A Unified Framework for Mobile Device Security

Wayne Jansen
The National Institute of Standards and Technology
Wjansen@nist.gov

Serban Gavrila
VDG, Inc.
Serban.Gavrila@nist.gov

Vlad Korolev
Booz-Allen Hamilton
Vkorolev@nist.gov

Thomas Heute, Clément Séveillac
The National Institute of Standards and Technology
{Thomas.Heute, Cseveillac}@nist.gov

Abstract: Present-day handheld devices, such as PDAs, are a useful blend of hardware and software oriented toward the mobile workforce. While they provide the capability to review documents, correspond via electronic mail, manage appointments and contacts, etc., they typically lack a number of important security features. Concerned individuals and organizations aware of the associated risks involved, mitigate them with such add-on mechanisms as improved user authentication, content encryption, organizational policy controls, virus protection, firewall and intrusion detection filtering, and virtual private network communication. Unfortunately, such piecemeal solutions often present problems in software integration, usability, and administration. This paper describes a framework for incorporating core security mechanisms in a unified manner that avoids these problems.

Keywords
Mobile Security, User Authentication, Policy Enforcement, File Encryption, Security Frameworks

1. Introduction
The use of mobile handheld devices within the workplace is expanding rapidly. These devices are no longer viewed as coveted gadgets for early technology adopters, but have instead become indispensable tools that offer competitive business advantages for the mobile workforce. While these devices provide productivity benefits, they also pose new risks to an organization's security, not only from the sensitive information held and the organizational networks accessible by them, but also from their propensity to become physically separated from the user.

Adequate user authentication is the first line of defense against unauthorized use of a lost or stolen handheld device. Multiple modes of authentication increase the work factor needed to compromise a device; however, very few devices support more than one mode, usually password-based authentication. Moreover, integrating a second mode of authentication under an operating system, especially a proprietary one, can be a daunting task [Mic03].

With enough time and effort, any authentication mechanism can be overcome or circumvented, especially if weaknesses are present [Kin01]. Content encryption serves as a second line of defense, opening the information repository to only those individuals with the correct cryptographic key. A cryptographic key can be determined dependently or independently of an authentication event. For example, for a user who authenticates with a USB or MMC smart card, the key for decrypting content could be contained on the card. Alternatively, for a user who authenticates with a biometric, the key could be determined subsequently from a separate graphical [Blo96, Jer99, sfr00] or textual password entry dialogue, using password-based encryption to form the key [Atr00].

Mobile devices lie at the periphery of an organization's infrastructure, which makes them difficult to administer. While mobile computing opens up new application areas, it also introduces new vulnerabilities. To reduce or eliminate common risks, a security officer requires the means to express, monitor, and enforce organizational security policies effectively, particularly over external communications and interfaces. Policies should not only restrict and filter external communications, but also constrain user privileges on handheld devices [Jan03a, Poi02, Uti03]. When implemented correctly, security policy management mechanisms can be applied to govern user behavior automatically and unobtrusively.

This paper describes a general framework for PDA security that incorporates multi-mode user authentication, content encryption, and policy enforcement in a unified fashion. The approach requires the application of organizational security policies, organized into distinct policy contexts known as echelons, among which a user may transition. Besides the policy context, an echelon can be associated with one or more authentication steps and with a distinct repository of encrypted information. The approach aims at helping users easily comply with their organization's security policy, yet be able to exercise a significant amount of flexibility and discretion. The design of the framework allows various types of authentication technologies to be incorporated readily and provides a simple interface for supporting different types of policy enforcement mechanisms. The goal is to provide a sound basis for the security of devices issued organizationally in sectors such as medical, financial, defense, and law enforcement.

2. Concept of Operation

For this discussion, only single-user systems apply. That is, the user is considered the sole operator of the device, once it is issued. Policy rules govern both the behavior of the user and the device.

An organizational security policy may involve several sets of policy rules that are organized into echelons. Conceptually, echelons are graded into sensitivity levels, level 1 being the least sensitive and level 3 being the most sensitive. Level 0 represents the most restrictive policy for device lockdown, and is the starting point whenever a device is powered on or rebooted. While the framework allows the user discretion in selecting among echelon levels at which to operate, it also can impose one or more authentication steps, where needed, before permitting a transition to a higher echelon level. Thus, authentication steps are cumulative and hierarchical – all lower level authentication steps plus those required at some desired level must be successfully completed to reach the desired level. Though the authentication steps are hierarchical, the policies at each echelon are independent of one another from the perspective of the framework. The framework also allows a distinct encrypted information repository to be associated with any echelon above level 0 and its contents to be made available once a successful transition to that level occurs.

