Provably Secure Chipcard Personalization or How To Fool Malicious Insiders

Helena Handschuh[1], David Naccache[1],
Pascal Paillier[1], Christophe Tymen[1,2]

[1] Gemplus Card International, Cryptography Group
34 rue Guynemer, 92447 Issy-les-Moulineaux, France
{helena.handschuh, david.naccache, pascal.paillier}@gemplus.com
[2] École Normale Supérieure, 45 rue d'Ulm, 75230 Paris, France
christophe.tymen@gemplus.com

http://www.gemplus.com/smart/

Abstract. We present 'malicious insider attacks' on chip-card personalization processes and suggest an improved way to securely generate secret-keys shared between an issuer and the user's smart card. Our procedure which results in a situation where even the card manufacturer producing the card cannot determine the value of the secret-keys that he personalizes into the card, uses public key techniques to provide integrity and privacy of the generated keys with respect to the complete initialisation chain. Our solution, which provides a non-interactive alternative to authenticated key agreement protocols, achieves provable security in the random oracle model under standard complexity assumptions. Our mechanism also features a certain genericity and, when coupled to a cryptosystem with fast encryption like RSA, allows low-cost intrusion-secure secret key generation.

1 Introduction

Tamper-resistant devices like smart-cards are used to store and process secret and personal data. Examples of applications making extensive use of smart cards include wireless communication systems such as the Global System for Mobile communications (GSM), or banking systems using the EMV (Europay, Mastercard and VISA) standard. These applications share the fact that they use secret key identification or authentication to achieve security and enable access to services. Thus some unique secret key K_I (we will adopt the notation K_I to denote a card's secret key by analogy with the widely known GSM terminology) needs to be shared between the issuer (the bank or the telecom operator) and the smart card. Usually this secret key material is downloaded into the card during the so-called chip personalization phase, i.e. the initialisation phase during which identical cards are configured in such a way that each and every of them corresponds to one specific user.

Usually, the card personalization center either writes secret keys into the cards according to a list provided by the issuer, or generates the keys itself and downloads them into the cards within it's own premises, and subsequently transmits a list of (encrypted) keys to the issuer. We refer to these scenarios as *typical* personalization protocols. In the sequel, we consider precisely the second scenario (key generation within the manufacturer's premises) and show that such a basic personalization procedure is vulnerable to *malicious insiders.*

We first discuss the potential security flaws in such a process, and then proceed to present a new personalization protocol in which the manufacturer is able to *provide evidence* to the issuer that no one except the issuer himself knows the secrets stored inside the cards.

Thus our new technique provides generally trusted keys for secret key applications.

The rest of the paper is organized as follows. In Section 2, we give an overview of a typical personalization protocol, and we point out its vulnerability to insider attacks when appropriate physical site protection measures are not enforced. Section 3 proposes our new personalization procedure. We provide a thorough security analysis in section 4 and conclude by a giving practical implementations of our technique in section 5.

2 Personalization Protocols

2.1 The current approach: typical protocols

Card personalization involves three parties: an issuer (telecommunications operator or bank), a card manufacturer (who actually personalizes smart cards for the issuer), and a smart card. Beyond graphical personalization – which may consist in printing the issuer's logo on the card for instance, the manufacturer has to electrically initiate the card and among such tasks, initialize the files meant to contain the card's secret key material K_I. In a typical scenario, each and every secret key K_I is generated uniformly at random by a personalization computer (PC) connected to the personalization system (such as a DataCard 9000 machine). Whenever a card enters the system, a fresh random key K_I is selected by the PC and downloaded into the card's non-volatile memory.

Simultaneously, the key gets encrypted on the PC, together with the card identifier Id (which might be some publicly available unique bitstring such as a serial number for instance) using the issuer's secret key K_s. Lists of encrypted (K_I, Id) pairs are then sent over an insecure channel to the issuer who decrypts the received files and recovers the pairs (K_I, Id).

Another way to proceed consists in encrypting the generated keys with the issuer's authenticated public key in an asymmetric key setting. This way, the issuer is the only entity able to decrypt the generated files, and the key K_s need not be known at the manufacturer's premises.

