Nuances of the JavaCard API on the cryptographic smart cards – JCAlgTest project

Petr Švenda

Masaryk University, Czech Republic
`svenda@fi.muni.cz`

Abstract

The main goal of this paper is to describe the JCAlgTest project. The JCAlgTest project is designed for an automatic testing of the degree of support for algorithms listed in the JavaCard API specification by a given smart card. Discussion of the results and trends drawn from the database of 28 different cryptographic smart cards is performed. Additional information about particular algorithm or protocol obtained via timing and power analysis is also described and (partially) available in JCAlgTest database.

1 Introduction

The three main open multi-application platforms for smart cards currently exist - MULTOS, .NET for smart cards and JavaCard. We will focus only to JavaCard platform with corresponding JavaCard API available from Oracle [1]. Oracle provides (free of charge) specifications of the JavaCard API (including cryptographic packages), development kits and specifications of JavaCard virtual machine including specification of JavaCard bytecode. The most recent version of specification is JC 3.0.4 (published in September 2011), while the the oldest specification one can obtain from the Oracle web page is JC 2.1 (published in June 1999). The specifications starting from JC 3.x provides two significantly different APIs – classic and connected edition. The classic edition continues to extend JavaCard API with the new cryptographic algorithms in a style of JC 2.x versions, whereas connected edition perceives smart card as a server providing web service interface (see [5] for tutorials and complementary materials).

When cryptographic smart card is planned for use as a trusted component in a designed system, several necessary cryptographic algorithms are required from the selected smart card. Although JavaCard API itself covers most of the commonly used cryptographic algorithms, no actual smart card implements the JavaCard API in its full spectrum. Because cryptographic functions usually needs to be hardware-accelerated in smart cards, set of the supported algorithms is usually fixed in production and cannot be easily extended or only with a significant performance overhead. A designer of the system should therefore search for suitable card with a proper support. However, manufacturers sales brochure usually shows only supported JavaCard version (e.g., JC 2.2.2), but not all actually implemented algorithms. Higher level information like support for RSA-2048 is usually listed, but without information about supported padding methods or supported combinations with underlaying hash functions. More detailed information can be obtained from the third party security certification reports like NIST FIPS 140 [6], but still without exhaustive listing of the supported algorithms. Moreover, only very rough information about a smart card performance is usually provided, if any at all.

This paper introduces automated testing framework JCAlgTest together with a detailed results for the database of 28 widely used smart cards. Based on results, trends in the range of supported algorithms are discussed in Section 1.1. The section 1.2 presents the performance comparison for selected cryptographic algorithms, providing partial insight into the implementation of particular algorithm. This insight is further extended via power analysis of smart card execution as described in Section 1.3. Finally, possibility for extension of supported algorithms is discussed in the Section 1.4, including how to use one supported algorithm to construct another unsupported one.

1.1 Methodology and results

The current version of JavaCard specification contains several main categories of algorithms (Cipher, Signature, MessageDigest, RandomData, KeyBuilder, KeyPair, KeyAgreement, Checksum...) and more than 200 different algorithms or their parametrization by key length etc. A manual testing for the support of a given algorithm is therefore not a viable option. The JCAlgTest project we developed [8] instead tries to create an instance of every single algorithm and reports result to the controlling computer for the post-processing. The testing itself relies on the exception handling (supported by JavaCard), can be automated and provides possibility to test all algorithms in a reasonable time (several minutes). Details can be found at JCAlgTest project web page. It requires only to upload testing applet to the target smart card and run the complementary application.

The JCAlgTest project was started in 2008 and over the time accumulated results from the 28 different smart cards with around half coming from our laboratory and rest provided by the volunteers worldwide. The results can be found on project web page in the form of csv files for every card or large post-processed html table [8].

