Chapter 6

Smartcard Integration

In this chapter we show how to implement our certificate issuing and showing protocol techniques in smartcards. We realize all the smartcard benefits listed in Section 1.1.6, without adding complexity and without downgrading security or privacy. We also show how to tune our protocols in the smartcard-enhanced setting to accommodate any degree of privacy desired.

6.1 Shortcomings of the smartcard-only paradigm

As we have seen in Section 1.1.6, smartcard-based implementations of PKI mechanisms offer numerous advantages over software-only implementations. Many of the benefits do not exist, though, when certificates are implemented entirely in smartcards. In this section we describe the shortcomings of smartcard-only implementations.

6.1.1 Privacy dangers

In a brochure [358], the Smart Card Forum states: "Smart card technology, if properly designed and implemented, can enhance both the fact and the perception of the individual's ability to exercise a much greater degree of control over personal information than is the case with any comparable delivery system." While it is true that smartcards enhance the perception of privacy, perception and fact are two very different things:

- The smartcard systems currently in use do nothing to prevent organizations from linking and tracing all communications and transactions by the same cardholder. For security reasons, they operate by transmitting in each transaction a unique card identifier that can be linked to central database entries that hold all kinds of identifiable personal data.

- A smartcard, or any other tamper-resistant device for that matter, operates as a black box to anyone but its manufacturer; its inner workings cannot be scrutinized or modified by its holder. (Most outsiders are not capable of breaking into smartcards to determine their internal behavior, and detected attempts to break in may result in criminal charges.) Smartcards can be programmed to operate as Trojan horses, offering convenience to their holders while at the same time covertly sending out their personal data.

The idea that government agencies may cause organizations to build in backdoors into devices is not far-fetched, as history reveals:

- In the early eighties the U.S. Justice Department misappropriated PROMIS, a software program developed by Inslaw Inc. that facilitated the integration of all sorts of databases. The U.S. intelligence community incorporated a backdoor and sold the modified version to intelligence organizations and banks in over 80 foreign countries; see Fricker [175], Kimery [231], Kunkin [243], Leon [248], and the Washington Weekly [372].

- In the mid eighties, the NSA worked together with Systematics, a major supplier of software for back office clearing and wire transfers, to introduce backdoors in much of the banking software supplied to U.S. financial institutions; see Grabbe [197] and Russell [332].

- In 1995, Der Spiegel and The Baltimore Sun reported that the NSA since 1957 had a secret deal with the Swiss cryptography company Crypto AG, under which Crypto AG built backdoor access into its encryption products; see Bamford [19, Chapter 8], Global Network Navigator [188], Strehle [364], the Baltimore Sun [371], and Madsen [254].

- In 1996, Walsh, a former deputy of the Australian Security Intelligence Organisation (ASIO, the Australian equivalent of the NSA), conducted an influential study [386] about Australian cryptography policy. In the uncensored version of the report, Walsh recommended: "Authority should be created for the AFP, the NCA and ASIO to alter proprietary software so that it performs additional functions to those specified by the manufacturer. [...] The software (or more rarely the hardware) may relate to communication, data storage, encoding, encryption or publishing devices. [...] some modifications may [...] create an intelligent memory, a permanent set of commands not specified in the program written by the manufacturer or a remote switching device with a capacity to issue commands at request. [...] In the event of an equipment or software malfunction or failure to perform to full specification, an investigation of a complaint could lead to discovery of the modification. The issue of liability of the Commonwealth may then potentially arise."

Covert data exchange between a tamper-resistant device and the outside world can take place through a number of channels. Two obvious channels are the following:

- The device can covertly send out or receive data by exploiting the van Eck effect or simply by sending out or receiving radio signals. We call this the *physical broadcast channel*. It is a particular problem in case of contactless smartcards, which can communicate over much larger distances than commonly believed. For instance, On Track Innovation Ltd. markets a contactless smartcard that it claims can communicate with a reader over a distance of up to 30 meters; the card can be remotely "revised, expanded and changed, and new applications can be added, without having to replace the card itself."
- Another channel becomes available when the device is engaged in a protocol execution with another device. Along with the data specified by the protocol, the device can leak out data. Alternatively, data can be leaked by changing the format of the messages as specified by the protocol; stop bits can be flipped, message fields set to values outside of specified ranges, and so on. We call this the *piggyback channel*.

The following channels are more surreptitious:

- In case a device engages in an interactive protocol, one or more bits of information can be leaked by halting during the protocol execution. For instance, by deciding whether or not to respond to a challenge before a time-out the device can leak one bit. More generally, in a protocol with t rounds, one of $t + 1$ preestablished messages can be leaked. This is called the *halting channel* (also known as the stopping channel).
- In a similar manner, a device can leak out bits of information by timing the delay before transmitting a message; this is called the *timing channel*. In contrast to the halting channel, this requires the device to have access to a timer. (This can be as simple as resetting a counter value in a chip register when a message is received, and sending out a response message when the counter reaches zero.) If messages are transmitted over a reliable channel with known performance characteristics, many bits can be leaked without noticeable delays.
- A related channel is the *computation error channel*. Here, a device deliberately causes one of its response messages to be incorrect. The fact whether or not a response is correct can be used to leak one bit of data, while an incorrect response can leak as many bits as its binary size.

Even more surreptitious are *subliminal channels*, first studied by Simmons [354, 355] in the setting of two prisoners who want to hatch an escape plan but can only communicate via a warden. Here, the data transmitted complies with all the range limitations, format specifications, and verification relations specified by the protocol, but

the manner in which the data is formed does not. In particular, data is leaked in such a manner that not even an interposed observer can distinguish a legitimate protocol execution from one in which bits are leaked. Following Chaum [97], we distinguish two subliminal channels:

- Any data that is subliminally leaked by a device is called *outflow*. The data can be leaked by encoding covert message bits in any data that may be chosen by the device. If the device encrypts its covert messages under a random key known only to the receiving device, an observer with unlimited computing power will not be able to tell the difference with a truly random number.
- Any data that is subliminally leaked to a device is called *inflow*. In the showing protocols in Chapter 3, the verifier can encode a covert message into the part of its challenge message that serves to protect against replay, possibly encrypted under a mutually known key. Any other data that is hashed along and under the control of the verifier can also be misused to cause inflow.

Certificate holders will in general be more concerned about outflow than inflow. Outflow can reveal their device identifier, access control code, communication and transaction history, attributes, data from other applications running on the same device, and so on. The device does not need to know with whom it is communicating; it can simply leak out the data with each protocol execution, possibly encrypted under a key of the intended recipient. Outflow gives verifiers and other parties the power to discriminate against certificate holders and to follow certificate holders around in real time.

Outflow is particularly powerful when triggered by inflow. For example, a receiving device could transmit its device identifier, so that the smartcard can decide whether or not to leak data. New rule specifications can be uploaded dynamically by means of inflow, and the certificate holder's device can be queried in an arbitrary manner.

When outflow is prevented, inflow is less of a problem, but even then it poses dangers:

- Inflow messages can cause the smartcard to enter a state of suspension or to default in other ways. For example, an inflow message could say "If the device identifier is #134522, then abort." Errors can be made to occur that appear random to the device holder.
- In cases where the device is expected to send out at least one message, it can always respond to inflow queries by means of the halting channel.
- When the smartcard is returned to its issuer, either to be replaced or for another reason, the card issuer can read out all the inflow collected by the card during the time it was in the hands of its holder. (The card issuer may for instance make the card dump its memory contents by entering a special access

code.) In this manner, the card issuer may be able to retroactively trace all the communications and transactions of the individual.

Inflow is most powerful when receiving devices are tamper-resistant as well.

The trend to store multiple applications on a single smartcard increases the scope for privacy invasion. It is impossible for anyone but the card issuing organizations to verify that different pieces of personal data are used only by the application for which they were collected.

The disparity between perception and fact makes the smartcard-only model extremely dangerous, especially since the interests of card issuers are not aligned with those of individuals. Even if the issuer and the manufacturer(s) of the smartcards could be trusted, implementing our issuing and showing protocol techniques entirely in tamper-resistant smartcards would not preserve privacy. Namely, most of today's smartcards can produce only pseudorandom numbers, on the basis of a random seed value that must be installed by (or under the supervision of) the issuer, to guarantee its uniqueness, randomness, and secrecy. Therefore, the issuer can compute all the pseudorandom numbers used by each device during its lifetime. Even if the issuer were to state that it does not know the seed values, or that its tamper-resistant devices produce random numbers by post-processing bits sampled from an internal "true" randomness source (such as a noise diode), this statement cannot be publicly verified. Also, the quality of a noise generator or other hardware methods for generating random numbers cannot be guaranteed.

Privacy concerns are one of the main reasons why smartcard implementations worldwide are stalling. As Schwartz [343] notes, "Ultimately, smart cards will not be able to succeed if consumers do not trust them. If the tracking ability of the cards weighs greater in the minds of consumers than convenience, the cards will not succeed in the market."

6.1.2 Other shortcomings

In addition to these serious privacy problems, there are other reasons to believe that the smartcard-only paradigm is inadequate and dangerous:

- Great care must be taken that the addition of complex circuitry and software does not introduce weaknesses in the tamper-resistance characteristics of the hardware. The cost of developing and incorporating new circuitry and software is much greater in the case of smartcards than in environments where tamper-resistance is not a concern. The ability to protect smartcards hinges on having enough capability and space for a software solution.

- With smartcard components already cramming for space, adding circuitry adversely affects smartcard reliability. A "dead" card at the very least inconveniences and frustrates its holder, and in many applications the consequences

can be quite dramatic. If multiple applications are stored on the same card, its holder may all of a sudden be locked out of a number of services.

- Because of size constraints, there will always be serious limitations to the functionality that can be squeezed into a smartcard. The amount of smartcard memory needed to keep track of a detailed log of all uses of a certificate (for the convenience of its holder) is substantial. The display and keyboard sizes of a so-called "super smartcard" are adequate only for communication of the most rudimentary data.

- The smartcard must be relied on to protect the security interests of its holder. Since standard smartcards do not have their own display and keyboard, user identification data must be entered on a terminal communicating with the card, and this terminal must be trusted not to capture the user's identification data. Likewise, any results that the card wants to communicate to its holder must be displayed on the terminal. The result is that a variety of *fake-terminal attacks* become possible. Even with a super smartcard there is no way for cardholders to verify that the data entered into and displayed by their card is the same as that processed and output by the card.

- If the secret keys of certificate holders are generated inside a smartcard, there is no way to verify that they are generated in such a manner that others cannot guess them. In particular, as we have seen in the previous section, it is very hard to guarantee that the CA cannot reconstruct all the secret keys. This makes the legal status of digital signatures highly doubtful.

- Card issuers for security reasons will need to migrate to a new tamper-resistant chip design every couple of years. During the renewal period, the individual is unable to use his or her certified key pairs, and is inconvenienced by the loss of the functionality stored on the card.

In sum, smartcards are far from the most logical place to install additional circuitry and software intended to expand functionality for the card holder.

In spite of the pitfalls, the security advantages of smartcards are so great that smartcards should not be abandoned. We now examine how to overcome the shortcomings of the smartcard-only model and realize all the benefits described in Section 1.1.6.

6.2 Combining smartcards and software-only devices

A smartcard is not a stand-alone computing device; by its very nature, it can operate only in combination with another device. Indeed, most smartcards do not have their own power source. In many communication and transaction settings, smartcards are

naturally used in conjunction with user-controlled software-only computing devices. For example, to use a smartcard over the Internet, it will have to be connected to a desktop computer, notebook, handheld, or otherwise.

The integration of smartcards and software-only computing devices (such as PCs, notebooks, and handhelds) is already well underway. Standardization efforts by major industry consortiums include the PC/SC Workgroup [297] and the OpenCard Framework[293]. Smartcard readers for PCs come in many forms: as a serial port add-on, embedded into the keyboard, as a device connected to the keyboard port, plugging into a floppy drive, combined with a modem, or as a PC Card. Prices are dropping rapidly owing to new design methods. In October 1998, Microsoft announced Smart Cards for Windows (subsequently renamed to Windows for Smart Cards), an 8-bit multi-application operating system for low-cost smart cards with 8 kilobytes of ROM, designed to extend the desktop computer environment into smartcard use.[1]

6.2.1 Benefits

Combining smartcards with user-controlled software-only computers offers many advantages:

- Fake terminal attacks can be prevented. If the user's computer has a keyboard and a display, the user can enter his or her password, PIN, or biometric using the keyboard, and can read out any transaction information from the computer's display; there is no need to rely on someone else's device for interacting with one's own smartcard.
- With a proper design, most of the computational and storage burden of the smartcard can be moved to the user-controlled computer, which can be much more powerful. In particular, high-strength public key cryptographic techniques may become entirely feasible using low-cost smartcards without cryptographic coprocessors. (See Section 6.5.1 for an example.)
- As a consequence of the fact that fake terminal attacks need not be prevented and sophisticated public-key cryptographic techniques are entirely feasible, verifiers do not need tamper-resistant terminals that represent the security interests of a CA.[2] This makes it much easier to become an acceptor of certificates.
- Any individual or software or hardware manufacturer can make or provide its own functional or other enhancements for the user's computer and sell them on the open market; this is an attractive feature for all parties alike.

