash during standard light conditions

Parameter features have advantages of providing some additional information about signature. This information, in wrong hand can on the contrary wok against signature security. For this reason we won't use any parameter features of signature.

We will use only function features of signature. Cons of this approach is this feature is harder to process, since its time-bound data. Pros is, that function features can't be captured from already written signatures.

5.3 Token from signature

As we know from Signature credential provider design 3.5, result of signature authentication should be access token, which we use from decrypting credentials. Question is, how we obtain this token from entered signature.

First option is to transform signature directly to access token. Similar solution, but for voice-based approach can be found in [33], where cryptographic keys are generated from spoken telephone number sequence. Signature hash described in [29] used 50 diferent signature features to transform signature to bionic hash.

Great advantages of this technique is there is no need to storing access token. However we must store user related parameters for hash calculation. One major disadvantage is that during signature transform, there is no easy way to determine which feature is responsible for possible mistake.

Another options use combination of signature hash with digital key. First solution in this category use embedding of digital key to biometric hash, for example with bit replacement. Second solution uses combination of digital key and signature (biometric in general) data into so called Bio Script ^{TM} in such way that neither information can be retrieved independently by the other. Both solution are based on signature hash (or directly use data divided from signature) thus they share same disadvantages. Additionally we must manage digital key.

Last option is solution which release token upon successfully signature verification. Advantage of this approach is that verified signature doesn't transform token in any ways. If is valid, token is sent and vice versa. From transformation signature token we move to signature authorization. By this approach, we have wider choice of technique to verify signatures, since there are lot of related works. Biggest disadvantage is related with security. Token and all related data for authentication must be securely stored.

We tried transforming signature directly to access token and releasing token upon successful verification. Transform option during implementation shown unreliable results, caused probably with trimmed feature set. Successful usage of this technique is major part of future work. We chose last option, because we prefer rather signature authorization before signature transformation, even at the cost of reduced security. As we will talk in chapter security, it's pointless to have steel strong security in mild environment.

5.4 Signature authentication

Signature authentication can return two results: given signature will be, or will be not valid. But how we will validate this signature? Form chapter1.5 we have basic knowledge about method used in signature authentication. We will use on-line signature verification for nature of capturing device. Discussed authentication methods will be: continues and semi-continues hidden Markov model (HMM), neural networks and dynamic time warp (DTW).

For authentication by hidden Markov model, support vector machines or neural networks we need large training data set. For example in this work [31] about signature verification bases on neural network classifier they used 40 samples per person, totally 240 writers. We don't have access to database of signatures of this size and prompt end-user to sign in 40 times before he can use our application is unrealistic. However if good training data is provided above mentioned method shown great results.

Last option is to use Dynamic time warp. This method doesn't need any training data, so it's ideal for our purpose. Dynamic Time Warping is used to compute a distance between two time series. Naive approach to calculating a matching distance between two series could be to resample one of them and then compare it sample by sample. DTW provide recovering of optimal alignment between sample points in two time series. The name Time Warping is derived from warping time axes of the two time series in way that corresponding samples appear at the same location. Drawback of this method is that we calculating distance between two series, so we must sustain user signatures that are entered upon initialization.

We will use dynamic time warp because no training data is needed, which we cannot provide. Using DTW for each time series of feature values we will get quantifiable similarity between one features of two signatures. How we use this basically number to verify similarity between two signatures?

5.5 Weight of signature features

Have signature features equal importance while measuring signature similarity? Each signature feature reflect uniqueness of signature in some ways, so they are as unique, as signatures itself. Some features will be more important to determine validity of signature as others.

We introduce feature weight, as measurements of signature features relevance. Options for obtaining feature weight that we will discuss:

1. One constant weight to all signature features.
2. Constant weight unique per feature.
3. Weight of features based on pattern signatures.

First option just increase (or decreses if weight is below 1) differences between features. This is not in general bad thing, but it doesn't capture uniqueness of signatures.

Second option will ensure basic features weight. Pros of this solution is that we can set feature weight: favor one feature before another one. Values of features weights are set manually in program. Feature weights can be based on research therefore it can cover different importance between features in general. But these weights are same for every user. Con of this solution is that it doesn't take into account uniqueness of signatures.

Last option will calculate feature weight for each user. Each features will have capability for standalone weight calculation. Weight calculation will take in favor: range of feature values and similarity based on this feature.

We chose this option despite more demanding implementation in favor to increase accuracy of our method.

5.6 Calculation of signature weight

When we calculate signature weight, we must look to two statistical measure: Intrapersonal (intra-pattern) stability, intrapersonal (intra-pattern) entropy.

5.6.1 Feature similarity deviation

Major problem with using signature (and any other bionic) features, is the trade-off between natural intrapersonal variability of feature values between several samples of a user and the requirement to have a persistent stable value (or interval) to authenticate. Example for this dilemma is the total writing time of a signature. This feature is very easy to calculate and very often used in low-resources verification systems [7] this feature shown a rather stable intrapersonal behavior. If for example a natural interpersonal variance of 5% . For signature that takes 5 seconds to write, all duration in interval [4.25s .. 5.25s] should be accepted to authenticate this feature. In short feature deviation can be describe as intrapersonal instability reflecting the degree of scatter of feature [29].

We use function features, so instead of one value as total writing time of signature in example, we have time-ordered vector of features values. Instead of calculating feature deviation over aggregate (min., max.,avrg.) values of function features per pattern, we can use value of similarity per feature.

Big deviation in aggregate value does not have to impact similarity. For example when user have punctuation in signature, pressure factor in punctuation is independent from rest of signature. By calculating similarity by DTW this punctuation pressure deviation will not have big impact on overall similarity, however it impact feature deviation. Instead of intrapersonal stability, we are looking of intra-pattern stability (Stability between user patterns).

We will use feature similarity deviation, as measurement of stability in similarity over one feature. We calculate feature similarity deviation as ratio of average feature similarity values over both extreme values of similarity

$$FeatureSimilarityDeviation_i = \frac{2*average(Similarity_i(patterns))}{Max(Similarity_i(patterns)+Min(Similarity_i(patterns)}$$

Function $Similarity_i$ return vector of similarities for feature i calculate for all pairs of unequal signature patterns.

By including feature similarity deviation into feature weigh we can raise weight of feature, that have stable similarity and vice versa.

5.6.2 Feature entropy

Information entropy had been introduced by Shannon in 1948 [28], and is a measure for information density with a set of values with known occurrence probabilities. Information entropy is the base for several efficient data coding and compression techniques like the Huffman code [27]. The question of effective information content is direct related to the uniqueness of signature feature. For features with a low interpersonal variability, it can be expected that many users will have similar or identical values, whereas a height interpersonal variability indicates a greater distinguishing character. Feature entropy in short can be describe as potential of information density in feature [29].

Applied on function features, we can assume that if feature have bigger range of feature values, then this feature have bigger entropy. Since all values of feature have same length, we can assume that with bigger value range it can contain more information.
We will calculate feature entropy as

$$FeatureEntropy_i = Max(features_i(patterns)) - Min(features_i(patterns))$$

Function $feature_i$ return vector of all features i values in patterns.

5.6.3 Feature weight calculation

Based on feature entropy and Feature similarity deviation we calculate overall feature weight as

$$FeatureWeight_i = FeatureSimilarityDeviation_i(patterns)*FeatureEntropy_i(patterns)$$

5.7 Signature similarity calculation

Until now, we deal with signature similarity. Result of verification isn't quantified similarity, but Boolean value that declares if input signature is or is not valid. To transform similarity to this statement, we use threshold. If signature similarity is below this threshold, we declare it as valid and vice-versa.

We calculate threshold as average of similarities between all combinations of two unequal signature patterns.

Since similarity is calculate between two signatures, we must choose which one from pattern we will use. We use most similar pattern to examination signature. If similarity is below threshold, we declare exanimate signature as valid. Otherwise we declare signature as invalid.

In other words, a signature can be declared valid, if it is similar to the most similar pattern, at least as much as pattern signatures to each other.

6. Signature authentication implementation

In this chapter, we will discuss implementation of our Signature Authentication service and Sign form. We chose this architecture in chapter Signature Credential design. Along with programs needed to perform signature login, we introduce support programs: Credential manager for recording and re-recording user signature patterns and user credentials and Signature studio which serve for visualization of signature verification and running measurements. Since signature verification is used in multiple project, we will encapsulate it in DLL called SignatureVerification. First we discuss implementation of this DLL, because rest of programs relies on it.

