Identity Card Key Generation in the Malicious Card Issuer Model

Arnis Parsovs

May 27, 2014

Abstract

The generally accepted practice, where the electronic identity card issuer generates private keys on behalf of the cardholder, allows for a malicious card issuer to compromise the security of cardholder's private keys. In this report we discuss several solutions for improving a cardholder's private key security against a malicious card issuer.

1 Introduction

Governments in several countries around the world issue Electronic Identity Cards (eID cards) to their residents. These cards are smart cards which usually contain RSA key pairs that are bound to a resident's identity and allow the resident to authenticate in online services, sign electronic documents with a digital signature, and perform encrypted communication between residents.

The security of an eID card depends on the cardholder being the only one who can use the private key which has been bound to their identity. Therefore, the private key is usually kept in a single copy inside a tamper resistant smart card and the private key operations are performed only after the cardholder has authenticated theirself to the smart card using a symmetric PIN code shared only between the cardholder and smart card.

In practice, the card issuer (usually the government or its delegated party) provides cardholders with fully personalized smart cards that contain an RSA key pair, an X.509 public key certificate and PIN codes generated for the cardholder.

By applying security threat modeling for smart cards [1] we see that since the card issuer in this case is not the data owner, the reliance on the issuer in providing security for a cardholder's private key opens up the possibility for a malicious card issuer to launch attacks against the cardholder.

The designers of the eID card presuppose that the card issuer holds the best interests of the cardholder. This, however, might not necessarily be the case, and as we know from security economics, where the party who is in a position to protect a system is not the party who would suffer the results of security failure, problems may be expected [2].

One such problem was recently found in the Taiwanese eID card. The hardware random-number generator built in the smart card was fatally flawed, allowing anyone to recover private keys stored on these cards [3].

In this report we will discuss possible solutions that could provide a cardholder's private key improved security even in the case of a careless or malicious card issuer. To simplify our analysis, we will assume that the card issuer, the card manufacturer, and the Certificate Authority (CA) issuing the certificates is the same, possibly malicious party.

2 Attacks by the issuer against the cardholder

The most straightforward attack against a cardholder is to compromise the secrecy of the cardholder's private key. As the card issuer is the one who produces the smart card and generates the keys for the cardholder, compromising a cardholder's private key is trivial. In the simplest form of the attack the issuer simply saves a copy of the cardholder's private key before loading it into the smart card.

To avoid such accusations, the card issuers usually claim that the key pair is generated on the card using the built-in secure hardware random number generator and that there is no way for a private key to ever leave the smart card. These claims are usually attested by independent smart card security certification, such as Common Criteria or FIPS 140.

Unfortunately, there is no way for anyone other than the card issuer to ensure that the chip and the software that has passed the certification is the one that is used in the card. Furthermore, as we learn from the Taiwanese eID card case, the certified card may allow insecure key generation with "FIPS mode" disabled [3] and only the issuer will know whether this was the case.

It is possible to argue that we have to trust the issuer anyway, since a malicious issuer can always impersonate the cardholder even without compromising his private key, for instance, by unlawfully issuing and using a certificate that cointains the name of the cardholder. However, while the use of a mis-issued certificate can be traced back to the CA with cryptographic precision, there is no way to trace back who holds a copy of the private key which has been abused. Under the European Directive 1999/93/EC (see Article 6 in [4]) the CA is liable for actions performed with a mis-issued certificate. Therefore, while a malicious card issuer can indeed temporarily impersonate the cardholder, the eventual discovery of misuse is a deterrent for the card issuer from mis-issuing certificates. Unfortunately, there is no effective attribution measure that would deter the issuer from abusing a cardholder's private keys.