[1] Users of Windows NT 5.0, for instance, can gain network access by presenting an X.509v3 certificate stored on a smartcard.

[2] In today's electronic purse systems and other off-line smartcard systems based on symmetric cryptography, verifier terminals must be strongly tamper-resistant to protect a master key that enables reconstruction of all smartcard secret keys.

- The user's computer can safeguard the secret keys of the user, and can make, store (for the purpose of non-repudiation), and verify any digital signatures. It can also keep its own chronological transaction log, and allow the user to review and print it. Special software could provide certificate management functions, such as help the user track his or her own transactions and categorize them. The transaction data could automatically flow into a software program, so that the user need never key in any data. For improved confidentiality, any sensitive data stored on the user's computer can be encrypted using a password-derived secret key.

- Schneier and Shostack [336] discuss the security ramifications of the fact that many different parties are involved with manufacturing and handling smartcards. Among other measures, they argue that a prime candidate for improvement is to place the user interface under the control of the user. This fits perfectly with the model considered here.

Compact operating systems, open source code, independent code reviews, digital certification of source code and executables, open market availability, and anti-virus software all contribute to the trust that users can place in the software running on their own computer.

Ideally, desktop computers are used only as devices to communicate over the Internet and to run application software, and the user-controlled computer is in the form of a personal digital assistant (PDA) or similar handheld device. The handheld device enables its holder to port his or her certified key pairs, and is better suited as a medium to run trusted code. Access to the Internet by PDAs is on the rise. In 1998, Semico Research predicted that the sales of small portable computers will reach 7.5 billion dollars in the year 2003, and the Yankee Group estimated that by 2002 there will be over 21 million subscribers of mobile, wireless access to the Internet. Companies including Microsoft and 3COM have already announced smartcard enhancements for handheld devices.

By interposing their computer between the smartcard and the outside world, users can straightforwardly prevent certain data leakages:

- The user's computer can prevent the smartcard from covertly sending out data using physical broadcast. In cyberspace, there should rarely be a need to protect against the emission or reception of radio signals or electromagnetic radiation, especially when communications and transactions are untraceable. Communications and transactions in the physical world, however, require protection by means of physical shielding. General measures to protect against van Eck monitoring are expensive and mostly classified, but effective solutions are available for smartcards and other user devices. For instance, Cord Technologies, Inc. ships a low-cost electromagnetic shield slightly larger than a smartcard, which (according to its claims) closely approximates a Faraday Cage; by blocking low frequency magnetic fields and radiofrequency communication,

it prevents an inserted chipcard from communicating with a card reader and from receiving the electrical power needed to operate. Handhelds and other software-only devices could have such protection built into their smartcard slots.

- Software on the user's computer can verify all the data passing to and from the smartcard. It can do range checks, check verification relations, and so on. Sniffer programs can intercept data sent to communication ports and compare this with specified data formats, allowing the user's computer to detect the existence of a piggyback channel.[3] More generally, software running on the user's computer can sift any data before passing it on, or simply halt a transmission in case data fields do not comply with the protocol specifications.

Thus, the integration of smartcards and other tamper-resistant user devices with user-controlled computers provides a natural environment to protect against data leakages that are efficiently detectable. From now on, we will assume that the smartcard-enhanced systems that we will develop prevent such data leakages.

6.2.2 How not to cope with subliminal channels

Simmons [354, 355] and Desmedt, Goutier, and Bengio [135, Section 5] showed how to use randomization to destroy subliminal channels, in a general setting. In 1988, Chaum [97] proposed very similar techniques to prevent inflow and outflow in the setting of consumer transactions, particularly electronic cash. Chaum's smartcard techniques [97, 103, 109] have numerous drawbacks, though, that make them unsuitable for practical use.

By far the most serious shortcoming relates to security: the systemic security breaks down completely when an attacker succeeds in compromising the tamper-resistance of a single smartcard. The problem is Chaum's reliance on blind signature protocols. Anyone who extracts the secret key of a smartcard can copy and lend out certificates at will. The blinding makes it hard or even impossible to detect the fraud, even if all the showing protocol executions involve the online participation of the CA. Even if fraud could be detected, its source cannot be traced and the fraud cannot be contained in any other way than by suspending the operation of the entire system: distribution of CRLs and online certificate validation do not help, because the attacker cannot be stopped from retrieving new certificates.

The security problem is worsened by the way in which Chaum encodes attribute information. Instead of having the CA encode attributes by cryptographic techniques (blind signatures cannot accomplish this), Chaum simply has the smartcard digitally sign statements about attributes the CA has stored in its memory. This flawed approach is central to all Chaum's work, as witnessed for instance by the following:

[3] In this manner, engineers in May 1995 discovered that Microsoft's Registration Wizard covertly sent the entire directory structure of the user's hard drive to Microsoft.

- Chaum [97, page 16] proposes that the tamper-resistant device "sign statements requested by C that T checks are true based on credential data it maintains." (Here, C denotes the user-controlled computer, and T the tamper-resistant device.)

- Chaum [88] states: "When the young Bob graduates with honors in medieval literature, for example, the university registrar gives his representative a digitally signed message asserting his academic credentials. When Bob applies to graduate school, [...] his representative asks its observer to sign a statement that he has a B.A. cum laude and that he qualifies for financial aid based on at least one of the university's criteria (but without revealing which ones). The observer, which has verified and stored each of Bob's credentials as they come in, simply checks its memory and signs the statement if it is true." (Chaum uses the terms "observer" and "representative" to refer to the smartcard and the user-controlled computer, respectively.)

- In the same manner, Chaum [88] deals with negative statements, such as felony convictions, license suspensions, or statements of pending bankruptcy: "Once the observer has registered a negative credential, an organization can find out about it simply by asking the observer (through the representative) to sign a message attesting to its presence or absence. Although a representative could muzzle the observer, it could not forge an assertion about the state of its credentials."

- In his smartcard-enhanced electronic cash systems [41, 103], Chaum gives each consumer smartcard the power to mint electronic money. Each smartcard stores electronic cash in the form of a counter value maintained in a chip register. For example, one hundred electronic dollars spendable up to cent granularity would be represented by a counter value of 10 000. To make a payment, a smartcard fills out the payable amount using a blinded electronic check and decreases its register value correspondingly.[4]

Consequently, the physical compromise of a single smartcard not only enables the attacker to anonymously flood the system with (copies of) untraceable certificates, but also to pretend to possess certificates for arbitrary attributes. Depending on the application, this can result in enormous damages. For instance, in a financial application the attacker may claim to have been issued digital bearer bonds worth huge sums of money, and may be able to impersonate people by assuming arbitrary identity attributes.

Chaum's blind trust in the tamper-proofness of smartcards is puzzling. Decades of experience have provided strong evidence that absolute tamper-proofness will

[4] Electronic purse systems based on symmetric cryptography also work this way, but the complete lack of privacy at least ensures that fraud can be contained.

likely never be fully realizable. Kocher [236], Anderson and Kuhn [12, 13], Boneh, DeMillo, and Lipton [38], and Biham and Shamir [29] discuss low-cost physical attacks that do not require direct physical access to the smartcard's internals. The most powerful non-invasive attack is Differential Power Analysis, due to Kocher, Jaffe, and Jun [237], in which the attacker gathers information correlated to secret keys by using statistical analysis and error correction techniques. Invasive physical attacks require days or weeks in a specialized laboratory and are much more costly to undertake, but they are also much harder to protect against. Microprobe workstations and other sophisticated equipment can be used to physically damage a smartcard chip in a controlled manner. See Kuhn [242, Section 1.2] and Kömmerling and Kuhn [239] for an overview of equipment to attack chips at the hardware level, and Biham and Shamir [30] for an invasive attack based on destroying a single wire of a carefully chosen memory cell using a laser beam. Even though manufacturers work hard to improve smartcard tamper-resistance, mass production smartcards will likely never be able to withstand invasive physical attacks for more than a couple of years following their release. New sophisticated apparatus will appear, and existing apparatus is being improved all the time. Organized crime can hire expertise comparable to that in national laboratories, and sophisticated tools are increasingly becoming accessible to hackers and undergraduate students at technical universities. Kömmerling and Kuhn [239] "see no really effective short-term protection against carefully planned invasive tampering involving focused ion-beam tools." They note that storage of secrets in battery-backed SRAM is highly effective to protect against invasive attacks (through zeroization), but this technology is not feasible for smartcards.

In a 1997 interview, Chaum [104] claimed: "On-line, software-only techniques sometimes become rather elaborate, whereas if you have a tamper-resistant device, you can do things in a relatively simple way." We now know, though, that the practice of relying solely on smartcard tamper-resistance is not acceptable. As Kocher, Jaffe, and Jun [237] point out with respect to Differential Power Analysis, the best way to deal with smartcard vulnerability is to design cryptographic systems under the assumption that secret key information will leak. In light of this, we will operate in this chapter under the assumption that smartcard tamper-resistance is a matter of economics. At the very least, we need the following security features:

- Each smartcard must have a unique independent secret key. (In practice, computational independence suffices, and so this principle is not violated by using diversified secret keys.)

- To guarantee that a smartcard design will be able to survive widespread physical compromise of smartcards, the design must provide for the ability to detect, trace and contain fraud even if all smartcards are compromised. That is, the smartcard-enhanced system should have all the security features of a secure software-only system. (This also facilitates a seamless migration path from software-only devices to a setting where certificate holders also hold smart-

cards, or vice versa.)

- Any sound smartcard system requires a strategy for migrating from one generation of smartcards to the next, to keep up with advances in smartcard tampering.

Chaum's smartcard techniques have yet another drawback: they cannot prevent the smartcard from storing data that the CA can use retroactively to trace transactions, once it gains access to the contents of the device. Any unique random number known to both the smartcard and the verifier in the showing protocol execution can be used to trace the transaction. Cramer and Pedersen [122] modified Chaum's protocols to prevent the smartcard and the verifier from developing *common coin flips*, i.e., mutually known information that is statistically correlated, other than inflow and outflow. A drawback of their protocol is that both the smartcard and the user's computer must perform several real-time exponentiations that cannot be preprocessed, and the issuing protocol consists of 10 moves. More seriously, the modified protocol does nothing to overcome the security problems of Chaum's techniques, and in fact degrades security even further. An anonymous extortioner in cyberspace can force an individual to retrieve certificates for which the extortioner supplies the blinding factors, and can subsequently show these by requiring the victim to assist in performing the showing protocol execution. The extortioner can blind any data from and to the verifier before relaying it, and in this manner can remain untraceable at all times to anyone including the victim.[5] When certificates represent money or other significant value that verifiers can redeem from the CA, this *perfect crime* exposes certificate holders to unacceptable risk.

In the remainder of this chapter we show how to overcome all these problems.

6.3 Secure smartcard integration

In this section we show how to overcome the security and efficiency shortcomings described in the previous section. The goal is to ensure that the user-controlled computer cannot show a certificate without the assistance of the smartcard. This will lay the foundation for the privacy techniques in Section 6.4.

6.3.1 Technique based on the DLREP function

We replace $\mathcal{P}$ in the showing protocols of Chapter 3 by $<\mathcal{S}, \mathcal{U}>$. $\mathcal{S}$ denotes the smartcard and $\mathcal{U}$ the user's software-only computer. $\mathcal{S}$ and $\mathcal{U}$ must be configured in such a manner that any protocol data to and from $\mathcal{S}$ flows through $\mathcal{U}$. The system

[5] A simpler form of this extortion strategy applies in the software-only setting, where the extortioner needs the "assistance" of the victim only in the issuing protocol. In the special case of electronic cash, this is known as "payee untraceability."

parameters and the public key h are generated as described in Section 3.2, and atomic propositions are demonstrated as depicted in Figures 3.1 and 3.5.