6.1 Signature verification DLL implementation

Signature verification content can by divided into two parts: Capturing signatures and verifying signatures.

6.1.1 Capturing signatures

For handwritten data, .net windows presentation foundation provide class InkCanvas. This Wpf UI element capture and visualize handwritten input in form of stroke collection. InkCanvas doesn't provide all features that we need: it only visualize, don't capture, stylus movement over the surface (hovering) and its accuracy is not sufficient.

We solve this problem by inheritance child object SignatureInkCanvas from InkCanvas into witch we added plugin SignaturePlugin that modify its functionality 6.1

Inside SignatureInkCanvas we create StrokeCollection named _wholeStroke. Function OnStylusAirMove si triggered when stylus is moving over surface. Inside this function we added all with over the surface stylus points to _wholeStroke. With this approach, we can capture whole movement of stylus during signing, not only when stylus touching surface of tablet. We can add complexity to the Signature that look simple on first sign 6.2. As we want to track whole sign movement, when user leave sign form with stylus, written signature is erased before we start writing new signature.

We added modification for signature precision improvements. First we tried to get timestamp information for each point. Inside SignaturePlugin we have access to raw stylus points. Raw stylus point contain Timestamp value, in which timestamp when impute occurred is stored. We didn't get sufficient result by this approach. Stylus points was clustered around few times.

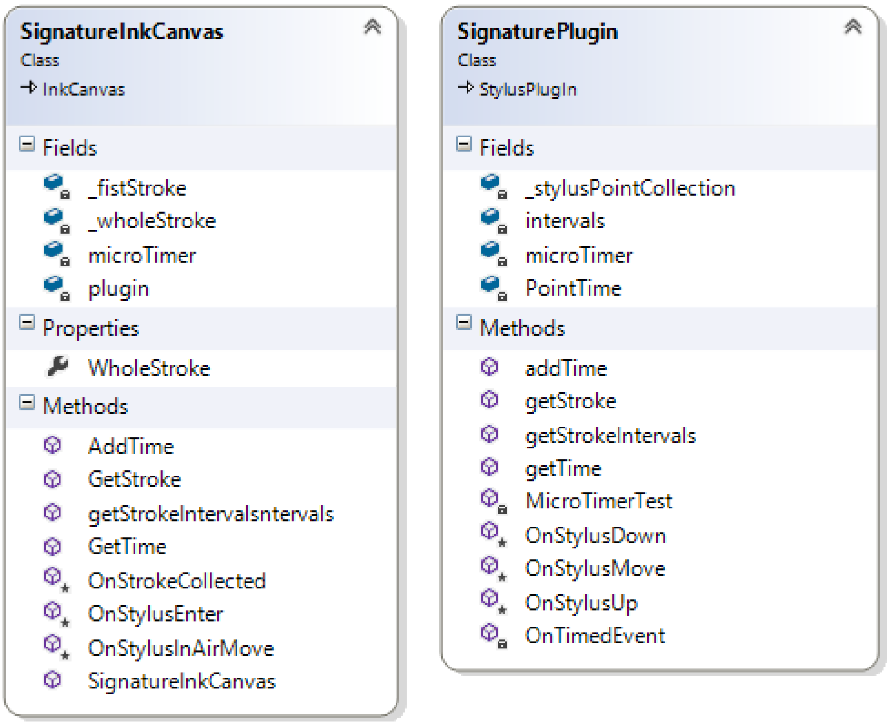


Figure 6.1: SignatureInkCanvas and SignaturePlugin class diagram

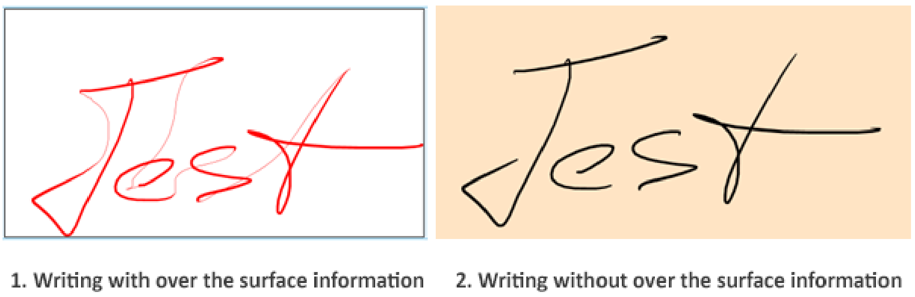

1. Writing with over the surface information
2. Writing without over the surface information

Figure 6.2: Comparison of writing with and without of the surface movement information

Then we try to pool stylus point at regular time bases. When the timeframe is small enough, we can get stylus points that was added during this timeframe. For this solution we set timer on constant time and when timer expired we clone content of InkCanvas. We didn't do difference of two last InkCanvas snapshot on this place for performance reasons. By this approach, we hit on the build in timer smallest possible time limit. Since period of timer is set as a number of milliseconds, one could expect that even can reach a frequency up to 1000Hz. This is not true, the maximum frequency is fall less and its system depend. We tried it on multiple systems and got minimum values around 15-16 millisecond's which 66 – 62 hertz is. This is low value comparably to tablet samples rate that can go from 100pps up to 200. (Samples rate on tablet is in pps : points per second)

To improve timer, we use MicroTimer[1] (related class can be found in subfolder MicroTimer) which can go up to theoretically $1\mu s$ (1Mhz). This is far beyond sample rate of digitalize, so we use only $5000\mu s$ (5ms, 200 Hz) to match sample rate of tablet digitalize. When we start pooling for stylus point, we get stable one point per 5 millisecond. Without pooling sample rate was from 5ms to 20ms. However we won't time data, since we have constant time between points, we provide this information from SignatureInkCanvas for alternative usage.

6.1.2 Verifying signatures

For calculating similarity per one feature we have parent class called SignatureComparer 6.3. We chose inheritance before interface because not all function that we required must have signatureComparer implement, for e.g. function DrawnUI is optional. Inside SignatureComparer we can find only one property ID. This is

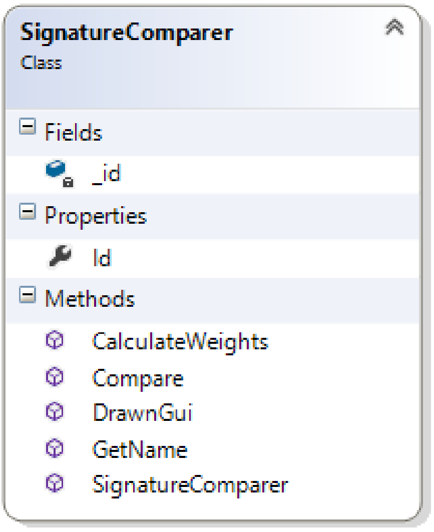


Figure 6.3: Class diagram of SignatureComparer

GUID of the comparer. By this GUID weight of feature that comparer examined is stored inside signature container6.1.4. We initialized credential with this GUID.

For calculating comparer weight (we can say feature, but we don't strict that one comparer use only one feature) there is function CalculateWeights that take List of user patterns and return double value of comparer weight.

For comparing two signatures is used private function Comparer that take two signatures and weight. It return signature similarity in form of double number.

Function DrawnGui is optional, it serve for only demonstrating purpose. DrawnGui is basically signature comparer, but rather than double value of similarity it return whole UIElement that can contain visualization of signature sim-

[1] Available from:`http://www.codeproject.com/Articles/98346/Microsecond-and-Millisecond-NET-Timer`

ilarity, partial results of comparing or it can show whole process of comparing.

Because parent class parent class SignatureComparer doesn't implement any complex functionality, we describe one specific signature comparer: _dynamixX. All comparer that use DTW for computing similarity are similar.

6.1.3 DynamicX signature comparer

DynamicX signature comparer signature use speed in X axle as feature for calculate similarity of two signatures. Similarity calculation is done as follow:

1. We take examined feature (in this case speed in X axle) of pattern and signature
2. We calculate similarity by DTW of this two time ordered series. Dtw parameters are two series and dtw settings. The more signature are similar, then result of dtw is closest to zero.
3. Similarity will be multiplied result of dtw by comparer weight

For calculation of dynamic time warp we using library nDtw (obtaied from `github.com/doblak/ndtw`)

Calculation of comparer weight is done as we describe it in chapter 5.6

For optional function we have besides GetName function DrawnGui. DrawnGui return UIElement that contain comparer name, visualization of dtw calculation both in series and matrix and overall information about feature weigh and calculated similarity (whit and whiteout weight) and visualization of feature 6.4.

