Device-Enabled Authorization in the Grey System[¶]

Lujo Bauer[*]
Michael K. Reiter[* † ‡]
Scott Garriss[†]
Jason Rouse[*]
Jonathan M. McCune[†]
Peter Rutenbar[§]

February 2005
CMU-CS-05-111

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

We describe the design and implementation of Grey, a set of software extensions that convert an off-the-shelf smartphone-class device into a tool by which its owner exercises and delegates her authority to both physical and virtual resources. We describe the software architecture and user interfaces of Grey, and then detail two initial case studies in which we have converted infrastructure to accommodate requests from Grey-enabled devices. The first is two floors (nearly 30,000 square feet) of office space, in which we are equipping over 65 doors for access control using Grey for a population of roughly 150 persons. The second is modifications to Windows XP that permit login via Grey-enabled phones. We provide preliminary evaluations of these efforts and directions for research to further the vision of a unified authorization framework for both physical and virtual resources.

[*]CyLab, Carnegie Mellon University, Pittsburgh, PA, USA
[†]Electrical & Computer Engineering Department, Carnegie Mellon University, Pittsburgh, PA, USA
[‡]Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, USA
[§]Winchester Thurston High School, Pittsburgh, PA, USA

[¶]We gratefully acknowledge support from the National Science Foundation grant number CNS-0433540, the U.S. Navy grant number N00014-04-1-0724, and the U.S. Army Research Office contract number DAAD19-02-1-0389.

Keywords: security, access control, mobile devices, smartphones

1 Introduction

Access control today is characterized by an expanse of mechanisms that do not interoperate and that are highly inflexible. Access to physical resources (e.g., home, office) is most commonly tied to the possession of a hardware key, and in office environments possibly a swipe card or proximity/RFID card. By contrast, access to virtual resources is most commonly tied to the knowledge of a password and/or possession of a physical token (e.g., SecureID) for producing time-varying passwords, particularly in the case of virtual private network access.

In this paper we introduce the Grey system, which utilizes converged mobile devices, or "smartphones", as the technology of choice for unifying access control to both physical and virtual resources. We focus on smartphones for two central reasons. First, their nearly ubiquitous adoption is inevitable: Gartner predicts shipment of 20 million smartphones already in 2006 (versus 13 million for PDAs) [33], and in the long term they stand to inherit the vast cellular phone market, which in 2004 shipped over 648 million units [45], or more than one phone per ten people in the world.[1] Second, the hardware capabilities of smartphones and the maturity of application programming environments for them have advanced to a stage that enables applications to take full advantage of rich computation, communication, and interface capabilities (e.g., a camera).

It is this convergence of market trends and technological advances that points to a not-too-distant future marked by pervasive adoption of highly capable and always-in-hand smartphones. Grey is an effort to enhance this platform to build a ubiquitous access-control technology spanning both physical and virtual resources. This vision is not ours alone; e.g., several groups have experimented with the use of mobile phones as digital keys [11, 38]; NTT Docomo is conducting trials on the use of mobile phones to authorize entry to apartments[2]; and mobile phones can already be used to purchase items from vending machines in several countries. However, to the extent that the capabilities of these systems can be inferred, we believe that Grey presents a more sound and flexible platform for building a ubiquitous access-control system and, eventually, for experimenting with advanced mobile applications.

As an example of the type of flexibility not possible in these other solutions, with Grey a user will be able to easily create and lend to her friend a temporary, virtual key to her car or apartment; this will happen seamlessly regardless of whether the user and her friend are standing next to each other or thousands of miles apart. Similarly, a manager could give to her secretary temporary access to her email without revealing any information (such as passwords) that could be used at a later time or to access a different resource. Going further, a user could specify, for example, that his office may be accessed by any three of his colleagues acting together, but at least three would have to cooperate to gain access. A detailed usage scenario will be given in Section 4.

The Grey platform is a novel integration of several technologies that results in a single tool for exercising and delegating authority that we believe is far more secure, flexible and usable than any alternative available today. At the core of Grey is a flexible and provably sound authorization framework based on *proof-carrying authorization* (PCA) [5, 9], extended with a new distributed proving technique that offers significant efficiency and usability advances [10]. In addition to enabling a user to exercise her authority, PCA provides a framework in which users can delegate authority in a convenient fashion. These techniques rely on sound cryptographic key

[1] The July 2004 estimate of world population by the *The World Factbook* (see `http://www.cia.gov/cia/publications/factbook/geos/xx.html`) is just over 6.379 billion.

[2] `http://www.i4u.com/article960.html`

management. For protection of phone-resident cryptographic keys in the event of phone capture, Grey incorporates *capture resilience* [29, 30]. And, on the user-interface front, we employ a technique for conveying key material and user identifiers in support of access management, and even for conveying network addresses, that is as simple as taking a picture with the phone's built-in camera [32, 44]. Phone-to-phone and phone-to-infrastructure data communication is conducted through an asynchronous messaging layer that we have developed to take advantage of the myriad networking technologies available to modern smartphones, including Bluetooth, cellular data service (e.g., GPRS), and messaging protocols (e.g., SMS and MMS).

In this paper we survey these component technologies, their implementations in Grey, and their performance on modern smartphones. In addition, we detail two applications of Grey that we are presently developing for deployment starting in March 2005, which coincides with the completion of a new building on our university campus. One of the applications involves instrumenting the physical space of this building, roughly 30,000 square feet, with infrastructure needed to control access to approximately 65 doors by roughly 150 people carrying Grey-enabled phones. A second application involves the use of Grey for accessing Windows XP sessions. In both of these applications, we argue that Grey offers a more secure, flexible and convenient basis for access control than existing solutions.

2 Related Work

Biometrics are one class of technologies commonly advocated for unifying access control across resources. Like a password or PIN, however, biometrics are exclusively a user-authentication technology, providing no inherent capability for computation or storage. As such, they are useful only for authenticating a user to a trusted platform where computation (e.g., cryptographic operations, policy creation and evaluation) can be done; indeed, our focus in this paper is designing such a platform and supporting architecture.

A smartcard (or RFID card) could serve as this platform, but there are numerous reasons why we believe smartphones offer a superior solution. First, a smartcard's inability to interact with the user directly (due to a lack of a keypad and display) typically limits its use to performing cryptographic operations, and indirectly to a second factor for two-factor user authentication. Richer uses have driven researchers to advocate the extension of smartcards with entry keypads and displays (e.g., [34]) or to abandon the smartcard format and move to PDA-class devices (e.g., [8]). Our work can be viewed as a continuation of this trend, highlighting the use of modern smartphones with additional capabilities still (e.g., a camera, diverse communication capabilities, and significant computational power) and their application to access control. Second, trends in smartcard technology have not kept pace with smartphones, in that smartphones now offer dramatically better capabilities and programming environments. Third, smartcard adoption pales in comparison to that of mobile phones, and this gap will only widen in the future. Finally, while a smartcard's tamper-resistance is often cited as a necessary feature, the communication and user-input capabilities of the smartphone enable us to nullify this advantage for a wide range of attacks through the use of capture resilience [29, 30].[3]

[3]We do not intend to overstate this case, however: due to its emphasis on security, a smartcard may offer physical protections against side-channel attacks, notably electromagnetic analysis [36], whereas defending against these attacks on commodity smartphones would presumably require significant modifications to the cryptographic algorithms.

Our use of smartphones to control physical environments evokes visions of *ubiquitous* or *pervasive* computing, and in this context there have been a number of works on access control, e.g., due to Al-Muhtadi et al. [3, 4], Covington et al. [16, 15], Hengartner and Steenkiste [23], Tripathi et al. [46] and Wullens et al. [48]. These works primarily focus on accommodating novel types of context in access control (e.g., a user's current activities and location) that are expected to be available in future pervasive-computing environments. More importantly from our perspective, these efforts are largely agnostic to the vehicle by which the user interacts with the computing environment. Thus, these efforts focus mostly on infrastructure in which policy is created, stored and/or interpreted, in some cases using centralized servers. In contrast, our focus is the vehicle (the smartphone) itself and its supporting software and architecture, and the decentralized creation, storage and interpretation of policy it enables.