According to Proposition 3.6.1(b), the demonstration of a satisfiable Boolean formula is a proof of knowledge of a DL-representation of h with respect to $(g_1, \ldots, g_l)$. From this we immediately obtain the following result.

Proposition 6.3.1. *If $(x_1, \ldots, x_l)$ is shared between $\mathcal{S}$ and $\mathcal{U}$ in such a manner that $\widehat{\mathcal{U}}$ cannot compute with non-negligible success probability a DL-representation of $h := \prod_{i=1}^{l} g_i^{x_i}$ with respect to $(g_1, \ldots, g_l)$, then $\mathcal{U}$ cannot demonstrate any Boolean formula without the assistance of $\mathcal{S}$.*

Note that $\mathcal{S}$'s assistance is required even to demonstrate formulae in which its share of the DL-representation does not appear, and in particular to demonstrate the formula TRUE.

Any manner of sharing the DL-representation of h will do; all that matters is that $\widehat{\mathcal{U}}$ cannot compute the entire representation by itself. Not all sharing methods are equally suitable, though. Non-linear sharing makes it difficult for $\mathcal{S}$ to assist $\mathcal{U}$ in the showing protocol in any other manner than by providing to $\mathcal{U}$ its share of the representation. The resulting showing protocol would be insecure, since $\mathcal{U}$ can reuse h in subsequent formula demonstrations without needing $\mathcal{S}$'s assistance. The following construction shows how $(x_1, \ldots, x_l)$ may be shared securely and efficiently.

Proposition 6.3.2. *Suppose that x_1 is generated at random from $\mathbb{Z}_q$, and is known to $\mathcal{S}$ but not to $\mathcal{U}$. Regardless of the distribution of $(x_2, \ldots, x_l)$, $<\mathcal{S}, \mathcal{U}>$ can demonstrate to $\mathcal{V}$ any Boolean formula of the form (3.7) in which x_1 does not appear, in such a manner that:*

- *The only task that $\mathcal{S}$ performs in each formula demonstration is one execution of the Schnorr proof of knowledge, in which it proves knowledge of x_1 to $\mathcal{U}$.*
- *The view of $\widetilde{\mathcal{V}}$ when interacting with $<\overline{\mathcal{S}}, \overline{\mathcal{U}}>$ is the same as it would be when interacting with $\overline{\mathcal{P}}$.*

Proof. Let $h = h_{\mathcal{S}} h_{\mathcal{U}}$, where $h_{\mathcal{S}} = g_1^{x_1}$ and $h_{\mathcal{U}} = \prod_{i=2}^{l} g_i^{x_i}$. To show how the demonstration of a formula takes place, we consider four cases:

Case 1. F is an atomic formula consisting of zero or more "AND" connectives and no other connectives, in the form (3.3). We assume the permutation $\pi(\cdot)$ of $\{1, \ldots, l\}$ has 1 as a fixed point. If x_1 does not appear in the formula to be demonstrated by $<\mathcal{S}, \mathcal{U}>$, then the 1-st column in the matrix on the left-hand side of the system (3.2) contains all zeros.

To compute a in Step 1 of the showing protocol, $\mathcal{S}$ generates a random number $w_1 \in \mathbb{Z}_q$, computes $a_{\mathcal{S}} := g_1^{w_1}$, and sends $a_{\mathcal{S}}$ to $\mathcal{U}$. $\mathcal{U}$ generates $l - t - 1$

random numbers, $w_2, \ldots, w_{l-t} \in \mathbb{Z}_q$, and computes

$$a := a_{\mathcal{S}} \prod_{i=2}^{l-t} g_{\pi(i)}^{w_i} \prod_{i=1}^{t} g_{\pi(l-t+i)}^{-\sum_{j=2}^{l-t} \alpha_{ij} w_j}.$$

To compute the $l-t$ responses in Step 2, $\mathcal{U}$ sends $\mathcal{V}$'s challenge c to $\mathcal{S}$. $\mathcal{S}$ computes $r_{\mathcal{S}} := cx_1 + w_1 \bmod q$, and sends $r_{\mathcal{S}}$ to $\mathcal{U}$. $\mathcal{U}$ sets $r_1 := r_{\mathcal{S}}$, and computes the other $l-t-1$ responses as specified in Step 2 of the protocol $(\mathcal{P}, \mathcal{V})$.

Clearly, the task performed by $\mathcal{S}$ is exactly an execution of the Schnorr proof of knowledge, in which it proves knowledge of $\log_{g_1} h_{\mathcal{S}}$ to $\mathcal{U}$, and the view of $\widetilde{\mathcal{V}}$ is the same as it would be in $(\overline{\mathcal{P}}, \widetilde{\mathcal{V}})$.

Case 2. F is an atomic formula consisting of zero or more "AND" connectives, one "NOT" connective, and no other connectives, in the form (3.4). As in Case 1, with the permutation $\pi(\cdot)$ having 1 as a fixed point, if x_1 does not appear in the formula to be demonstrated then the 1-st column in the matrix on the left-hand side of the system (3.4) contains all zeros.

To compute a in Step 1 of the showing protocol, $\mathcal{S}$ generates a random number $w_1 \in \mathbb{Z}_q$, computes $a_{\mathcal{S}} := g_1^{w_1}$, and sends $a_{\mathcal{S}}$ to $\mathcal{U}$. $\mathcal{U}$ generates $l-t$ random numbers, $w_0, w_2, \ldots, w_{l-t} \in \mathbb{Z}_q$, and computes

$$a := a_{\mathcal{S}}^{\delta} h^{-w_0} \prod_{i=2}^{l-t} g_{\pi(i)}^{w_i} \prod_{i=1}^{t} g_{\pi(l-t+i)}^{b_i w_0 - \sum_{j=2}^{l-t} \alpha_{ij} w_j}.$$

To compute the $l-t+1$ responses in Step 2, $\mathcal{U}$ sends $\mathcal{V}$'s challenge c to $\mathcal{S}$. $\mathcal{S}$ computes $r_{\mathcal{S}} := cx_1 + w_1 \bmod q$, and sends $r_{\mathcal{S}}$ to $\mathcal{U}$. $\mathcal{U}$ computes $r_1 := r_{\mathcal{S}}\delta \bmod q$, and computes the other $l-t$ responses as specified in Step 2 of the protocol $(\mathcal{P}, \mathcal{V})$.

Again, the two claimed properties can easily be seen to hold.

Case 3. F is a formula connecting atomic subformulae by zero or more "OR" connectives and no other connectives, in the form (3.6). Recall from Section 3.5 that $\mathcal{P}$ simulates all but one of the subformula demonstrations, even if multiple subformulae hold true. The advantage of this is now clear: $\mathcal{U}$ needs the assistance of $\mathcal{S}$ only for the one subformulae demonstration that cannot be simulated, using the method of either Case 1 or Case 2.

Case 4. F is an arbitrary Boolean formula, in the form (3.7). Recall from Section 3.6 that $\mathcal{V}$'s challenge is the same for all j subformula demonstrations. The advantage of this is now clear: each of the j subformulae that must be demonstrated

contains (at least) 1 subsubformula that holds true, yet $\mathcal{S}$ need only perform one execution of the Schnorr proof of knowledge to provide its assistance. Specifically, $\mathcal{U}$ can use $\mathcal{S}$'s response $r_{\mathcal{S}}$ for each of the j subformula demonstrations, using the method of either Case 1 or Case 2.

This completes the proof. □

This construction works regardless of whether $\mathcal{V}$ is to receive a signed proof, a 4-move zero-knowledge proof, or otherwise.

Note that $\mathcal{U}$ can precompute all the required exponentiations, except for the operation of raising $a_{\mathcal{S}}$ to the power δ in Case 2 of the proof. In case $\mathcal{U}$ does not know the formula to be demonstrated until it communicates with $\mathcal{V}$, it does not help for $\mathcal{S}$ to provide its initial witness well before the start of the showing protocol execution. In many practical applications δ will be small, and the burden of raising $a_{\mathcal{S}}$ to the power δ is insignificant. In PKIs where speed is of utmost importance, $\mathcal{U}$ may prefer to avoid the exponentiation of $a_{\mathcal{S}}$ altogether by simply multiplying c by δ before passing it on to $\mathcal{S}$.[6] A drawback of the latter approach is that $\mathcal{V}$ cannot encode information into its challenge that is intended for $\mathcal{S}$, because $\mathcal{U}$ modifies it before passing it on. In Section 6.4 we will see why this may be undesirable.

The construction in Proposition 6.3.2 ensures not only that $\mathcal{S}$ need not provide its assistance for each (sub)subformula, but also lays the foundation for $\mathcal{U}$'s privacy. Note, for example, that $\mathcal{S}$ does not need to know h, l, the attributes $(x_2, \ldots, x_l)$, or the formula to be demonstrated. Section 6.4 will elaborate on this observation.

Security follows from the following result.

Proposition 6.3.3. *If the Schnorr proof of knowledge is witness-hiding, then $\widehat{\mathcal{U}}$ in the construction in Proposition 6.3.2 needs the assistance of $\overline{\mathcal{S}}$ in the t-th formula demonstration with respect to h, for any integer t.*

In other words, the only way to get around $\mathcal{S}$ is to physically break its tamper-resistance and extract x_1. The formal proof of this security result proceeds by generating $g_2, \ldots, g_l$ all as random powers of g_1. From a successful formula demonstration in which $\widehat{\mathcal{U}}$ does not use the assistance of $\overline{\mathcal{S}}$, a DL-representation of h with respect to $(g_1, \ldots, g_l)$ can be extracted. With overwhelming success probability this can be converted into $\overline{\mathcal{S}}$'s secret key x_1, resulting in a contradiction. (For signed proofs, we need to assume the random oracle model.)

In a practical implementation one might care to ensure that $g_1 \neq 1$, but from a theoretical viewpoint there is no need to exclude this possibility.

In case $\mathcal{U}$ is allowed to use the same h in arbitrarily many showing protocol executions, a construction for $\mathcal{S}$ that offers weaker security than Proposition 6.3.3

[6] Extra operations may be needed to make this work. Consider for example a formula connecting two atomic subformulae, both with a "NOT" connective, by an "AND" connective. With δ_1 denoting the δ for the first subformula and δ_2 that for the second, $\mathcal{U}$ computes γ such that $c\delta_1 + \gamma = c\delta_2$ and adjusts for γ when computing a and its responses for the first subformula.

clearly will not do. The property that the assistance of $\overline{\mathcal{S}}$ is needed after arbitrarily many protocol executions is desirable, however, even in case of public keys for which $\overline{\mathcal{S}}$ refuses to further assist $\mathcal{U}$ once it has assisted in a predetermined number of showing protocol executions. Namely, it allows $\mathcal{S}$ to use the same x_1 as its contribution to arbitrarily many different public keys, rather than having to use a new random x_1 for each. (As we will see in Section 6.4.3, this approach also improves privacy.) $\mathcal{S}$'s share x_1 can be regarded as its secret key, and $h_{\mathcal{S}}$ as its "public" key.

An even stronger security result can be proved by letting $\mathcal{S}$ perform an execution of Okamoto's extension of the Schnorr proof of knowledge [288, page 36], or more efficiently its optimization described in Section 2.4.3.

Proposition 6.3.4. *Let $l \geq 2$, and suppose that x_1 is generated at random from $\mathbb{Z}_q$, that x_2 is the outcome of a random coin flip, and that (x_1, x_2) is known to $\mathcal{S}$ but not to $\mathcal{U}$. Regardless of the distribution of $(x_3, \ldots, x_l)$, $<\mathcal{S}, \mathcal{U}>$ can demonstrate to $\mathcal{V}$ any Boolean formula of the form (3.7) in which x_1 and x_2 do not appear, in such a manner that:*

- *The only task that $\mathcal{S}$ performs in each formula demonstration is one execution of the three-move proof of knowledge described in Section 2.4.3, in which it proves knowledge to $\mathcal{U}$ of a DL-representation of $g_1^{x_1} g_2^{x_2}$ with respect to (g_1, g_2).*
- *The view of $\widetilde{\mathcal{V}}$ when interacting with $<\overline{\mathcal{S}}, \overline{\mathcal{U}}>$ is the same as it would be when interacting with $\overline{\mathcal{P}}$.*

The construction is readily apparent when studying the construction in the proof of Proposition 6.3.2. Because $\mathcal{S}$'s proof of knowledge is provably witness-hiding, Proposition 6.3.3 can be proved under the weaker assumption that the DL function used to implement the commitment to $(x_1, \ldots, x_l)$ is one-way. In a practical implementation one might care to ensure that $g_1, g_2 \neq 1$, but from a theoretical viewpoint there is no need to exclude this possibility.