The implementation of the framework uses our own policy enforcement engine to govern behavior [Jan03a]; however, its design allows it to interface with other policy enforcement engines that may be available, such as the Linux Intrusion Detection System or Security Enhanced Linux. The implementation also supports up to thirty-one echelon levels, if needed. In practice, however, three levels or less have normally been sufficient. At its simplest, the framework can support a single authentication mechanism and an associated policy, comparable to present-day PDA configurations.

Figure 1 gives an example of a 3-echelon configuration, where each echelon comprises a distinct set of policy rules, prerequisite authentication steps, and an optional security repository. For level 1, authentication steps 1A and 1B are needed. For level 3, authentication step 3A is needed, and the condition for authentication steps 1A and 1B must still hold. A successful transition to level 3, allows the user to gain access to a repository of encrypted files. No additional authentication steps are needed to transition from level 1 to level 2, though the condition for steps 1A and 1B must still hold. Transition among levels is initiated at the discretion of the user. Though echelon levels range from low to high to differentiate escalating sensitivity, hierarchical policy rules are not a requirement.

	Required Authentication	Effective Policy	Cryptographic Repository
L3 - High	3A	Policy for L3	Available
L2 - Medium	None – user choice	Policy for L2	Unavailable
L1 - Low	1A, 1B	Policy for L1	Unavailable
L0 - Locked	None – default at power on and boot up	Most Restrictive	Not Applicable

Figure 1: Echelon Example

That is, after successfully completing a number of authentication steps, the user may be granted more or less restrictive permissions than at a lower echelon level. For example, the policy at a higher level may disable communications that were available at a lower level, but be granted access to additional applications that were previously unavailable.

The framework supports both polled and non-polled forms of authentication. Non-polled authentication is resolute – once the verdict is determined, it is inviolate until the next authentication attempt. Examples of non-polled authentication include passwords, fingerprints, and voice verification. Polled authentication on the other hand is irresolute – once the initial verdict is determined, the status can change based on the absence or presence of some logical or physical token involved in reaching the initial verdict. Examples of polled authentication include smart cards, security tokens, and communications signals (e.g., a trusted beacon), whereby the absence of the device or signal triggers a non-authenticated condition.

3. Transition Flow

Figure 2 illustrates the transition flow between echelons for the example configuration. The darkened circles represent the echelon levels 0 through 3 and the lightened circles represent any required authentication steps. The arrows represent the transitions among levels and authentication steps. For clarity, a box circumscribes the set of authentication steps needed to be accomplished to transition between adjacent levels, and apply cumulatively between non-adjacent levels. Some transitions can occur manually (i.e., Man Tran) or automatically (i.e., Auto Tran), and are labeled accordingly.

On reboot and power on events, the system begins at the default level, level 0, and automatically attempts to transition the user to echelon level 1. Level 1 requires the successful completion of two authentication steps for entry: 1A and 1B. If any of the authentication steps fail, the user remains at level 0 and the system automatically reattempts to transition again to level 1. However, access is blocked for a period of time, as a penalty for failing to successfully transition there in the previous attempt. Similarly, if the user successfully transitions to level 1, but the status of a polled authentication step used to reach that level changes (e.g., 1B authenticated the user through a token, which is then removed), triggering a non-authenticated condition, the user is returned to level 0. The system then automatically reattempts to transition the user to level 1, after blocking access for a specified period of time.

Blocking access during a transition attempt for which an authentication failure previously occurred is done by bracketing the execution of those authentication steps with pre- and post-authentication steps (e.g., 1^p and 1_p), designated respectively with superscript and subscript letter P. The pre- and post-authentication steps are a useful but optional technique for the framework. These pseudo authentication steps work in tandem to allow authentication failures to be dealt with collectively by the framework, rather than individually by each authentication mechanism.