There are fewer works focused on user devices and their use for exercising authority in pervasive-computing scenarios. Perhaps the most closely related such works are those of Beaufour and Bonnet [11] and Ravi et al. [38]. Both consider the use of smartphones for implementing physical access control (c.f., Section 6.1), and Ravi et al. also provide an approach for integrating payment and access control with service discovery via smartphones. However, neither develops a general framework for implementing access control across applications or offering convenient delegation, as we do here. Moreover, our advances in user interfaces for smartphones, and our development of techniques to defend the cryptographic keys they hold, have no parallel in their works. More distantly related work is due to Sailer and Giles, who develop protocols for a single user device, called the Personal Authentication Gateway, to temporarily permit other devices to utilize the user's authority [42] so that, e.g., the user's watch could display information from a service for which the gateway holds credentials. Multiple user devices are not required in our work, though if a user employs multiple devices, then her authority can easily be transfered among these devices as a consequence of the general authorization framework we employ.

3 Component Technologies

Grey is a novel integration of a number of recently-developed technologies that utilize the capabilities of modern smartphones. In the interest of providing background to the reader, we summarize these component technologies here.

3.1 Graphical Identifiers

A common feature of modern smartphones is a camera. In Grey we utilize this camera as a data input device for the smartphone, i.e., by asking the user to take a picture of an item or person she intends to interact with. Information conveyed by photographing two-dimensional barcodes is a theme common to several ubiquitous computing efforts (e.g., [28, 31, 40]), typically to convey service information or a URL where such information can be obtained. Our use of the camera in Grey is not limited to (but does include) two-dimensional barcodes, and is primarily a user-friendly means for importing identifiers into the system. More specifically, there are three types of identifiers that are commonly input via the camera:

- **An identifier for a person**. A photograph of a person is used as an identifier for the person, in lieu of or in addition to a text string name, in credentials such as certificates (c.f., [43]).

In the absence of unique and standardized names for all persons, pictures provide a less failure-prone way for identifying a person. For example, a user can have more confidence in a credential giving rights to a person with a picture that she recognizes, than in one giving rights to the string "Bob", which may be interpreted as different persons by different people. Of course, in some cases this picture should be accompanied with text (e.g., "Bob") to disambiguate persons with similar appearances (e.g., twins).

- **An identifier for a public key**. A useful identifier for a key is the collision-resistant hash of the key (e.g., [26]). In Grey, a two-dimensional barcode is used to encode the hash of a public key and can be displayed on a sticker attached to an item (e.g., on a door) or, for a device with a display (e.g., smartphone or computer), presented on the display. A camera-equipped smartphone can then be used to photograph this identifier and authenticate the public key obtained by any other means (e.g., over a wireless link) [32]. This provides a more natural and user-friendly way for obtaining an authentic public key than other alternatives.

- **A network address**. Similar to encoding a hash of a public key in a two-dimensional barcode, a barcode can also be used to encode a network address. As above, a camera-equipped smartphone can then obtain the network address by photographing the barcode. This idea has been utilized in order to circumvent high-latency device discovery in Bluetooth [44], and we use it similarly in Grey. In addition, this idea offers similar usability advantages to that above, as it is an intuitive operation for a user to photograph the device with which she intends to communicate.

The pervasiveness of graphical identifiers in Grey lends itself well to purely graphical management interfaces for collecting identifiers and managing access. We will provide an overview of the interfaces we have developed in Section 5.

3.2 Capture-Resilient Cryptography

A user's Grey-enabled smartphone utilizes a private signature key in the course of exercising the user's authority. The capture of a smartphone thus risks permitting an attacker who reverse-engineers the smartphone to utilize this private key and, as a result, the user's authority. To defend against this threat, Grey *capture protects* the phone's private key [29, 30]. At a high level, capture protection utilizes a remote *capture-protection server* to confirm that the device is being held by the person who initialized the device (e.g., using a PIN, face recognition via the phone's camera, or other biometric if the phone supports it), before it permits the key on the phone to be used. This server can also disable the use of the key permanently when informed that the device has been lost, or temporarily to protect the key from an online dictionary attack on the PIN (or other authentication technique). At the same time, this capture-protection server is untrusted in that it gains no information about the user's key.

In keeping with the theme that Grey is a wholly decentralized system, the capture-protection server is not a centralized resource. That is, each user can utilize her own capture-protection server (e.g., her desktop computer), and indeed there is no management required of this server in the sense of establishing user accounts. Rather, this server need only have a public key that is made available to the user's phone when the phone's key is created—perhaps by taking a picture of it

displayed on the server's screen, as described in Section 3.1—and must to be reachable when the phone needs to utilize its private key.

A concern that arises with the use of a phone for exercising personal authority is the sheer inconvenience of losing one's phone, in the sense of being unable to exercise one's own authority. While this can occur with any form of access control that utilizes a token or other hardware, we note that capture protection provides a remedy. Since the capture-protection server ensures that a key can be used only by a device in possession of the person present when the key was created, a user may back up her key material with little risk of exposing it in an indefensible way.

3.3 Proof-Carrying Authorization

Underpinning any distributed access-control system must be a scheme for expressing and reasoning about authority. In computer systems, the key building block of such a scheme is the digital certificate, by which a signer expresses his belief that the content of the certificate is true. The typical use of digital certificates is to bind a person's name to her public key [24], but certificates can be used as arbitrary credentials, including to express the signer's intention or permission to access a resource, to delegate authority, and to describe resources.

Prior research in distributed authorization has produced a number of systems [39, 18, 19, 12] that provide ways to implement and use complex security policies that are distributed across multiple entities. The process of gaining access to a resource typically involves locating and gathering credentials and verifying that a set of credentials satisfies some access-control policy. Both the gathering and the verification is typically carried out by the entity or host that is trying to decide whether to allow or deny access.

These credentials and the algorithms for deciding whether a set of credentials satisfies some security policy can be described using formal logics (e.g., [1, 21]). Such logics provide a foundation for expressing rich access-control mechanisms including roles, groups, delegation, etc., as extensions of a few basic concepts, including a fundamental *speaks-for* relation between principals [26]; for example, a certificate might indicate the public key that "speaks for" a named person. In early work in this vein, the design of access-control systems starts with the specification of a security logic, after which a system is built that implements as exactly as possible the abstractions and algorithms that the logic describes [47, 7]. While this approach can dramatically increase confidence in the systems' correctness [2], at best the system *emulates* the access-control ideal as captured in the formal logic. That is, since the correspondence between the formal logic and the implementation is only informal, any guarantees derived from the formal logic might fail to extend to the implemented system.

An alternative introduced in the concept of *proof-carrying authorization* (PCA) [5, 9] is to utilize this formal logic directly in the implementation of the system. In PCA the system directly manipulates fragments of logic that represent credentials; the proofs of access are likewise constructed directly in formal logic. This integration of formal logic into the implemented system provides increased assurance that the system will behave as expected. This is the high-level approach that we adopt in Grey. As such, each Grey component (including a smartphone) includes an automated theorem prover for generating proofs in the logic, and a checker for verifying proofs.

A fundamental tension in access control is that the more expressive a system is (that is, the greater the range of security policies that its credentials can describe), the more difficult it becomes to make access-control decisions. To ensure that the access-control decision can always be made,

most systems restrict the range of security policies that can be expressed, ruling out many potentially useful policies. Since Grey is meant to be used in a highly heterogeneous environment and supports ad-hoc creation of policy components, this type of inflexibility could be very limiting. An insight behind PCA is that the access-control policy concerning any particular client is likely to be far simpler to reason about than the sum of all the policies of all clients. PCA takes advantage of this insight by making it the client's responsibility to prove that access should be granted. To gain access, a client must provide the server with a logical proof that access should be allowed; the server must only verify that the proof is valid, which is a much simpler task. The common language in which proofs are expressed is a higher-order logic [13]; when constructing proofs, each client uses only a tractable subset of the higher-order logic that fits its own needs. The mechanism for verifying proofs is lightweight, which increases confidence in its correctness [6] and also enables even computationally impoverished devices to be protected by Grey.

4 A Usage Scenario

Grey's integration of the technologies described in Section 3 (and others) enables a range of interactions that enhance access control to render it more user friendly, decentralized and flexible. To illustrate this, we describe an example scenario that utilizes several of the pieces we have introduced.