In addition, it is possible to prove the following result.

Proposition 6.3.5. *If the DL function is one-way, and $\overline{\mathcal{S}}$ performs no more than poly-logarithmically many executions of its protocol with $\mathcal{U}$, then signed proofs of formula demonstrations with respect to h are unforgeable by $\widehat{\mathcal{U}}$ in the random oracle model, regardless of the distribution of $(x_3, \ldots, x_l)$.*

The same should be true for the construction of Proposition 6.3.2, but it is unclear how to prove this.

It is straightforward to combine our smartcard-enhanced showing protocols with any of the DLREP-based certificate issuing protocols described in Chapter 4. In case unlimited-show certificates are issued, $\mathcal{S}$ is not needed in the issuing protocol; the CA simply looks up (or reconstructs) the secret key of $\mathcal{S}$, and encodes that into the certified key pair it issues. In the case of limited-show certificates, $\mathcal{U}$ in Step 2 of

the issuing protocol must hash along its (master) initial witness (set) for the showing protocol, and so $\mathcal{S}$ must provide (each) $a_{\mathcal{S}}$ already at this stage. (Providing the initial witnesses in an early stage is desirable anyway, to enable $\mathcal{U}$ to precompute as many exponentiations as possible.) In addition, $\mathcal{S}$ must keep track of the number of times it has cooperated in demonstrating a formula with respect to a limited-show certificate, and refuse to cooperate more times than allowed. In practice, limiting the number of times a certificate can be shown is trivial: for each of its initial witnesses, $\mathcal{S}$ responds just once anyway.

Implementing our limited-show techniques in smartcards offers two security layers: showing a certificate more times than allowed is not possible without physically extracting the smartcard's secret key, and perpetrators (who manage to physically extract the secret keys from their smartcards and then commit fraud) can be traced. Thus, we get both the tamper-resistance protection and the software-only protection.

In the case of dynamic limited-show certificates (see Section 5.4.2), the correction factors can all be determined by $\mathcal{U}$, as desired.

Any linear sharing method other than the methods in Propositions 6.3.2 and 6.3.4 can be used, with the obvious modifications. Different sharing methods may each have their own advantages, depending on other design aspects. For instance, when using DLREP-based scheme I or II, it may be preferable to let $(x_1, \ldots, x_{l-1})$ be known to $\mathcal{U}$ and let x_l be the sum of a secret key of $\mathcal{S}$ and the random blinding factor α_1 chosen by $\mathcal{U}$ in the issuing protocol; this enables the CA to encode $l-1$ attributes into each certified key pair,[7] instead of only $l-2$. Another advantage is that it is significantly easier to guarantee that a secret key of a certified key pair is generated secretly at random, and therefore that the device holder is the only one in possession of the secret key; it does not matter if another party can guess or determine the share of the secret key that is generated by one of the two devices, as long as the other device generates (part of) its share at random. Consequently, the legal status of digital signatures is easier to establish than in the software-only and smartcard-only paradigms.

In general, for any or all $i \in \{1, \ldots, l\}$, one can set $x_i = \alpha_i x_{i\mathcal{S}} + \beta_i x_{i\mathcal{U}} \bmod q$, for any $\alpha_i, \beta_i \in \mathbb{Z}_q$, where $x_{i\mathcal{S}}$ is $\mathcal{S}$'s share and $x_{i\mathcal{U}}$ that of $\mathcal{U}$.

When combining our smartcard-enhanced showing protocols with a secret-key certificate issuing protocol, the delegation strategy is never an issue. The simple software-only measures described in Section 5.1.2, if necessary at all, suffice to ensure that $\widehat{\mathcal{U}}$ cannot abuse delegation in order to circumvent the assistance of $\mathcal{S}$ in the showing protocol, even in those cases where delegation is not prevented but merely made harmless.

It is worth noting that $\overline{\mathcal{S}}$ can provide its assistance in alternative manners that differ from a Schnorr proof of knowledge or the related proofs of knowledge described

[7] When a limited-show certificate is shown more times than allowed, this sharing method does not allow the CA to trace the perpetrator by computing the secret key of $\mathcal{S}$. To achieve this property, one of $x_1, \ldots, x_{l-1}$ should be an identifier.

in Section 2.4.3. For instance, in the construction of Proposition 6.3.2 we may replace the verification relation of the Schnorr proof of knowledge, $g_1^{r_{\mathcal{S}}} = h_{\mathcal{S}}^{c} a_{\mathcal{S}}$, by one that is closely related to that of the DSA signature scheme: $g_1^{r_{\mathcal{S}}} = h_{\mathcal{S}}^{\overline{a_{\mathcal{S}}}} a_{\mathcal{S}}^{c}$. $\mathcal{S}$ must then compute its response as $r_{\mathcal{S}} := cw_1 + x_1\overline{a_{\mathcal{S}}} \bmod q$; it is easy to modify the actions of $\mathcal{U}$ correspondingly.

6.3.2 Technique based on the RSAREP function

For commitments based on the RSAREP function, the direct analogue of Proposition 6.3.1 applies. Specifically, if $(x_1, \ldots, x_{l+1})$ is shared between $\mathcal{S}$ and $\mathcal{U}$ in such a manner that $\widehat{\mathcal{U}}$ cannot compute with non-negligible success probability an RSA-representation of $h := \prod_{i=1}^{l} g_i^{x_i} x_{l+1}^{v}$ with respect to $(g_1, \ldots, g_l, v)$, then $\widehat{\mathcal{U}}$ cannot demonstrate any Boolean formula without the assistance of $\mathcal{S}$. The technique of letting x_1 be a secret key of $\mathcal{S}$ and $g_1^{x_1}$ its public key is not recommendable, though.[8] An efficient construction that achieves security for $\overline{\mathcal{S}}$ is the following.

Proposition 6.3.6. *Suppose that x_{l+1} is generated at random from $\mathbb{Z}_n^*$, and is known to $\mathcal{S}$ but not to $\mathcal{U}$. Regardless of the distribution of $(x_1, \ldots, x_l)$, $<\mathcal{S}, \mathcal{U}>$ can demonstrate to $\mathcal{V}$ any formula of the form (3.7), in such a manner that:*

- *The only task that $\mathcal{S}$ performs in each formula demonstration is one execution of the Guillou-Quisquater proof of knowledge, in which it proves knowledge of x_{l+1} to $\mathcal{U}$.*
- *The view of $\widetilde{\mathcal{V}}$ when interacting with $<\overline{\mathcal{S}}, \overline{\mathcal{U}}>$ is the same as it would be when interacting with $\overline{\mathcal{P}}$.*

Proof. Let $h = h_{\mathcal{S}} h_{\mathcal{U}}$, where $h_{\mathcal{S}} = x_{l+1}^{v}$ and $h_{\mathcal{U}} = \prod_{i=1}^{l} g_i^{x_i}$. To show how the demonstration of a formula takes place, we consider four cases:

Case 1. F is an atomic formula consisting of zero or more "AND" connectives and no other connectives, in the form (3.3). To compute a in Step 1 of the showing protocol, $\mathcal{S}$ generates a random number $w_{l+1} \in \mathbb{Z}_n^*$, computes $a_{\mathcal{S}} := w_{l+1}^{v}$, and sends $a_{\mathcal{S}}$ to $\mathcal{U}$. $\mathcal{U}$ generates $l - t$ random numbers, $w_1, \ldots, w_{l-t} \in \mathbb{Z}_v$, and computes

$$a := a_{\mathcal{S}} \prod_{i=1}^{l-t} g_{\pi(i)}^{w_i} \prod_{i=1}^{t} g_{\pi(l-t+i)}^{-\sum_{j=1}^{l-t} \alpha_{ij} w_j}.$$

[8] The following proof of knowledge, proposed by Girault [185] (see also Poupard and Stern [310]), is conjectured to be witness-hiding: $\mathcal{S}$ sends an initial witness $g_1^{w_1}$ to $\mathcal{U}$, receives a challenge c, and sends the response $cx_1 + w_1$ (no modular reduction) to $\mathcal{U}$. Unfortunately, the response size in a practical implementation is much larger than $|v|$ bits.

To compute the $l-t+1$ responses in Step 2, $\mathcal{U}$ sends $\mathcal{V}$'s challenge c to $\mathcal{S}$. $\mathcal{S}$ computes $r_{\mathcal{S}} := x_{l+1}^{c} w_{l+1}$, and sends $r_{\mathcal{S}}$ to $\mathcal{U}$. $\mathcal{U}$ sets

$$r_{l+1} := r_{\mathcal{S}} \prod_{j=1}^{l-t} (g_{\pi(j)} \prod_{i=1}^{t} g_{\pi(l-t+i)}^{-\alpha_{ij}})^{(cx_{\pi(j)}+w_j) \operatorname{div} v},$$

and computes the other $l-t$ responses in the manner specified in Step 2 of the protocol ($\mathcal{P}$, $\mathcal{V}$).

The task performed by $\mathcal{S}$ is exactly an execution of the Guillou-Quisquater proof of knowledge, in which it proves knowledge of the v-th root of $h_{\mathcal{S}}$ to $\mathcal{U}$, and the view of $\widetilde{\mathcal{V}}$ is the same as it would be in ($\overline{\mathcal{P}}$, $\widetilde{\mathcal{V}}$).

Case 2. F is an atomic formula consisting of zero or more "AND" connectives, one "NOT" connective, and no other connectives, in the form (3.4). It is easy to see that $\mathcal{S}$ can provide its assistance in the same manner as in Case 1, and again the two claimed properties are readily seen to hold.

Case 3. As in Case 3 of the proof of Proposition 6.3.2.

Case 4. As in Case 4 of the proof of Proposition 6.3.2.

This completes the proof. □

This construction works regardless of whether $\mathcal{V}$ is to receive a signed proof, a 4-move zero-knowledge proof, or otherwise.

Proposition 6.3.7. *If the Guillou-Quisquater proof of knowledge is witness-hiding, then $\widehat{\mathcal{U}}$ in the construction in Proposition 6.3.6 needs the assistance of $\overline{\mathcal{S}}$ in the t-th formula demonstration with respect to h, for any integer t.*

Assuming that interactively issued Guillou-Quisquater signatures are unforgeable, one can also prove that if $\mathcal{S}$ follows the protocol, then signed proofs of formula demonstrations with respect to h are unforgeable by $\widehat{\mathcal{U}}$, regardless of the distribution of $(x_1, \ldots, x_l)$.

As with the DLREP-based variants, stronger provability can be achieved by letting $\mathcal{S}$ perform an execution of Okamoto's extension of the Guillou-Quisquater proof of knowledge [288, page 39], or more efficiently its optimization described in Section 2.4.4.

Proposition 6.3.8. *Let $l \geq 1$, and suppose that x_1 is the outcome of a random coin flip, that x_{l+1} is generated at random from $\mathbb{Z}_n^*$, and that (x_1, x_{l+1}) is known to $\mathcal{S}$ but not to $\mathcal{U}$. Regardless of the distribution of $(x_2, \ldots, x_l)$, $<\mathcal{S}, \mathcal{U}>$ can demonstrate to $\mathcal{V}$ any Boolean formula of the form (3.7) in which x_1 does not appear, in such a manner that:*

- *The only task that $\mathcal{S}$ performs in each formula demonstration is one execution of the three-move proof of knowledge described in Section 2.4.4, in which it proves knowledge to $\mathcal{U}$ of an RSA-representation of $g_1^{x_1} x_{l+1}^{v}$ with respect to (g_1, v).*

- *The view of $\widetilde{\mathcal{V}}$ when interacting with $<\overline{\mathcal{S}}, \overline{\mathcal{U}}>$ is the same as it would be when interacting with $\overline{\mathcal{P}}$.*

This time, Proposition 6.3.7 can be proved under the weaker assumption that the RSA function used to implement the commitment is one-way. In addition, if the RSA function is one-way and $\overline{\mathcal{S}}$ performs no more than polylogarithmically many executions of the protocol with $\mathcal{U}$, signed proofs of formula demonstrations with respect to h are unforgeable by $\widehat{\mathcal{U}}$ in the random oracle model, regardless of the distribution of $(x_2, \ldots, x_l)$.