Reason why in SignatureVefification folder is Project Ndtw.Visualization.Wpf is that vizualization of dtw calculation didn't work. We have to fix some issue related with oxyplot (dll used for drawing graph).

We added function SetSetting beyond SignatureComparer. By this function we can set and modify setting for DTW calculation as way of distance measure (Euclidian, Manhattan, maximum or squared Euclidian) and if we boundary start and end to be Constrain align or not.

We implemented follow signature comparer on the basis of signature feature 6.5

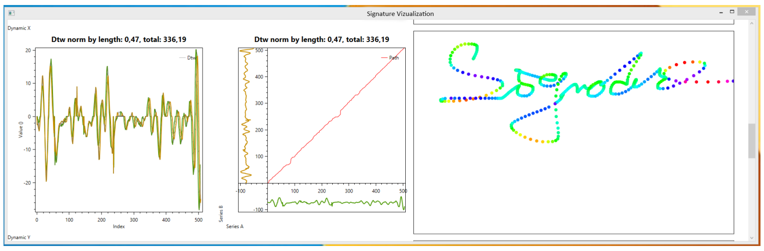

Signature Vizualization
Dynamic X
Dtw norm by length: 0,47, total: 336,19
Dtw norm by length: 0,47, total: 336,19
Index
Series A
Dynamic Y

Figure 6.4: Vizualization provided by DynamicX comparerr

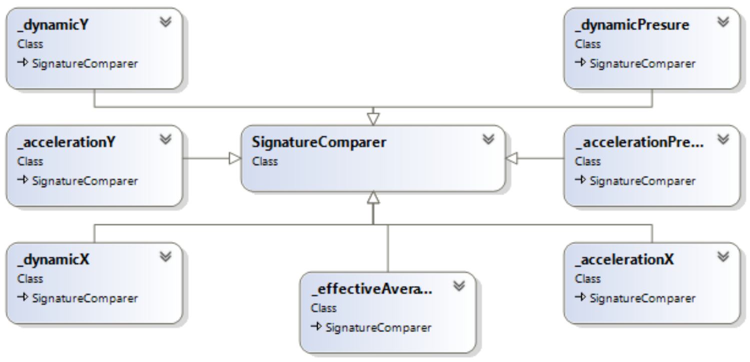


Figure 6.5: Diagram of all implemented signature comparers

6.1.4 Signature Container

We store user patterns, comparer's weights and access token in binary serializable class named Signature Container 6.6. Signature Container doesn't expose these data, but provide ways that data can be modify and obtain.

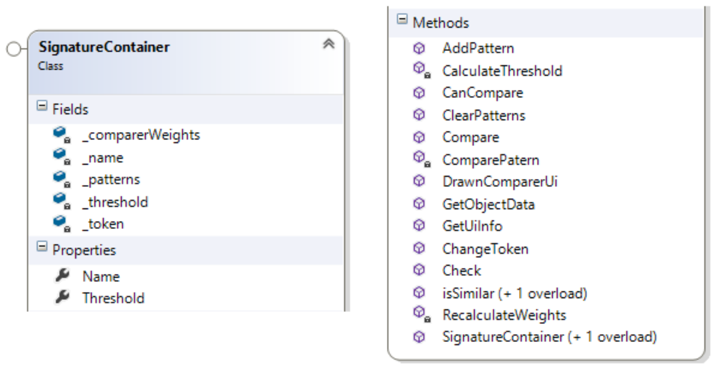


Figure 6.6: Diagram of signature container

First we wall through property:

- Name is name of SignatureContainer. This Name is used as filename.
- Threshold is for demonstrate purpose, it show maximum value of similarity value that can have signature to be valid.

Function can be divided to tree groups

- Managing functions: AddPattern, ClearPatterns ,RecalulateWeights, ChangeToken.
- Comparing function: isSimilar with one overload. Function can return Boolean value if given signature is valid or additionally return quantified similarity (double) and access token (if signature is valid) in out parameters.

- Demonstrating functions: DrawnComparerUi and GetUiInfo are function that return UIElements , its use for demonstrating purpose

We will describe this functions more in detail during usage scenarios wall through.

6.2 Sign form

Sign form is used for capturing user signatures and transferring it to Signature Authentication service. It contain few classes:

- Speaker: provide communication through name pipes. In this class is also implemented visual behavior of sign form based on signature verification result.
- MainWindow is main class of this program. This class is responsible for serializing and sending user signature. MainWindows contain XAML file that describe UI of sign form. For capturing User signature we use SignatureInkCanvas describet in chapter6.1.1

6.2.1 Usage scenario

When Sign form is initialized, Speaker instance connect to namePipe. This channel is duplex for receiving message about signature authentication. User then enter signature and click on send button. After send button is pressed, we get signature from SignatureInkCanvas by read only property .WholeStroke. In wholeStroke is stored signature whit off-surface movements. Only way to serialize stroke is through Stroke collection. Stroke collection have capability to save itself to stream. We use this way of serialization as follows: we create memory stream, in which we save StrokeCollection that contain only whole stroke. Then we get byte array from memStrem and encode it to string. This string will we send to Signature Authentication service.

There can be more than simple true / false result from authentication service. We distinguish four result states and follow behavior:

- Signature verified : Sign form flash in green color and close itself
- Signature denied : Sign form tremble itself , it stay open for another signature
- Signature recorded: Sign form flash in blue color. This happen during recording new signature's. This indicate that signature was recorded correctly.
- Signature verified silent: Sign form flash in dark green, but stay open. This happen when signature is verify not against signature authentication service but against Signature studio6.5. or when surficial count of signature is recorded.

6.3 Signature authentication service

Signature authentication service serve for authenticating signatures. Signatures came from Sign Form through named pipe. Besides authenticating signatures and sending access token to Signature Credential provider it serves for creating and updating user's data stored in SignatureContainers.

We walk through tree usage scenario: service loading, authenticating signature and recording new signatures.

6.3.1 Signature authenticator loading

In OnStart function (every service must implement OnStart and OnStop function) we create and start namePipeServer , open host , and we load userPatterns(stored in Signature Containers) and create signature comparers.

Loading user pattern is done by function LoadUserPatterns. We scan `\C:\Signature\UserPaterns"` directory and we try desterilize every file in it. After success deserialization we added this signature container inside list SignatureContainers.

Signature Comparers are created by function AddComparer. We simple added new instance of every used signature comparer to list of signatures Comparers.

6.3.2 Signature authentication

First of all, we need to obtain signature to authentication. Signature came from Sign Form to signature authentication services through namepipe. Besides signatures from Sign Form we can receive command messages from Credential Manage 6.3.3 we will discuss this in chapter 6.3.3, we will split code execution by sender name. For now we will focus on signatures only.

Fist we recreate Stroke from received string. We Decode string to byte array and from this byte array we create memory stream. We recreate StrokeCollection from this memory stream. Its exactly opposite way as we use in sign form to serialization 6.1.1.

Then we check if we are in record state

First we look out on our signature container, if any can compare. If yes, we authenticate signature by function Compare Signature with recreated stroke as parameter.

For each signatureContainer, we run similarity calculation by function isSimilar. We will store thresholds and access tokens alongside with similarity results.

There must be exactly one similar signature found. If there is more or less than one match, we can't declare positive user authentication. In this case we write Error to Event log and send message about unsuccessful signature verification to sign form to provide visual feedback to user.

In case that signature was authenticate successfully we send access token to Signature Credential provider through name pipe .We also send message about successful authentication to sign form.