The scenario we consider begins with two researchers, Alice and Bob, who meet at a conference and begin a research collaboration. Anticipating communicating electronically when they return to their home institutions, each enters the other in his/her smartphone "address book". The address book interface is such that creating the entry for Bob requires Alice to snap two pictures with her smartphone: one of Bob and one of a two-dimensional barcode displayed on his smartphone that encodes Bob's public key (or a collision-resistant hash thereof). Bob does similarly. After Alice returns to her home institution, her phone automatically synchronizes its address book with her PC. This could permit her, for example, to authenticate electronic mail from Bob using standard protocols (e.g., [37]).

Figure 1: Bob and Alice exchange public keys. The phones' displays and cameras serve as a secondary validation channel.

As their submission deadline approaches, Alice and Bob decide that a face-to-face meeting would be warranted, and so Bob makes plans to visit Alice. On the day that Bob arrives at Alice's institution, Alice is delayed at home. Bob thus arrives to Alice's locked office door. Inside the glass next to Alice's door is a barcode sticker that encodes the Bluetooth address of a computer that can actuate Alice's door to open, if convinced to do so. Bob photographs the barcode, prompting his smartphone to connect to the computer, which challenges Bob's phone to prove his rights to access the door—a feat which his phone cannot do alone, since Bob lacks the needed credentials. The

theorem prover in his phone, however, discerns that Alice's phone could assist, and initiates a communication with it.

Upon receiving Bob's phone's request, the theorem prover in Alice's phone automatically generates several options by which Alice can permit Bob to enter the door, based on credentials that she has previously created and that are stored in the phone: she can (i) simply grant him a credential to open the door only this time; (ii) add him to a group `visitors` that she previously created and granted rights to, among other things, open her door; or (iii) give him the rights of her secretary, to whom she also granted the ability to open her door. Alice's phone presents this list to Alice, who selects (ii). The phone then signs a credential to this effect and returns it to Bob's phone, enabling it to complete the proof of access.

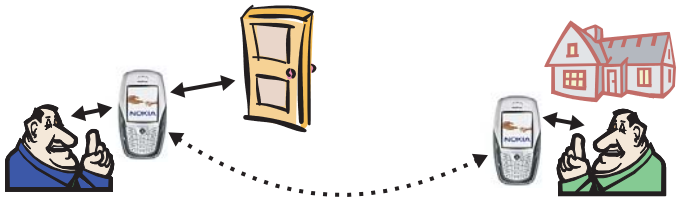

Figure 2: Bob entering Alice's office. In the course of proving access, Bob's phone contacts Alice's phone for help.

It is worthwhile to reflect on the presentation of this process to each of Alice and Bob. Bob, upon photographing the door barcode, is asked to enter a PIN in order to utilize his private key to sign a request to open the door—an operation protected by capture protection; see Section 3.2—and the door opens with no further interaction (albeit with some waiting while Alice makes her decision). Alice is consulted merely with a list offering her several options by which she can permit Bob to enter her office. Upon selecting one and also typing her PIN—again to activate her capture-protected key—her task is completed.

Bob's credential indicating that he is a member of Alice's `visitors` group turns out to be handy while he awaits Alice's arrival. In addition to permitting him to open Alice's office, it could grant his laptop access to the campus 802.11 network, to the floor printer, and to a back room where there is a vending machine with snacks and sodas. All these privileges are afforded to Bob due to Alice's prior creation of credentials that grant these privileges to her `visitors`.

5 Software Architecture

At a high level of abstraction, every Grey host or device is composed of some subset of the following elements: a compact and trustworthy *verifier* that mediates access to a protected resource; an extensible *prover* that attempts to construct proofs of access; a lightweight, asynchronous *communication framework* that facilitates the distributed construction of proofs and management of certificates; and a collection of *graphical interfaces* that allows the convenient and seamless integration of Grey into everyday life. Grey is implemented in Java, which allows it to easily extend across multiple platforms (workstations, smartphones, embedded PCs, etc.) and operating systems.

5.1 Graphical User Interfaces

An emphasis in Grey is usability. In this subsection we describe the primary user interfaces involved in Grey at the time of this writing.

In order to maximize our user population, we have targeted Grey for the widest range of smartphones possible, including those of modest size—and correspondingly modest screen size. For example, our primary development platform to date has been the Nokia 6620, a smartphone with dimensions $4.28 \times 2.29 \times 0.93$ inches and a 176×208 pixel display. Due to the limited screen size on this class of smartphones, we have divided tasks into those performed on the phone by necessity, and those that can be offloaded to a companion tool run on a personal computer, after which the necessary state can be transferred to the phone via a synchronization operation. At a high level, tasks such as the creation of groups and roles (as defined in [26]), and proactive policy creation, are offloaded to the companion tool. Because these tasks are standard in a variety of access-control settings, here we focus on the phone-resident interfaces, as these are the ones that we believe to be more innovative.

The tasks performed on the smartphone with user interaction include: collecting identifiers (of persons, keys, or addresses) and creating speaks-for relationships [26] among them to create an "address book"; making an access request to a resource; and reactive policy creation, i.e., responding to a request for a credential to permit another person to complete an access proof.

Address book The first of these tasks, building an address book of identifiers and bindings among them, is performed using the camera and, to a lesser extent, the keypad of the phone. As described in Section 3.1, the identifiers that can be input via the camera include pictures of persons and of public keys (and of network addresses, but these are not involved in address-book creation). The keypad permits the input of text strings, which we refer to as *names*. The address-book interface presently enables the creation of speaks-for relationships of the following forms: (i) a key speaks for a picture (person); (ii) a key speaks for a name; and (iii) a name speaks for a picture. Relationship (ii) is a typical certificate, and (i) serves a similar purpose. (iii) is a way to provide a local name to a photograph of a person. Grey does not support a means to specify that a photograph speaks for a name. While this has an intuitive meaning, we eschew this possibility to avoid confusion with group creation (not discussed here), which involves issuing credentials authorizing an identifier (which in our case includes a photo) to speak for a group name.

The manner of creating these speaks-for relationships differs depending on the type of relationship being created. In case (i) or (ii), the speaks-for relationship is created by photographing the key and then either selecting an already-present identifier for which it speaks or inputting the identifier at that time. In particular, after photographing the two-dimensional barcode encoding the key, the key is permanently hidden from the user. While user-friendly representations of keys using "snowflakes" [20, 27], flags [17] or random art [35] have been proposed, we believe that exposing keys in the interface is unnecessary and potentially confusing. This is different from names and pictures of persons, which are identifiers that appear in menus and can be used to create speaks-for relationships of type (iii) at any time.

Requesting access to a resource A user requesting access to a resource for the first time must obtain the network address of the computer that controls access to that resource. Collecting this network address can presently be done in two ways: either with Bluetooth discovery or, as discussed

Figure 3: Alice's entry in Bob's address book.

in Section 3.1, using the phone's camera to photograph a two-dimensional barcode encoding the Bluetooth address (Figure 4). The latter technique is more reliable, since Bluetooth discovery can net multiple devices, and selecting the proper device is a user choice that is vulnerable to misinterpretation or the user being misled. Once the network address for a resource is captured, it is kept in a resource menu on the phone. A single click on a resource in this menu initiates an attempt to connect to the corresponding computer and start the sequence to access the resource (see Figure 5).

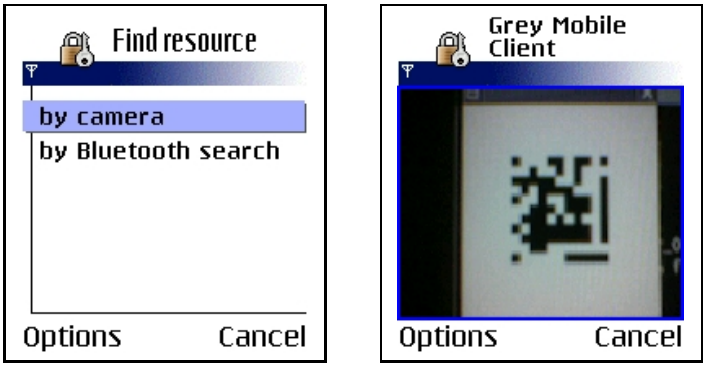

Figure 4: New Grey resources can be found via the camera or through Bluetooth discovery. Bob learns the Bluetooth address of Alice's door by taking a picture of the two-dimensional barcode visible near Alice's door.