However, both solutions are vulnerable to insider attacks where a malicious entity having access to the manufacturer's premises would get hold of the key. We may think of a malicious insider as some malevolent employee willing to clone SIM cards or as a hacker that discretely eavesdrops the computer network from outside the personalization center. This strongly motivates the search for protocols featuring a guaranteed level of security against this kind of threat.

2.2 Security Notions for Card Personalization

Let us examine the setting and determine which security goals are desirable to reach from the issuer's standpoint. When the personalization protocol takes place, parties are

- a tamper-resistant secret-less chip-card to be personalized with identifier Id,
- an issuer (supposedly remote),
- a personalization system (PC + DC9000) in which the issuer has no reason to put trust.

Ultimately, the goals the personalization protocol is meant to achieve are the following. At the end of the process,

1. the card must contain some secret key K_I belonging to some fixed key space (we call this property *correctness*),
2. the issuer must know the correct pair (Id, K_I) (we refer to this property as *key integrity*),
3. the issuer should be confident that he is the only entity who shares the knowledge of the (Id, K_I) pair with the card (this is defined as *key privacy*).

Correctness is easily achieved. The question is whether requirements 2 and 3 are actually achieved by current typical personalization protocols, and the answer is obviously no. The above protocols do not meet key integrity nor even key privacy. Indeed, the computer, if handled by a malicious person, may very well generate a given K_I and transmit a different one to the issuer. This can be considered a denial of service attack, as the end-user would get a non-functional card. Alternatively, the computer might respect the integrity property by providing the right pair (Id, K_I) to the issuer, but reveal this pair to an intruder getting hold of the PC. In this case, card cloning becomes possible. We call such attacks 'malicious insider attacks'.

2.3 The Interlock protocol

One obvious attempt to address this problem consists in executing a key agreement protocol such as *Interlock* [4] between the card and the issuer.

Interlock is described as follows. Assuming that two entities A and B, with public-keys pk_A and pk_B, want to exchange a secret through an insecure channel, A and B proceed as follows. First, A and B exchange their public keys through the channel. Then, A (resp. B) chooses a random r_A (resp. r_B), and encrypts it with pk_B (resp. pk_A) to obtain a ciphertext c_A (resp. c_B). c_A (resp. c_B) is a bitstring which can be cut into two equal parts (c_A^1, c_A^2) (resp. (c_B^1, c_B^2)). Thus, A sends c_A^1 to B, and sends the remaining part c_A^2 only after having received c_B^1. Finally, B sends c_B^2. At the end of the sequence, A and B share the pair (r_A, r_B).

Clearly, this protocol thwarts passive man-in-the-middle attacks. However, it is interactive, which represents an unacceptable hurdle in the context of a personalization process. The only way to achieve an equivalent non-interactive protocol would be to use public-key certificates and signature verification which calls for far too complex (and heavy) public-key infrastructures.

Besides, security requirements explicitly demand resistance against active attacks, where the attacker may not only eavesdrop exchanged pieces of information but also modify them in some way, and may impersonate parties as well. Because it does not provide authentication, Interlock does not resist active attacks.

The contribution of this paper consists in providing a non-interactive alternative to the Interlock protocol which, in our context, resists active attacks and needs no certificates or signatures whatsoever.

3 A Provably Secure Card Personalization Protocol

Let us go back to the typical scenario. Obviously, the security breach resides in the possibility to attack the PC. Thus each and every secret should be generated *inside the card itself*, which, by assumption, provides the advantage of being tamper-resistant over an open PC.

3.1 A First approach

Thus a first idea is to generate the secret key K_I inside the card, download the issuer's public key into the card, encrypt the generated secret under the public key and output the result. Next, the encrypted secrets are collected along with the Id's in a file and sent to the issuer who decrypts the list with his private key SK and recovers the associated pairs in clear (alternatively the Id's could also be encrypted together with the secret K_I inside the card). This protocol is shown in figure 1. The public key of the issuer is noted PK; typically, one could use stand-alone RSA public key encryption [5] for instance. We suppose the key pair $(\mathsf{PK}, \mathsf{SK})$ is generated once and for all by the issuer himself, and then transmitted to the personalization center which uses it for a certain period of time.

