Device-Enabled Authorization in the Grey System[⋆]

(Extended Abstract)

Lujo Bauer, Scott Garriss, Jonathan M. McCune,
Michael K. Reiter, Jason Rouse, and Peter Rutenbar

Carnegie Mellon University, Pittsburgh, Pennsylvania, USA

Abstract. We describe the design 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 focus on the software components and user interfaces of Grey, highlighting the features of each. We also discuss an initial case study for Grey, in which we are equipping over 65 doors on two floors of office space for access control using Grey-enabled devices, for a population of roughly 150 persons. Further details of Grey, and this and other applications, can be found in a companion technical report.

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 RFID card. By contrast, access to virtual resources is typically tied to the knowledge of a password and/or possession of a physical token (e.g., SecureID) for producing time-varying passwords.

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, as in the long term they stand to inherit the vast cellular phone market, which in 2004 shipped over 648 million units [30]. 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).

This convergence of market trends and technological advances points to a future marked by pervasive adoption of highly capable and always-in-hand smartphones. Grey is an effort to use this platform to build a ubiquitous access-control

[⋆] 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.

technology spanning both physical and virtual resources. This vision is not ours alone: several groups have experimented with the use of mobile phones as digital keys [9, 26]; NTT Docomo is conducting trials on the use of mobile phones to authorize entry to apartments[*]; and mobile phones can already be used to purchase items from vending machines in several countries. However, to the extent that we can infer the capabilities of these systems, 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 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 (e.g., passwords) that could be used at a later time or to access a different resource. Going further, a user could specify 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.

Grey 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) [3], extended with a new distributed proving technique that offers significant efficiency advances [7]. In addition to enabling a user to exercise her authority, PCA provides a framework in which users can delegate authority in a convenient fashion. For protection of phone-resident cryptographic keys in the event of phone capture, Grey incorporates *capture resilience* [22], which renders a lost or stolen phone resistant to misuse. And, on the user-interface front, we employ a technique for conveying key material and network addresses, that is as simple as taking a picture with the phone's built-in camera [23, 29]. Phone-to-phone and phone-to-infrastructure data communication utilizes 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 describe the adaptation of these components into a practical access-control system called Grey. At the time of this writing, we are deploying Grey to create a platform for future research on practical smartphone-based access-control systems. Our initial deployment on two floors of a new building on our university campus will involve roughly 150 users and consist of two applications: (1) controlling access to 65 offices by Grey-enabled phones; (2) using Grey for accessing Windows XP sessions. In these applications, Grey offers a more secure, flexible and convenient basis for access control than existing solutions.

Due to space limitations, we were forced to omit the descriptions of several important aspects of Grey. For more detail, including a thorough discussion of related work, a more comprehensive description of the software architecture,

[*] http://www.i4u.com/article960.html

more extensive performance results, and a description of the Grey Windows XP login plugin, please see our companion technical report [6].

2 Component Technologies

Grey is a novel integration of a number of recently-developed technologies that utilize the capabilities of modern smartphones; we summarize these component technologies here.

2.1 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, e.g., by asking the user to take a picture of an item she intends to interact with. Information conveyed by photographing two-dimensional barcodes is a theme common to several ubiquitous computing efforts (e.g., [13, 28]), typically to convey service information or a URL where such information can be obtained. In Grey, there are two types of identifiers that are commonly input via the camera:

An identifier for a public key. A useful identifier for a key is the collision-resistant hash of the key (e.g., [20]). 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 photograph this identifier and authenticate the public key obtained by other means (e.g., over a wireless link) [23]. This provides a natural and user-friendly way for obtaining an authentic public key.

A network address. 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 to circumvent high-latency device discovery in Bluetooth [29], and we use it in this way 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 graphical management interfaces for collecting identifiers and managing access. We will provide an overview of the interfaces we have developed in Section 4.

2.2 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 [22]. 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 2.1—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 with little risk of exposing it in an indefensible way.

2.3 Proof-Carrying Authorization

Prior research in distributed authorization has produced a number of systems [27, 16, 15, 10] that provide ways to implement and use complex security policies that are distributed across multiple entities. 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 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, 18]). 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 [31, 5]. 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) [3, 8] 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 [11]; 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 [4] and also enables even computationally impoverished devices to be protected by Grey.

3 A Usage Scenario

Grey's integration of the technologies described in Section 2 (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". To populate her address book entry for Bob, Alice needs merely to snap a picture of the two-dimensional barcode displayed on Bob's phone. The barcode encodes both the Bluetooth address of Bob's phone, enabling Alice's phone to connect to it, and a hash of Bob's public key, which can be used to authenticate the full key that is transferred via Bluetooth along with Bob's contact information. 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., [25]).

As their submission deadline approaches, Alice and Bob decide to meet in person, 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.