Perhaps the most innovative aspect of this part of the user interface is its use of learned patterns of resource accesses. Most users exhibit a pattern of accesses, and this is particularly true for physical resources; e.g., a typical workday begins with the user opening a building door, then a door on the floor on which she works, then her office door, and finally logging into her desktop computer. If all these resources are accessed using Grey, the user's smartphone will learn the temporal proximity and order of these accesses as a pattern, and can offer this pattern as an option when the user initiates the first access in the pattern (e.g., `Work_Garage` to `HH_D202_PC` in Figure 5 is such a pattern). If the user selects the pattern, the phone will attempt to connect to and access each of the resources in sequence, with each step contingent on the previous access in the pattern succeeding. In this way, merely two clicks and a PIN entry as the user approaches her building will enable her to reach her office and will log her into her desktop.

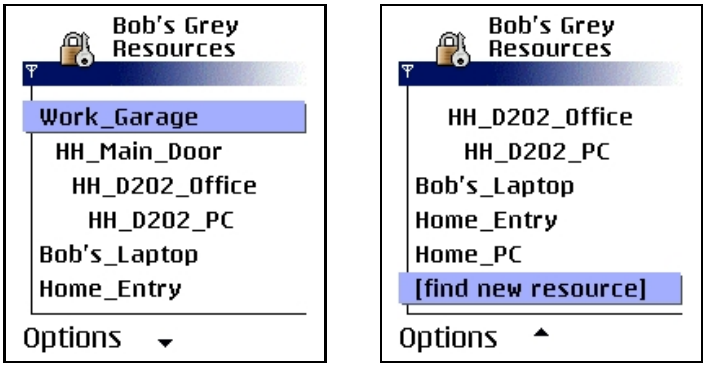

Figure 5: Resource list on Bob's phone.

Reactive policy creation The third type of interface presented by the phone to the user permits the reactive creation of policy. This interface is launched by the prover in the user's smartphone after the prover has generated a list of credentials to which the user could consent to enable an access that is being attempted by another person. For example, in the usage scenario of Section 4, this is the interface by which Alice adds Bob to her `visitors` group by selecting this option from the menu generated by the prover (see Section 5.2).

Because this interface interrupts the user (unlike the other interfaces, which are user driven), it is important that the user can apply access control to this step and silence these interrupts at times she prefers to not be interrupted. For the former (access control), we employ the same access-control infrastructure that we use for other resources, utilizing a default, but user-configurable, policy that permits only those in the phone's address book to request assistance. The latter, i.e., silencing all such requests, is a simple toggle, and, once activated, received requests will be silently queued for the user to handle later. The party requesting credentials from her will simply time out awaiting her response, and will not be able to access the requested resource (or at least not with her help). However, if she later consents to the request, the appropriate credential will still be sent to the requester for use in the future.

5.2 Prover

As described in the example in Section 4, after arriving in front of Alice's office, Bob instructs his phone to unlock the door. The door's first reply contains a *challenge*—a statement, in logic, of the theorem that Bob's phone must prove before the door will unlock. The challenge that typically needs to be proved is that the door's owner believes that it is OK for access to be granted. In this case, expressed in logic, the challenge is $Alice$ **says goal**(`A-111`), i.e., Bob must prove that Alice believes that it is OK to access her office, A-111.[4]

The straightforward way for Bob to answer the door's challenge would be to scour the network for useful credentials and then attempt to form them into a proof; most distributed authorization systems use a close facsimile of this approach. There are some inherent problems, however, with this method of constructing a proof. Bob might guess, for example, that Alice has credentials that

[4]In order to enforce the timeliness of Bob's response and to protect against replay attacks, the logical statement that must be proved also contains a nonce. This and other low-level details that are not novel are described in related work; we omit them from this paper in order to focus on the more abstract ideas.

he could use, but he does not know exactly which of the credentials that she possesses will be helpful for this particular proof. It would be inefficient for Alice to send Bob *all* her credentials, since she might have hundreds. Moreover, sending all her credentials to Bob would reveal exactly the extent of Alice's authority, which is unlikely to meet with Alice's approval. Finally, there may be cases, such as in our example, when the credential that Bob needs has not yet been created; in these situations a simple search, no matter how thorough, would fail to yield sufficient credentials for Bob to access Alice's office.

An answer to these problems can be found in *distributed proving*—a scheme in which Bob's phone does not just search for individual credentials, but also solicits help in proving simpler subproofs that he can assemble into a proof of the challenge [10]. Using this approach, Bob's phone might ask Alice's phone to prove a theorem like Bob **says goal**$(...) \rightarrow Alice$ **says goal**$(...)$. Alice's phone now has the opportunity to decide which of her credentials to use or which new credentials to create in order to prove this theorem; these credentials will be returned to Bob's phone along with the proof. This scheme of farming out subproofs to other entities spans two extremes: *eager* proving, in which a client farms out a theorem only if he is completely unable to make progress on it himself; and *lazy* proving, in which the client asks for help as soon as he isolates a theorem that someone else might be able to help with. Distributed proving can be combined with several optimizations, including caching of credentials and subproofs and deriving proof strategies based on the shape of previously encountered proofs [10].

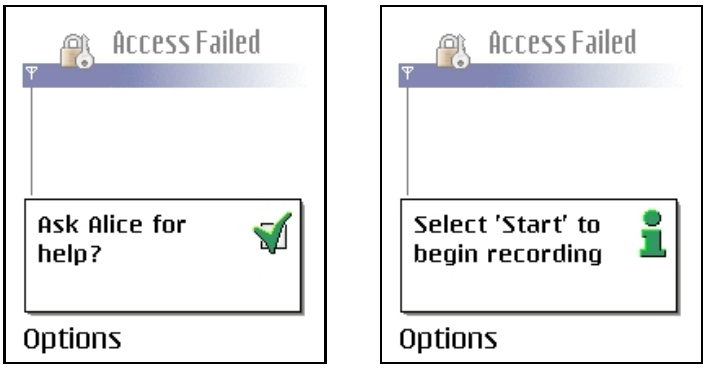

Figure 6: When a proof cannot be generated locally, the user is asked to approve a help request, which can be annotated with an audio clip.

The use of distributed proving in Grey and the details of constructing proofs in general are largely out of the view of the user. Bob's phone processes the door's challenge until it arrives at a potentially useful subtheorem; at that point, the phone consults the address book to determine how Alice can be reached (by phone or by URL, for example). Since Bob might have to pay for the communication (typically, some combination of SMS and GPRS connectivity is needed, and use of either may incur some cost) and to prevent other users from being unintentionally disturbed, Bob's phone prompts Bob to approve the help request. Alice may need reminding or convincing before she will be willing to help, and so Bob is given the option of annotating his request for a subproof with a recorded or text message (see Figure 6).

Upon receiving Bob's request, Alice's phone first verifies that Alice is in fact willing to help Bob (Figure 7). If Alice agrees, her phone begins to compute the subproof, which can in many cases be done without further input from Alice. Sometimes, however, construction of the sub-

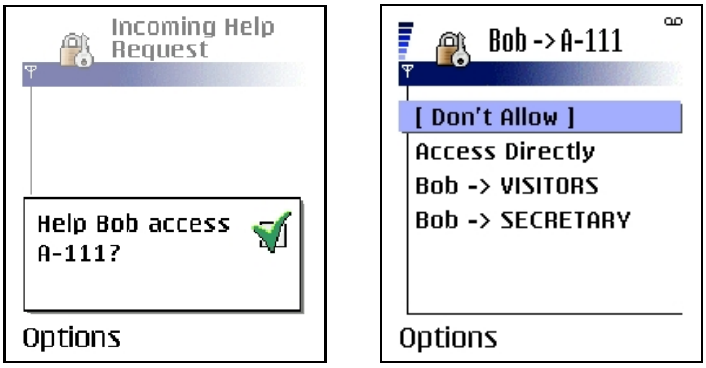

Figure 7: A phone seeks approval before replying to a help inquiry. Alice is given the opportunity to chose the type of credential that she is willing to grant to Bob.

proof will require Alice to generate a new credential. In these cases, Alice is shown a list of the credentials that can be used to complete the subproof. Alice can either choose the credential she wishes to create, or decide that none of them are appropriate. When Alice makes her selection, her smartphone finishes constructing the subproof and sends it to Bob. Bob's phone incorporates Alice's subproof into the main proof and sends the proof to the door (Figure 8).