To understand how this works, an explanation of "handlers" is needed. Handlers are code modules that carry out each authentication step, such as 1A or 1B. One handler exists for every authentication step and also for the pre- and post- authentication steps, if used. Because handlers are synonymous with authentication steps, the same name assigned to one is normally used to refer to

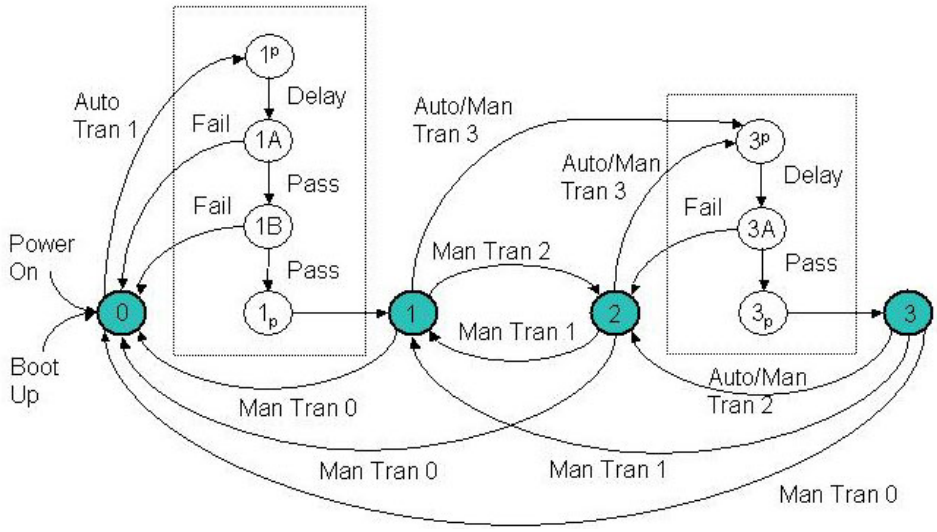

Figure 2: Transition Flow

the other (e.g., pre-authentication step 1^p versus pre-authentication handler 1^p or, in shorthand, pre-handler 1^p).

The pre-handler maintains a penalty file where the number of failures is recorded. If the file does not exist, it creates the file and sets the value to 1, in anticipation of an upcoming failure. Otherwise, failures have previously occurred, and the pre-handler imposes a delay penalty commensurate with the recorded value, before incrementing the value by 1. The delay penalty can be programmed for linear, exponential, or any other scheme the implementer chooses, including total lockout of a level. If the authentication steps proceed successfully, the post-handler deletes the penalty file when it executes to clear out the count. This approach allows the number of attempts for a user to authenticate to be controlled independently at each authentication step by the associated handler, yet a penalty to be applied collectively for the entire set of authentication steps by the pre- and post-handlers.

Once echelon level 1 is reached, transitioning among the remaining levels is done at the user's discretion. Downward transitions do not involve any authentication steps. It may require the successful completion of one or more additional authentication steps, however, to make the transition upwards to a higher level. All of the sets of authentication steps from the current echelon level up to the requested level, including any intervening levels, are initiated in their logical sequence. If any of the authentication steps fail, the user transitions to the highest level permitted, based on authentication condition of those steps that still hold.

For the example configuration, transitioning to level 3 can be attempted from either level 1 or 2, and requires the successful completion of one additional authentication step, 3A. A user attempting to transition from level 1 to level 3 could attain level 2, should the user successfully complete all authentication steps for level 2 (i.e., in this case none), but fail the authentication step for level 3. However, because the user unsuccessfully attempted a transition to level 3, access will be blocked for a period of time on the next transition attempt to that level. Note that the blocking is done selectively, permitting the user to continue operation at level 2, as the penalty time for level 3 expires. After the user successfully transitions to level 3, a cryptographic repository is deciphered and its contents are made available.

While the user typically initiates a transition among echelon levels, the framework also allows the possibility for polled authentication mechanisms to request that a transition attempt be initiated. For example, the presence of a smart card in the smart card reader could be made to trigger the system to attempt a transition to the level for which this authentication step is associated. This facility is fairly intuitive and appropriate for most authentication mechanisms; however, it could be troublesome for tokens that are not under the control of the user. For example, if the presence of a signal broadcast from a trusted beacon is used as one of the authentication steps for completing a transition to a higher level, an unwanted transition attempt might occur. The framework allows the user to control whether a polled authentication mechanism can initiate a transition attempt, by providing the option to cap the level from which requests from polled handlers are enacted upon by the system, so that those requests are ignored.