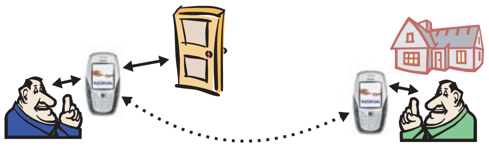

Fig. 1. Bob entering Alice's office. In the course of proving access, Bob's phone contacts Alice's phone for help.

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.

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 2.2—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`.

4 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 (for details please see our companion technical report [6]); 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.

4.1 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 [20]), 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); 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 the keypad of the phone. As described in Section 2.1, the identifiers that can be input via the camera include pictures of public keys (and of network addresses, but these are not involved in address-book creation). The keypad permits the input of text strings. The address-book interface enables the creation of speaks-for relationships between names and keys: a user photographs the key and then either selects an already-present identifier for which the key speaks or inputs the identifier at that time. After a user photographs the two-dimensional barcode encoding a key, the key is permanently hidden from the her. While user-friendly representations of keys using "snowflakes" [17, 21], flags [14] or random art [24] have been proposed, we believe that exposing keys in the interface is unnecessary and potentially confusing.

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 in Section 2.1, using the phone's camera to photograph a two-dimensional barcode encoding the Bluetooth address (Figure 2). 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 3).

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; 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 3 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.

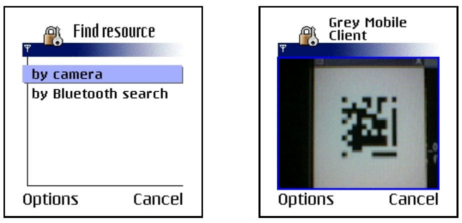

Fig. 2. Bob learns the Bluetooth address of Alice's door by taking a picture of the two-dimensional barcode visible near Alice's door.

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 3, 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 4.2).

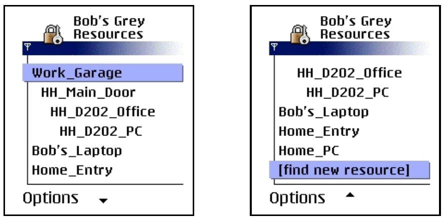

Fig. 3. Resource list on Bob's phone.

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 be informed that a response is not forthcoming, 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.

4.2 Prover

As described in the example in Section 3, after arriving at 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.[**]

The straightforward way for Bob to answer the door's challenge is 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 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 [7]. 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

[**] 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 elsewhere; we omit them from this paper in order to focus on the more abstract ideas.

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 [7].

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.

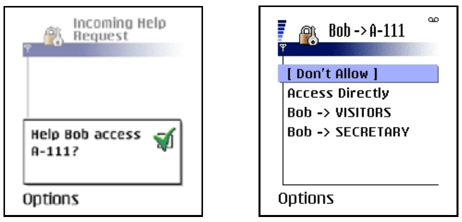

Fig. 4. Alice is given the opportunity to chose the type of credential to grant to Bob.

Upon receiving Bob's request, Alice's phone first verifies that Alice is in fact willing to help Bob (Figure 4). 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 subproof 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.

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 [7].

Figure 5 depicts the structure of the Grey application that runs on Bob's phone. The entire application is implemented in Java Micro Edition (J2ME), 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 to traverse the space of possible 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.

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[***] to implement the higher-level Grey cryptographic primitives.

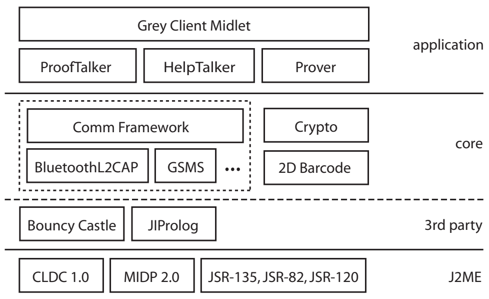

Fig. 5. The structure of the Grey application that runs on smartphones.

4.3 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 fits our needs well; in PCA, 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, login).

Figure 6 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 initi-

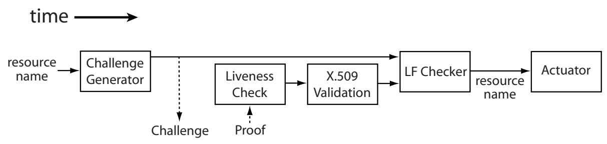

Fig. 6. Flow of the verification process.

[***] http://www.bouncycastle.org

ated 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 described in Section 2.3, 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 [19].

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 (using the nonce) that it was created 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 [12, 4]. If this proof is successfully verified, the LF checker signals an actuator to open the door.

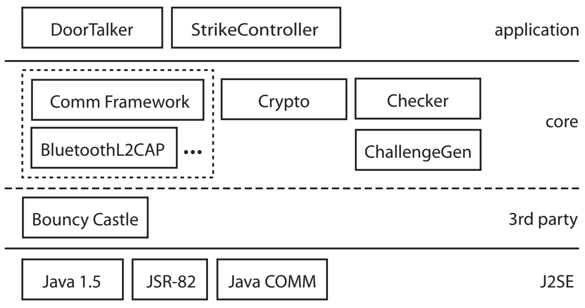

Fig. 7. The structure of the Java application that allows office doors to be Grey-enabled.