Unfortunately, this solution is vulnerable to the well-known man-in-the-middle attack. Suppose the attacker controls the PC again. She is then able to generate her own public

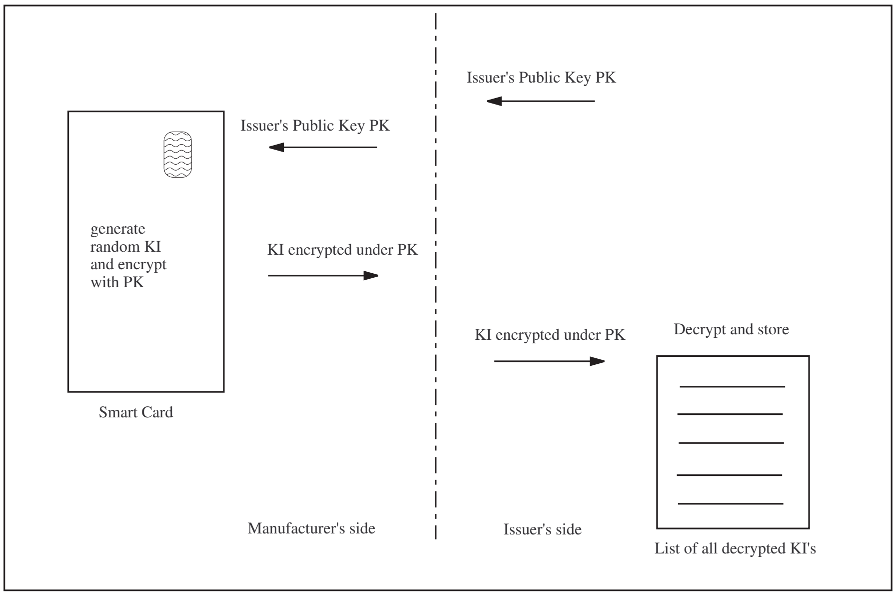

Fig. 1. Secure personalization protocol : first approach

and private RSA key pair and to fool the card by sending to it her own public key. She recovers the encrypted K_I values, decrypts them, and re-encrypts them with the issuer's public key. Thus key integrity is preserved, but key privacy is violated. The attack is shown in figure 2 where the attacker's public key is noted PK'.

3.2 Proposed Protocol

Let us now proceed to describe our protocol. The security analysis will be discussed in the next section. Basically, the personalization process now includes the following steps :

1. the PC transmits the issuer's public key PK to the card,
2. the card generates a random r, computes $K_I = H(r, \mathsf{PK})$ where H is a hash function such as SHA-1 [7], and memorizes K_I in non volatile memory,
3. the card encrypts r as $c = \mathcal{E}_{\mathsf{PK}}(r)$ where $\mathcal{E}_{\mathsf{PK}}$ denotes public key encryption under PK, and outputs c,
4. the PC collects the pair (Id, c) and sends it to the issuer who later decrypts c using SK, recovers $r = \mathcal{D}_{\mathsf{SK}}(c)$ and computes $K_I = H(r, \mathsf{PK})$.

This protocol is shown in figure 3.

4 Security Analysis

4.1 Main Results

Although looking simple, our protocol achieves a very satisfactory security property, namely that

– both key integrity and key privacy are preserved under a passive attack,
– if key privacy is not preserved under an active attack then key integrity cannot be preserved either.

The proof of that fact is given below. From a practical viewpoint, this means that if an intruder simply eavesdrops what is transmitted through the PC, our protocol fully reaches the security goals of section 2.2, namely key integrity and privacy. Additionally, if the intruder actively operates changes over transmitted data, she is given no other choice than