Certain trends can be observed (for rough categorization, cards obtained before year 2008 are referred as *older*, cards obtained after year 2008 as *the newer* and cards after 2012 as *the newest*):

- Hardware generator of truly random numbers (ALG_SECURE_RANDOM), 3DES algorithm (ALG_DES_CBC_NOPAD), RSA with 1024bit keys (ALG_RSA LENGTH_RSA_1024), SHA-1 (ALG_SHA) and MD5 (ALG_MD5) algorithms are always supported even by the old cards.
- AES-128/196/256 (TYPE_AES LENGTH_AES_128/196/256) and RSA with 2048bit keys (ALG_RSA LENGTH_RSA_2048) are supported by all newer cards.
- Korean SEED encryption algorithm (ALG_KOREAN_SEED_CBC_NOPAD) according to RFC 4269 is usually supported by the newer cards.
- SHA2-256 (ALG_SHA_256) is usually supported by the newer cards with support for SHA2-384/512 (ALG_SHA_384, ALG_SHA_512) being provided only by the newest cards.
- RSA in an encryption mode is almost always supported as plain RSA (ALG_RSA_NOPAD) or formatted according to PKCS#1 (ALG_RSA_PKCS1). The newer cards also support OAEP mode (ALG_RSA_PKCS1_OAEP). RSA algorithm in signature mode is almost always supported with formatting according to PKCS#1 (ALG_RSA_SHA_PKCS1) and ISO/IEC 9796 modes (ALG_RSA_SHA_ISO9796).
- On-card key generation is almost always supported (both for symmetric and asymmetric cryptography algorithms), when given key length is also supported.
- The newest cards commonly supports RSA only in CRT mode (ALG_RSA_CRT, speedup based on Chinese remainder theorem) whereas older cards usually supported both. As a naïve implementation of CRT is known to be vulnerable to fault induction attack [2],

support only for ALG_RSA_CRT mode might signalize presence of the strong defense mechanism against this type of attack without necessity to support also slower RSA without CRT speedup.

- The newer cards commonly supports elliptic curves cryptography, either as EC operations over fields of characteristic 2 with polynomial basis (TYPE_EC_F2M_PRIVATE) or more commonly as EC operations over large prime fields (TYPE_EC_FP_PRIVATE). Key lengths 128, 160 and 192 are widely supported, with newest cards offering support of key lengths up to 384 bits (LENGTH_EC_FP_384).
- EC-based Diffie-Hellman key agreement protocol (ALG_EC_SVDP_DH) according to IEEE P1363 is usually supported by the newer cards.

1.2 Performance analysis

By measuring the time necessary to finish an execution of a given algorithm running on card, interesting statistics can be obtained. Overhead related to data input and output transfer to and from smart card must be taken into account, as well as overhead inherent to PC/SC subsystem. We are currently finishing the implementation of an extension to JCAlgTest for a fully automatic performance testing including generation of graphical figures as demonstrated on Figure 1. Note that a very precise time measurement can be obtained from the power consumption traces as discussed in Section 1.3, but with the necessity of an additional hardware.