Figure 8: Bob is kept apprised of the proving process.

Although a single help request is sufficient for our example with Alice and Bob, Bob's phone may in general need to request subproofs from several other users; in addition, each of those users may in turn also need to solicit help. Through a combination of optimizations derived from observing both successful and unsuccessful past behaviors, a user's Grey smartphone can guide proof search to minimize the number of times help is requested. If multiple avenues can lead to constructing a proof, the ones most likely to be successful and quick will be the ones pursued first [10].

Figure 9 depicts the structure of the Grey application that runs on Bob's phone. The entire application is implemented in Java Micro Edition (J2ME)[5], the restricted flavor of Java that runs on many smartphones. The process of generating proofs is managed by different components depending on whether Bob is trying to access a resource himself (ProofTalker) or help another user (HelpTalker). In addition to directing a Prolog engine (JIProlog[6]) to traverse the space of possible

[5] `http://java.sun.com/j2me`
[6] `http://www.ugosweb.com/jiprolog`

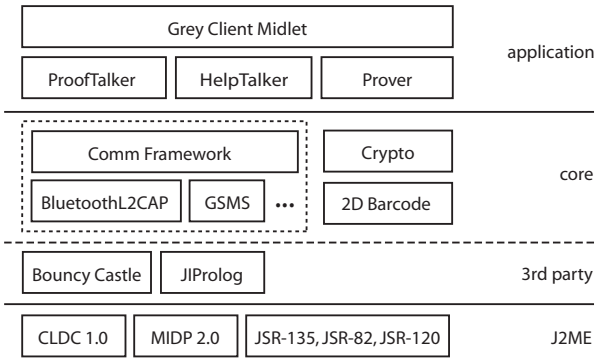

Figure 9: The structure of the Grey application that runs on smartphones.

proofs, these components manage communication with the resource Bob is trying to access and with other users via the communication framework. They also create and manage credentials using the Crypto module, perhaps employing two-dimensional barcodes captured by the phone's camera as an unspoofable channel (2D Barcode).

Grey makes use of a rich set of standard extensions to the core J2ME APIs to enable use of Bluetooth and other communications protocols (JSR-82 and JSR-120) and the phone's camera (JSR-135). In addition, we use the BouncyCastle libraries[7] to implement the higher-level Grey cryptographic primitives.

5.3 Verifier

One of the goals of Grey is to encompass many diverse resources that a user might wish to access. Some of these resources, such as doors and computer logins, we traditionally associate with the need for access control. Others, like thermostats, are not normally thought of the same way. However, with the ability to actuate such resources remotely, via the network or via a smartphone, also comes the need to regulate access. For example, Alice may want to adjust her office temperature before she arrives at work, but she most likely does not want passers-by to do the same.

To enable Grey to conveniently apply to a wide range of devices, it was necessary for its verification module—the component that mediates access to resources—to be simple, relatively lightweight, and device independent. At the same time, we wanted to maintain a high level of assurance that access is not granted improperly. The proof-carrying authorization paradigm fit our needs well; in proof-carrying authorization, access to a resource is allowed if the client presents a proof that he is authorized to use it. The verification of such proofs is a straightforward mechanical process, with none of the complexity and potential intractability of generating proofs. This distinction is fortunate, since the verifier is in the trusted computing base, while proof generation is not. Moreover, the verification process itself is independent of the security policy protecting the resource, and so also of the resource's type (e.g., door, thermostat).

Figure 10 shows the components and control flow of the verification module, which are described in more detail in the following paragraphs. The process of gaining access to a resource is initiated by a user request. In response to the request, a *challenge* is generated. The challenge is the statement, in formal logic, of the theorem whose proof a potential user must provide. As

[7] `http://www.bouncycastle.org`

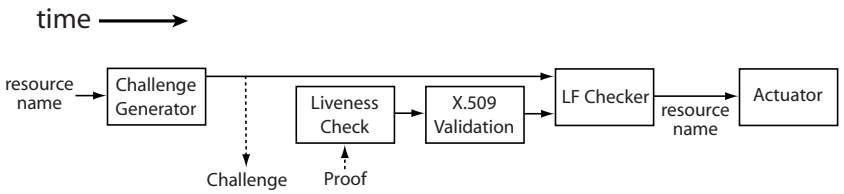

Figure 10: Flow of the verification process.

described in Section 3.3, the challenge is specified in higher-order logic; this in turn is encoded in LF, the notation of one of the most widely used frameworks for specifying logics [22].

When Bob attempts to access Alice's office, the verification module generates a challenge that includes the name of the resource, A-111, and a nonce. This challenge is sent to Bob, but also recorded for use in later stages of verification.

Bob's eventual reply to the challenge will contain a set of credentials (e.g., Bob is a member of `visitors`), and a proof, in formal logic, that the credentials satisfy the door's challenge. The first step of verifying the proof is to ensure that it has arrived within a brief period after the door issued the challenge. Next, the credentials, which are X.509v3 certificates with customized extensions, are verified: their digital signatures and expiration times are checked. Finally, the formal proof is passed to an LF type checker, which ensures that the structure of the proof is valid (e.g., that it contains no false implications) and that the correct theorem (the one that was issued as the challenge) was proved. This algorithm is widely studied and well understood, providing high assurance that an invalid proof will never be accepted [14, 6]. If this proof is successfully verified, the LF checker signals an actuator to open the door.

Some resources are simpler to manage than others. Alice's door, for example, has a single security policy and a binary actuator. Other resources, like computer logins, are likely to have multiple security policies (e.g., one for each computer account) and different ways of actuating (e.g., "log in Alice", instead of just "log in"). For this reason we provide the challenge generator and actuator with the name of the resource being accessed. Note that these two components are the only ones that are customized to a particular type of Grey device; the other components of the verification module can be reused without any modification.

5.4 The Communication Framework

One of the components critical to Grey's success is a flexible and reliable framework for managing communications in a network of both mobile (e.g., smartphones) and fixed devices (e.g., computers controlling access to fixed resources). In designing and developing Grey's communications infrastructure we had the following goals.

Support for multiple transport layers Smartphones have at their disposal a plethora of messaging protocols—SMS, MMS, voice calls, Bluetooth—that are highly heterogenous in terms of latency, reliability, and capacity, and yet all need to be seamlessly integrated into a single framework.

Reliable, flexible message routing Some Grey devices will have access to only a subset of the possible messaging protocols (e.g., only Bluetooth); other devices will experience periods of

limited connectivity (e.g., while in the basement of an office building). The communication framework must compensate by routing messages via whichever protocols are available.

Light user burden Our emphasis on usability demands that the user remain oblivious, whenever possible, to any decisions regarding ad-hoc routing, retrying failed transmissions, etc. For this reason, the communication framework needs more than ordinary insight into the characteristics of messaging protocols and the communications abilities of various Grey devices.

Figure 11 depicts the structure of the communication framework and its integration with other components of a Grey application. The communication framework is comprised of three main parts: a Talker interface for accepting messages from the application and returning diagnostic information; the message queue and carrier "management layer"; and a collection of carrier modules for the different messaging protocols.

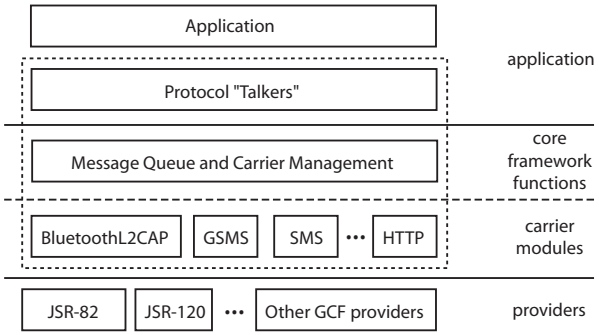

Figure 11: Structure of the multi-platform communication framework.