– either knowing the key K_I generated by the card; but then the issuer recovers

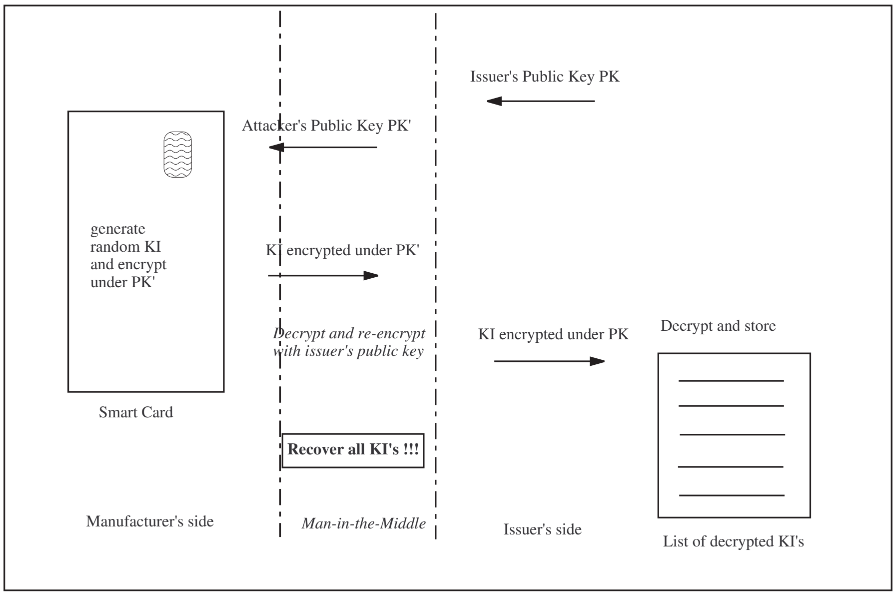

Fig. 2. Man-in-the-middle attack on key generation process

nothing else than a faulty key $K_I' \neq K_I$. Subsequently, the card just cannot work properly because user authentication will be unsuccessful each time the end user attempts to access the issuer's service. The issuer may then recognize the card as a fake or abnormal one and blacklist it.

– or letting the card generate K_I properly and later have normal access to the issuer's service; but then, no information whatsoever can be obtained on K_I.

In other words, our protocol prevents insiders from cloning normal cards since only useless cards are exposed to key divulgation. Trying to gain information on the card's key simply forbids its future use in normal conditions. We guarantee this under any type of attacks, be they very sophisticated. The insider is left only with malevolence i.e. the ability to force the personalization of useless cards. We argue that this scenario is not of interest to an active adversary. We assess these results without considering collusions in the first place, and address these further in section 4.5.

4.2 Security Proof Against Passive Insiders

We state, in a somewhat more formal way:

Theorem 1 (Passive Attacks). *Assume that the encryption scheme $\mathcal{E}_{\mathsf{PK}}$ is deterministic and one-way under chosen plaintext attacks (OW-CPA). Then no polynomial time attacker given* PK *and* $c = \mathcal{E}_{\mathsf{PK}}(r)$ *can recover* $K_I = H(r, \mathsf{PK})$ *with non-negligible probability in the random oracle model.*

Proof. We assume the existence of an attacker $\mathcal{A}$ with success probability ϵ and show how to invert $\mathcal{E}_{\mathsf{PK}}$ with probability ϵ'. We build a reduction algorithm $\mathcal{B}$ as follows. $\mathcal{B}$ is given an instance $\widetilde{c} = \mathcal{E}_{\mathsf{PK}}(\widetilde{r})$ and must return $\widetilde{r}$ with non-negligible probability. $\mathcal{B}$ randomly selects $\widetilde{K_I}$ and runs $\mathcal{A}(\mathsf{PK}, \widetilde{c})$.

Now, each time $\mathcal{A}$ queries the random oracle H for an input (r, pk), $\mathcal{B}$ checks in the history of queries if (r, pk) was queried by $\mathcal{A}$ in the past, in which case the same answer is returned to $\mathcal{A}$. Otherwise, if $pk = \mathsf{PK}$ and $\mathcal{E}_{\mathsf{PK}}(r) = \widetilde{c}$, then $\mathcal{B}$ sets $\widetilde{r} = r$ and returns $\widetilde{K_I}$. If none of these cases occur, $\mathcal{B}$ selects h uniformly at random, returns h and updates the