4. High-Level Design

The design of the multi-mode authentication solution involves four main types of components: kernel modules, authentication handlers, user interface (UI) components, and the level selector. Figure 3 below illustrates the different components of the solution and the flow of data between them. The echelon level selector is the user's control window for the framework. It allows the user to transition among echelons and to exert control over the behavior of some polled authentication mechanisms, such as a communications signal, essentially preventing them from attempting to transition automatically up to a level that the user does not yet want to attain. The level selector is visible as an active icon in the system tray that shows the current status as an accumulative stack of colored bars (i.e., locked, low, medium, and high), and expands to a full interaction window when selected.

The user interface for the various authentication mechanisms is implemented as components of a plug-in module. Their function is to interact with the user, for example, to accept a password or prompt the user to insert a smart card. The plug-in module supports a socket interface to receive commands from the authentication handler components that run as separate processes, and route the commands to the correct interior user interface component. Similarly, the reverse process is also supported between components and the module for responses.

As mentioned earlier, handlers embody the mechanism that performs the actual authentication. They interact with the user interface components to tell them to bring up the specific screens, accept input, display messages, etc. They also communicate with any peripheral hardware devices needed for authentication, such as a security token, and access the file system to store and retrieve information as needed. For example, for the picture password visual authentication mechanism shown in Figure 3 [Jan03b], the handler uses files containing the theme identifier, button code mapping, and the cyphertext password to verify the sequence of user-selected images. Handlers communicate with the kernel module, listening

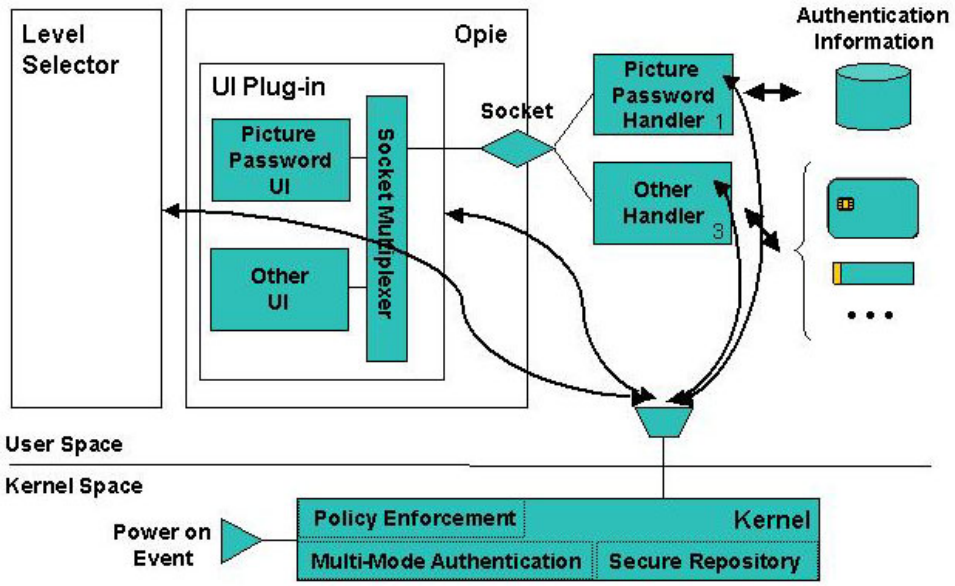

Figure 3: Top-Level Components

for when to initiate authentication, and reporting back whether authentication was successful.

The kernel has been augmented with three key modifications: the multi-mode authentication functionality, the secure repository functionality, and the policy enforcement functionality. Policy enforcement's main responsibility is to impose different sets of policy rules on the device, as signaled by multi-mode authentication. For example, it blocks the I/O ports on the device and other means to bypass the authentication sequence until the user is authenticated at level 1. It also protects authentication information files, the user interface and handler components, and policy enforcement information. Moreover, it also has the means to register and start up registered components if they are not running or restart them if they terminate for some reason, which is used for the authentication handlers.

The main responsibility of multi-mode authentication functionality within the kernel is to govern the authentication steps as they relate to the various echelon levels that are configured. It is the source of all knowledge about the mappings between authentication mechanisms and echelon levels, simplifying the complexity of the authentication handlers that carry out the authentication steps. One of its key functions is to initiate user authentication when the device is powered on. It also controls the order and frequency in which the handlers are awakened from suspended state and begin execution, and ensures that messages from only recognized handlers are accepted and processed.