Application layer The Talker exports an asynchronous, message-based communications API to the client application. An application typically instantiates an extension of Talker for each protocol or function that the application wishes to implement. For example, the Grey application that runs on smartphones uses a ProofTalker to encapsulate the protocol for accessing a resource on behalf of the user. Any replies to messages dispatched by a particular Talker are automatically routed to that Talker's event handler by the communication framework. In addition, a Talker can explicitly register with the communication framework to receive particular types of messages, where the type can be specified by source, content, or carrier. From the standpoint of the application, this allows all communication relevant to a particular process to be automatically routed to the appropriate Talker (e.g., the ProofTalker). Two other Talkers commonly used by Grey applications are the HelpTalker, which manages help requests sent out by other users, and the AdminTalker, which handles synchronization between the smartphone and a workstation. A feature often implemented by Talkers is end-to-end encryption using symmetric or public-key techniques.

In addition to dispatching and receiving messages, Talkers also receive feedback from the underlying communication layers about the successful or unsuccessful receipt of the message by the addressee, as well as diagnostic information indicating the status of the message (e.g., that the transit time is expected to be long, that the preferred carrier is unavailable). This diagnostic information can be safely ignored, but Talkers may choose to use it to adjust their protocol (e.g., by asking a different device for help) or to inform the user (e.g., by displaying the expected waiting time).

Management layer The middle layer of the communication framework is a database management system that allows the communication framework to track and manage its messages and active carriers. The queue- and carrier-management tasks generally take the form of mediation between the requester (Talker) and the provider (carrier).

All messages dispatched by Talkers are added to a message queue and then processed in turn. Processing a message involves determining the exact address to which it should be sent and the carrier that should be used. Determining the exact address could be trivial, since the address (e.g., the Bluetooth address of a door) is in many cases provided by the Talker. However, a Talker may identify the addressee of a message only abstractly (e.g., "Alice"). In this case, the communication framework must first consult the user's address book to determine the possible destination addresses (e.g., Bob's address book may contain Alice's phone number, the URL of her workstation, and the Bluetooth address of her smartphone). Hints from the Talker assist the communication framework in selecting the most appropriate address.

Once a concrete destination address is identified, the communication framework selects the best carrier for dispatching the message. In the typical case this is, again, straightforward. For example, if a message is addressed to the Bluetooth address of a door's embedded computer, the communication framework will instantiate the Bluetooth carrier (BluetoothL2CAP) and send the message. However, if the addressee cannot be contacted directly (e.g., the destination address is a phone number but the sender is out of reach of a wireless carrier) then, if configured to do so, the communication framework will attempt to send the message to other willing Grey devices. These devices will in turn forward the message to the desired recipient, subject to hop-count limitations.

The tasks of selecting the destination address and the carrier are not always independent; for example, if Bob's address book contains three addresses for Alice and all of the corresponding carriers are currently able to send messages, Grey chooses which carrier to use in order of speed of delivery—in general, Bluetooth first, followed by GSMS or SMS, followed finally by indirect routing of a message via other Grey nodes, if configured to do so.

In addition to simple message dispatch, the management layer of the communication framework performs a variety of other housekeeping tasks, such as confirming message receipt by sending and waiting for acknowledgments, resending messages if the original dispatch fails, discarding duplicate messages, etc.

Carriers Each carrier handles one type of point-to-point communication link. A Grey device may have multiple instances of the same type of carrier. The modular approach allows us to simply "plug" new carriers into Grey as new communications capabilities become available to smartphones in the future. Likewise, this approach allows a Grey application to use only the carriers it needs or that are supported by the underlying hardware.

One of the challenges in designing the carriers was to create a single interface that could encompass very different behaviors—some carriers, like SMS, are inherently message based and can have very high latency, while others, like HTTP, are connection based and have lower latency and higher capacity. A testament to the soundness of our communication framework design is that adding carriers, even complex ones that use several communications protocols (e.g., the GSMS carrier, which emulates multimedia messaging by using a combination of SMS, GPRS, and HTTP communication) has proven straightforward.

Access	Time (s)	Variance
Door Access	5.36	0.33
Windows XP Login	9.31	2.20

Table 1: User-level benchmarks in seconds.

5.5 Performance on Modern Smartphones

In this section we provide performance measurements for certain tasks in Grey. Our primary interest is measuring delays as experienced by the user to access a resource in the common case. We report such numbers here, and additionally measure costs associated with underlying operations to shed light on the sources of these delays.

Our first macrobenchmark is the time required to open a door. The computer controlling the door lock was an embedded PC with a 1.4GHz Pentium M processor; more detail on this pilot application is given in Section 6.1. Each timing was measured starting when the user selected the door from the resource list on her phone (a Nokia 6620), and ended when the door unlocked. As shown in Table 1, this delay was approximately six seconds excluding any user interaction (more on this below), with a small variance resulting from background work on the phone, such as alarms, housekeeping, and other applications. The second macrobenchmark is the time required for a user to log into a 2GHz Windows XP workstation. The methodology in this experiment was similar to that for the door, with timing beginning when the user selected the resource in her resource list on the phone, and ending when the "Start" menu became available to the user. As shown in Table 1, this delay is roughly nine to eleven seconds. The bulk of the extra time was taken up by the load time for `explorer.exe` and desktop preparation.

We emphasize that these are common-case numbers in three senses. First, neither of these tests involved a remote help request. Help requests can take significantly longer (e.g., a minute), and vary depending on cellular network conditions and user responsiveness. Second, these measurements did not involve the use of a capture-resilient signing key on the phone, and as such the signing operation by the phone did not involve user input (i.e., a PIN) or interaction with a capture-protection server. In our present implementation, we have adopted a design by which the user can configure the frequency with which she is prompted for her PIN (and the capture-protection server is contacted), rather than being prompted per resource access. Her capture-resilient key is then used at these intervals to create a short-lived certificate for a non-capture-resilient public key (a step which does require PIN entry) that is used to sign access requests. As such, the common case incurs only the latency of a signature with this non-capture-resilient key; the measurements in Table 1 reflect this. Third, the network address for each of the computers regulating access was already stored in the resource list of the phone and so, e.g., the one-time barcode-processing overhead incurred if it is first captured via the camera (roughly 1.5 seconds) is not reflected in these numbers.

Typical latencies of under six seconds to open a door and roughly nine seconds to complete a computer login are already comparable to the latencies of these processes using more traditional access control (e.g., physical keys and passwords), particularly considering that much of the latency for the XP login is due to login steps that follow the access-control decision. However, we emphasize that Grey permits these latencies to be hidden from the user more effectively than al-

ternatives. Our current systems utilize class 2 Bluetooth devices, meaning that, e.g., a smartphone could initiate an access once it is within 10 meters of the resource (the door or computer). By the time the user reaches the resource in order to make use of it, the access typically would have completed. In our own experience with using the system, access is consequently far quicker than with the alternatives that Grey replaces.

	Nokia 6620		Audiovox SMT 5600	
Action	Time (ms)	Variance	Time (ms)	Variance
RSA PSS	1780	480	1260	30
RMS Read (1.5KB)	697	100	60	4
RMS Write (1.5KB)	480	88	170	3
RMS Read (30KB)	900	115	90	20
RMS Write (30KB)	1109	78	200	28

Table 2: Microbenchmarks in milliseconds.

The microbenchmarks shown in Table 2 illustrate some of the performance bottlenecks of using a resource-constrained platform such as the smartphone. Among the most significant sources of delay are RSA signatures and accessing the Record Management Store (RMS), which is a simplified file system and the only long-term storage available to the Java VM. The left columns of Table 2 describe these costs for the Nokia 6620, the device used in the experiments described in Table 1. The 30KB RMS read and write benchmarks measure the time it takes to read and write a standard address-book picture, each about 30KB in size. In the case of the RSA signature [41], the RSA modulus was 1024 bits in length.

In light of Table 2, an obvious approach to optimize the signature cost is to employ a signature algorithm for which signing is less expensive, such as DSA [25]. However, trends in device technology are reducing these costs, across the board, at a dramatic pace. As an example, the right columns of Table 2 provide the same measurements for an Audiovox SMT 5600, a more modern smartphone with an even smaller form factor than the Nokia 6620. Our measurements show a 30% improvement in RSA signing times, a threefold or more improvement in RMS write times, and nearly an order of magnitude improvement in RMS read times. Sadly, we cannot yet use the Audiovox model, because it does not currently support JSR-82 (Bluetooth).