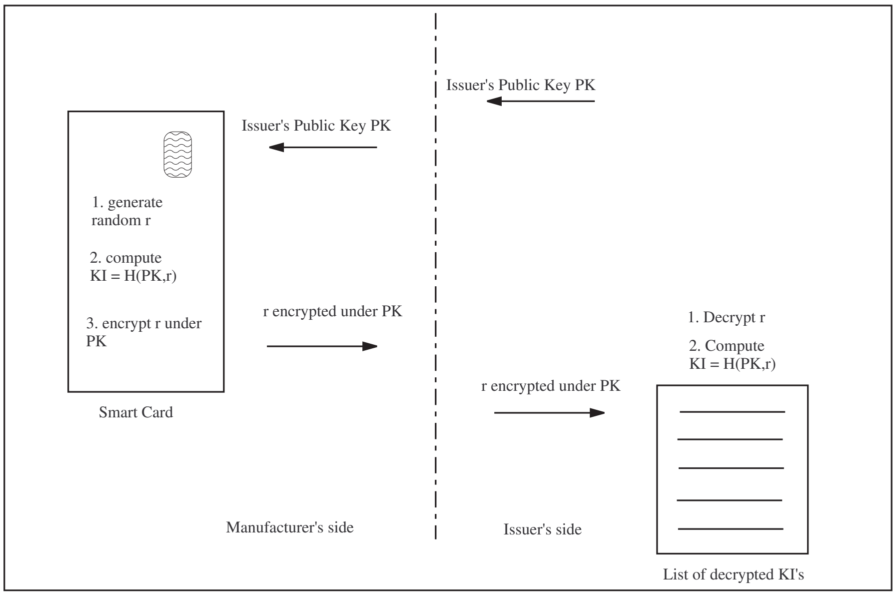

Fig. 3. Provably Secure Card Personalization Protocol

history of queries. Now when $\mathcal{A}$ has finished, $\mathcal{B}$ checks whether $\widetilde{r}$ was initialized during the game, simply returns $\widetilde{r}$ if so or fails otherwise. This completes the description of the reduction algorithm $\mathcal{B}$.

Since the simulation of H is perfect, it is clear that B is sound. We denote by Ask the event that $\mathcal{A}$ submits $\widetilde{r}$ to the simulation of H. Now if Ask never happens, $\widetilde{K_I}$ is a uniformly distributed random value unknown to $\mathcal{A}$, so

$$\Pr\left[\mathcal{A} = \widetilde{K_I} \mid \neg\mathsf{Ask}\right] \leq \frac{1}{\sharp H} \ ,$$

where $\sharp H$ denotes the number of elements in the output space of H. By assumption,

$$\begin{aligned} \epsilon &\leq \Pr\left[\mathcal{A} = \widetilde{K_I}\right] \\ &\leq \Pr\left[\mathcal{A} = \widetilde{K_I} \mid \neg\mathsf{Ask}\right] + \Pr\left[\mathsf{Ask}\right] \\ &\leq \frac{1}{\sharp H} + \Pr\left[\mathsf{Ask}\right] \end{aligned}$$

which yields

$$\begin{aligned} \epsilon' &= \Pr\left[\mathcal{B} = \widetilde{r}\right] \\ &= \Pr\left[\mathsf{Ask}\right] \\ &\geq \epsilon - 1/\sharp H \end{aligned}$$

Therefore, if ϵ is non negligible, ϵ' is non negligible either. □

Interestingly, we also get a slightly different result for *non deterministic* encryption schemes, i.e. when the protocol relies on a probabilistic encryption function $r \mapsto \mathcal{E}_{\mathsf{PK}}(r; u)$. We include this result here for the sake of completeness. We state:

Theorem 2 (Passive Attacks). *Assume that the probabilistic encryption scheme $\mathcal{E}_{\mathsf{PK}}$ is semantically secure under chosen plaintext attacks (IND-CPA). Then no polynomial time attacker given PK and $c = \mathcal{E}_{\mathsf{PK}}(r)$ can recover $K_I = H(r, \mathsf{PK})$ with non-negligible probability in the random oracle model.*

Proof. Here again, we assume the existence of the same attacker $\mathcal{A}$ with non negligible success probability ϵ and show how to distinguish encryptions $\mathcal{E}_{\mathsf{PK}}$ with non negligible advantage ϵ'. The reduction algorithm $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ is as follows. $\mathcal{B}_1$ (the find stage) chooses two distinct messages (r_0, r_1) uniformly at random and outputs them. Then $\mathcal{B}_2$ inputs $c_b = \mathcal{E}_{\mathsf{PK}}(r_b; u)$ for a certain bit b and random tape u. $\mathcal{B}$ must guess b with non negligible advantage.