6.3.3 Management of signatures

As we told, we can receive messages from Credential Manager. These messages are send when we want record new users pattern or create whole new user (SignatureContainers). We can receive massage of follow type

- **Record**: We change state of service to record. This mean all incoming signature will not be authenticate, but they will be added to signatureContainer. If user already have recorded and loaded signature container, we find it, clear user patterns and start recording to it. If we record new user, we create new signatureContainer. Alongside control message, we also received access token. By this token we can find SignatureContainers related to user, or create new signature Container. At the beginning we set record timer that change state back from record after time automatically, if something unexpected happen.
- **Change**: This message change token inside signatureContainer. Alongside control message, we also received old and new access token. By old token we can find SignatureContainers related to user in which we change token to new one
- **Stop recording**: This message change state of service from record to authenticate. It do not save new recorder signature Container.
- **Discard**: This message change state of service from record to authenticate, disable record timer and for consistency we reload all user patterns from disk.
- **Save**: This message change state of service from record to authenticate. We save edited (or newly created) to User Pattern folder, disable record timer and for consistency we reload all user patterns from disk.

6.4 Credential manager

Signature manager in form of wizard is used for managing and recording signatures. Users will use signature manager by done three tasks:

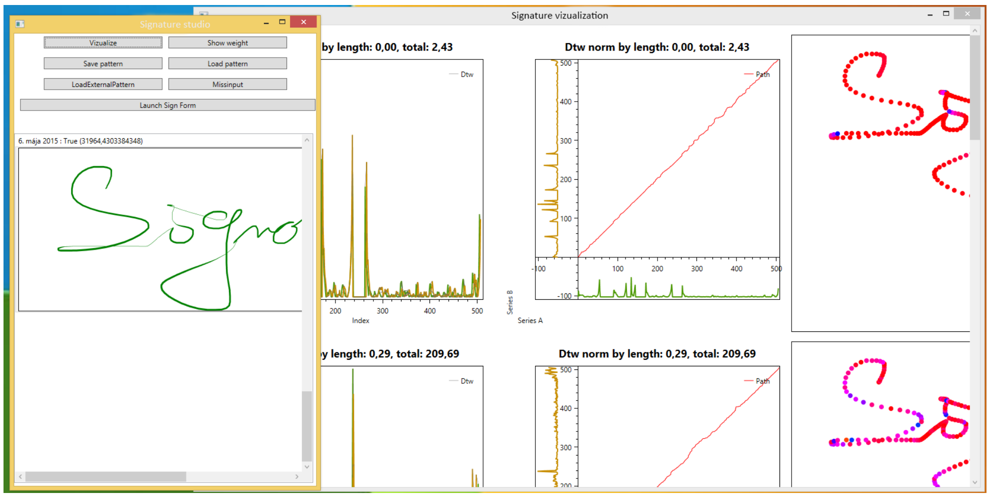
- Change stored windows password
- Change stores signature
- Both above

When user want to change any information, first of all he must verify by current windows password. From password token is generated. If we don't found any matching stored user credentials, we ask for last password. For new users (or users that forge old password) there is change to generate new credentials. Users that just want update password can skip signature recording part.

Then record control message is send to signature authorization service and sign form is launched. After user enter signature pattern sign form notify about successful record. After 3 signatures are recorded (smallest sufficient amount) user is notify. He can continue recording more signature patterns or close sign form and go to next step, stop record control message is send. Then user fill credentials and finish wizard, save control message is sender and user credential are updated (or created). If application is closed Discard control message is send. Screenshots of Credential manager can be found in User documentation.

6.5 Signature studio

Signature studio 6.7 application was originally built for running measurements on signatures. During development it's became center of signature authentication visualization, performing measurements on given signature sets and even can substitute signature authentication service in some ways. It also serve as counterpart when debugging some part of signature authentication service.



6. mája 2015 : True (31964,4303384348)

by length: 0,00, total: 2,43

Dtw norm by length: 0,00, total: 2,43

y length: 0,29, total: 209,69

Dtw norm by length: 0,29, total: 209,69

Figure 6.7: Signature stuio with open signature vizualization

7. Evaluation

In addition in this section, we will evaluate our signature authentication implementation. First we look on difficulties about gathering test data and then we evaluate our signature authenticator in set of test with variable count of user patterns and different threshold computing.

7.1 Test data

Gathering testing data was difficult for two reasons:

- To compare test signatures, all signatures must be capture on same (or at least similar) device. This is due different features of capturing devices. For example test subjects that have signature captured by big tablet (size of A4 +) have usually larger signatures than test subject that have signatures capture by small one. Other difference is between signing on tablet with digitizer that provide no visual feedback and on tablet pc.
- Users, especial in these days, are aware of computer security threads. Is nearly hard to gather one signature samples, harder obtain at least three signatures samples and nearly impossible obtain signature, when we tell to test subjects, that we will try to forge given signatures.

We test data obtained from SVC 2004: First International Signature Verification Competition [30].Competition consist from two task about signature verification. Information provided to first task contain only coordinate information only. For second task information contain additional information including pen orientation and pressure. Because we use pressure as one of signature feature for verification, we will use data from task 2.

"Each data contributor was asked to produce 20 genuine signatures and 20 skilled forgeries in two separate sessions. For privacy reasons, the contributors were advised not to use their real signatures in daily use. Instead, they were suggested to design a new signature and to practice the writing of it sufficiently so that it remained relatively consistent over different signature instances, just like real signatures. Contributors were also reminded that consistency should not be limited to spatial consistency in the signature shape but should also include temporal consistency due to dynamic features.

In the first session, each contributor was asked to contribute 10 genuine signatures. Contributors were advised to write naturally on the digitizing tablet (WACOM Intuos tablet) as if they were enrolling themselves to a real signature verification system. They were also suggested to practice thoroughly before the actual data collection started. Moreover, contributors were provided the option of not accepting a signature instance if they were not satisfied with it. In the second session, which was at least one week after the first one, each contributor came again to contribute another 10 genuine signatures. In addition, he/she also contributed four skilled forgeries for each of five other contributors. Skilled forgeries were collected in the following fashion. Using a viewer, a contributor could see genuine signatures of other contributors that he/she would attempt to forge. The viewer could replay the writing sequence of the signatures on the computer screen. Contributors were also advised to practice the skilled forgeries for a few times until they were confident to proceed to the actual data collection. The signatures are mostly in either English or Chinese" [30].

We only use data that can also provide tablet pc digitizer (X/Y coordinate, pressure). We don't use azimuth and altitude information. Overall we have 1600 signatures (20 genuine and 20 by skilled forger from 40 test subject).

7.2 Test methods

We will measure these factors:

- **FRR** False Rejection Rate : ratio of the number of false rejections divided by the number of total identification attempts.
- **FAR** False Accept Rate : ratio of number of false acceptance divided by the number of total identification attempts.
- **EER** Equal Error rate: a point where the FAR and FRR intersect. This point describes probability when FAR and FRR are equal. The risk of accepting an impostor is equally as small as the risk of rejecting a legitimate user.

Since our method use relatively small count of user signature patterns (n), we define one iteration of n patterns as: knowledge base of authentication sustain of n patterns randomly pick from 20 genuine signature. We run follow test: Miss impute test and Forger test.

7.3 Miss input test

In this test, we measure robustness of our authentication method against "random" user input. For every test subject, we run 100 iteration of 3 to 6 user patterns. We used substitution from random input signatures of all other test subject. This approach measure if one user signature can be authenticate as two person. If signature match to more than one parson, we declare it as invalid, because we can't guarantee its authenticity.ňň

In total 2,5 million of signature comparison have been made. Only four random user input was considered as valid.

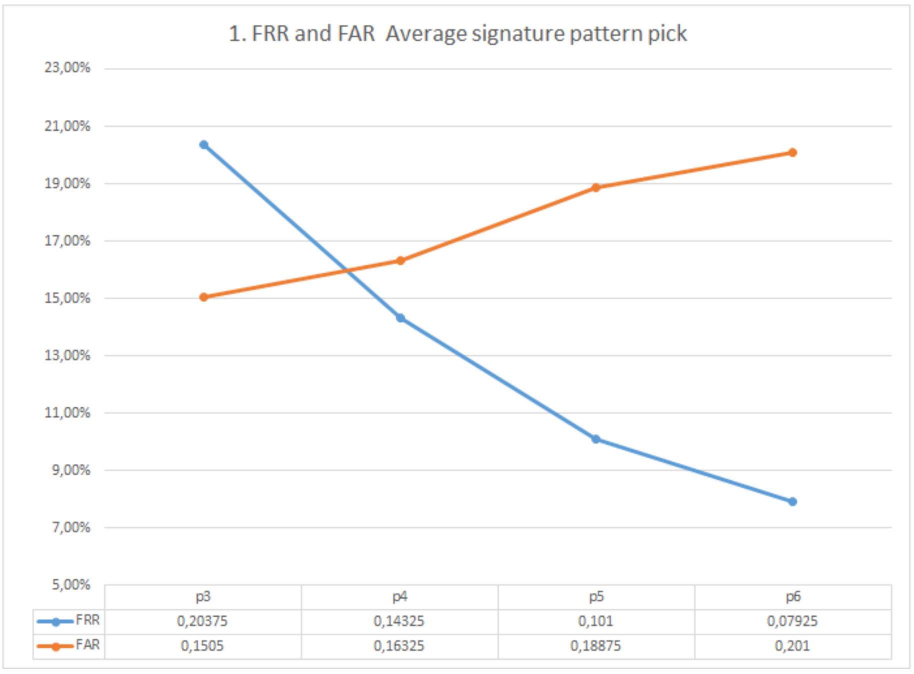
7.4 Forger test

In this test, we measure robustness of our authentication method again skilled forger. FRR measures the rate of genuine signatures classified as forgeries while

FAR represents the rate of forgeries recognized as genuine ones.