6 Pilot Applications

In this section we provide greater detail on the pilot applications that we are developing for deployment with Grey.

6.1 Office Access

Our primary pilot application for Grey, which has been discussed throughout this paper, is enabling access to office doors. The required physical infrastructure is relatively minimal: a standard electric door strike actuated by an embedded PC located in the wall near each door. Our prototype embedded PC measures $4.55 \times 3.75 \times 1.70$ inches—small enough to fit *within* each door, an option we seriously considered. It is equipped with a Bluetooth adapter and an RS-485 relay controller,

and to improve reliability has no moving parts (i.e., cooling is passive, and flash memory is used for non-volatile storage). The prototype embedded PC uses a commodity Pentium M on a PC-104+ mainboard; for a wide deployment of Grey a significantly more compact, custom embedded system could be designed.

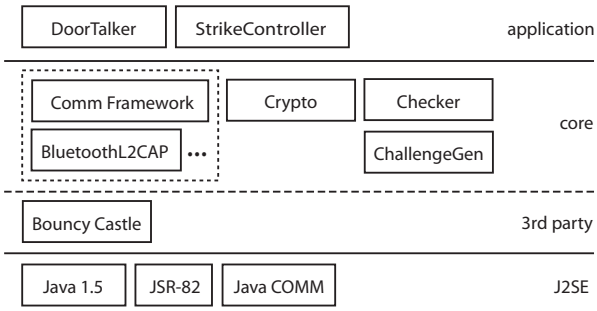

Figure 12: The structure of the Java application that allows office doors to be Grey-enabled.

Figure 12 shows the structure of the Grey application that controls access to a door. The application is constructed in a modular fashion—the bulk of it are the communications and verification components discussed in Sections 5.4 and 5.3, and the only customization necessary was the front end (DoorTalker) that encapsulates these modules and the actuator module (StrikeController) that sends commands specific to the relay controller we use.

Enabling a door with Grey does not preclude legacy access technologies (e.g., keys, proximity cards) from being used; Grey merely provides a parallel way to unlock the door. Of course, Grey can also be used as the sole method of controlling access.

6.2 Windows XP Login

The second of our two pilot applications was integrating Grey with the Windows XP login architecture to enable Grey users to login to their workstations from a smartphone. The structure of the verification module in this application remains unchanged. However, unlike access to doors where the actuator performs a binary action (i.e., either unlocking the door or not), in this case the actuator presents the desired username to the Windows component that starts a new user session.

The Windows XP login architecture and Grey's integration into it are shown in Figure 13. The regular method of starting a user session on Windows XP is as follows: After the workstation boots (or a previous user logs out), `Winlogon.exe` starts the GINA, a GUI module that is responsible for all authentication-related interaction with the user. The GINA collects a user's credentials (normally a username and password) and sends them to the Local Security Authority (LSA). The LSA uses an authentication package to verify the credentials and starts a user session (i.e., logging the user in).

In the door-access-control application, the entire verification module, from the challenge generator to the actuator, lives in "trusted" space. One of the difficulties of integrating Grey into the Windows XP login architecture was that Windows enforces a strict division between untrusted modules (the GINA) and trusted modules (the LSA and its authentication packages), which required us to separate the Grey communications components from the rest of the verification mod-

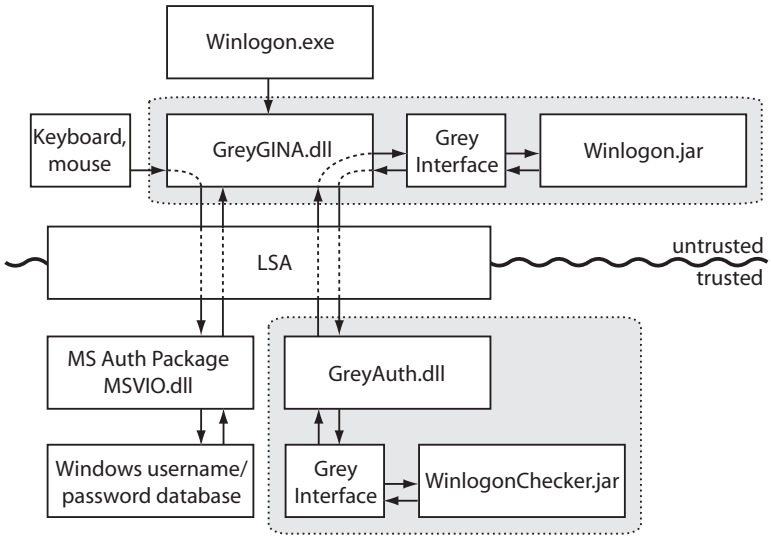

Figure 13: The integration of Grey into the Windows authorization architecture.

ule.[8]

One component resides in "untrusted space" and enables the display of barcodes and Bluetooth communication with the user device. A customized GINA allows this module to be used in parallel with the standard login dialog window. Both the generation of the challenge and the verification of the proof must be carried out correctly to prevent users from logging in without authorization, and so those components reside in trusted space and are accessed by the LSA as another authentication package. A small interface module (Grey Interface) mediates communication between the standard Windows components, which are written in C/C++, and the Grey components that are written in Java.

This boundary between untrusted execution and trusted execution space is crossed twice. First, the communications component (`Winlogon.jar`) requests a challenge from the verification module (`WinlogonChecker.jar`). Second, the communications component passes the user's proof to the verification module.

Note that Grey-enabling the Windows XP login does not prevent users from continuing to login with a username and password or any other token (e.g., smartcard) they had previously been using. In addition, the functionality of gaining access via Grey is available whenever the Windows login architecture is used, for example, when the screen is locked while a user is still logged in.

To continue our example from Section 4: After Bob has proved access and entered Alice's office, he can use his smartphone to login to the "guest" account on Alice's workstation. To start the process, Bob uses his phone's camera to take a picture of the barcode embedded in the otherwise standard Windows XP login dialog box (Figure 14). Alice has delegated permission to login via the "Guest" account to everyone in her `visitors` group. This delegation credential was part of the proof of access that Bob constructed to enter Alice's office; hence, Bob's smartphone already has a copy of the credential. Without further ado, Bob's smartphone locally constructs a proof of access and sends it to the computer, which starts a new login session.

[8]The major difficulty was the lack of provision in the Windows XP architecture for adding third-party authorization modules.

Figure 14: The Windows XP login interface, modified for use by Grey.

7 Conclusion and Status

Smartphones offer a number of features that make them attractive as a basis for pervasive-computing applications, not the least of which is their impending ubiquity. Grey is an effort to leverage these devices beyond the games, personal information management, and basic communication (voice, email) for which they are primarily used today. We believe, in particular, that these devices can form the basis of a sound access-control infrastructure offering both usability and unparalleled flexibility in policy creation.

Grey is a collection of software extensions to commodity mobile phones that forms the basis for such an infrastructure. At the core of Grey is the novel integration of several new advances in areas ranging from device technologies (e.g., cameras) and applications thereof, to theorem proving in the context of access-control logics. This integration yields, we believe, a compelling and usable tool for performing device-enabled access control to both physical and virtual resources. We briefly described two such applications that we have implemented, namely access to doors and to Windows XP logins, and the novel forms of reactive delegation that this tool enables.

Grey is being deployed to control access to the physical space on two floors of a building currently under construction on our university campus. Construction of this building is planned to be completed in March 2005, after which Grey will be phased into the building on an opt-in basis. At the time of this writing, Grey is already sufficiently mature to permit our group members to use it on a daily basis to access a door that serves as a prototype of those being deployed in this building. In addition, we continue to refine a prototype of the Grey-enabled Windows XP login, which is operational in our laboratory. We will continue to mature these prototypes as we approach deployment.

References

[1] M. Abadi. On SDSI's linked local name spaces. *Journal of Computer Security*, October 1998.

[2] M. Abadi, M. Burrows, B. Lampson, and G. D. Plotkin. A calculus for access control in distributed systems. *ACM Transactions on Programming Languages and Systems*, September 1993.

[3] J. Al-Muhtadi, M. Anand, M. D. Mickunas and R. H. Campbell. Secure smart homes using Jini and UIUS SESAME. In *Proccedings of the 2000 Computer Security Applications Conference*, December 2000.