Other ways to share $(x_1, \ldots, x_{l+1})$ can be used as well. In general, for any or all $i \in \{1, \ldots, l\}$, one can use the linear sharing $x_i = \alpha_i x_{i\mathcal{S}} + \beta_i x_{i\mathcal{S}} \bmod v$ and the multiplicative sharing $x_{l+1} = x_{\mathcal{S}}^{\alpha_{l+1}} x_{\mathcal{U}}^{\beta_{l+1}}$, for any α_i, β_i. A sharing method that fits well when combining the showing protocol with one of the RSA-based issuing protocols described in Chapter 4 is to let $(x_1, \ldots, x_l)$ be the attributes of $\mathcal{U}$, and let x_{l+1} be the product of the secret key of $\mathcal{S}$ and the blinding factor α_1 used by $\mathcal{U}$ in the certificate issuing protocol (x_1 may be either an identifier of $\mathcal{U}$ or a part of the secret key of $\mathcal{S}$). Indeed, if the construction of Proposition 6.3.6 is used in combination with an RSAREP-based issuing protocol, $\mathcal{U}$ for reason of privacy may not let $\mathcal{S}$ determine the blinding factor α_1.

The same considerations as described in Section 6.3.1 apply when combining the showing protocol techniques with one of the issuing protocols in Chapter 3. In particular, the delegation strategy is never an issue.

6.4 Privacy protection

In Section 6.1.1 we have seen that there are many ways in the smartcard-enhanced setting to get around the privacy protections of the showing protocol. The techniques that we will develop in this section build on top of those developed in the previous section and enable $\overline{\mathcal{U}}$ to prevent inflow, outflow, and even the development of common coin flips. They also ensure that the leakage through the halting, timing, and computation error channels is at most 1 bit in total; this is the best result achievable. Furthermore, the smartcard cannot learn any information about the formulae demonstrated, and cannot determine the number of encoded attributes or whether or not certificates are limited-show; all it can learn is (an upper bound on) the number of times its holder has engaged in a showing protocol execution.

All the privacy guarantees hold in the strongest possible sense, namely in the presence of conspiring $\mathcal{S}$, CA, and $\mathcal{V}$ that have unlimited computing resources and

may establish secret information in an preparatory phase (e.g., a strategy for deviating from the protocol and keys for authenticating and encrypting covert messages). We also show how to tune our smartcard techniques to design systems for which one or more of the privacy protections cannot be achieved, while guaranteeing that $\mathcal{U}$ can verify and control any information exchanged by $\mathcal{S}$ and $\mathcal{V}$.

The security accomplishments of the previous section are fully preserved by the new techniques.

6.4.1 Inflow prevention

Inflow cannot occur in any of the certificate issuing protocols, since the smartcard does not receive any messages from the CA. In the smartcard-enhanced showing protocols in Section 6.3, the only possibility for inflow is $\mathcal{V}$'s challenge, c. If the demonstration is in the form of a zero-knowledge proof, the entire s-bit data channel (as well as part of $\mathcal{V}$'s commitment) can be used for inflow. Even in signed proofs, there may be ample opportunity for $\mathcal{V}$ to cause inflow. In this case, $\mathcal{V}$'s challenge is the result of the application of a sufficiently strong one-way hash function to certain data that may be at least somewhat under the control of $\mathcal{V}$, such as a nonce (e.g., a random number or an estimate of the time and date) or a random number γ as in Section 5.3. Any such data fields can be misused by $\mathcal{V}$ to encode a covert message, possibly encrypted under a key known to $\mathcal{S}$.

Proposition 6.4.1. *In the construction of Proposition 6.3.2, $\mathcal{U}$ can prevent inflow at the cost of essentially one additional exponentiation, which can be precomputed. $\mathcal{S}$ must now accept any challenge in $\mathbb{Z}_q$, but the tasks of $\mathcal{S}$ and $\mathcal{V}$ remain unchanged in all other respects.*

Proof. As in the proof of Proposition 6.3.2, we distinguish four cases:

Case 1. F is an atomic formula consisting of zero or more "AND" connectives and no other connectives, in the form (3.3). $\mathcal{U}$ proceeds as in Case 1 of the proof of Proposition 6.3.2, but when computing a from $a_{\mathcal{S}}$ it also multiplies in a factor $h_{\mathcal{S}}^{\beta}$, for a randomly chosen $\beta \in \mathbb{Z}_q$. Furthermore, instead of sending $\mathcal{V}$'s challenge c to $\mathcal{S}$, $\mathcal{U}$ sets $c_{\mathcal{S}} := c + \beta \bmod q$ and sends $c_{\mathcal{S}}$ to $\mathcal{S}$. No further changes are needed.

Case 2. F is an atomic formula consisting of zero or more "AND" connectives, one "NOT" connective, and no other connectives, in the form (3.4). $\mathcal{U}$ proceeds as in Case 2 of the proof of Proposition 6.3.2, but when computing a from $a_{\mathcal{S}}$ it also multiplies in a factor $h_{\mathcal{S}}^{\beta\delta}$, for a randomly chosen $\beta \in \mathbb{Z}_q$. Furthermore, instead of sending $\mathcal{V}$'s challenge c to $\mathcal{S}$, $\mathcal{U}$ sets $c_{\mathcal{S}} := c + \beta \bmod q$ and sends $c_{\mathcal{S}}$ to $\mathcal{S}$. No further changes are needed.

Case 3. F is a formula connecting atomic subformulae by zero or more "OR" connectives and no other connectives, in the form (3.6). $\mathcal{U}$ proceeds as in Case 3

of the proof of Proposition 6.3.2, applying the modification of either Case 1 or Case 2.

Case 4. F is an arbitrary Boolean formula, in the form (3.7). $\mathcal{U}$ proceeds as in Case 4 of the proof of Proposition 6.3.2, with the restriction that it uses the same random choice of β when forming a in each of the atomic subsubformulae for which a genuine proof is due. This ensures that the same blinded form of $\mathcal{V}$'s challenge works for all subsubformulae demonstrations for which $\mathcal{S}$'s assistance is needed.

Because β is chosen at random, there can be no inflow. □

This inflow prevention technique can also be applied to the construction in Proposition 6.3.4, and more generally to showing protocols based on any other kind of linear sharing discussed in Section 6.3. It can also readily be adapted to prevent inflow in any of the RSAREP-based smartcard-enhanced showing protocols in Section 6.3.2.

An even simpler technique to prevent inflow is to let $\mathcal{U}$ and $\mathcal{V}$ jointly develop $\mathcal{V}$'s challenge, in such a manner that it cannot contain covert message bits chosen by $\widetilde{\mathcal{V}}$. This approach is not recommendable, though. It increases the number of message exchanges between $\mathcal{U}$ and $\mathcal{V}$, and makes it harder to encode meaningful data into $\mathcal{V}$'s challenge message (for the purpose of a signed proof). Also, the approach is not always practical. For example, in the technique described in Section 5.3, $\mathcal{V}$ must form a commitment γ; randomizing this requires the commitment function to be of a number-theoretic nature (e.g., a DLREP or RSAREP function), which excludes much faster alternatives. Another unique and important advantage of our inflow prevention technique is that it destroys other data leakage channels as a by-product, as we will see in Section 6.4.3.

6.4.2 Outflow prevention

Outflow cannot occur in any of the certificate issuing protocols, since the CA does not receive any messages from $\mathcal{S}$. In the showing protocols there is some opportunity for outflow, through the response(s) provided by $\mathcal{S}$. If $\mathcal{V}$ in the smartcard-enhanced showing protocols in Section 6.3 knows $\mathcal{S}$'s share of the representation of h, then $\mathcal{S}$ can encode its outflow message in the "random" number(s) that it uses to compute $a_{\mathcal{S}}$, possibly encrypted under a key of $\mathcal{V}$ or of the CA. For example, in the construction of Proposition 6.3.2 or 6.3.6, $\mathcal{U}$ demonstrates an atomic formula without "NOT" connectives by passing $\mathcal{S}$'s response $r_{\mathcal{S}}$ on to $\mathcal{V}$, enabling $\mathcal{V}$ to extract $\mathcal{S}$'s covert message by computing w_1. In practice, $\mathcal{V}$ can guess $\mathcal{S}$'s share with non-negligible success probability if it knows the list of all smartcard secret keys. It could even determine the correct secret key by verifying whether w_1 satisfies a redundancy pattern or the like.

This outflow method does not work if an atomic formula with a "NOT" connective is demonstrated, because $\mathcal{U}$ applies δ to $r_{\mathcal{S}}$ before passing it on. According to

Proposition 3.4.3, if x_l (or, in the RSAREP-based protocol, $\mathcal{U}$'s share of x_{l+1}) is a random number that does not occur in the formula, then the demonstration does not leak any information about δ. $\mathcal{S}$ can still cause outflow if δ has smaller entropy than a random number in $\mathbb{Z}_q$ or $\mathbb{Z}_v$; this will be the case in most practical applications. In fact, $\mathcal{S}$ can first leak δ (without knowing it), by generating $a_{\mathcal{S}}$ in a manner known to $\mathcal{V}$, and in subsequent demonstrations with the same h (if any) use the full response channel for outflow.

Also, if the formula that is demonstrated is of the form (3.7), then $\mathcal{V}$ may learn information about which of the atomic subsubformula are valid. For each of the atomic subsubformulae for which $\mathcal{U}$ requires $\mathcal{S}$'s assistance, $\mathcal{V}$ sees either $r_{\mathcal{S}}$ or $r_{\mathcal{S}}\delta_i$ for some δ_i, and can check for correlations. (See the construction of Proposition 6.3.2.)

Proposition 6.4.2. *Suppose that $\mathcal{U}$ generates x_l at random from $\mathbb{Z}_q$ and only demonstrates formulae in which x_l does not appear. In the construction of Proposition 6.3.2, $\mathcal{U}$ can prevent outflow at the cost of essentially one additional exponentiation, which can be precomputed. The tasks of $\mathcal{S}$ and $\mathcal{V}$ remain unchanged.*

Proof. As in the proof of Proposition 6.3.2, we distinguish four cases:

Case 1. F is an atomic formula consisting of zero or more "AND" connectives and no other connectives, in the form (3.3). $\mathcal{U}$ proceeds as in Case 1 of the proof of Proposition 6.3.2, but when computing a from $a_{\mathcal{S}}$ it also multiplies in a factor g_1^{γ}, for a randomly chosen $\gamma \in \mathbb{Z}_q$. Furthermore, instead of passing $\mathcal{S}$'s response $r_{\mathcal{S}}$ on to $\mathcal{V}$, it sets $r_1 := r_{\mathcal{S}} + \gamma \bmod q$. No further changes are needed.

Case 2. F is an atomic formula consisting of zero or more "AND" connectives, one "NOT" connective, and no other connectives, in the form (3.4). $\mathcal{U}$ proceeds as in Case 2 of the proof of Proposition 6.3.2, but when computing a from $a_{\mathcal{S}}$ it also multiplies in a factor g_1^{γ}, for a randomly chosen $\gamma \in \mathbb{Z}_q$. Furthermore, $\mathcal{U}$ computes $r_1 := r_{\mathcal{S}}\delta + \gamma \bmod q$. No further changes are needed.

Case 3. F is a formula connecting atomic subformulae by zero or more "OR" connectives and no other connectives, in the form (3.6). $\mathcal{U}$ proceeds as in Case 3 of the proof of Proposition 6.3.2, applying the modification of either Case 1 or Case 2.

Case 4. F is an arbitrary Boolean formula, in the form (3.7). $\mathcal{U}$ proceeds as in Case 4 of the proof of Proposition 6.3.2. (For each atomic subsubformula for which a genuine proof is due, $\mathcal{U}$ uses an independently generated γ.)

Because the γ's are chosen at random, there can be no outflow. □

Because $\widetilde{\mathcal{S}}$ cannot cause outflow, accepting views of $\widetilde{\mathcal{V}}$ when interacting with $\overline{\mathcal{U}}$ are perfectly indistinguishable from those that result when interacting with $\overline{\mathcal{P}}$ in the showing protocols in Chapter 3. This immediately leads to the following results.

Corollary 6.4.3. *If $\mathcal{U}$ follows the construction described in the proof of Proposition 6.4.2, then Proposition 3.6.1(c) holds, regardless of the behavior of $\widetilde{\mathcal{S}}$ and any joint initial set-up by $\widetilde{\mathcal{S}}$ and $\widetilde{\mathcal{V}}$.*

Corollary 6.4.4. *If $\mathcal{U}$ follows the construction described in the proof of Proposition 6.4.2, then Proposition 3.6.2 and 3.6.3 both hold, regardless of the behavior of $\widetilde{\mathcal{S}}$ and any joint initial set-up by $\widetilde{\mathcal{S}}$ and $\widetilde{\mathcal{V}}$.*

In Section 6.3.1 we already saw that these security results are true assuming $\mathcal{S}$ follows the protocol but $\widehat{\mathcal{U}}$ need not. In most applications both types of results are desirable: $\overline{\mathcal{S}}$ should prevent $\widehat{\mathcal{U}}$ and $\widehat{\mathcal{V}}$ from forging signed proofs and from making them appear to pertain to formulae that do not apply, while $\overline{\mathcal{U}}$ should prevent $\widetilde{\mathcal{S}}$ and $\widehat{\mathcal{V}}$ from framing its holder by forming a signed proof that $\mathcal{U}$ did not originate.