For every test subject, we run 100 iteration of 3 till 6 user patterns. As input to compare, we use remaining genuine signatures and all forged signatures. We don't take pattern signature as input, because from nature of our authenticate method, they will always be correctly classified.

We are averse of importance of selection of right quantity and quality of signature pattern in our method. Alongside average result of each random pattern pick, we record best and worst case. This can give us better perspective in what range our authenticate method works.



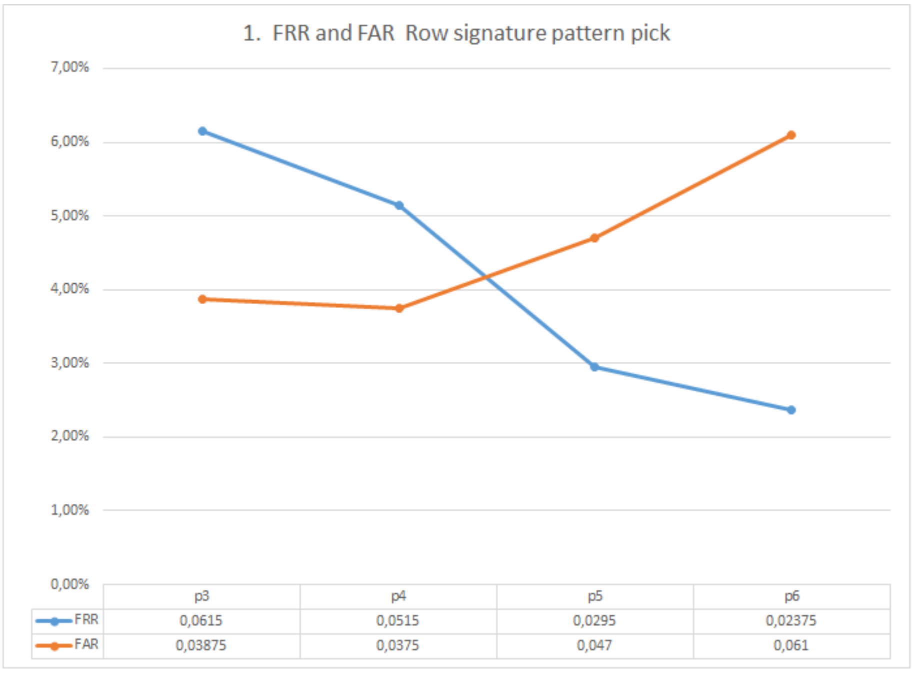
	p3	p4	p5	p6
FRR	0,20375	0,14325	0,101	0,07925
FAR	0,1505	0,16325	0,18875	0,201

Figure 7.1: Graph of FRR and FAR of random signature pattern pick

In figure 7.1 we can see FAR/FRR of average signature pattern pick. EER in average signature pick is 15.97%

Random pattern pick is good for providing overall result of our signature authentification method, but signatures pick doesn't correspondent with our usage scenario. Random pattern pick have big probability to pick signatures that have been recorded week apart. In our usage scenario, we will record pattern signatures in a row, so each signatures will be recorded in short succession.

In next test, we randomly pick one genuine signature from one week and then we add following patterns until we have required amount of patterns. When pick patterns in this way, we use remaining genuine signatures alongside with forged signatures as input to compare. This patent pick better reflects how users of our method will record signatures.



	p3	p4	p5	p6
FRR	0,0615	0,0515	0,0295	0,02375
FAR	0,03875	0,0375	0,047	0,061

Figure 7.2: Graph of FRR and FAR of row signature pattern pick

Results can be found in figure 7.2. As we can see, recording signatures in a row produce better results as random pattern pick. EER by pattern pick in a row is: 4,17%

Overall EER of implemented method is 15.97%, but it doesn't reflect ways that users pattern will be recorded in our solution. Much closer to real time usage is row pattern pick method with EER 4,17% . As result we will mention both results, random pick result as overall result of method, and row pick result real time usage result.

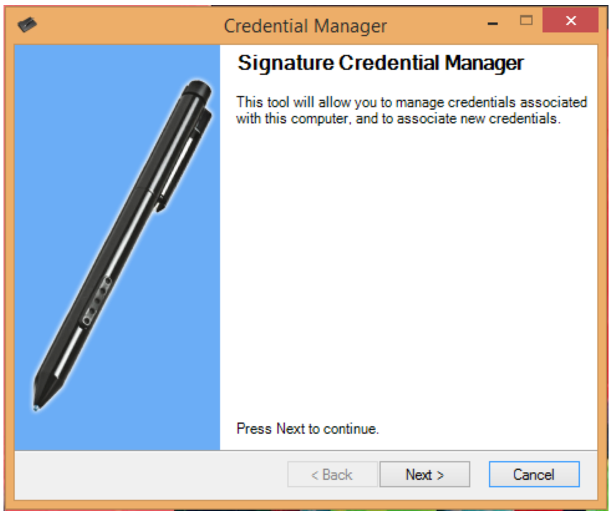
8. User documentation

8.1 Instalation

User install our signature login solution by Installer. No user input is needed during installation. At the end, user have check box that run Credential Manager after installer exit.

8.2 Signature and credentials managing

Signature management is done by program Credential manager. Credential manager, due to registry access, must by run with as administrator.



Credential Manager

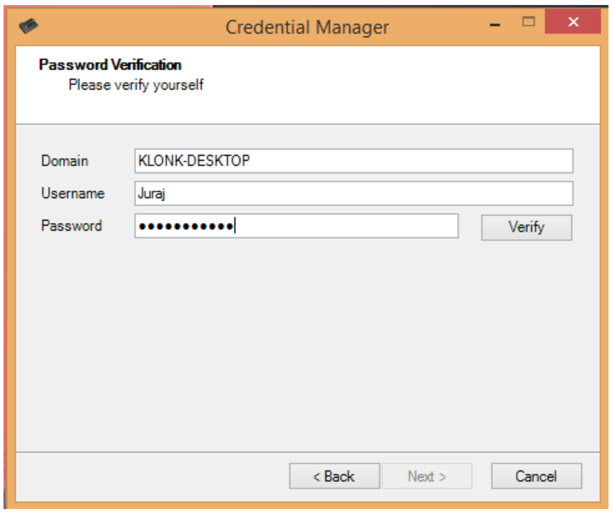
Signature Credential Manager

This tool will allow you to manage credentials associated with this computer, and to associate new credentials.

Press Next to continue.