3 Possible solutions

3.1 Private key generated by the cardholder

The simplest solution would be to completely remove the issuer's role in the cardholder's private key generation and storage, making it the sole responsibility of the cardholder (as is practiced when issuing TLS server certificates). The cardholder could generate the private key using his own media and include the corresponding public key in the application form when applying for the certificate. This, however, would be too complicated for most of the cardholders and the corresponding private keys would likely end up being stored in an unprotected media which would be against our original intention to improve the private key security. In fact, the European Directive 1999/93/EC requires the signature-creation-data to be stored in a secure signature-creation device, which must ensure that the signature-creation-data can practically occur only once and signature generation can be reliably protected by the legitimate signatory against the use of others (see ANNEX III in [4]). Therefore, storing the private key somewhere other, than in a tamper resistant device, would be against the directive.

A straightforward improvement would be to ship the smart cards without the private keys loaded, but with the cardholder initiating the key generation on the first use of the eID card. This would guarantee that the keys are generated on the card. However, a malicious issuer can make the random number generator predictable (for example, by seeding PRNG with the card's serial number) or even generate the RSA key pair such that the private key can be efficiently reconstructed knowing only the public key [5].

3.2 Use of two keys

A desirable security level against a malicious card issuer would be achieved if the cryptographic operations performed by the cardholder would require use of two keys – one generated and stored on a smart card produced by the issuer, and another generated and stored on any medium owned by the cardholder. The digital signature would be considered valid only if data has been signed using both keys. Similarly, for hybrid encryption the symmetric transport key could be split into two parts using a secret sharing scheme and both parts encrypted using distinct public keys. This would prevent a malicious issuer from abusing the private key that has been generated for the cardholder and would protect the cardholder if anyone (except the issuer) has compromised his carelessly stored private key.

While optimal, the solution is not practical, since it requires significant changes in the user interface, standards, and protocols as they are used today.

3.3 Threshold RSA

The same security benefits as in the two key use scenario can be achieved using a single key, but by performing key generation and private key operations in a distributed manner. In this scenario, as the RSA public key would be generated by the issuer's smart card and the cardholder's device, no single device could perform private key operations without collaborating.

The first two-party RSA protocol has been suggested in [6] by Gilboa. Straub in [7] proposed a more efficient protocol that generates 3-prime RSA. Recently, Hazay et al. [8] proposed the first protocol secure against an active adversary. The semi-honest protocol implementation of [8] described in [9] shows that a 2048-bit RSA key can be generated in 15 minutes on average on the Intel Core i5 dual core 2.3 GHz CPU. Unfortunately, if one of the parties is a smart card with low computational resources, the estimated generation time would probably reach several days, which makes this approach undeployable in practice.

Even if it was efficient, the solution still requires the private key share to be stored on the cardholder's computer. This creates portability issues and has a risk that the eID solution becomes unusable if the cardholder's computer becomes unusable.

3.4 Abuse-free RSA key generation

In this solution the private key is fully stored on the smart card provided by the card issuer. However, the private key is generated in an abuse-free way in cooperation with the cardholder's computer, such that neither the issuer nor the cardholder can learn the private key.

In this section we will discuss several protocols which can be used to achieve this. It is important to note, however, that this approach compared to the previous solutions, does not give the same protection against a malicious issuer.

A malicious issuer can deliberately weaken the smart card or implement backdoors that will leak the private key through various side channels. In the simplest form of the attack the smart card can simply return the cardholder's private key to the terminal if the terminal sends a special command to the card. Fortunately, these attacks require the card issuer to be in direct contact with the card after the key has been generated. In the current eID card use case the terminal is usually owned by the cardholder (terminal being the cardholder's card reader and computer) and the card is rarely[1] inserted in a terminal that is under the card issuer's control.

A malicious smart card could also try to use a signature as a covert channel to leak the private key. However, RSA PKCS#1 v1.5 signing scheme uses deterministic padding[2], thus the only place where the private key can be encoded would be the message itself, which would result in a failure to verify the signature of the signed document.

[1] One such case is the PIN escrow mechanism provided by the issuer which allows the issuer to reset a card's PIN codes in case the cardholder has forgotten them. The procedure is performed after the cardholder's identity has been verified at the issuer's customer service point.