The same outflow prevention technique can be applied to the construction in Proposition 6.3.4, and more generally to any other kind of linear sharing discussed in Section 6.3. In cases where $\overline{\mathcal{S}}$ provides more than one response to $\mathcal{U}$, the latter must independently randomize each of these responses, and the corresponding adjustments must be made when forming a. It is also straightforward to apply the outflow prevention technique to any of the RSAREP-based smartcard-enhanced showing protocols discussed in Section 6.3.2.

Our outflow prevention technique actively prevents outflow by randomizing data on the flight. It is much more powerful than the approach of Chaum [103], which can only detect outflow after the fact with low probability. It is also highly preferable over Chaum's [97, 109] other outflow prevention technique (also applied by Cramer and Pedersen [122]), which in our situation would proceed by letting $\mathcal{S}$ and $\mathcal{U}$ form $a_{\mathcal{S}}$ in a mutually random manner by means of a coin flipping protocol. This would seriously degrade the communication and computation complexity, and would prevent the optimization that will be introduced in Section 6.5.1. Another important advantage of our outflow prevention technique is that other data leakage channels are destroyed as a by-product, as we will now show.

6.4.3 Prevention of other data leakage channels

Inflow and outflow prevention are orthogonal measures. $\mathcal{U}$ can prevent both inflow and outflow by randomizing $\mathcal{V}$'s challenge c as well as any responses provided by $\mathcal{S}$, and multiplying the corresponding expressions into a when forming $a_{\mathcal{S}}$. Assuming simultaneous repeated squaring, the computational cost for demonstrating an atomic formula is approximately one exponentiation that can be precomputed.

Our issuing and showing protocols in Chapters 3 and 4 by their very design greatly reduce the scope for leakage of covert messages: no interaction takes place between $\mathcal{S}$ and the CA, and the interaction between $\mathcal{S}$ and $\mathcal{V}$ is minimal. What's more, the application of our inflow and outflow prevention techniques is so powerful that any other data leakage channels in the issuing and showing protocols are destroyed as a by-product:

- The smartcard learns virtually nothing:
  - $\widetilde{\mathcal{S}}$ cannot learn $\mathcal{U}$'s attributes, nor how many are encoded by the CA, regardless of the behavior of $\widetilde{\mathcal{V}}$. $\widetilde{\mathcal{S}}$ cannot even learn the g_i's that it does not use itself, unless they are known in advance. (In Section 6.5.1 we will describe an optimization in which $\widetilde{\mathcal{S}}$ merely learns q and a secret seed value to generate pseudorandom numbers.)
  - When interacting with $\overline{\mathcal{U}}$, $\widetilde{\mathcal{S}}$ cannot learn any information about the formula that is demonstrated, regardless of the behavior of $\widetilde{\mathcal{V}}$. This holds even if $\mathcal{U}$ issues a signed proof and a unique description of the formula is hashed along when computing $\mathcal{V}$'s challenge. $\widetilde{\mathcal{S}}$ cannot even learn the message that it helps sign.
  - $\widetilde{\mathcal{S}}$ cannot learn whether it is assisting in showing a limited-show certificate or an unlimited-show certificate, regardless of the behavior of $\widetilde{\mathcal{V}}$, assuming $\mathcal{S}$ in both cases provides its initial witness in advance (e.g., during the corresponding issuing protocol). (Note, though, that $\mathcal{U}$ cannot show a limited-show certificate more times than allowed, because $\mathcal{S}$ uses a new random initial witness in each showing protocol execution.) Likewise, $\widetilde{\mathcal{S}}$ cannot decide whether multiple invocations of its assistance are for the purpose of showing the same certificate or different certificates.
  - $\widetilde{\mathcal{S}}$ cannot even learn any information on the number of certificates issued to $\mathcal{U}$, since $\mathcal{U}$ can query $\mathcal{S}$ even if it has not been issued any certificates. (This practice is not recommendable when limited-show certificates represent value, such as in an electronic cash system.)

  In other words, all that $\widetilde{\mathcal{S}}$ can learn about the actions of its holder is an upper bound on the number of showing protocol executions.

- The verifier learns virtually nothing either (beyond the status of the formulae demonstrated):
  - $\widetilde{\mathcal{V}}$ cannot decide from its accepting views in protocol executions whether or not $\mathcal{P}$ is assisted by a smartcard, regardless of the behavior of $\widetilde{\mathcal{S}}$. In PKIs in which certificate applicants may but need not hold a smartcard, this reduces the scope for discrimination and improves privacy.
  - The computation error channel is destroyed as a by-product of our outflow prevention technique: $\widetilde{\mathcal{S}}$ cannot leak more than 1 bit. Therefore, $\mathcal{U}$ may skip the verification of $\mathcal{S}$'s response(s), thereby circumventing the need for an exponentiation that cannot be precomputed. This holds even if $\mathcal{S}$ provides more than one response to $\mathcal{U}$'s challenge: the independent randomization applied by $\mathcal{U}$ prevents $\widetilde{\mathcal{V}}$ from learning any information about how many or which of $\mathcal{U}$'s responses were formed using an incorrect response supplied by $\widetilde{\mathcal{S}}$.

– Owing to the minimal interaction required in our smartcard-enhanced protocols, $\widetilde{\mathcal{S}}$ can leak at most 1 bit using the halting channel, assuming $\mathcal{U}$ does not engage in a showing protocol execution until after it has obtained $a_{\mathcal{S}}$. This holds even if $\mathcal{U}$ interactively issues a signed proof or performs a 4-move zero-knowledge demonstration. No bits at all can be leaked when $\mathcal{U}$ non-interactively issues a signed proof to $\mathcal{V}$.

– To prevent the timing channel, $\mathcal{U}$ should time the delay between incoming and outgoing messages exchanged between the $\mathcal{S}$ and the other device. If the delay exceeds a reasonable time bound, $\mathcal{U}$ aborts the protocol execution; otherwise it first adds its own delay to ensure that the total delay time is approximately constant. What makes this approach truly practical for the DLREP-based protocols is that $\mathcal{S}$ and $\mathcal{V}$ need never perform any exponentiations that cannot be precomputed. (An upper bound on computation time will be much looser for time-consuming operations, and increases the delay that must be added.) As with the halting channel, the minimal interaction in our protocols ensures that at most 1 bit can leak.

It is easy to see that $\widetilde{\mathcal{S}}$ can leak at most 1 bit in total to $\widetilde{\mathcal{V}}$ via the halting channel, the computation error channel, and the timing channel. This is the best result that can be achieved, since $\mathcal{S}$ can always cause a protocol execution to abort by not providing its response(s) to $\mathcal{U}$. In practice the maximum leakage will be significantly less than 1 bit, since there will be other possible causes for incorrect responses to $\mathcal{V}$ or protocol abortion, such as incorrect behavior by $\mathcal{U}$ or a communication failure. In fact, any attempt by $\widetilde{\mathcal{S}}$ to leak bits through the halting channel or the computation error channel is futile, since it leaves $\widetilde{\mathcal{V}}$ unconvinced of the formula demonstrated.

Even common coin flips are prevented as a by-product. It is not hard to prove that if $\mathcal{U}$ follows the issuing and showing protocols, then the views of $\widetilde{\mathcal{S}}$ are statistically independent from the accepting views of the $\widetilde{\mathsf{CA}}$ and $\widetilde{\mathcal{V}}$, regardless of the formulae demonstrated by $\overline{\mathcal{U}}$. Thus, physical access to the contents of a smartcard does not help the $\widetilde{\mathsf{CA}}$ in cooperation with all verifiers to retroactively trace and link communications and transactions; all that can be learned is what certificate holders willingly disclose in executions of the showing protocol. More importantly, the card cannot leak information even if it could send out radio signals.

In general, however, it is impossible without breaking the tamper-resistance to verify that a smartcard does not have a sense of time, place, temperature, sound, or anything else that can be exploited by the CA once it gains physical access to the contents of the card. For instance, ultra-thin batteries that meet the required physical dimensions for smartcards are expected to become widely available within a few years. In light of this, a much more effective approach is for certificate holders to never return their smartcards to the CA (or anyone else) in the first place. This simple measure protects privacy in the strongest possible manner. In Section 6.5.1

we will show that our DLREP-based issuing and showing protocols can be efficiently implemented using low-cost smartcards without a cryptographic coprocessor. These cards might as well be destroyed, archived, or thrown away by their holders once their physical security has become outdated or their expiry date has been reached; there are no valid financial excuses for the CA to demand them returned.

An alternative is to enable cardholders to zeroize the contents of their smartcards before returning them. By feeding $\mathcal{S}$ a (pseudo)randomly generated file as large as $\mathcal{S}$'s memory, and requiring $\mathcal{S}$ to output the file, $\mathcal{U}$ can force $\mathcal{S}$ to overwrite its memory contents. $\mathcal{S}$ can be programmed to perform this action only if $\mathcal{U}$ also provides it with a MAC of the CA on a hash of the random file. Forensic experts may still be able to reconstruct the previous memory contents, though, and sting operations are not prevented (some cards may be given more memory than publicly specified).

We will now show that there are other persuasive reasons to prefer the model of not returning cards over the prevention of common coin flips.

6.4.4 Restricting the level of privacy protection

As we have seen in Section 6.2.2, $\mathcal{U}$'s ability to prevent $\widetilde{\mathcal{S}}$ and $\widetilde{\mathcal{V}}$ from developing common coin flips can be misused by an anonymous extortioner. When certificates represent significant value, it is in $\mathcal{U}$'s own interest that this level of privacy cannot be achieved. In case an extortioner strikes, this would enable $\mathcal{U}$ to find out how and where the extorted certificates have been misused, by comparing the common coin flips stored by the smartcard with the views of all the verifiers; in case the CA stores a copy of the relevant parts of all the showing protocol transcripts, this task is easy to accomplish with the cooperation of the CA. In an electronic coin system, for instance, this tracing ability strongly discourages the extortioner from making a large payment to his or her own account or to that of an accomplice; the extortioner will have little choice but to spend the money on goods or services that he or she can receive while remaining untraceable.

Even more effectively, by designing the showing protocol in such a manner that the verifier's challenge contains a verifier identifier that cannot be hidden from $\mathcal{S}$, the victim of an extortioner can determine in real time to which verifier(s) the extortioner is showing his or her certificates, since the identifiers must pass in unencrypted form through the user-controlled computer. This enables the victim to immediately notify the verifier that a fraud is going on, and so it will be very hard for the extortioner to profit from his or her crime. Verifier traceability by the certificate holder is desirable anyway, for many reasons: the certificate may need to obtain a digital receipt from the verifier in the showing protocol; the certificate holder may need the ability to complain about bad goods or services, or at least to warn others about the behavior of an unscrupulous service provider or to have an investigation instigated; and, absent verifier traceability it is impossible to recover (if needed) should the connection with the verifier become permanently lost during the showing protocol execution. It is

important to note that in this approach only the originator of the showing protocol execution has the power to trace the verifier. To prevent smartcards from locking their holders out of legitimate transactions to designated verifiers, smartcards should learn only a salted hash or an encryption of the verifier's identity.

In high-risk PKIs, this approach may be exploited to give law enforcement the ability to trace the actions of a designated cardholder. Requirements should be in place to ensure that a court order or search warrant must be presented to the designated cardholder, so that the cardholder can challenge the investigation. Although a certificate holder can falsely claim that his or her smartcard has been lost, broken, or stolen, this does not invalidate the approach. To most honest certificate applicants, the inconvenience of having to obtain a new smartcard to retrieve new certificates is sufficient reason to refrain from false claims. Furthermore, the police can resort to traditional investigative techniques to locate a card claimed to have been lost, and to forensic methods to retrieve the memory contents of a smartcard that has been wrecked. Any audit logs by the user-controlled computer could be examined as well. Also, if the CA requires timely reporting of the loss or theft of smartcards, it would be suspect if an individual claims to have lost control over his or her smartcard right after a court order was presented.