Figure 7 shows the structure of the Grey application that controls access to a door. Similarly to the prover application described in Section 4.3, this application is constructed in a modular fashion—the only customization necessary was the front end (DoorTalker) that encapsulates these modules and the actuator module (Strike-Controller) that sends commands specific to the relay controller we use.

The required physical infrastructure for Grey-enabling a door 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.

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.

4.4 Performance on 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 our companion technical report [6]. 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. On average, this delay was 5.36 seconds excluding any user interaction (more on this below), with an variance of 0.33 due to background work on the phone. The second macrobenchmark is the time required for a user to log into a 2GHz Windows XP workstation using Grey [6]. The methodology in this experiment was similar to that for the door. This delay averaged to 9.31 seconds, with a variance of 2.20. 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. 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 sec.) 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 more traditional access control (e.g., physical keys and passwords). However, we emphasize that Grey permits these latencies to be hidden from the user more effectively than alternatives. 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 for us.

5 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.

Grey is being deployed to control access to the physical space on two floors of a building recently constructed on our university campus. Construction of this building was completed in June 2005, and Grey is being phased into the building on an opt-in basis. This deployment will serve as a platform for continued research on usability, credential management, theorem proving and other technologies in the function of access control.

References

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

[2] M. Abadi, M. Burrows, B. Lampson, and G. D. Plotkin. A calculus for access control in distributed systems. *ACM Trans. Prog. Lang. and Sys.*, Sept. 1993.

[3] A. W. Appel and E. W. Felten. Proof-carrying authentication. In *Proc. 6th ACM Conference on Computer and Communications Security*, Nov. 1999.

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

[5] D. Balfanz, D. Dean, and M. Spreitzer. A security infrastructure for distributed Java applications. In *Proc. 21st IEEE Symposium on Security and Privacy*, 2002.

[6] L. Bauer, S. Garriss, J. M. McCune, M. K. Reiter, J. Rouse, and P. Rutenbar. Device-enabled authorization in the Grey system. Technical Report CMU-CS-05-111, Computer Science Department, Carnegie Mellon University, Feb. 2005.

[7] L. Bauer, S. Garriss, and M. K. Reiter. Distributed proving in access-control systems. In *Proc. 2005 IEEE Symposium on Security and Privacy*, May 2005.

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

[9] A. Beaufour and P. Bonnet. Personal servers as digital keys. In *Proc. 2nd IEEE International Conference of Pervasive Computing and Communications*, Mar. 2004.

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

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

[12] T. Coquand. An algorithm for testing conversion in type theory. In G. Huet and G. Plotkin, editors, *Logical Frameworks*, pages 255–280. 1991.

[13] D. L. de Ipiña, P. Mendonça, and A. Hopper. TRIP: a low-cost vision-based location system for ubiquitous computing. *Pers. and Ubiq. Comp.*, 6(3), 2002.

[14] S. Dohrmann and C. Ellison. Public key support for collaborative groups. In *Proc. First Annual PKI Research Workshop*, Apr. 2002.

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

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

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

[18] J. Y. Halpern and R. van der Meyden. A logic for SDSI's linked local name spaces. In *Proc. 12th IEEE Computer Security Foundations Workshop*, 1999.

[19] R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. *J. ACM*, 40(1):143–184, Jan. 1993.

[20] B. Lampson, M. Abadi, M. Burrows, and E. Wobber. Authentication in distributed systems: Theory and practice. *ACM Trans. Comp. Sys.*, 10(4):265–310, Nov. 1992.

[21] R. Levin. PGP snowflake. Personal communication, 1996.

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

[23] J. M. McCune, A. Perrig, and M. K. Reiter. Seeing-is-believing: Using camera phones for human-verifiable authentication. In *Proc. 2005 IEEE Symposium on Security and Privacy*, May 2005.

[24] A. Perrig and D. Song. Hash visualization: A new technique to improve real-world security. In *Proc. 1999 Intern. Work. Crypto. Techn. and E-Comm.*, July 1999.

[25] B. Ramsdell. Secure/multipurpose internet mail extensions (S/MIME) version 3.1: Message specification. RFC 3850, July 2004.

[26] N. Ravi, P. Stern, N. Desai, and L. Iftode. Accessing ubiquitous services using smart phones. In *Proc. 3rd Intern. Conf. Pervasive Comp. and Comm.*, 2005.

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

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

[29] D. Scott, R. Sharp, A. Madhavapeddy, and E. Upton. Using visual tags to bypass Bluetooth device discovery. *Mobile Comp. and Comm. Review*, 1(2), Jan. 2005.

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

[31] E. Wobber, M. Abadi, M. Burrows, and B. Lampson. Authentication in the Taos operating system. *ACM Trans. Comp. Sys.*, 12(1):3–32, Feb. 1994.