[2] The less popular padding scheme RSA-PSS uses random padding, which could be used by a smart card to leak the private key.

Therefore, while the abuse-free RSA key generation approach does not protect against a malicious issuer who can obtain direct access to the card later in the smart card lifecycle, the use of this approach will eliminate the risk of private key compromise that is caused by intentional or unintentional randomness flaws in the smart card. This will require a malicious issuer to carry out a more sophisticated attack, increasing the conspiracy level required and thus the risk of being caught.

3.4.1 Threshold RSA

We can obtain abuse-free key generation immediately by using protocols described in the threshold RSA solution, since by definition threshold RSA generates private keys in an abuse-free manner. After the key is generated using the threshold RSA protocol, the cardholder's computer simply sends his private exponent share to the smart card.

In the following subsections we will discuss abuse-free key generation approaches that do not require distributed private key operations and therefore are more efficient than the two party threshold RSA protocols.

3.4.2 Verifiable randomness

In these protocols the smart card would prove to the cardholder that the randomness received from the cardholder was mixed with the smart card's randomness to generate unpredictable RSA factors. The first generic protocol was suggested by Desmedt in [10]. There, the cardholder's computer would send randomness to the smart card, and the smart card, using commitments and zero-knowledge proofs, would prove that the randomness from the cardholder's computer was used to generate the RSA modulus in an abuse-free way.

A specific protocol was proposed by Juels-Guajardo in [11]. The work [12] estimates that the Juels-Guajardo protocol would likely take over 40 minutes to execute on a home router, which likely would take several days on a low performance smart card, making the protocol impractical for our purpose.

3.4.3 Multi-prime RSA

In this protocol, proposed by the author, the cardholder's computer and smart card would generate a 4096-bit 4-prime balanced RSA modulus, which would be constructed from four 1024-bit prime factors (p_1, q_1, p_2, q_2). The card issuer would the ship the smart card containing p_1, q_1 and $e = 65537$[3]. On the first eID use, the following protocol would be executed between the cardholder's computer and the smart card[4].

1. The smart card sends $p_1 \cdot q_1$ (or it's commitment) to the cardholder's computer.

[3]Public exponent e is fixed to prevent it from being used as a covert channel.
[4]Note that this does not require any interaction from the cardholder.

2. The cardholder's computer generates p_2, q_2, such that $gcd(p_2 - 1, e) = 1$ and $gcd(q_2 - 1, e) = 1$, and sends it to the smart card.

3. The smart card calculates n, d and returns n to the computer, revealing the committed value from step 1.

4. The cardholder's computer will verifies whether $p_2 \cdot q_2$ divides n without a remainder to ensure that the modulus contains the cardholder's factors.

5. The cardholder's computer verifies whether $\frac{n}{p_2 \cdot q_2}$ is equal to the $p_1 \cdot q_1$ received in the first step to ensure that the smart card's factors p_1, q_1 do not depend on the cardholder's p_2, q_2.

6. The cardholder's computer sends n to the card issuer.

7. The issuer verifies whether the $p_1 \cdot q_1$ stored in the issuer's database divides n without a remainder (to ensure that the modulus contains the issuer's factors – this prevents the cardholder from obtaining a certificate for a private key that does not reside on the smart card).

8. The issuer returns an X.509 certificate to the cardholder's computer.

9. The cardholder's computer loads the X.509 certificate into the smart card.

As can be seen from the protocol, the card issuer is unable to learn the factorization of $p_2 \cdot q_2$, and in turn the cardholder is not able to learn the factorization of $p_1 \cdot q_1$. Therefore, for the malicious issuer, the cardholder or anyone who has compromised the cardholder's computer (except the issuer), the effective security of the private key is equivalent to 2048-bit RSA, while for all other attackers the private key has 4096-bit RSA security (see Section 2 in [13]).