It may be cumbersome for law enforcement to make a physical presence to gain access to the contents of a device. Moreover, physical access takes away the device holder's control over which parts of the transaction log are being inspected. It is possible to give authorized third parties the ability to remotely (over a network) query the contents of the smartcards of designated certificate holders. When presented with a digitally signed electronic search warrant (or court order) demanding that a part of the transaction log be disclosed (e.g., all transactions above a specified threshold, all the transactions to a designated organizations, or the ten most recent transactions), the user's computer checks the warrant's authenticity and contents, passes the query on to the smartcard, receives the requested information from the smartcard (authenticated using a MAC or a digital signature), checks the supplied data against its own copy of the transaction log, and passes it onward to the requesting party only if the verification holds.[9]

More generally, there may be a legitimate need for $\mathcal{S}$ to know information other than (just) the verifier's identity, to decide whether or not to assist $\mathcal{U}$ in performing the showing protocol execution. For instance, $\overline{\mathcal{S}}$ may need to know (part of) the formula demonstrated, (part of) the message it is helping to sign, or the type of verifier to which a signed proof is to be issued. To accommodate these and other legitimate needs for restricting the level of privacy, we reject the goal of preventing common coin flips and instead open up (in a controlled manner) one communication channel from $\mathcal{V}$ to $\mathcal{S}$: $\mathcal{V}$'s challenge message, c. To prevent $\widehat{\mathcal{U}}$ from altering c before passing it on to $\mathcal{S}$, we have $\mathcal{S}$ form by itself the challenge message it will respond to, by

[9] MAC outflow can be prevented by using commitments, while randomization or deterministic signature generation suffice for digital signatures.

applying a sufficiently strong (publicly known) one-way hash function to the data provided by $\mathcal{U}$. For example, if $\mathcal{U}$ is to issue a signed proof to $\mathcal{V}$, the protocol ($\mathcal{S}$, $\mathcal{U}$) should be such that $\mathcal{S}$ computes c by applying $\mathcal{H}_i(\cdot)$ to data provided by $\mathcal{U}$. This forces $\mathcal{U}$ to provide (h, a, F, m) and any other data to be included in the hash.

To guarantee that $\mathcal{S}$ learns only the minimum information needed, we use *challenge semantics* based on hash structures. If, for example, $\mathcal{S}$ should be able to learn only the message m in a signed proof, $\mathcal{V}$'s challenge message could be set equal to $\mathcal{H}_i(\mathcal{F}_i(h, a, F), m)$, where $\mathcal{F}(\cdot)$ and $\mathcal{H}(\cdot)$ may be the same hash function. When $\mathcal{S}$ provides responses only to challenges c formed by itself by applying $\mathcal{H}_i(\cdot)$ to data provided by $\mathcal{U}$, the latter is forced to provide at least $\mathcal{F}_i(h, a, F)$ and m to $\mathcal{S}$. To improve privacy, $\mathcal{U}$ may hash along a random salt when applying $\mathcal{F}_i(\cdot)$. (Formally, $\mathcal{F}(\cdot)$ should be a commitment function that unconditionally hides the data; a DLREP function can be used.) More generally, c can be defined as an arbitrary structure of nested hashes of data fields, and the role of $\mathcal{S}$ in forming c can be arbitrarily defined as well. In case of a dispute, the hash structure can be opened selectively.

Note that $\widehat{\mathcal{U}}$ cannot block a message transfer from $\mathcal{V}$ to $\mathcal{S}$ without blocking the entire showing protocol execution. On the other hand, $\mathcal{U}$ can unconditionally prevent inflow and outflow, and can at all times see and control the common coin flips developed by $\mathcal{V}$ and $\mathcal{S}$.

Another benefit of rejecting the goal of preventing coin flips is related to attacks on $\mathcal{U}$. If $\mathcal{S}$ sees the message it is helping to sign and shows this on a display of its own, and waits for its holder to accord its next step through an on-board key pad, then a virus or a Trojan horse attack on $\mathcal{U}$ cannot trick $\mathcal{S}$ into signing a false message.

This technique of combining inflow and outflow protection with "controlled common coin flips" suffices to accommodate most legitimate needs to reduce the level of privacy attainable. It does not, however, enable the verifier in the showing protocol to distinguish whether or not the prover's demonstration is conducted using the assistance of a smartcard. This capability may be needed in case of high-risk limited-show certificates for which the techniques in Section 5.5 to trace and contain fraud offer inadequate security. One way to enable this capability is for the CA to encode into each certified key pair a "policy attribute" that indicates at least whether or not the verifier should resort to online certificate authorization. This policy attribute must always be disclosed when performing a demonstration. (Any other data that must always be disclosed, such as a certificate expiration date, can be encoded into the same attribute.)

6.5 Other techniques

In this section we describe how to implement our DLREP-based techniques using low-cost smartcards. We also design a protocol that enables certificate holders to return to the CA any retrieved certificates that have not yet been shown. Furthermore,

we show how to discourage certificate holders from using their certificates to help remote parties gain access to services for which they do not hold the proper certificates themselves. Finally, we design secure bearer certificates with optimal privacy and address some loose ends.

6.5.1 Implementation in low-cost smartcards

The cheapest and most widely available smartcards contain a simple 8-bit microprocessor that can perform a DES operation in software in about 4 milliseconds. A single modular multiplication as needed in public-key cryptographic operations takes many minutes, and may not even fit in memory. (RAM is typically between 124 and 1024 bytes, EEPROM between 1 and 16 kilobytes, and ROM between 4 and 16 kilobytes.) Even can today's fastest coprocessor smartcards cannot compute more than 3 full RSA exponentiations per second, assuming a 1024-bit modulus and no use of the Chinese Remainder Theorem.[10]

Moreover, cryptographic coprocessors decrease reliability, are more vulnerable to timing attacks (see Lenoir [245]), take up precious space, and add about three to five dollars to the cost of each smartcard.

When using our DLREP-based techniques, two noteworthy optimizations are available:

- An elliptic curve implementation over a field of the form GF_{2^m}, with $m \approx 160$, guarantees that a conventional 8-bit smartcard microprocessor can rapidly perform a high-strength public-key exponentiation in G_q. For instance, the Schlumberger Multiflex smartcard used in the first smartcard-based pilot of SET [257] can compute an elliptic-curve DSA signature in exactly one second, using a 163-bit modulus; this chip contains an 8-bit Motorola chip with 240 bytes of RAM, 8 kilobytes of EEPROM, and 12 kilobytes of ROM. Since the Schnorr proof of knowledge requires virtually the same operations as the DSA scheme, the same performance in a low-cost smartcard can be achieved when using our techniques. For details on implementing elliptic curves in smartcards, see Certicom [85].

- The bulk of the workload of $\mathcal{S}$ can be shifted to the CA, by having the latter instead of the former provide $a_{\mathcal{S}}$ to $\mathcal{U}$; this removes the need for $\mathcal{S}$ to perform any exponentiations. The CA can provide $a_{\mathcal{S}}$ during the execution of the issuing protocol in which it issues the certified key pair that uses $a_{\mathcal{S}}$. Alternatively, the CA provides a great many $a_{\mathcal{S}}$-values at once to $\mathcal{U}$, possibly whenever the latter makes a request. The only remaining task for $\mathcal{S}$ in an execution of the showing protocol is to compute a linear relation over $\mathbb{Z}_q$ and perhaps a few

[10] In November 1999, Bull and other major hardware manufacturers demonstrated a smartcard that can perform the exponentiation in less than 350 milli-seconds. Their design features a RISC 32-bit processor, 64 kilobytes of EEPROM, 96 kilobytes of ROM, and 4 kilobytes of RAM.

hashes. Even the cheapest of 8-bit smartcards can compute dozens of such responses within a second. Note that this optimization does not require the use of elliptic curves.

To implement the latter optimization, the CA and $\mathcal{S}$ must share the list of random numbers that $\mathcal{S}$ will deploy. In practice these numbers may be generated in a pseudorandom fashion from a seed value that is known to both $\mathcal{S}$ and the CA, which the CA may form by diversifying a smartcard identifier using a master key. (The use of pseudorandom numbers is recommendable also to avoid physical attacks on noise generators.) The preferred method of generating pseudorandom numbers depends on certain characteristics of the certificates:

- In case certificates are shown in the order in which they are retrieved, $\mathcal{S}$ can regenerate in constant time and space its contributions to the secret keys of up to 2^t certified key pairs by storing merely a t-bit index for the pseudorandom number generator, and incrementing this upon each successful showing of a certificate. The DSS [277, Appendix 3] specifies two algorithms for generating pseudorandom numbers in $\mathbb{Z}_q$ for use with the DSA. Both algorithms feed a secret seed value and the previous pseudorandom number into a one-way function in order to derive the next pseudorandom number. Since $\mathcal{U}$ in a practical implementation need merely store some 40 bytes per certified key pair, regardless of the number of attributes encoded into it, the user's computer and the smartcard can retrieve and store a number of certified key pairs that is, for all practical purposes, virtually unlimited.

- In PKIs where the showing order of limited-show certificates may be unrelated to the certificate retrieval order, it is preferable to generate each new pseudorandom number by feeding the seed value and an increment of an index value to a sufficiently strong one-way hash function, to facilitate random addressing. $\mathcal{S}$ must keep track of all the indices for which it has already provided its assistance, because it may not respond to two different challenges using the same initial witness. The indices may be known to $\mathcal{U}$, and form a convenient manner for $\mathcal{U}$ to query $\mathcal{S}$ in the showing protocol: the CA in the issuing protocol can provide $\mathcal{U}$ with the indices used to generate the pseudorandom contributions of $\mathcal{S}$, and when $\mathcal{U}$ needs to show a certificate it can send $\mathcal{S}$ the corresponding index in order to have $\mathcal{S}$ regenerate its random contribution to the secret key of the appropriate certified key pair.

Having the CA instead of $\mathcal{S}$ provide $a_{\mathcal{S}}$ to $\mathcal{U}$ also allows $\mathcal{U}$'s holder to leave $\mathcal{S}$ at a safe place during certificate retrieval, for improved protection against theft. Another advantage is that the same smartcard can be used for different PKIs, in fact even for applications not known at the time of card issuance. Hereto each CA should use the same q and preferably a different g_1, and the pseudorandom numbers of

each smartcard should be generated by feeding an additional CA identifier into the pseudorandom generator.

Card performance can be pushed to the limit by either speeding up or circumventing the memory-intensive reduction modulo q that must be performed by $\mathcal{S}$. To achieve the former, one should use a modulus q of the form $2^t \pm B$, for small B. To circumvent the modular reduction altogether, a technique due to Shamir [347] can be applied. The idea is to replace the computation of $cx_1 + w_1 \bmod q$ by the computation of $cx_1 + w_1 + tq$, for t chosen at random from a suitable range; the result can be computed and output one byte after another by using a double convolution process.

6.5.2 Returning certificates

According to Proposition 6.3.1, the loss or theft of a smartcard prevents its holder from showing his or her certificates, in spite of any access control mechanism that may prevent others from operating the smartcard. In many PKIs it is desirable that any previously retrieved certificates that have not yet been shown can be returned to CA. The existence of a certificate return protocol would give the victims of lost, stolen, or crashed smartcards the opportunity to recover value that might be associated with their certificates. It also enables the CA to publish the certificates on an update of its CRL, and in this manner prevents attackers able to get around the smartcard access control mechanism from using the certified key pairs.

A return protocol may also be useful in the unlikely event that the cryptography is broken (see Section 5.5.5); since the CA can keep a positive list at certificate issuing time, even an adversary with unlimited computing power cannot return forged certificates. (In this particular case, it is recommended that each certificate applicant initially establish a secret key or a one-time pad with the CA, for the purpose of symmetric encryption and authentication of data sent in the return protocol.)

Furthermore, a return protocol is useful to resolve permanently lost connections between a certificate holder and a verifier; if the transcripts of all the showing protocol executions are deposited with the CA, fault tolerance can be implemented by returning the certificate to the CA so that the latter can find out whether the verifier has received the certificate.

Other uses of a certificate return protocol include: to exchange expired certificates for fresh ones; to cancel a transaction that has already been performed; and, to have recourse to the CA if a smartcard enters into suspension mode or produces an error.

Returning a certificate to the CA is easy, assuming certificate holders keep backup copies of their certified public keys (and, for secret-key certificates, their share of the corresponding secret keys). Backups can be stored conveniently on the user-controlled computer, on a floppy disc, in encrypted form in a cyberspace "strongbox," or in any other manner deemed secure by the certificate holder.