To do this, $\mathcal{B}_2$ is designed as follows. $\mathcal{B}_2$ randomly selects $\widetilde{K_I}$ and runs $\mathcal{A}(\mathsf{PK}, c_b)$. Each time $\mathcal{A}$ queries the random oracle H for an input (r, pk), $\mathcal{B}_2$ checks in the history of queries if (r, pk) was queried by $\mathcal{A}$ in the past, in which case the same answer is returned to $\mathcal{A}$. Otherwise, if $pk = \mathsf{PK}$ and $r = r_b$ for $b \in \{0,1\}$, then $\mathcal{B}_2$ stops and output b. If none of these cases occur, $\mathcal{B}$ selects h uniformly at random, returns h and updates the history of queries. If $\mathcal{A}$ finishes, $\mathcal{B}$ stops, chooses $\beta \in \{0,1\}$ at random and returns β. This completes the description of the reduction algorithm $\mathcal{B}$.

The simulation of H is almost perfect. We denote by Good the event that $\mathcal{A}$ submits r_b to the simulation of H and by Bad the event that $\mathcal{A}$ submits $r_{\overline{b}}$ to the simulation of H. Now if neither Good nor Bad ever happens, $\widetilde{K_I}$ is a uniformly distributed random value unknown to $\mathcal{A}$, so

$$\Pr\left[\mathcal{A} = \widetilde{K_I} \mid \neg(\mathsf{Good} \vee \mathsf{Bad})\right] \leq \frac{1}{\sharp H} \ .$$

By assumption,

$$\begin{aligned}
\epsilon &\leq \Pr\left[\mathcal{A} = \widetilde{K_I}\right] \\
&\leq \Pr\left[\mathcal{A} = \widetilde{K_I} \mid \neg(\mathsf{Good} \vee \mathsf{Bad})\right] \\
&+ \Pr\left[\mathsf{Good} \vee \mathsf{Bad}\right] \\
&\leq \frac{1}{\sharp H} + \Pr\left[\mathsf{Good} \vee \mathsf{Bad}\right] \ .
\end{aligned}$$

Since the choice of (r_0, r_1) is independent from $\mathcal{A}$'s view, the probability that $r_{\overline{b}}$ is submitted by $\mathcal{A}$ to the random oracle H is upper bounded by $1/\sharp r$. Given that Good and Bad exclude each other, we get

$$\begin{aligned}
\Pr\left[\mathsf{Good}\right] &= \Pr\left[\mathsf{Good} \vee \mathsf{Bad}\right] - \Pr\left[\mathsf{Bad}\right] \\
&\geq \Pr\left[\mathsf{Good} \vee \mathsf{Bad}\right] - \frac{1}{\sharp r} \ .
\end{aligned}$$

Therefore

$$\begin{aligned}
\frac{1+\epsilon'}{2} &= \Pr\left[\mathcal{B} = b\right] \\
&= \Pr\left[\mathsf{Good}\right] \\
&+ \Pr\left[\neg(\mathsf{Good} \vee \mathsf{Bad}) \wedge \beta = b\right] \\
&= \Pr\left[\mathsf{Good}\right] + \frac{1}{2}\Pr\left[\neg(\mathsf{Good} \vee \mathsf{Bad})\right] \\
&= \frac{1}{2} + \Pr\left[\mathsf{Good}\right] - \frac{1}{2}\Pr\left[\mathsf{Good} \vee \mathsf{Bad}\right] \\
&\geq \frac{1}{2} + \frac{1}{2}\Pr\left[\mathsf{Good} \vee \mathsf{Bad}\right] - \frac{1}{\sharp r} \\
&\geq \frac{1}{2} + \frac{1}{2}(\epsilon - \frac{1}{\sharp H}) - \frac{1}{\sharp r} \ ,
\end{aligned}$$

and finally $\epsilon' \geq \epsilon - 1/\sharp H - 2/\sharp r$ as wanted. □

4.3 Security Proof Against Active Insiders

We now focus on security against active insiders. We have:

Theorem 3 (Active Attacks). *Assume the encryption scheme $\mathcal{E}_{\mathsf{PK}}$ is deterministic and one-way or probabilistic and semantically secure (under chosen ciphertext attacks). Then obtaining information about K_I requires the attacker to corrupt the value of* PK*. Then $K_I \neq H(r, \mathsf{PK})$ with overwhelming probability.*

Proof. Essentially, we follow the initial work of [6]. Suppose indeed, that the attacker does not alter the value of PK which is transmitted to the card. Two situations may occur:

1. either the insider corrupts the value of $c = \mathcal{E}_{\mathsf{PK}}(r)$ by changing it into c', but this of of no use whatsoever to her,
2. or she does not corrupt c; in this case, the insider is passive and theorem 1 or 2 applies, depending on $\mathcal{E}_{\mathsf{PK}}$. This means that no information about K_I can be obtained.

On the other hand, if the insider controls the PC and cheats on PK, she may recover

K_I by submitting another public key PK' but the issuer then gets a different value $H(r, \mathsf{PK}) \neq H(r, \mathsf{PK}')$ with overwhelming probability. Thus the card will not be functional and no damage (other than denial of service) will incur to the issuer. This provides evidence that either the protocol is correct, or the card will not function at all. □

4.4 Can Denial of Service Be Avoided?

What is desirable is that the protocol would preserve both key integrity and privacy under any attack circumstances, as this would thwart denial of service attacks discussed above. For theoretical reasons, however, no protocol can achieve such a better security level without an authenticated communication channel between the card and the issuer. The only cheap way to achieve authentication would consist in masking the issuer's public key PK into the read only memory (ROM) of the card. We would then reach both key integrity and privacy in any case. But we recall that denial of service does not serve the attacker's interests anyway because it precisely testifies the presence of an active attack during the personalization process.

4.5 Collusion attacks

An intuitive way to break the system would be to envision the collusion between a malicious insider and a malicious issuer. For example, the insider might substitute the genuine issuer's public key with the malicious issuer's public key. In this case, under the unusual assumption that both issuers use the same operating system on the card, the personalized cards would work on the malicious issuer's network whereas they would *not* work on the genuine network. Although this scenario theoretically exists, one cannot help wondering what benefit the malicious issuer could possibly get out of this setting. First, the cards are shipped to the initially intended recipients or more generally speaking directly to the user. Thus the malicious issuer will never get hold of the cards. Second, this issuer would then have cards in the field that can and will be used on his own network, but he could not plausibly recover any fees associated to this usage. So the users would simply (say) use wireless communication networks without paying a dime to the malicious operator.

Interestingly, we could also envision attacks combining an active intrusion with a partial or total access to the issuer's decryption server. This would allow the attacker to query the server for r-values of her choice given c, possibly excepting the ones that correspond to already listed K_I's (as this could cause some kind of collision detection by the server). This is exactly a chosen-ciphertext attack scenario and in this case, again, our protocol remains fully secure in the same sense, provided that the underlying encryption scheme $\mathcal{E}_{\mathsf{PK}}$ be OW-CCA or INC-CCA (instead of OW-CPA or IND-CPA). This is easily obtained as a natural extension of theorems 1 and 2. Then chosen-ciphertext secure encryption schemes like RSA-OAEP [1] or Cramer-Shoup [2] must be employed.

A denial of service attacker can always interact with the chip-card in such a way that in the end the card is invalid. But, as stressed before, we assume that this scenario is not of interest to an active adversary. We also stress the fact that more elaborate attacks where the complete set of employees of the manufacturer collude against the issuer are not considered in this paper. As an illustration, these include situations where the card's operating system itself is flawed or corrupted and does not fully respect the protocol.

In light of the above discussion, we believe that no other protocol can further enhance the one we propose in this setting, except if additional key authentication is implemented in some way or an other.