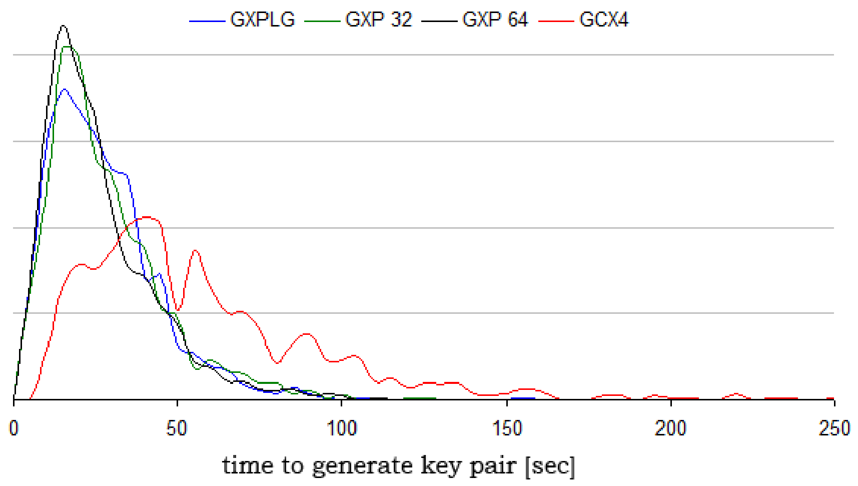

Figure 1: The histogram of the distribution of times required to generate single key pair for RSA with 2048 bits key length [4]. Histogram of times is computed from 1024 randomly generated key pairs on every tested card. Note that the generation time for a given card is not constant, as randomly generated candidate values for p and q of RSA algorithm must be tested being primes; if prime test fail, candidate value is discarded and eventually generated again, increasing overall time necessary to generate a key pair. Interestingly, results for the card Gemalto GXP4 are around two times slower (on average), yet this card has the fastest random number generator and also the fastest CPU (from the four displayed cards) – better prime testing is probably used or additional security mechanism is applied.

1.3 Power analysis as an auditing tool

Power analysis of smart cards via oscilloscope is now well known attack vector [3]. Usually used to recover a side-channel information about a manipulated secret data, it can be also used as a very precise time-measurement method if measured operation can be isolated in the resulting power trace. Additionally, power analysis can be used as an auditing tool to inspect an implementation of the protocols when the source code is otherwise unavailable. We are continuously building database of the power traces corresponding to algorithms or protocols executed on given smart card to further increase an information available via the JCAlgTest project database. When such database is created, the reverse engineering of the main parts of an unknown applet executed on the smart card of known type profiled before can be performed using suitable pattern matching algorithm. As show on Figure 2, one can detect, where authentication cryptogram is computed or verified and what other actions are taken. Whereas extracting secret key is very difficult for the modern smart cards, identifying main cryptographic operations in the power trace is usually much easier.

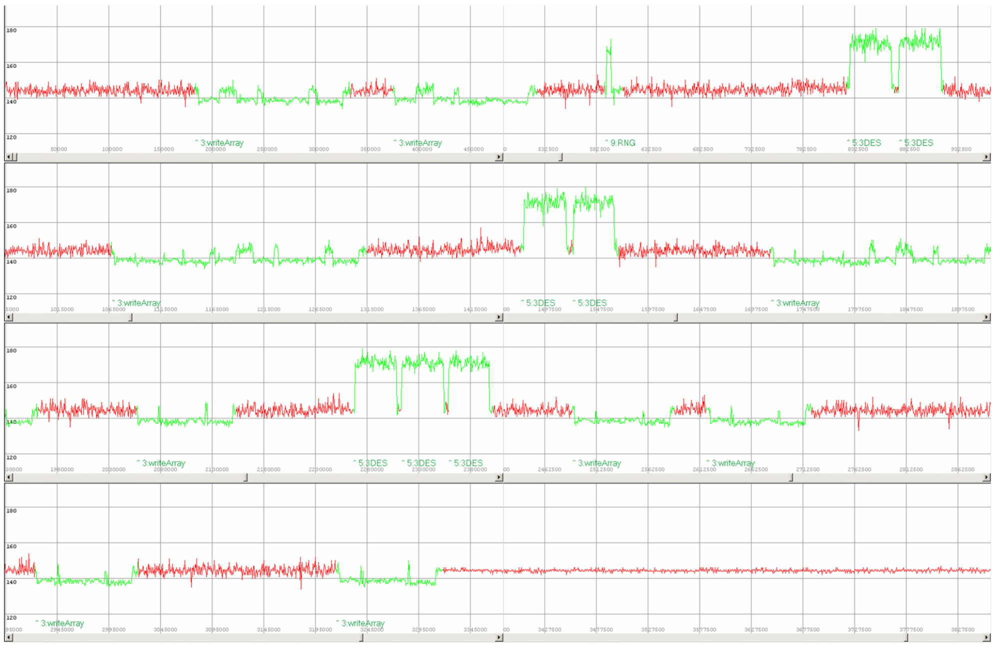

Figure 2: A power trace with INIT_UPDATE operation during the secure channel SCP'01 establishment on Gemalto GemXpresso Pro-R3 E64 PK card. Parts of the power trace can be directly mapped to a sequence of operations generating a card challenge, a derivation of the session keys and a computation of a card authentication cryptogram.