The communication between the kernel and either an authentication handler or the echelon selector is done via the /proc file system. There are several different messages that can be communicated. Overall, two classes of messages exist: those related to authentication and those related to echelon transitions. The purpose of these messages is described below. To simplify developing handlers, a common messaging programming interface was established.

- *Handler Ready* - When a handler process is ready to perform authentication it signals the kernel with this message. When the kernel receives this message it first verifies that this process is indeed a registered handler and, if it is, it puts the process on a wait queue until there is a need to perform the authentication step.

- *Authenticate* - The kernel uses this message to wake up authentication handlers and have them perform an authentication step. The handlers are awakened one at a time in the same order they occur in the handler's table.

- *Poll* - This message is used by the kernel to wake up authentication handlers periodically to have them check for the presence of a token.

- *Error* - With this message, the kernel signals a specific handler to exit.

- *Authentication Failed* - This message is conveyed to the kernel to indicate a failed authentication attempt.

The kernel verifies that the message comes from a registered handler.

- *Authentication Successful* - This message is conveyed to the kernel to indicate a successful authentication attempt. The kernel checks if this message indeed comes from the handler that is supposed to be running at a present time. When all of the registered handlers report successful authentication at some level, the kernel transitions the device accordingly.

- *Level n* - This message is conveyed to the kernel from the echelon level selector and indicates that a transition to echelon level n should be attempted. An authentication handler may also use the message in special situations (e.g., a polling handler sensing the presence or absence of a token and needing to trigger a transition).

- *Maxl n* - This message is conveyed to the kernel from the echelon level selector and indicates that the maximum echelon level for automatic transition attempts should be set to n.

- *Key k* - This message is conveyed to the kernel from a handler to convey the cryptographic key for opening the cryptographic repository for the level at which the handler is operating. When the kernel receives this message it verifies that this process is a registered handler and uses the key to open the repository.

5. Conclusions

While mobile devices provide productivity benefits, they also pose new risks. This paper demonstrates that a unified framework for incorporating core security mechanisms in mobile devices is both possible and practical. The approach provides users the flexibility to switch among security contexts to perform their tasks, yet ensures that organizational policy can be administered and enforced at the edges of the organization.

References

[Atr00] Mohan Atreya, Password Based Encryption, <URL: http://www.rsasecurity.com/products/bsafe/whitepapers/Article3-PBE.pdf>.

[Blo96] Greg E. Blonder; Graphical password, US Patent 5559961, Lucent Technologies Inc., Murray Hill, NJ, August 30, 1995.

[Jan03a] Wayne Jansen, Tom Karygiannis, Michaela Iorga, Serban Gavrila, and Vlad Korolev, Security Policy Management for Handheld Devices, The 2003 International Conference on Security and Management (SAM'03), June 2003.

[Jan03b] Wayne Jansen, Serban Gavrila, Vlad Korolev, Rick Ayers, and Ryan Swanstrom, Picture Password: A Visual Login Technique for Mobile Devices, NISTIR 7030, July 2003, <URL: http://csrc.nist.gov/publications/nistir/nistir-7030.pdf >.

[Jer99] Ian Jermyn, Alain May, Fabian Monrose, Michael Riter, and Avi Rubin, The Design and Analysis of Graphical Passwords, Proceedings of the 8th USENIX Security Symposium, August 1999.

[Kin01] Kingpin and Mudge, Security Analysis of the Palm Operating System and its Weaknesses Against Malicious Code Threats, USENIX Security Symposium, August 2001.

[Mic03] Let Me In: Pocket PC User Interface Password Redirect Sample, Microsoft Knowledge Base Article – 314989, Microsoft Corporation, July 2003, <URL: http://support.microsoft.com/default.aspx?scid=kb;en-us;314989>.

[Poi02] Pointsec for Pocket PC, Pointsec Mobile Technologies, November 2002, <URL: http://www.pointsec.com/news/download/Pointsec_PPC_POP_Nov_02.pdf>.

[sfr00] visual Key – Technology, sfr GmbH, 2000, <URL: http://www.viskey.com/technik.html>.

[Uti03] SafeGuard PDA, Utimaco Safeware AG, March 2003, <URL: http://www.utimaco.com/eng/content_pdf/sg_pda_eng.pdf>.