To perform the actual return of a certificate, $\mathcal{U}$ transmits the certified public key to the CA. In case of a secret-key certificate, $\mathcal{U}$ must also provide the CA with sufficient

information to enable it to determine $\mathcal{S}$'s secret key, and must prove knowledge of its share of the secret key of the certified key pair. To return a certificate obtained using DLREP-based scheme I, for instance, $\mathcal{U}$ sends $h', (c'_0, r'_0)$ to the CA and proves knowledge of a DL-representation of $h'/h_{\mathcal{S}}$. In the proof of knowledge, $\mathcal{U}$ may selectively disclose properties about the encoded attributes, as in an execution of the showing protocol.

Note that this return protocol is software-only: it does not involve the cooperation of the smartcard. This is desirable not only to cope with loss, theft, and destruction of smartcards, but also for certificate holders who have never been issued a smartcard at all.

It is desirable, both for secret-key and for public-key certificates, that $\mathcal{U}$ always provide the CA with a signed proof on a message stating the reason it is returning a certificate. If the CA stores this signed message in a database, any disputes that may arise later on can be resolved.

In case $\mathcal{U}$ is to be reimbursed in some way or another, and executions of the showing protocol need not be authorized online, the CA may prefer to delay reimbursement until it is certain that the returned certificate has not been shown.

Care must be taken that a thief cannot return the certificate data stored on stolen backups for his or her own benefit. In an off-line electronic coin system, for instance, the CA should best redeem an account holder for a returned coin by crediting his or her account.

6.5.3 How to discourage remote lending

In Section 5.5.2 we presented software-only measures to discourage lending of certified key pairs. For stronger protection, certificate holders should be required to use smartcards with a biometric access control mechanism. In a face-to-face situation the verifier can visually inspect that the biometric data is properly entered. In general, a liveness detection mechanism is needed to prevent unauthorized cardholders from using biometric templates; a cardholder's fingerprints, for instance, may be all over his or her smartcard. (The same measure also protects against loss and theft.)

This leaves open the possibility of *remote lending*. Here, the holder of a certified key pair provides the "borrower" with the certified public key, and assists in the showing protocol execution by providing the required responses. The lender can provide his or her assistance over a radio link or a network such as the Internet. Remote lending is similar to the anonymous extortion attack described in Section 6.2.2, with the difference that the borrower voluntarily cooperates with the lender and may not be interested in hiding his or her identity and transaction details from the lender. (Remote lending differs also from the "mafia fraud" studied first by Desmedt [134]; in the latter, the prover is not aware that the protocol execution is being relayed.)

In communications and transactions that are not face-to-face, remote lending cannot be prevented, regardless of whether privacy-protecting certificates or fully trace-

able identity certificates are used. Indeed, the "lender" might as well perform the entire showing protocol execution and simply relay the provided service or goods to the "borrower." In many PKIs this is not considered a security problem, especially not if lending occurs only incidentally; it may even be a feature. In PKIs where large-scale lending is to be discouraged, the CA could issue limited-show certificates and establish one account per certificate applicant (see Section 5.2.1). Large-scale lenders will then be exposed because of their abnormal demand for certificates. Another measure is to charge a fee per certificate issued, so that a large-scale lender faces the problem of being compensated for the cost of his or her service without risking exposure.[11] A further measure to discourage large-scale lending is to program each smartcard in such a manner that it will only respond to a challenge when its biometric access control mechanism detects the live presence of the cardholder.

In face-to-face situations, additional measures are available. The CA could encode biometric characteristics and apply the measure described in Section 5.5.2. Even better, remote lending can be prevented altogether by having $\mathcal{V}$ bound its distance to the true certificate holder. The idea is for $\mathcal{S}$ and $\mathcal{V}$ in the showing protocol to rapidly exchange a series of random bits. Each bit of $\mathcal{S}$ is to be sent out immediately after receiving a bit from $\mathcal{V}$. To ensure that $\mathcal{S}$ cannot simply send out its i-th bit before receiving the i-th bit of $\mathcal{V}$, the i-th bit of $\mathcal{S}$ must depend on the i-th bit of $\mathcal{V}$. One way to accomplish this is to require $\mathcal{S}$ to send bits chosen in such a manner that the exclusive-or of these bits and those provided by $\mathcal{V}$ equals a previously established binary string. After the rapid bit exchange has taken place, $\mathcal{S}$'s binary string must be authenticated using the secret key of the certified key pair; one way to accomplish this is to hash along the binary string when forming $\mathcal{V}$'s challenge. $\mathcal{V}$ accepts if and only if the authentication is correct and its distance to the smartcard is sufficiently small. To compute an upper bound on the distance, $\mathcal{V}$ takes the maximum of the delay times between sending out its i-th bit and receiving the i-th bit of $\mathcal{S}$; this ensures that the cheating probability decreases exponentially fast in the number of bits exchanged. What makes this approach practical is that today's electronics can easily handle timings of a few nanoseconds, and light can travel only about 30cm during one nanosecond. Even the timing between two consecutive periods of a 50 MgHz clock allows light to travel only three meters and back. It is easy to integrate this *distance bounding technique* with the techniques described in Sections 6.3 and 6.4.

The smartcard techniques in this section may be used in addition to the software-only techniques in Section 5.5.2, rather than serving as a replacement.

6.5.4 Bearer certificates

As we have seen in Section 5.5.1, strong fraud prevention for software-only bearer certificates requires online clearing. The closest we can come to realizing the digital

[11] A business model relying on certificate prepayment has other advantages as well in many circumstances, and is naturally based on the issuance of limited-show certificates.

equivalent of paper-based bearer certificates is to implement digital certificates using smartcards, to ensure that each certificate can be at only one place at any moment in time.

Consider the following design of a digital bearer certificate system. The CA endows each smartcard with a unique card identifier and a secret key (possibly derived by diversifying the card identifier using a master secret key). Anyone can purchase one or more of these smartcards via a myriad of retail outlets, without needing to identify. Each smartcard can be used to open an anonymous account with the CA, by communicating with the CA in cyberspace. To enable the CA to associate the secret key of a smartcard with the account opened, the user's computer sends the card identifier to the CA, possibly together with a MAC of the smartcard on a challenge message of the CA. The CA issues only limited-show certificates, and encodes into each certified key pair the secret key of the applicant's smartcard. Retrieval and showing of digital certificates take place using the techniques described in Sections 6.3 and 6.4. The additional option available to smartcard holders is to sell, lend, or otherwise transfer their cards to others. For protection against theft, smartcards should have an access control mechanism (based on a PIN, a password, or a biometric) that can be changed only when authorized by the current holder. Verifiers may accept digital certificates without needing online authorization by the CA, but are required to deposit transcripts of their showing protocol executions. If a smartcard holder manages to physically extract the card's secret key and show certificates more times than allowed, the CA can compute the smartcard identifier, close the corresponding account, and blacklist the certificates using its CRL. This ensures that the perpetrator can continue only by physically compromising another smartcard.

This design offers adequate security assuming that the average cost of physically extracting the secret key from a smartcard exceeds the expected profit that can be made until the CRL update has been distributed and the account closed. At the same time, the privacy of users is maximally protected: their showing protocol executions are untraceable and unlinkable, and their certificate retrievals anonymous.

In PKIs with high-risk certificates, it is preferable that the perpetrators themselves can be traced, to stop them from opening new accounts and perhaps also to recover some of the losses. Hereto our smartcard techniques should be combined with the personalization techniques described in Section 5.5. This does not preclude the possibility of anonymous withdrawal of certificates, as we have seen.

6.5.5 Loose ends

To prevent organizations and other verifiers from discriminating against certificate holders who do not disclose their built-in identifier, the secret key of a smartcard should serve as the cardholder's identifier. The link between the cardholder's identity and the smartcard's identifier can be made either when the card is issued or when the account is opened. With this approach, the privacy interests of the cardholder and the

security interests of the CA are perfectly aligned; identity bartering is out, because cardholders cannot disclose the identifiers that have been encoded into their certified key pairs.

In some situations it may be desirable that certificate holders can demonstrate not to be the originator of a transaction. The technique described in Section 5.5.1 to accomplish this is not in conflict with that of the previous paragraph. Namely, it is possible for $<\mathcal{S},\mathcal{U}>$ to demonstrate that the secret key of $\mathcal{S}$ is not equal to the secret key of one or more perpetrator smartcards. Consider by way of example the setting of Proposition 6.3.2. With $h := \prod_{i=1}^{l} g_i^{x_i}$, the goal is for $\mathcal{U}$ (with the assistance of $\mathcal{S}$) to demonstrate that $x_1 \neq y \bmod q$, for some $y \in \mathbb{Z}_q$, without $\widehat{\mathcal{U}}$ learning the secret key x_1 of $\mathcal{S}$. Hereto $\mathcal{S}$ demonstrates to $\mathcal{U}$ that $x_1 \neq y \bmod q$ by means of our standard interactive showing protocol for atomic formulae with a "NOT" connective: $\mathcal{S}$ sends $a_{\mathcal{S}} := (g_1^y/h_{\mathcal{S}})^{w_1}$ to $\mathcal{U}$, receives $\mathcal{V}$'s challenge c from $\mathcal{U}$, and responds by sending $r := c\delta + w_1 \bmod q$, where $\delta := 1/(y - x_1) \bmod q$. It is not hard to see that $\mathcal{U}$ can perform its part of the protocol with respect to $\mathcal{V}$ if $\mathcal{U}$ would know $g_2^{w_1}, \ldots, g_l^{w_1}$. $\mathcal{U}$ would then form

$$a := a_{\mathcal{S}}(g_1^y/h_{\mathcal{S}})^{\gamma_1} \prod_{i=2}^{l} \Big((g_i^{w_1})^{w_i} g_i^{\gamma_i} \Big)$$

for random $w_2, \ldots, w_l \in \mathbb{Z}_q$ and random $\gamma_1, \ldots, \gamma_l \in \mathbb{Z}_q$, and compute $r_1 := r + \gamma_1 \bmod q$ and $r_i := rw_i + \gamma_i \bmod q$, for all $i \in \{2, \ldots, l\}$. (If desired, $\mathcal{U}$ can be allowed to blind c before passing it on, as detailed in Section 6.4.2.) Having $\mathcal{S}$ compute and provide $g_2^{w_1}, \ldots, g_l^{w_1}$ to $\mathcal{U}$ would result in serious overhead to $\mathcal{S}$, but as with the technique in Section 6.5.1 this burden can be moved to the CA. Alternatively, assuming that demonstrating a property of $\mathcal{S}$'s secret key is an exceptional task, $\mathcal{S}$ could store a precomputed tuple $(w_1, g_1^{w_1}, \ldots, g_l^{w_1})$ into its memory. Knowledge of $g_2^{w_1}, \ldots, g_l^{w_1}$ is believed to be of no help in attacking the protocol performed by $\mathcal{S}$; these numbers are essentially Diffie-Hellman keys.

The technique described in Section 5.5.3 can be applied straightforwardly to achieve non-repudiation. For example, I can be formed as the sum of the secret key of $\mathcal{S}$ and a random secret of $\mathcal{U}$. Alternatively, non-repudiation can be based on the tamper-resistance of the smartcard: if smartcards are tamper-evident, then a smartcard holder can prove his or her innocence by showing that the smartcard has not been tampered with in an attempt to physically extract the secret key.

6.6 Bibliographic notes

The smartcard techniques in Sections 6.3 and 6.4 originate from Brands [54]. The presentation fills in some details that were previously only hinted at, and the formal statements of security and their proofs appear here for the first time. Prior to Brands [54], Brands [46, 48] applied a special case of the smartcard techniques

for the purpose of designing a secure off-line electronic coin system; coin payment requires the demonstration of the formula TRUE, and the static one-show blinding technique ensures that the bank can trace a doublespender who has defeated the tamper-resistance of his or her smartcard. In a variation [46, 58], the bank encodes an additional attribute into each coin to specify its denomination and expiry date.

The method in Section 6.5.1 for avoiding exponentiation by the smartcard is due to Brands [57]. Naccache, M'Raïhi, Vaudenay and Raphaeli [271, Section 6] proposed a similar method in a different context ("use-and-throw coupons" for smartcards).

The return protocol in Section 6.5.2 is similar to a protocol designed by Pfitzmann and Waidner [304] for the purpose of loss-tolerance in the electronic coin system of Brands [48]. New is the observation that certificate return is useful for many other purposes.

The distance bounding technique in Section 6.5.3 originates from Chaum [97], with improvements by Brands and Chaum [59].

Finally, the technique in Section 6.5.4 is based on Brands [54].