Figure 8.1: Initial page of Credential Manager

8.1 is initial screen of credential manager. Only basic information are provided on this page. User click Next to continue



Credential Manager

Password Verification

Please verify yourself

Domain KLONK-DESKTOP

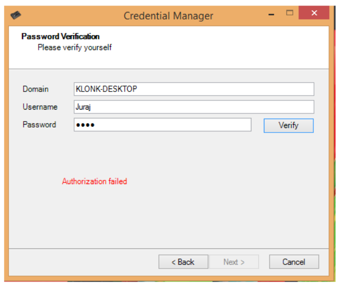
Username Juraj

Password ●●●●●●●●●●●

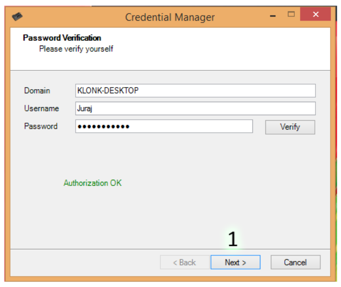
Figure 8.2: User verification page of Credential Manager

On next page, User is requested to verify himself. After user enter password, he click on Verify button. Three scenarios can happen:

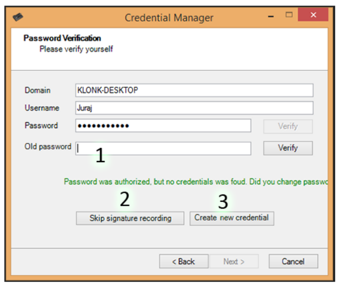
1. User password is incorrect: Correct password must be entered. 8.3.1
2. User password is correct and user already have stored credentials: Click next to continue. 8.3.2.1
3. User password is correct, but no credentials were found. This case can happen for follow reason:
   (a) New user, click on button new user 8.3.3.3
   (b) User password is different that stored one, User is prompt to enter old password and verify it 8.3.3.1. By this, user can just change password and skip Signature recording 8.3.3.2 to Edit credentials page 8.5


Credential Manager
Password Verification
Please verify yourself
Domain: KLONK-DESKTOP
Username: Juraj
Password: ••••
Authorization failed

1. Fail page


Credential Manager
Password Verification
Please verify yourself
Domain: KLONK-DESKTOP
Username: Juraj
Password: •••••••••••
Authorization OK

2. Verification OK page


Credential Manager
Password Verification
Please verify yourself
Domain: KLONK-DESKTOP
Username: Juraj
Password: •••••••••••
Old password:
Password was authorized, but no credentials was foud. Did you change passw

3. Verification OK,missing credentials page

Figure 8.3: User verification page variaton of Credential Manager

On next pag 8.4, Sign form is launched. User enter pattern signature and click send. When signature is successful recorded, visual feedback in form on blue flash is provided. After sufficient amount of signature is recorded (3 signatures) visual feedback in form of dark green flash is provided. User can record more signatures. Then user close sign form and click next to proceed to next page 8.5 or cancel to discard any changes.

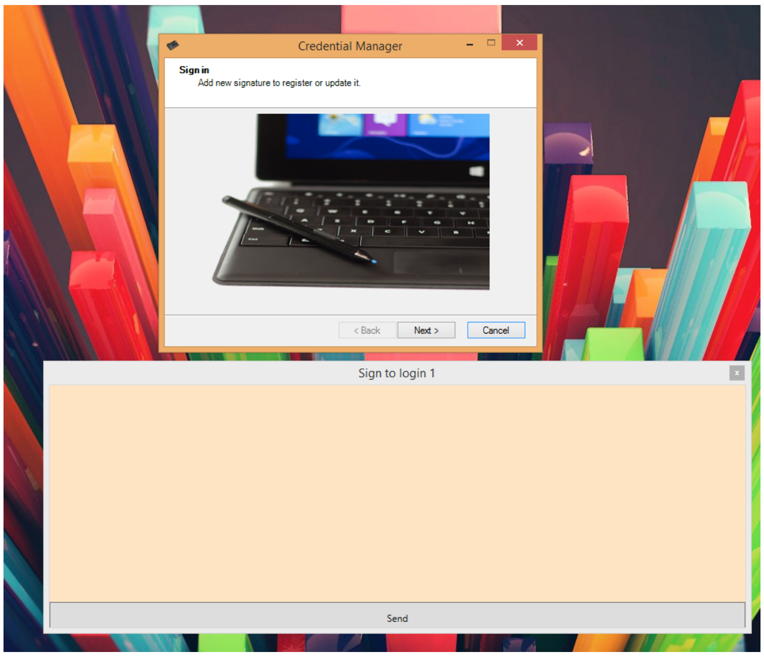
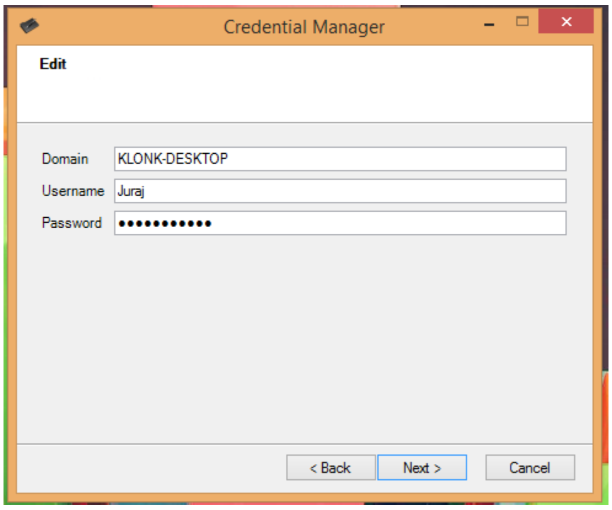


Figure 8.4: Signature recording page of Credential Manager

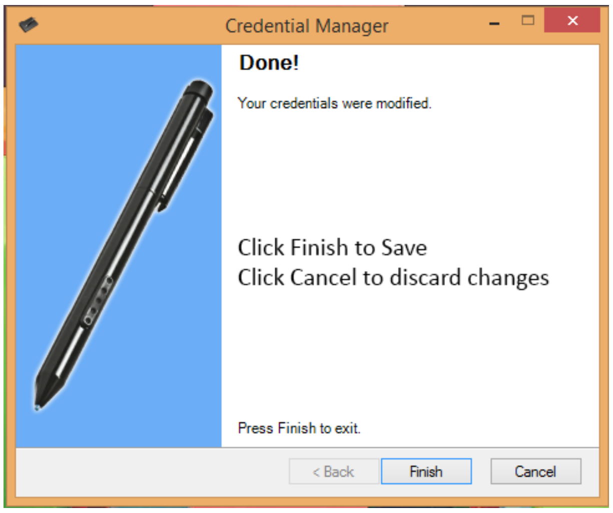
In this page 8.5, user enter valid credentials and can verify if user name and domain is correct.Click next to preceed to last page 8.6



Domain KLONK-DESKTOP
Username Juraj
Password ●●●●●●●●●●●

Figure 8.5: User credential edit page of Credential Manager

This is end page 8.6. After finish button is clicked, all changes will be saved. When cancel (or close button) is pressed no changes will be made.



Done!

Your credentials were modified.

Click Finish to Save
Click Cancel to discard changes

Press Finish to exit.

Figure 8.6: Finish page of Credential Manager

8.3 Login by signature

Login is little different in by V1 and V2 signature credential provider. V1 credent provider will on right side 8.7.1, V2 on left side 8.7.2.Fist user click on title of signature credential provider.

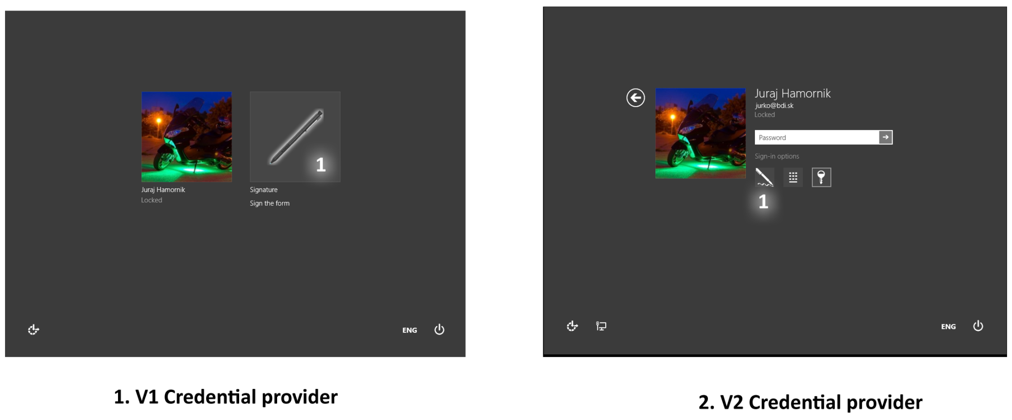


Figure 8.7: Logon screen of V1 and V2 Signature credential

In V1 Credential provider sign form is launched automatically. In V2 Credential provider user must click on tile “launch sign form”. 8.8.2.1
User enter genuine signature to sign form and click Send. If Signature is accepted, Visual feedback in form of light green flash is provided and sign form is closed and user is singed in windows. If Signature is not accepted, visual feedback in form of signature form shake is provided.