The protocol involves only a few simple calculations, however, the drawback being the rather large size of the public key – 4096 bits. Compared to 2048-bit RSA, the 4096-bit RSA public key operations are approximately 4 times slower, while private key operations are only 2 times slower if the Chinese remainder theorem is used.

4 Conclusion

In this report we have discussed several solutions that could improve a cardholder's private key security against a malicious card issuer. While practically deployable solutions do not provide full security against a malicious issuer, they do eliminate private key compromise caused by unintentional or intentional random number generator flaws and thus require more sophisticated attacks by the card issuer.

References

[1] Bruce Schneier and Adam Shostack. Breaking Up is Hard to Do: Modeling Security Threats for Smart Cards. In *Proceedings of the USENIX Workshop on Smartcard Technology on USENIX Workshop on Smartcard Technology*, WOST'99, pages 19–19, Berkeley, CA, USA, 1999. USENIX Association.

[2] R. Anderson. Why information security is hard - an economic perspective. In *Computer Security Applications Conference, 2001. ACSAC 2001. Proceedings 17th Annual*, pages 358–365, Dec 2001.

[3] Daniel J. Bernstein, Yun-An Chang, Chen-Mou Cheng, Li-Ping Chou, Nadia Heninger, Tanja Lange, and Nicko van Someren. Factoring RSA Keys from Certified Smart Cards: Coppersmith in the Wild. In Kazue Sako and Palash Sarkar, editors, *ASIACRYPT (2)*, volume 8270 of *Lecture Notes in Computer Science*, pages 341–360. Springer, 2013.

[4] Directive 1999/93/EC of the European Parliament and of the Council of 13 December 1999 on a Community framework for electronic signatures. Official Journal L13/12, 1999.

[5] Claude Crpeau and Alain Slakmon. Simple Backdoors for RSA Key Generation. In Marc Joye, editor, *Topics in Cryptology CT-RSA 2003*, volume 2612 of *Lecture Notes in Computer Science*, pages 403–416. Springer Berlin Heidelberg, 2003.

[6] Niv Gilboa. Two Party RSA Key Generation. In *In Crypto 99, LNCS 1666*, pages 116–129. Springer-Verlag, 1999.

[7] Tobias Straub. Efficient Two Party Multi-Prime RSA Key Generation. In *IASTED International Conference on Communication, Network, and Information Security*, 2003.

[8] Carmit Hazay, GertLsse Mikkelsen, Tal Rabin, and Tomas Toft. Efficient RSA Key Generation and Threshold Paillier in the Two-Party Setting. In Orr Dunkelman, editor, *Topics in Cryptology CT-RSA 2012*, volume 7178 of *Lecture Notes in Computer Science*, pages 313–331. Springer Berlin Heidelberg, 2012.

[9] Angelo Agatino Nicolosi. *Efficient RSA Key Generation Protocol in a Two-Party Setting and its Application into the Secure Multiparty Computation Environment*. Master thesis, University of Århus, 2011. `https://users-cs.au.dk/amenuor/AngAgaNic-MasThes.pdf`.

[10] Yvo Desmedt. Abuses in Cryptography and How to Fight Them. In Shafi Goldwasser, editor, *Advances in Cryptology CRYPTO 88*, volume 403 of *Lecture Notes in Computer Science*, pages 375–389. Springer New York, 1990.

[11] Ari Juels and Jorge Guajardo. RSA Key Generation with Verifiable Randomness. In David Naccache and Pascal Paillier, editors, *Public Key Cryptography*, volume 2274 of *Lecture Notes in Computer Science*, pages 357–374. Springer Berlin Heidelberg, 2002.

[12] Henry Corrigan-Gibbs, Wendy Mu, Dan Boneh, and Bryan Ford. Ensuring High-quality Randomness in Cryptographic Key Generation. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, CCS '13, pages 685–696, New York, NY, USA, 2013. ACM.

[13] M. Jason Hinek. On the security of multi-prime RSA. *Journal of Mathematical Cryptology*, 2(2):109–207, 2008.