5 Practical Examples

5.1 An Example Using Low-Exponent RSA

We recall the protocol steps in this context, taking SHA-1 as an embodiment of H. First, the issuer generates an RSA key pair $(\mathsf{PK}, \mathsf{SK})$ where $\mathsf{PK} = (n, e)$ and $\mathsf{SK} = (n, d)$ with $n = pq$ for two large primes p and q, $e = 3$ for instance and $d = e^{-1} \mod (p-1)(q-1)$ (RSA key generation imposes that $\gcd((p-1)(q-1), e) = 1$). The manufacturer is given n and for each card to be personalized, engages the PC in the following protocol:

1. the PC transmits n to the card with identifier Id,
2. the card selects r uniformly at random and computes $K_I = \text{SHA-1}(r, n)$,
3. the card computes $c = r^3 \mod n$ and outputs c,
4. the PC collects the pair (Id, c) and sends it later to the issuer,
5. the issuer recovers $r = c^d \mod n$, computes $K_I = \text{SHA-1}(r, n)$ and stores the pair (Id, K_I).

Note that this is extremely efficient, as the card only performs a couple of modular multiplications and a single call to SHA-1. Moreover, we have the following security statement.

Corollary 1 (of theorems 1 and 3). *Assuming the random oracle model, under the RSA assumption, malicious insiders cannot retrieve the secret key K_I of a functional card.*

5.2 An Example Based on the Diffie-Hellman Problem

It is possible to adapt the above protocol in order to use the Decision Diffie-Hellman (DDH) as the underlying intractability assumption. This is done by choosing El-Gamal encryption [3] to instantiate $\mathcal{E}_{\mathsf{PK}}$ instead of RSA, as follows.

The issuer chooses an abelian group G, denoted multiplicatively, of large order q, in which the discrete logarithm is intractable. An elliptic curve defined over a finite field, or the group of integers modulo a large prime p are examples of such a group. The issuer then chooses a base $g \in G$, a random integer $1 < x < q$, stores $\mathsf{SK} = x$ and transmits $\mathsf{PK} = (g, g^x) := (g, h)$. The personalization process now works as follows:

1. the PC transmits the issuer's public-key (g, h) to the card with identifier Id,
2. the card selects r uniformly at random and computes the pair (g^r, h^r),
3. the card computes $K_I = \text{SHA-1}(h^r, g, h)$, memorizes K_I in non-volatile memory and outputs g^r,
4. the PC sends the pair (Id, g^r) to the issuer, who later recovers K_I by computing $K_I = \text{SHA-1}((g^r)^x, g, h)$.

In this case, we get the following security result.

Corollary 2 (of theorems 2 and 3). *Assuming the random oracle model, under the DDH assumption, malicious insiders cannot retrieve the secret key K_I of a functional card.*

6 Conclusion

We have presented a simple provably secure protocol which enables a smart-card manufacturer to act as a trusted personalization center without knowing any secret data belonging to the issuer. The proposed solution does not require a public-key infrastructure, and avoids all the secret-key management procedures usually required to guarantee the security of the personalization process.

Acknowledgments

We thank Jacques Stern for his help on the initial version of the security proofs and anonymous reviewers for their comments and suggested improvements of the paper.

References

1. M. Bellare and P. Rogaway. Optimal asymmetric encryption. In A. De Santis, editor, *Advances in Cryptology – EUROCRYPT '94*, volume 950 of *Lecture Notes in Computer Science*, pp. 92–111, Springer-Verlag, 1995.
2. R. Cramer and V. Shoup. A Practical Public-Key Cryptosystem Provably Secure Against Adaptive Chosen-Ciphertext Attacks. In *Advances in Cryptology – CRYPT0 '98*, volume 1462 of *Lecture Notes in Computer Science*, pp. 13–25, Springer-Verlag, 1998.
3. T. El Gamal. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms In IEEE Trans. Inform. Theory, vol. 31, pp. 469–472, 1985.
4. R. Rivest and A.Shamir, How to expose an Eavesdropper, Communication of the ACM, v.27, n.4, Apr. 1984, pp. 393–395
5. R. Rivest, A. Shamir and L. Adleman, A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. In Communications of the ACM, vol. 21, n 2, p.120–126, February 1978.
6. J. Stern, Analysis of a Secure Chip-card Personalization Protocol. Unpublished Manuscript, January 2001.
7. US Department of Commerce, N.I.S.T. Secure Hash Algorithm. In FIPS 180-1, 1995.