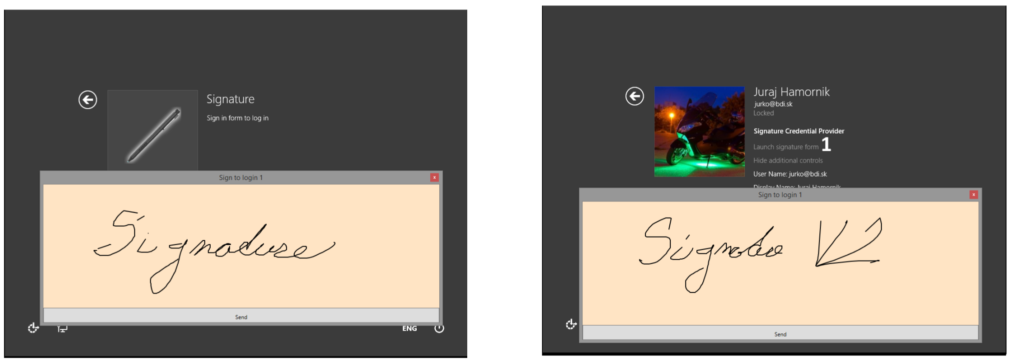


Figure 8.8: Sign screen of V1 and V2 Signature credential

9. Related work

There are no related work in term of allowing user to log in to windows by handwritten signature. There is no solution on other platform (android, iOS, Linux) that allows user to log in by handwritten signatures. However some usage of handwritten signatures and alternative ways to log on can be found. As this whole paper, this chapter will be too divided to section about signatures and about credentials provider

9.1 Signatures

There are countless related works in academic sphere about signature verification, authorization and capturing and classifying handwritten data, both by in on-line and off-line methods. We highlight one most similar work: Dynamic signature recognition based on velocity changes of some features [32]. In this work, authors use similar set of signature function features and DTW as method for comparing signatures. Additionally they use signature rotation for better similarity match. In this work even same test data set is used, so we can compare results on same data. However only fraction (320 from 1600) signatures were used. From nature of proposed technique of they work, best result is used. With our ERR value of raw pick 4.17% is comparable with value of ERR 1.4% in their method. Additionally in comparison with other works compared in the paper, our method is comparatively accurate. Proposed technique of they work is better as our method when choosing patterns randomly with ERR value of average pick 15.97%

In commercial solutions, handwritten signature is almost exclusively used for signing documents. Documents are signed only with image of signature or additional information about signer is included. In commercial solution stand out solution SignDock. This solution use handwritten signature to verify signer against pre-created online account. If verification is successful, on signed document are beside image of signature added information about signer. Most wide spread software that allow placement of signature (without verifying signature) is adobe acrobat.

Handwritten signature used for authenticating user is used in some mobile application. In application Bio Wallet Signature claim itself as exclusive signature password manager. This application use handwritten signature for authentication user for access to sensitive information such as credit card information or internet banking app.

9.2 Credentials providers

There are many alternative ways to log in to the Windows.

One category of alternative credential provider use bionic trait to authorize user. In this category follow bionic trait is used: fingerprint, hand, veins and iris scan.

Other category use possession of item to authorize user. Token in form of RFID card, NFC tag or USB key. Additionally some usb key solution have capability to automatically lock and unlock workstation based on proximity of usb-key contra part [Gatekeeper , Gkchain.con]

par Two-way authentication is also used. For example by solution from company duo security.

Last category are plugin-based credentials as pGina. When we want to custom background to our authentication process, with pGina we can authorize user against sql server, CAS (central authentication service) or even text file.

Conclusion

In this thesis, we implemented standalone solution that allow user log in to Windows by handwritten signature. This solution can be divided into two parts: Signature authentication and Signature credential provider.

Because credential providers, by their design, have limited capability to capture handwritten data, we split implementation to three parts: Sign form for capturing handwritten signatures. Signature authentication service to authenticate signatures and Signature credential provider that provide login by information from Signature authentication service. As support programs for this implementation are: Credential manager for managing credentials and recording signatures and Signature studio for signature authentication visualization.

We implemented both V1 credential provider for Windows Vista and above and V2 credential provide for Windows 8 and above. To implementation of both credential providers led problem with Microsoft live account in V1 credential provider. Biggest challenge in both credential providers were to achieve log in to the system without any spare user interaction (capability to be able do auto login). Demanding task was also how to secure store user credentials. We solve this by storing user's credentials in encrypted format in registries with access right restrictions.

Based on difficulties with obtaining training signature, we chose method that required smallest amount of signatures patterns: signature authentication by dynamic time warp features similarly. We also used feature weight based on feature entropy and feature deviation to achieve better results.

Our signature authentication method provide sufficient security properties with equal error rate (risk of accepting an impostor is equally as small as the risk of rejecting a legitimate user) with row pattern pick of 4.17%. This EER is comparable with other methods that use small amount of user patterns. Used signature authentication method with random pattern pick have EER at 15.97%. This method, because it use few user patterns, is however susceptible ambitus user signatures that serve as pattern.

Safety of solution raise and falls on security of signature credential provider. We are storing user encrypted credentials in registry that are over and above protected with access right. We are also storing user pattern and token in binary serializable class, but we don't expose these data directly, but through functions that required valid signature as input.

Feature work

In feature work, as we mentioned in chapter signature design, we want to achieve direct transformation from signature to access token, preferably without any user-related addition data.

For little better user experience, some modification in V2 signature credential provider have to be made. In feature work, we want to launch sign form in right time, without any additional user interaction as clicking on link that is needed in current implementation.

Last point of feature works, will also involves signature credential provider. As we can see in chapter alternative credential provider usage, there is small if none direct binding between signature and credentials. We want to redraw signature credential provider in the form of credential provider framework as platform similar as pGina, but with capability to launch external programs from credential environment. By this approach some exciting ways of user authentication can be develop such as log in by whistle or log in by tapping sequence on table

Bibliography

[1] POSTGATE, J. . *Early Mesopotamia: society and economy at the dawn of history.* . New York: Routledge, 1994, xxiii, 367 p. ISBN 04-150-0843-3.

[2] STEINSALTZ, Commentary by Rabbi Adin. *The Talmud.* 1. ed. New York: Random House, 1994. ISBN 0679428992.

[3] NICHOLAS, Barry. *An introduction to Roman law.* 1. ed. Oxford: Clarendon Press, 1991, pp. 254-255. Clarendon law series. ISBN 0198760639.

[4] SCOTT, S. *The civil law: including the Twelve Tables, the Institutes of Gaius, the Rules of Ulpian, the Opinions of Paulus, the Enactments of Justinian, and the Constitutions of Leo.* Union, N.J.: Lawbook Exchange, 2001, pp. 217. ISBN 1584771305.

[5] OSBORN, Albert Sherman. *Questioned documents.* 2d ed., complete and unabridged. Chicago: Nelson-Hall Co, [1974], viii, xxiv, 1042 p. ISBN 08-822-9190-4.