1.4 Alternative usage of API – supporting unsupported

When missing support for the required algorithms is detected, one can add missing features programmatically as JavaCard allows developer to write own code running at smart cards. However, additional performance and memory overhead is usually introduced. Three main options are possible:

A complete reimplementation of a missing algorithm – algorithm is executed in JavaCard VM instead of a hardware-accelerated coprocessor. E.g., AES-128 can be added [7], but runs usually around two orders of a magnitude slower (e.g., 10ms vs. 1000ms for AES on Gemplus GXP E64 PK card).

Only a part of a code is reimplemented (usually some formatting of input/output data) – with a main part still executed by a hardware-accelerated coprocessor. E.g., ALG_HMAC_SHA1 is not supported by any of the tested cards, but can be easily constructed by formatting input block for HMAC in JavaCard and then calling a hardware-accelerated hash algorithm (ALG_SHA1). This approach works well when the construction of new operations requires only formatting and usage of an already supported algorithm.

An implementation of a new algorithm by alternative usage of an existing algorithm – e.g., although cards must support fast hardware-accelerated modular exponentiation necessary for the RSA algorithm internally, JavaCard API was not exposing this important operation directly to a developer (situation changed with the introduction of BigInteger type, but is still largely currently unsupported). Yet if RSA in the plain mode (ALG_RSA_NOPAD) is supported and card itself supports setting of the arbitrary values of private or public exponent and modulus, this operation can be still exposed as a single RSA operation over the user supplied data. Fast implementation of Diffie-Hellman algorithm can be constructed, even when implementation of the modular exponentiation in software-only (JavaCard VM) would be prohibitively slow (at least 4-5 orders of magnitude). The JCAlgTest has a specific test for this useful feature and the newer cards are usually supporting this plain RSA "trick".

1.5 How to contribute

The JCAlgTest project is an open-source project (visit https://github.com/petrs/JCAlgTest for further instructions) and relies on the volunteers for the further extension of the database of results. You can contribute by a) downloading JCAlgTest, scanning your card and providing back results; b) helping to extend and improve the code (Github issues) or suggesting new features; c) spreading the word – let other knows that JCAlgTest project exists and results are available.

References

[1] Oracle corp. Java card technology. `http://www.oracle.com/technetwork/java/javame/javacard/index.html`, 2014.

[2] Marc Joye, Arjen K. Lenstra, and Jean jacques Quisquater. Chinese remaindering based cryptosystems in the presence of faults. *Journal of Cryptology*, 12:241–245, 1999.

[3] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *Advances in CryptologyCRYPTO99*, pages 388–397. Springer, 1999.

[4] Karol Kubanda. Comparison of the speed of cryptographic operations running on smartcards. Master's thesis, Masaryk University, Czech Republic, 2008.

[5] CROCS laboratory. Javacard applet development. `https://minotaur.fi.muni.cz:8443/~xsvenda/docuwiki/doku.php?id=public:smartcard:javacardcompilation`, 2014.

[6] NIST. Validated fips 140-1 and fips 140-2 cryptographic modules. `http://csrc.nist.gov/groups/STM/cmvp/documents/140-1/140val-all.htm`, 2014.

[7] Petr Švenda. Cryptographic algorithms re-implementation for javacard. `http://www.fi.muni.cz/~xsvenda/jcalgs.html`, 2014.

[8] Petr Švenda. JCAlgtest – javacard algorithm support testing project. `http://www.fi.muni.cz/~xsvenda/jcsupport.html`, 2014.