[4] J. Al-Muhtadi, A. Ranganathan, R. Campbell and M. D. Mickunas. Cerberus: A context-aware security scheme for smart spaces. In *Proceedings of the 1st IEEE Conference on Pervasive Computing and Communications*, pp. 489–496, March 2003.

[5] A. W. Appel and E. W. Felten. Proof-carrying authentication. In *Proceedings of the 6th ACM Conference on Computer and Communications Security*, November 1999.

[6] A. W. Appel, N. Michael, A. Stump and R. Virga. A trustworthy proof checker. *Journal of Automated Reasoning* 31(3-4):231–260, 2003.

[7] D. Balfanz, D. Dean, and M. Spreitzer. A security infrastructure for distributed Java applications. In *Proceedings of the 21st IEEE Symposium on Security and Privacy*, May 2002.

[8] D. Balfanz and E. Felten. Hand-held computers can be better smart cards. In *Proceedings of the 1999 USENIX Security Symposium*, August 1999.

[9] L. Bauer, M. A. Schneider, and E. W. Felten. A general and flexible access-control system for the Web. In *Proceedings of the 11th USENIX Security Symposium*, August 2002.

[10] L. Bauer, S. Garriss and M. K. Reiter. Distributed proving in access-control systems. In *Proceedings of the 2005 IEEE Symposium on Security and Privacy*, May 2005. To appear.

[11] A. Beaufour and P. Bonnet. Personal servers as digital keys. In *Proceedings of the 2nd IEEE International Conference of Pervasive Computing and Communications (PerCom)*, March 2004.

[12] M. Blaze, J. Feigenbaum, and M. Strauss. Compliance checking in the PolicyMaker trust-management system. In *Proceedings of the 2nd Financial Crypto Conference*, 1998.

[13] A. Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 1940.

[14] T. Coquand. An algorithm for testing conversion in type theory. In G. Huet and G. Plotkin (Eds.): *Logical Frameworks*, pp. 255–280, Cambridge University Press, 1991.

[15] M. J. Covington, P. Fogla, Z. Zhan and M. Ahamad. A context-aware security architecture for emerging applications. In *Proceedings of the 2002 Computer Security Applications Conference*, December 2002.

[16] M. J. Covington, W. Long, S. Srinivasan, A. Dey, M. Ahamad, and G. Abowd. Securing context-aware applications using environment roles. In *Proceedings of the 6th ACM Symposium on Access Control Models and Technologies*, May 2001.

[17] S. Dohrmann and C. Ellison. Public key support for collaborative groups. In *Proceedings of the First Annual PKI Research Workshop*, April 2002.

[18] C. M. Ellison, B. Frantz, B. Lampson, R. Rivest, B. M. Thomas, and T. Ylonen. Simple public key certificate. Internet Engineering Task Force Draft, July 1997.

[19] C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, and T. Ylonen. SPKI certificate theory. RFC 2693, September 1999.

[20] I. Goldberg. Visual key fingerprint code. Available at `http://www.cs.berkeley.edu/iang/visprint.c`, 1996.

[21] J. Y. Halpern and R. van der Meyden. A logic for SDSI's linked local name space. In *Proceedings of the 12th IEEE Computer Security Foundations Workshop*, June 1999.

[22] R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. *Journal of the Association for Computing Machinery* 40(1):143–184, January 1993.

[23] U. Hengartner and P. Steenkiste. Protecting access to people location information. In *Proceedings of the 1st International Conference on Security in Pervasive Computing*, pp. 25–38, March 2003.

[24] R. Housley, W. Polk, W. Ford, and D. Solo. Internet X.509 public key infrastucture certificate and CRL profile. RFC 3280, April 2002.

[25] D. W. Kravitz. Digital signature algorithm. U.S. Patent 5,231,668, 27 July 1993.

[26] B. Lampson, M. Abadi, M. Burrows, E. Wobber. Authentication in distributed systems: Theory and practice. ACM Transactions on Computer Systems, 10(4):265–310, November 1992.

[27] R. Levien. PGP snowflake. Personal communication, 1996.

[28] D. López de Ipiña, P. Mendonça, and A. Hopper. TRIP: a low-cost vision-based location system for ubiquitous computing. *Personal and Ubiquitous Computing* 6(3):206–219, May 2002.

[29] P. MacKenzie and M. K. Reiter. Networked cryptographic devices resilient to capture. *International Journal of Information Security* 2(1):1–20, November 2003.

[30] P. MacKenzie and M. K. Reiter. Delegation of cryptographic servers for capture-resilient devices. *Distributed Computing* 16(4):307–327, December 2003.

[31] A. Madhavapeddy, D. Scott, R. Sharp, and E. Upton. Using camera-phones to enhance human-computer interaction. In *6th International Conference on Ubiquitous Computing (Adjunct Proceedings: Demos)*, 2004.

[32] J. M. McCune, A. Perrig and M. K. Reiter. Seeing-is-believing: Using camera phones for human-verifiable authentication. To appear in *Proceedings of the 2005 IEEE Symposium on Security and Privacy*, May 2005. Draft available at `http://reports-archive.adm.cs.cmu.edu/cs2004.html`.

[33] Mobile Pipeline News. 'Smartphone' era to start next year, researcher predicts. Information Week, November 24, 2004. Available at `http://informationweek.com/story/showArticle.jhtml?articleID=54200479`.

[34] R. Molva and G. Tsudik. Authentication method with impersonal token cards. In *Proceedings of the 1993 IEEE Symposium on Security and Privacy*, May 1993.

[35] A. Perrig and D. Song. Hash visualization: A new technique to improve real-world security. In *Proceedings of the 1999 International Workshop on Cryptographic Techniques and E-Commerce (CrypTEC '99)*, pp. 131–138, July 1999.

[36] J.-J. Quisquater and D. Samyde. Electromagnetic analysis (EMA): Measures and countermeasures for smart cards. In I. Attali and T. Jensen (Eds.): *E-smart 2001*, LNCS 2140, pp. 200–210, 2001.

[37] B. Ramsdell. Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.1: Message Specification. RFC 3850, July 2004.

[38] N. Ravi, P. Stern, N. Desai, and L. Iftode. Accessing ubiquitous services using smart phones. In *Proceedings of the 3rd International Conference on Pervasive Computing and Communications (PerCom)*, March 2005.

[39] R. L. Rivest and B. Lampson. SDSI—A simple distributed security infrastructure. Presented at CRYPTO '96 Rumpsession, April 1996.

[40] M. Rohs and B. Gfeller. Using camera-equipped mobile phones for interacting with real-world objects. *Advances in Pervasive Computing*, pp. 265–271, April 2004.

[41] RSA Laboratories. PKCS #1: RSA Cryptography Standard, `ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-1/pkcs-1v2-1.pdf`.

[42] R. Sailer and J. R. Giles. Pervasive authentication domains for automatic pervasive device authorization. In *Proceedings of the 2nd IEEE Conference on Pervasive Computing and Communications Workshops*, 2004.

[43] S. Santesson, R. Housley, and T. Freeman. Internet X.509 Public Key Infrastructure: Logotypes in X.509 Certificates. RFC 3709, February 2004.

[44] D. Scott, R. Sharp, A. Madhavapeddy and E. Upton. Using visual tags to bypass Bluetooth device discovery. *Mobile Computing and Communications Review* 1(2), January 2005.

[45] A. Slawsby and A. Leibovitch. Worldwide mobile phone 2004–2008 forecast update and 1H04 vendor shares. `http://www.idc.com/getdoc.jsp?containerId=32336`, December 2004.

[46] A. Tripathi, T. Ahmed, D. Kulkarni, R. Kumar, and K. Kashiramka. Context-based secure resource access in pervasive computing environments. In *Proceedings of the 2nd IEEE Conference on Pervasive Computing and Communications Workshops*, 2004.

[47] E. Wobber, M. Abadi, M. Burrows and B. Lampson. Authentication in the Taos operating system. *ACM Transactions on Computer Systems* 12(1):3–32, February 1994.

[48] C. Wullems, M. Looi and A. Clark. Towards context-aware security: An authorization architecture for intranet environments. In *Proceedings of the 2nd IEEE Conference on Pervasive Computing and Communications Workshops*, 2004.