[6] JAIN, Anil, Lin HONG a Sharath PANKANTI. Biometric identification. *Communications of the ACM.* vol. 43, issue 2, pp. 90-98. DOI: 10.1145/328236.328110. Available from: `http://portal.acm.org/citation.cfm?doid=328236.328110`

[7] DULLINK, VAN DAALEN, J NIJHUIS, L SPAANENBURG a H ZUIDHOF. Implementing a DSP Kernel for Online Dynamic Handwritten Signature Verification Using the TMS320 DSP Family. *DSP Solution Challenge 1995 European Team.* 1995, pp. . 1. Available from: `http://www.ti.com/lit/an/spra304/spra304.pdf`

[8] YAMPOLSKIY, Roman V. a Venu GOVINDARAJU. Behavioural biometrics: a survey and classification. *International Journal of Biometrics.* 2008, vol. 1, issue 1, pp. 81-. DOI: 10.1504/IJBM.2008.018665. Available from: `http://www.inderscience.com/link.php?id=18665`

[9] MCCABE, Alan a Jarrod TREVATHAN. Markov Model-Based Handwritten Signature Verification. *2008 IEEE/IFIP International Conference on Embedded and Ubiquitous Computing.* IEEE, 2008, pp. 173-179. DOI: 10.1109/EUC.2008.138. Available from: `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4755225`

[10] ANTON-HARO, C., J.A.R. FONOLLOSA, C. FAULI a J.R. FONOLLOSA. On the inclusion of channel's time dependence in a hidden Markov model for blind channel estimation. *IEEE Transactions on Vehicular Technology.* 2001, vol. 50, issue 3, pp. 867-873. DOI: 10.1109/25.933319.Available from: `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=933319`

[11] ÖZGÜNDÜZ, Emre, Tülin ŞENTÜRK a M. Elif KARSLIGIL. OFF-LINE SIGNATURE VERIFICATION AND RECOGNITION BY SUPPORT VECTOR MACHINE. *13th European Signal Processing Conference*. 2005, n. 13. Available from: `http://www.eurasip.org/Proceedings/Eusipco/Eusipco2005/defevent/papers/cr2010.pdf`

[12] MCCORMACK, Daniel K. R., B. M. BROWN, John F. PEDERSEN, Bruce G. BATCHELOR, Susan Snell SOLOMON a Frederick M. WALTZ. Neural network signature verification using Haar wavelet and Fourier transforms. *Machine vision applications, architectures, and systems integration II: 7-9 September 1993, Boston, Massachusetts*. Bellingham, Wash., USA: SPIE, 1993, n. 2064, pp. 14-25. DOI: 10.1117/12.150284. Available from: `http://proceedings.spiedigitallibrary.org/proceeding.aspx?articleid=941886`

[13] KECMAN, V. *Learning and soft computing*. Vyd. 1. Massachusetts: MIT Press, 2001, 526 s. ISBN 02-621-1255-8.

[14] WIROTIUS, M., J. Y. RAMEL a N. VINCENT. Improving DTW for Online Handwritten Signature Verification. *Image analysis and recognition: international conference, ICIAR 2004, Porto, Portugal, September 29-October 1, 2004 : proceedings*. Berlin: SpringerVerlag, 2004, n. 1, pp. 786.DOI: 10.1007/978-3-540-30126-4_95.Available from: `http://link.springer.com/10.1007/978-3-540-30126-4_95`

[15] IMPEDOVO, Donato a Giuseppe PIRLO. Automatic Signature Verification: The State of the Art. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*. 2008, vol. 38, issue 5, pp. 609-635. DOI: 10.1109/TSMCC.2008.923866. Available from: `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4603099`

[16] PLAMONDON, R. a S.N. SRIHARI. Online and off-line handwriting recognition: a comprehensive survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 2000, vol. 22, issue 1, pp. 63-84. DOI: 10.1109/34.824821.Available from: `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=824821`

[17] VADACKUMCHERY, James. *Bankers' safety: methods and techniques*. 2nd ed. New Delhi: Concept Pub. Co, 2002, 70 - 81. ISBN 9788170229377.

[18] Create Custom Login Experiences With Credential Providers For Windows Vista. GRIFFIN, Dan. *THE MICROSOFT JOURNAL FOR DEVELOPERS* [online]. 2007 [cit. 2015-05-02]. Available from: https://msdn.microsoft.com/en-us/magazine/cc163489.aspx

[19] MICROSOFT. *Credential Provider Framework Changes in Windows 8*. 2012. Available from: : `http://download.microsoft.com/download/F/3/5/F3536898-FF3C-4548-8418-08D79555A0DB/Credential%20Provider%20Framework%20Changes%20in%20Windows%208.docx`

[20] *Credential Provider update–Windows 8 SDK breaks a few things...* [online]. 2013 [cit. 2015-05-02]. Available from: `http://blogs.msmvps.com/alunj/2013/04/07/credential-provider-update-windows-8-sdk-breaks-a-few-things/`

[21] FAIRHURST, M.C. Signature verification revisited: promoting practical exploitation of biometric technology. *Electronics.* 1997-12-01, vol. 9, issue 6, pp. 273-280. DOI: 10.1049/ecej:19970606.Available from: `http://digital-library.theiet.org/content/journals/10.1049/ecej_19970606`

[22] Windows Vista Credential Provider Samples: Five Credential Provider Samples for Windows Vista RTM and Windows Server Codename: Longhorn. *Microsoft* [online]. 2006 [cit. 2015-05-02].Available from: `http://www.microsoft.com/en-US/download/details.aspx?id=4057`

[23] FIPS PUB 180-4. *Secure Hash Standard (SHS).* Gaithersburg: National Institute of Standards and Technology, 2012.Available from: `http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf`

[24] PARK, James J. *Advances in information security and assurance: third international conference and workshops, ISA 2009, Seoul, Korea, June 25-27, 2009 : proceedings.* New York: Springer, 2009, pp. 674-678. Lecture notes in computer science, 5576. ISBN 9783642026164.

[25] Wintab Backgrounder. WACOM. *Wacom* [online]. 2010 [cit. 2015-05-03].Available from: `http://www.wacomeng.com/windows/docs/WintabBackground.htm`

[26] AVIN, Adam, Katherine GIBSON, Evan MOSSOP, Matt BLAZE a Jonathan M. SMITH. Smudge Attacks on Smartphone Touch Screens. *WOOT'10 Proceedings of the 4th USENIX conference on Offensive technologies.* 2010, n. 4, pp. 1-10. DOI: 10.1.1.176.4678.Available from: `http://static.usenix.org/events/woot10/tech/full_papers/Aviv.pdf`

[27] CORMEN, Thomas H. *Introduction to algorithms.* 2nd ed. Cambridge: MIT Press, c2001, pp. 385-392. ISBN 0-262-03293-7.

[28] SHANNON, Claude Elwood a Warren WEAVER. *The mathematical theory of communication.* Chicago: University of Illionois Press, 1998, 125 p. ISBN 978-0-252-72548-7.

[29] VIELHAUER, Claus a Ralf STEINMETZ. Handwriting: Feature Correlation Analysis for Biometric Hashes. *EURASIP Journal on Advances in Signal Processing.* 2004, vol. 2004, issue 4, pp. 542-558. DOI: 10.1155/S1110865704309248.Available from: `http://asp.eurasipjournals.com/content/2004/4/389304`

[30] YEUNG, Dit-Yan, Hong CHANG, Yimin XIONG, Susan GEORGE, Ramanujan KASHI, Takashi MATSUMOTO a Gerhard RIGOLL. SVC2004: First International Signature Verification Competition. *Biometric authentication: first international conference, ICBA 2004, Hong Kong, China, July*

15-17, 2004 : proceedings. New York: Springer, 2004, n. 1, pp. 16. DOI: 10.10079783540259480_3. Available from: http://link.springer.com/10.1007/978-3-540-25948-0_3

[31] BALTZAKIS, H. a N. PAPAMARKOS. A new signature verification technique based on a two-stage neural network classifier. *Engineering Applications of Artificial Intelligence*. 2001, vol. 14, issue 1, pp. 95-103. DOI: 10.1016/S0952-1976(00)00064-6.Available from: http://linkinghub.elsevier.com/retrieve/pii/S0952197600000646

[32] DOROZ, Rafal, Piotr PORWIK, Tomasz PARA a Krzysztof WROBEL. Dynamic signature recognition based on velocity changes of some features. *International Journal of Biometrics*. 2008, vol. 1, issue 1, pp. 47-. DOI: 10.1504/IJBM.2008.018663.Available from: http://www.inderscience.com/link.php?id=18663

[33] CARRARA, Brent a Carlisle ADAMS. You are the key: Generating cryptographic keys from voice biometrics. *2010 Eighth International Conference on Privacy, Security and Trust*. IEEE, 2010, n. 1, pp. 213-222. DOI: 10.1109/PST.2010.5593251.Available from: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5593251

Attachments : DVD contantes

The dvd disk included with this thesis has the following structure:

- Thesis.pdf - the electronic version of this thesis.
- readme.txt - usage instruction
- Installers
  - * Demo Mode - Installer of signature authentification demonstrate mode
  - * Windows 7 (V1 credential provider) - Instaler of V1 signature credential provider
  - * Windows 8 (V2 credential provider) - Instaler of V1 signature credential provider
- TestData - Signatures of 40 subjects used in SVC 2004: First International Signature Verification Competition
- SignatureCredentialProvider soucrce codes - source codes of our signature credentil solution
- Programing documentation - Programing documentation of our solution